MQSeries® Integrator for OS/390®

System Management Guide

Version 1.1

SC34-5748-00

Note: Before using this information, and the product it supports, be sure to read the general information under *Notices* on page 225**.**

First edition (December 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.

Contents

Chapter 1
Introduction

The System Management Guide for OS/390 is for those persons responsible for MQSeries Integrator administration. The system administrator should have an overall understanding of MQSeries Integrator 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, who administers the databases interacting with MQSeries Integrator, and a network administrator, who ensures that network communications are set up to include MQSeries Integrator.

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

MQSeries Integrator Overview

MQSeries Integrator, from IBM and New Era of Networks, Inc. (NEON), provides the flexibility and scalability that allows true application integration. MQSeries Integrator consists of four components:

- IBM MQSeries
- NEONFormatter
- NEONRules
- MQIntegrator Rules daemon

MQSeries Integrator is a cross-platform, guaranteed delivery, messaging middleware product designed to facilitate the synchronization, management, and distribution of information (messages) across large-scale, heterogeneous networks.

MQSeries Integrator is configurable and uses a content-based rules message evaluation, formatting, and routing paradigm. MQSeries Integrator also provides a powerful data content-based, source-target mechanism with dynamic format parsing and conversion capability.

The application program interfaces (APIs) and graphical user interfaces (GUIs) allow you to use these systems. Refer to the ***Programming Reference*** documents for instructions on using the APIs and the ***User's Guide*** for instructions on using the GUIs.

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. When a message is provided as input to Formatter, the message is parsed and data values are returned.

Message formats in the NEONFormatter database are defined through the graphical user interface (GUI). The GUI leads you through the definitions of format components, for example, tags, delimiters, and patterns, to the building of complete message definitions.

Rules

NEONRules lets you develop rules for managing message destination IDs, receiver locations, expected message formats, and any processes initiated upon message delivery. Creation and dispatch of multiple messages to multiple destinations from a single input message is supported.

Note:
For more in-depth descriptions of NEONFormatter and NEONRules, refer to the overviews in Chapters 3 and 4 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.

MQSeries

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

Product Documentation Set

The MQSeries Integrator for OS/390 documentation set includes:

- ***MQSeries Integrator for OS/390 Installation and Configuration Guide*** details the installation and initial implementation of MQSeries Integrator and the MQSeries Integrator applications.
- ***User's Guide*** helps MQSeries Integrator users understand and apply the program through its graphical user interfaces (GUIs).
- ***System Management*** is for SPs and DBAs who work with MQSeries Integrator on a day-to-day basis.
- ***Programming References*** are intended for users who build and maintain the links between MQSeries Integrator and other applications. This document includes the following volumes:
  - ***Application Development Guide*** assists programmers in writing applications that use MQSeries Integrator APIs.
  - ***Programming Reference for NEONFormatter*** is a reference to NEONFormatter APIs for those who write applications to translate messages from one format to another.
  - ***Programming Reference for NEONRules*** is a reference to NEONRules APIs for those who write applications to perform actions based on message contents.

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/

Before You Contact Technical Support

If you have difficulty executing one of the MQSeries Integrator programs, analyze your environment using the following steps. Be prepared to send the listed information and files to technical support.

1. Has this program ever worked in your environment?

   If so, identify what has changed.

2. Check the values specified in the SQLSVSES (DD-name SQLSVSES) file that the failing job is using to make sure it refers to an existing DB2 subsystem and an existing DB2 database within that subsystem.

3. Check the values specified in the CLIINI (DD-name DSNAOINI) file that the failing job is using to make sure it refers to an existing DB2 subsystem and an existing DB2 database within that subsystem.

4. Check whether the System Affinity is causing your job to execute on a system that does not contain the DB2 subsystem, MQSeries queue manager, or IBM datasets that MQSeries Integrator is trying to access.

5. In the CLIINI file (DD-name DSNAOINI), edit the following line:

   ```
   CLITRACE=0
   ```

   Change it to:

   ```
   CLITRACE=1
   ```

   Rerun your job. The CLITRACE produced (DD-name CLITRACE) is invaluable in diagnosing problems between the DB2 database and the MQSeries Integrator application. Your JCL should have a DD-statement that defines CLITRACE to either a disk file or SYSOUT class. This file is required by technical support to diagnose problems.

Note:
It is assumed that the DB2 CLI is installed, the DSNACLI Plan has been bound, and you are granted execute authority on it.

6. Examine all files produced by MQSeries Integrator for error or informational messages. Some error messages are written to SYSOUT, some to SYSPRINT, and some to STATLOG.
7. Look for Operating System messages that may indicate why the job has failed, such as missing files, no room to log messages (E-37, B-37 type failures), full queue conditions, and so on.
8. If failing to put or get from an MQSeries queue, make sure the queue is enabled for sharing:

   ```
   Permit shared access . . . . Y  Y=Yes,N=No
   Default share option . . . . S E=Exclusive,S=Shared
   ```

9. If the problem is related to poor Rules daemon performance, check the values of the timers specified in the input stream (DD-name SYSIN) file of the RULENG job. Setting these timers too high can result in poor performance of the Rules Engine.

When contacting technical support be prepared to send the following information via email or ftp:

- The complete listing of your jobs execution, including SYSOUTs, SYSPRINTs, STATLOG, JESMSGS, and so on.
- The contents of the CLITRACE file
- Any dump files produced (CEEDUMP or SYSUDUMP)
- Your site's SQLSVSES file
- Your site's CLIINI file

Chapter 2

Configuring MQSeries Integrator

There are operational differences between the OS/390 version of MQSeries Integrator and the UNIX and NT versions.

OS/390 Operational Differences

Command Line Parameters

There is no command line environment to run the executables, so they are executed in batch using Job Control Language (JCL). See *Tailoring Jobs for Your Site* on page 17. Because JCL limits the size of the PARM field on the EXEC card, several modules that previously accepted long command line argument lists now accept their input from the standard input stream (STDIN) SYSIN.

For those modules that accept parameters in the PARM string, the PARM string must be enclosed in quotes, and each parameter must have a blank space between it and the next parameter.

Filenames on OS/390

Filenames are specified on OS/390 as DD-names. For example, when a PARM field allows the specification of a filename, the format is similar to:

```
PARM='-f DD:FILENAME'
```

A subsequent line of JCL is required with a DD-name that matches the specified name:

```
//FILENAME   DD DSN=<dataset-name>,DISP=SHR
```

Executable Names

Because OS/390 limits the size of member names in a PDS, some executables have different names on OS/390, for example:

Name on UNIX and NT	Name on OS/390
NNRDBARuleOwnership	RULOWNER

Metadata Changes

Three metadata changes for the OS/390 platform might not be reflected on UNIX or NT versions of MQSeries Integrator. This can affect your ability to import data from UNIX or NT platforms to OS/390. These metadata changes are described in the following table:

UNIX or NT Value	OS/390 Value	Description
ASCII String	String	String data is always in the native character set of the machine on which the engine is running: ASCII on UNIX and NT, and EBCDIC on OS/390.
ASCII Numeric	Numeric	Numeric really means graphic characters 0-9. Numeric characters are represented in the native character set of the platform on which the engine is running: ASCII on UNIX and NT, and EBCDIC on OS/390.
EBCDIC Data	ASCII Data	This should be interpreted as “the character set different from my own.” On an ASCII platform, EBCDIC data is the other character set; and on OS/390, ASCII data is the other character set.

When you use the NEONFormatter GUI to import format definitions created on a non-OS/390 system, you might have to edit the exported input and output control files using a text editor. Change all the occurrences of ASCII String to String; ASCII Numeric to Numeric; and all EBCDIC data to ASCII data. This allows the import process to find all the corresponding metadata values and continue to import your data. Otherwise, an invalid datatype error may be reported.

Tailoring Jobs for Your Site

Job Control Language (JCL) that contains a statement enclosed by chevrons (< >) means that the user must provide a valid value in that statement prior to submitting the job. For example, the APITEST job contains the following line:

```
//MSGIN   DD DSN=<your-message-file>,DISP=SHR
```

The user must provide an OS/390 dataset name for the file containing the message data.

Each job uses in-stream procedures that contain symbolic parameters. These symbolic parameters might have to be tailored from the default installation values to match dataset names and dataset high-level qualifiers (HLQs) for your site. Each job uses some combination of the following parameters:

Parameter	Description	Default Value
PRM=(‘ ‘)	Run-time parameters passed to the program at startup.	varies
SMPHLQ	HLQ for the MQSeries Integrator distribution libraries.	
MQSHLQ	HLQ for IBM MQSeries run-time libraries.	
CEEHLQ	HLQ for IBM Language Environment run-time libraries.	

Parameter	Description	Default Value
CSSHLQ	HLQ for IBM Callable System Services (CSSLIB) library	SYS1
SQLMEM	The member of the SNEOCNTL library containing control cards for DB2 access.	SQLSVSES
OPCLAS	Output class for SYSOUT statement.	*
INIMEM	Controls access to DB2.	CLIINI
MPF=	The member of the SNEOMPF library containing control cards for MQSeries access.	PUTDATA, RULENGP, or GETDATA
TCPHLQ	HLQ for TCP connection; member of SNEOPROC library.	TCPIP
LEHLQ	HLQ for current version of language environment used by the compiler; member of SNEOPROC library.	

Configuring SQLSVSES

The SQLSVSES configuration file contains information used in several modules, including APITEST, MSGTEST, NNRIE, NNFIE, and RULETEST. The SQLSVSES file contains information about the database and database server used with executables. A sample file is included in the <smphlq>.SNEOCNTL library.

The SQLSVSES file is accessed using DD-name SQLSVSES. This DD-name can be specified in JCL as either a permanent DASD dataset or instream with the JCL. The SQLSVSES DD-name must be present in your JCL and refer to a valid sequential file or member of a PDS:

```
//SQLSVSES DD DISP=SHR,DSN=<smphlq>.SNEOCNTL(SQLSVSES)
```

The syntax for each record in the SQLSVSES file is:

```
OpenDbmsSessionName:ddf-Location:userid:pasword:sqlid:
```

Parameters

Parameter	Description
OpenDbmsSession Name	Database session name used by executables. This can be any string as long as it is unique within the file. This matches the string specified in the OpenDbmsSession() API call. The default session name is new_format_demo.
ddf-Location	Identifies the DB2 subsystem to which this application connects. This value is also in the SYSIBM.LOCATIONS table in the DB2 catalog.
userid	This field is not currently used, but must be specified. Security is handled by your site's RACF, ACF2, or DB2 security exits currently in effect. Specify xxxx as a placeholder for this field.
password	This field is not currently used, but must be specified. Security is handled by your site's RACF, ACF2, or DB2 security exits currently in effect. Specify xxxx as a placeholder for this field.
sqlid	Qualifier for the database to which this application connects. This value is used in a 'SET CURRENT SQLID' statement after the process successfully connects to DB2. Must be a valid primary or secondary AUTH-ID for the database accessed.

Editing the SQLSVSES File

Use ISPF Edit to modify your SQLSVSES file. Make sure there is a session-name for each MQSeries Integrator application you plan to run. The session-name must match (including case) the value specified for any OpenDbmsSession() API calls.

Sample Text Lines in the SQLSVSES File:

```
SESSION_TAG:DDF-LOCATION:N/A:N/A:SQLID:
new_format_demo:<ddf-location>:xxxxx:yyyyy:<sqlid>:
nnfie:<ddf-location>:xxxx:yyyy:<sqlid>:
nnrmie:<ddf-location>:xxxx:yyyy:<sqlid>:
rules:<ddf-location>:xxxx:yyyy:<sqlid>:
output:<ddf-location>:xxxx:yyyy:<sqlid>:
input:<ddf-location>:xxxx:yyyy:<sqlid>:
```

Implementing Changes to SQLSVSES

SQLSVSES is read at application startup. To implement changes to the SQLSVSES file, you must restart any applications using MQSeries Integrator components for the changes to be recognized by those applications.

Configuring DSNAOINI

Any program that accesses DB2 databases must have a DD-name for DSNAOINI in the JCL and refer to a valid sequential file or member of a PDS. The DSNAOINI file controls connection attributes to DB2. Refer to IBM documentation for configuration details.

```
//DSNAOINI  DD DISP=SHR,DSN=<smphlq>.SNEOCNTL(CLIINI)
```

The contents of the DSNAOINI file is documented in the IBM ***Call Level Interface Guide and Reference*** manual (SC26-8959).

File Encryption

The NNcrypt program is distributed in the \UTIL directory on the Windows NT CD. NNcrypt reads an encrypted file from the filename specified as the first parameter and writes decrypted data to the file specified as the second parameter string. The input file and the output file must be different datasets. You cannot decrypt into the same dataset containing the encrypted files.

You must decrypt the export files on Windows NT before you FTP or transfer them to OS/390. Use the following steps as a guide for this process:

1. Run NNcrypt on Windows NT.
2. FTP or transfer the NNFIE or NNRIE file to OS/390.
3. Import the NNFIE or NNRIE file.

Chapter 3

Migrating Formats and Rules

Summary

Use the following steps to migrate your database from MQIntegrator r. 3.2 to MQSeries Integrator 1.1:

1. Instantiate the MQSeries Integrator 1.1 database. Load the 1.1 metadata. For more information, see the ***Installation and Configuration Guide***.
2. Use the NEOFIX32 SPUFI script in the SNEOSQL library to create a backup copy of the r. 3.2 database.
3. Run the MQIntegrator r. 3.2 Consistency Checker SPUFI scripts to insure that the data is consistent.

   The Consistency Checker SPUFI scripts are FORMATCC, RULECC, and PERMCC in the MQIntegrator r. 3.2 SNEOSQL library. You must correct any inconsistencies in the data before you export the data.
4. Run NNFIE r. 3.2 to export formats to a sequential file.

   The export file should be preallocated with the following DCB attributes:

   DSORG=PS or PO, RECFM=VB,LRECL=32756,BLKSIZE=32760

5. Run NNRIE r. 3.2 to export rules to a sequential file.

   The export file should be preallocated with the following DCB attributes:

   DSORG=PS or PO, RECFM=VB,LRECL=32756,BLKSIZE=32760

6. Run NNFIE 1.1 to import formats from the export file created in step 4. See *Migrating Formats* on page 25.

7. Run NNRIE 1.1 to import rules from the export file created in step 5. See *Migrating Rules* on page 73.

8. You might want to run the RENAME batch job against the MQSeries Integrator 1.1 database.

   Use RENAME to rename components that start with NNDef_ to start with a prefix you specify. See *NNRENAME* on page 34.

9. Run the MQSeries Integrator Consistency Checker against the 1.1 database to verify the consistency of the data.

   The Consistency Checker SPUFI scripts are FORMATCC, RULECC, and PERMCC in the 1.1 SNEOSQL library. The NEOMQCC batch job is in the 1.1 SNEOJCL library. See *Consistency Checker* on page 179.

Note:
After migrating your data to MQSeries Integrator 1.1, you must recompile your applications. The .h (include) files in version 1.1 are different from the .h files in release 3.2.

Migrating Formats

Before you migrate any data, run the r. 3.2 Consistency Checkers on the MQIntegrator r. 3.2 database to check for database errors. See *Consistency Checker* on page 179.

- Run the Format Consistency Checker against your 3.2 database.

  This is the FORMATCC member in the r. 3.2 SNEOSQL library. Repair any inconsistencies using the NEONFormatter graphical user interface (GUI).

- Run the Rules Consistency Checker against your 3.2 database.

  This is the RULECC member in the r. 3.2 SNEOSQL library. Repair any inconsistencies using the NEONRules GUI.

- Run the Permissions Consistency Checker against your 3.2 database.

  This is the PERMCC member in the r. 3.2 SNEOSQL library. Repair any inconsistencies using the NEONRules GUI.

- Run the Repair SQL script to repair any known problems in the 3.2 database.

  This is the NEOFIX32 member in the MQSeries Integrator 1.1 SNEOSQL library.

To migrate existing formats from a MQIntegrator r. 3.2 database to a MQSeries Integrator 1.1 database, use the MQIntegrator r. 3.2 Formatter GUI export function or the 3.2 Formatter Import/Export Utility (NNFIE) to export the existing formats. After you install MQSeries Integrator 1.1, use the NEONFormatter GUI import function or the 1.1 NNFIE to load MQIntegrator release 3.2 formats into the 1.1 database.

To export formats, you must use the NNFIE version that matches the version of your database. For example, use NNFIE version 3.x to export from a 3.x database. NNFIE 1.1 can only export from a 1.1 database.

See the *Formatter* chapter in the ***User's Guide*** for instructions on using export and import functions of the NEONFormatter graphical user interface (GUI.

Importing and Exporting Formats

NNFIE is used to export information from database tables and import information to database tables associated with the NEONFormatter. NNFIE creates a flat file during export and reads the same file structure for import. Earlier versions of NNFIE used encrypted files; the NNFIE files are no longer encrypted. The user can export individual formats or all formats.

During the import phase, all formats and associated controls in the import file are loaded. NNFIE detects situations where an existing component that is modified during an import can cause the import of that component to fail. If an existing component will be overwritten, and the component being imported is identical, then the import can succeed. All formats and controls that contain a component that fails to import will fail.

The NNFIE export file contains components defined by Formatter Management API structures. Many useful pieces of information that define the component are in numeric form instead of text form. If a user is not familiar with the ordinal type values and specific component definitions, the export file can be difficult to decipher. NNFIE contains an inventory option that produces a component inventory listing in the DD:NNFIELOG file.

When an NNFIE export file is created, a header is added to the beginning of the file. This header includes source and date information. The user can specify additional comments. The header and comments are preceded by a pound sign (#) and are ignored by NNFIE during import.

All output controls associated with an output format must be exported. To export output controls that use conditional branching, use the "export by name" option.

In earlier releases, the record length of a Formatter component was determined by the API structure that defined the component. The component definition can become so long that generic tools, such as text editors and stream tools, corrupt the data by truncating the longest lines. A text editor is unable to read the export file and to modify records. By inserting a continuation character, the component definition can be divided into several lines within the export file. The backslash (\) character immediately preceding the end-of-line character indicates that the following line is concatenated by the export file reader. The default line width is 80 characters, but the user can specify an optional line length.

When a component conflict was detected in import files using earlier versions of NNFIE, the conflict was logged, and the component was not imported. Identifying conflicts without importing data allows users to verify the contents of export files with working databases. NNFIE 1.1 has greater flexibility in conflict management, which allows NNFIE to be used as a migration tool. Overwrite, Ignore / Skip, and Rename functionality is available for resolving conflicts in existing database components. During the import of format definitions, all decisions to resolve conflicts are reported to an DD:NNFIELOG file. If a component fails to import, the line containing an error from the export file is written to DD:NNFIELOG. During the export phase, the user can specify a text comment to include in the export file.

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

NNFIE can import data from a MQIntegrator r 3.2 export file into an MQSeries Integrator 1.1 database. The input file is created using the NNFIE export facility of a MQIntegrator r. 3.2 node. The input file DD:IMPORTFL contains the exported formats and format components from MQIntegrator r. 3.2. The file is then moved to OS / 390 and translated from ASCII to EBCDIC in the process

Using NNFIE requires the following preparation:

- DB2 must be installed.
- The operating system must support standard input, standard output, and standard error stream sources and sinks (SYSIN, SYSPRINT, SYSOUT).
- The Rules database schema and the Formatter database schema must be created.
- Formats and related format components must be exported from a valid database.
- The target database has been created.

The export file for NNFIE is not interchangeable with the files created using the graphical user interface (GUI). NNFIE can import data from an export file

into an 1.1 database. NNFIE 1.1 exports data only from a Version 1.1 database.

WARNING!

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

If you import components exported from a case-sensitive database into a case-insensitive database, NNFIE may fail during import if a conflict arises between two components named the same with only case differences.

The SQLSVSES DD-name must reference a valid dataset containing valid SQLSVSES entries, or the application fails to connect to DB2 and terminates.

When exporting, the DCB attributes of the export files should be set to DSORG=PS, RECFM=VB, LRECL=32756, BLKSIZE=32760. The export records may be very large.

Note:

The WORKFILE, FAILFILE, IMPORTFL, and any other non-print class files should be allocated with the same DCB attributes before the job is executed.

NNFIE

The following sample job control language (JCL) is provided to illustrate how to run the NEON Formatter Import/Export Utility (NNFIE) job in batch and pass startup parameters to it. The JCL at your site will be different. See *Tailoring Jobs for Your Site* on page 17 for information about the symbolic parameters in this sample.

```
//*  <tailor member JOBCARD and insert here>
//*
//*********************************************************************
//*                                                                   *
//* Licensed Materials - Property of New Era of Networks, Inc.        *
//* Copyright (c) 1998-1999, New Era of Networks, Inc.                *
//* All Rights Reserved.                                              *
//*                                                                   *
//* Release 4.1.1                                                     *
//*********************************************************************
//*********************************************************************
//*                                                                   *
//*NNFIE: Formatter Import/Export Utility                             *
//*                                                                   *
//*********************************************************************
//NNFIE    PROC PRM=('-export -all -file DD:EXPORTFL'),
//         SMPHLQ='<smphlq>',    HLQ for NEONet distrib libs
//         MQSHLQ='<mqshlq>',    HLQ for MQS run-time libs
//         CEEHLQ='<lehlq>',     HLQ for Lang Envir libs
//         CSSHLQ='SYS1',        HLQ for Callable Sys Svcs (CSS-) Lib
//         SQLMEM='SQLSVSES',    MEMbername for SQLSVSES cntl cards
//         INIMEM='CLIINI',      MEMbername for CLIINI cntl cards
//         OPCLAS='*'            SYSOUT CLASS
//*
//STP0101  EXEC PGM=NNFIE,
//   PARM=&PRM
//*
//*   <tailor member STEPLIB and copy it here>
//*
//SQLSVSES DD  DSN=&SMPHLQ..SNEOCNTL(&SQLMEM),DISP=SHR
//DSNAOINI DD  DSN=&SMPHLQ..SNEOCNTL(&INIMEM),DISP=SHR
//SYSOUT   DD  SYSOUT=&OPCLAS
//SYSPRINT DD  SYSOUT=&OPCLAS
//STATLOG  DD  SYSOUT=&OPCLAS
//CLITRACE DD  SYSOUT=&OPCLAS  used for DB2 v5 CLI high-level tracing
```

```
//SYSIN    DD  DUMMY
//         PEND
//*
//*
//* All datasets used by MQSeries Integrator must be preallocated and
//* cataloged prior to running any MQSeries Integrator jobs. The
//* recommended DCB attributes are:
//* DSORG=PS,RECFM=VB,LRECL=32756,BLKSIZE=32760
//*
//NNFIE    EXEC NNFIE,PRM=(' -C DD:CMDFILE')
//NNFIELOG DD  DISP=SHR,DSN=<your-log-file-here>
//NNFIEERR DD  DISP=SHR,DSN=<your-error-file-here>
//IMPORTFL DD  DISP=SHR,DSN=<your-format-import-file>
//CMDFILE DD *
-i DD:IMPORTFL
/*
//
//
```

Syntax

NNFIE commands and options must be entered in the following order:

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

Parameters

Name	Mandatory/ Optional	Description
-C <command filename>	Optional	Alternate command filename; default file is DD:CMDFILE. If -C is provided, NNFIE reads command line options from the specified file instead of the command line. Using -C puts import or export command options in a text file. Do not enclose component names in quotation marks in the text file. Do not use back slashes in command lines.
-i | -import <import filename>	Mandatory	Required parameter to import data from the named file; mutually exclusive from -e. The default file is DD:NNFIEEXP.
-s <session name>	Optional	Name of session in SQLSVSES. Defaults to NNFIE.
-e | -export <export filename>	Mandatory	Required parameter export data from the named file; mutually exclusive from -i. The default file is DD:NNFIEEXP.
-t <import filename>	Mandatory	Writes an inventory of the import file to DD:NNFIELOG.
-I<import filename>	Mandatory	Writes description of all conflicts in import file to DD:NNFIELOG.
Filenames for both import and export must be no longer than 255 characters.		

Importing Formats

The following options are available for importing formats:

```
(-i | import <import filename> [-T] [-o|-g|-n|-4]
                               [-s <session name>])
```

Import Options

Parameter	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.
-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.

Troubleshooting Import Failures

If NNFIE fails to import from a given export file, view the DD:NNFIELOG file to determine the cause for import failure. An import can fail if the data conflicts with the data existing in the database, or if there is incorrect or missing data.

Missing or Incorrect Data Error Message

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

```
ERROR: <error message>
```

Missing or Incorrect Data Error Message for a Specific Formatter Element

This error message contains 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>
```

Resolving Component Conflicts

A conflict occurs when an imported component does not match an existing component of the same name and type in the database. The user can overwrite the component definition, ignore or skip the component, or rename the inported component.

When a component is overwritten, the component definition within the export file is imported into the database. Overwriting existing components may cause supporting components to be unused. When a component is ignored or skipped, the component in the export file is not imported. However, the component is added to the internal inventory of imported components. By skipping components, supporting components that have already been imported might be unused. Skipping or overwriting components does not affect the integrity of the database. If the user renames a component, all references to that component in the export file are updated.

The user can choose to resolve conflicts in interactive mode or batch mode.

All conflicts and resolutions are reported to the DD:NNFIELOG log file.

Note:
Interactive mode is only available for use on a Windows NT workstation or UNIX-based workstation. It is not available on the OS/390 platform.

Resolving Conflicts in Batch Mode

Overwrite and Ignore / Skip options are available for resolving conflicts in batch mode. The selected option is used to resolve all conflicts.

Use the following code to implement batch Overwrite conflict resolution:

```
NNFIE -i <filename> -o
```

Use the following code to implement batch Ignore / Skip conflict resolution:

```
NNFIE -i <filename> -g
```

Resolving Conflicts in Interactive Mode

Overwrite, Ignore / Skip, and Rename options are available for resolving conflicts in interactive mode on a Windows NT workstation or on a UNIX-based workstation. If the user implements interactive conflict resolution, descriptions of the existing components and the import components are displayed.

Use the following code to implement interactive conflict resolution:

```
NNFIE -i <filename> -n
```

The following sample code illustrates interactive conflict resolution:

```
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
```

NNRENAME

When you run NNFIE to unload your data from r.3.2 and reload your data into MQSeries Integrator 1.1, some Formatter components are created that did not exist in r.3.2. These components are assigned a default name:

```
NNDef_xxxx_nnnn
```

where xxxx is the type of format component. For example, a literal value might be NNDef_Literal_nnnn; a Default output operation might be NNDef_Default_nnnn.

The NNRENAME utility takes the field value of the component that is being renamed and creates a name:

PREFIX_VALUE_nn

Parameters

Field	Description
PREFIX	Default value is NN_. The user can specify a PREFIX, or can specify PREFIX=NONE.
VALUE	The value of the component field. Non-printable characters are converted to '#'. Any character that repeats 5 or more times sequentially will have the repeat count and then the value. For example, the field "abcddddddd" is converted to "abc8d". The length of the VALUE field is truncated so that the total length of the new name is no more than 32 characters.
nn	Value between 01 and 99. The value is selected sequentially to handle any duplicates. If there are more than 99 duplicates, only the first 99 are inserted. The rest of the duplicates remain in the database without a name change.

The following sample job control language (JCL) is provide to illustrate how to run the RENAME job in batch and pass startup parameters to it. The JCL at your site will be different. See *Tailoring Jobs for Your Site* on page 17 for information about the symbolic parameters in this sample.

```
//*  <insert a valid jobcard here >
//*
//*******************************************************************
//*                                                                 *
//* Licensed Materials - Property of New Era of Networks, Inc.      *
//* Copyright (c) 1998-1999, New Era of Networks, Inc.              *
//* All Rights Reserved.                                            *
//*                                                                 *
//* Release 4.1.1                                                   *
//*******************************************************************
//*******************************************************************
//*                                                                 *
//* NNRENAME : Rename components that begin with 'NNDef_'          *
```

```
//*                                                                   *
//*******************************************************************
//NNRENAME PROC SMPHLQ='<smphlq>', HLQ for NEONet distrib libs
//         MQSHLQ='<mqshlq>',      HLQ for MQS runtime libs
//         CEEHLQ='<lehlq>',       HLQ for Lang Envir  libs
//         CSSHLQ='SYS1',          HLQ for Callable Sys Svcs (CSS-)Lib
//         INIMEM='CLIINI',        MEMbername for CLI INI  cntl cards
//         OPCLAS='*'              SYSOUT CLASS
//*
//STP0101  EXEC PGM=NNRENAME
//*
//*   <tailor the member STEPLIB and copy it here>
//*
//DSNAOINI DD  DSN=&SMPHLQ..SCTLSTMT(&INIMEM),DISP=SHR
//SYSOUT   DD  SYSOUT=&OPCLAS
//SYSPRINT DD  SYSOUT=&OPCLAS
//CLITRACE DD  SYSOUT=&OPCLAS     used for DB2-CLI high-level tracing
//         PEND
//*
//* All datasets used by MQSeries Integrator must be preallocated and
//* cataloged prior to running any MQSeries Integrator jobs. The
//* recommended DCB attributes are:
//* DSORG=PS,RECFM=VB,LRECL=32756,BLKSIZE=32760
//*
//NNRENAME EXEC NNRENAME
//*
//DB2PARMS DD  *
<db2-sysid> <db2-sqlid>
<PREFIX=(default is NN_) >
/*
//
```

Exporting Formats

The following options are available for exporting formats:

```
(-e | -export <export filename> [-m <format name>+]
                                [-q "comment"]
                                [-Q <Comment filename>]
                                [-w <number>]
                                [-s <session name>])
```

Export Options

Parameter	Mandatory/ Optional	Description
-q "comment"	Optional	Adds comments enclosed in quotation marks to beginning of the export file.
-Q <comment file>	Optional	Adds contents of <comment file> to beginning of export file.
-w <number>	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.

Examples

The following code illustrates exporting an entire database:

```
NNFIE -e [<export filename>] [-s <session name>]
```

The following code illustrates exporting a single format:

```
NNFIE -e [<export filename>] [-m <format name>]
         [-s <session name>]
```

The following code illustrates exporting several formats:

```
NNFIE -e [<export filename>]
         [-m <format name> <format name> ...]
         [-s <session name>]
```

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.

Troubleshooting Export Failures

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 DD:NNFIELOG file. To use this conflict report, type the following syntax:

```
NNFIE -I <filename> -k
```

Producing a Component Inventory

The NNFIE export file contains useful pieces of information that are in numeric form instead of text form. NNFIE contains an inventory option that produces a component inventory from the export file in the NNFIE log file. Use the following code to produce a component inventory listing in the DD:NNFIELOG file:

```
NNFIE -t <filename>
```

In earlier releases, the only access to the NEONFormatter database configurations have been through the NEONFormatter GUI. With the export files in a readable form, the user can write or modify scripts that create NEONFormatter components.

Sample Component Inventory

The following code samples illustrate an NNFIE export file and a component inventory in the DD:NNFIELOG file.

NNFIE Export File

```
F!4.0!7!NEON.Space,0x20,1!
F!4.0!7!NEON.LOADER.KEY.929054218,0x4E454F4E6164617074657220666
F7220434F424F4C205B56657273696F6E203A20322E302028322E3020554E4B
4E4F574E5F4255494C445F56455253494F4E292C2028554E4B4E4F574E5F425
5494C445F5441472F57494E444F5753295D,94!
F!4.0!5!CCB.B-1,B-1!
F!4.0!5!CCB.B2-1,B2-1!
F!4.0!16!NEON.Space.Left,NEON.Space,0x20,1,1!
F!4.0!16!NEON.Space.Right,NEON.Space,0x20,1,2!
F!4.0!16!NEON.Space.Both,NEON.Space,0x20,1,3!
F!4.0!12!NEON.Space.1,1,NEON.Space,0x20,1!
```

```
F!4.0!8!COBOL:X.1,0,1,String,,1,0,NONE,0x00,0,0,0,0,NONE,0x00,
0,NEON.Space.1,4!
F!4.0.1!2!COBOL:X.1,0,1,String,1,,2,NONE,1,0,0,0,NONE,0x00,
NONE,0,0,0,NONE,0,0,!0!
F!4.0!1!CCB.IF.COBOL-OCCURS:1,1,0!1,0,0,NONE!1!CCB.IF.COBOL-
OCCURS:1,CCB.B-1,COBOL:X.1!
F!4.0!1!CCB.IF.B-2-TABLE,1,0!1,0,0,NONE!1!CCB.IF.B-2-TABLE,
CCB.B2-1,COBOL:X.1!
F!4.0!1!CCB.OF.COBOL-OCCURS:1,0,0!1,0,0,NONE!1!CCB.OF.COBOL-
OCCURS:1,CCB.B-1,COBOL:X.1,1,0,CCB.B-1!
F!4.0!1!CCB.OF.B-2-TABLE,0,0!1,0,0,NONE!1!CCB.OF.B-2-TABLE,
CCB.B2-1,COBOL:X.1,1,0,CCB.B2-1!
F!4.0!1!CCB.IC.COBOL-OCCURS,1,1!2!CCB.IC.COBOL-OCCURS,
CCB.IF.COBOL-OCCURS:1,0,0,0,NONE,0,NONE!CCB.IC.COBOL-OCCURS,
CCB.IF.B-2-TABLE,0,1,3,NONE,3,NONE!
F!4.0!1!CCB.OC.COBOL-OCCURS,0,1!2!CCB.OC.COBOL-OCCURS,
CCB.OF.COBOL-OCCURS:1,0,0,0,NONE,0,NONE!CCB.OC.COBOL-OCCURS,
CCB.OF.B-2-TABLE,0,1,0,NONE,0,NONE!
```

Sample Component Inventory

```
Component Type: Literal
-----------------------
Component Name: NEON.Space
              Length: 1
              Value: " "
Component Name: NEON.LOADER.KEY.929054218
              Length: 94
              Value: "NEONadapter for COBOL [Version : 2.0 (2.0
 UNKNOWN_BUILD_VERSION), (UNKNOWN_BUILD_TAG/WINDOWS)]"
Component Type: Field
---------------------
Component Name: CCB.B-1
              Description: "B-1"
Component Name: CCB.B2-1
              Description: "B2-1"
Component Type: Trim Output Operation
-------------------------------------
Component Name: NEON.Space.Left
              Trim Type: Left
              Trim Literal: NEON.Space
              Trim Value Length: 1
              Trim Value: " "
```

```
Component Name: NEON.Space.Right
              Trim Type: Right
              Trim Literal: NEON.Space
              Trim Value Length: 1
              Trim Value: " "
Component Name: NEON.Space.Both
              Trim Type: Left and Right
              Trim Literal: NEON.Space
              Trim Value Length: 1
              Trim Value: " "
Component Type: Length Output Operation
---------------------------------------
Component Name: NEON.Space.1
              Pad Literal Name: NEON.Space
              Pad Literal Value: " "
Component Type: Output Control Master
-------------------------------------
Component Name: COBOL:X.1
              Optional: No
              Control Type: Data Field Name Search
              Data Type: Ascii String
              Child Control Name: NEON.Space.1
              Child Control Type: Length
Component Type: Input Control
-----------------------------
Component Name: COBOL:X.1
              Optional: No
              Control Type: Data Only
              Data Type: Ascii String
                  Termination: Exact Length
                  Length: 1
Component Type: Flat Input Format
---------------------------------
Component Name: CCB.IF.COBOL-OCCURS:1
              Input Field 1: CCB.B-1
                  Input Control: COBOL:X.1
Component Name: CCB.IF.B-2-TABLE
              Input Field 1: CCB.B2-1
                  Input Control: COBOL:X.1
Component Type: Flat Output Format
----------------------------------
Component Name: CCB.OF.COBOL-OCCURS:1
```

```
                Decomposition: Ordered
                Termination: Not Applicable
                Output Field 1: CCB.B-1
                    Output Control: COBOL:X.1
                    Access Mode: Normal Access
                    Input Field: CCB.B-1
Component Name: CCB.OF.B-2-TABLE
                Decomposition: Ordered
                Termination: Not Applicable
                Output Field 1: CCB.B2-1
                    Output Control: COBOL:X.1
                    Access Mode: Normal Access
                    Input Field: CCB.B2-1
Component Type: Compound Input Format
-------------------------------------
Component Name: CCB.IC.COBOL-OCCURS
                Child Format 1: CCB.IF.COBOL-OCCURS:1
                    Optional: No
                    Repeating: No
                Child Format 2: CCB.IF.B-2-TABLE
                    Optional: No
                    Repeating: Yes
                    Repeat Termination: Exact Count
                    Repeat Count: 3
Component Type: Compound Output Format
--------------------------------------
Component Name: CCB.OC.COBOL-OCCURS
                Child Format 1: CCB.OF.COBOL-OCCURS:1
                    Optional: No
                    Repeating: No
                Child Format 2: CCB.OF.B-2-TABLE
                    Optional: No
                    Repeating: Yes
                    Repeat Termination: Not Applicable
```

NNFIE Readable Files

Earlier versions of NNFIE exported and imported encrypted files. In earlier releases, the only access to the Formatter database configurations was through the Formatter GUI. With the export files in a readable form, the user can write or modify scripts that create NEONFormatter components.

NNFIE Header

When an NNFIE export file is created, a header file can be added to the beginning of the file. This file logs information about the data source and creation of the export file.

Use the following code to add a header file:

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

The header file contains the following information:

1. Time of creation (using Greenwich Mean Time)
2. Version of NNFIE
3. Database logon information
4. Database server version
5. Operating system version

The first character in the header file is a pound sign (#), which indicates that the comments should be ignored by NNFIE during import. The user can specify additional comments using the following export option:

```
NNFIE -e <filename> -q "comment"
```

Formatting Export Data

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

A continuation marker divides the format component definition into several lines within the export file. A backslash character (\) immediately preceding an end-of-line character indicates that the next line is concatenated by the export file text editor. The default line width is 80 characters, but the user can specify an optional width using the following command line option:

```
NNFIE -e <filename> -w <number>
```

Identifying NEONFormatter Components

Each NEONFormatter component definition begins with three identification fields, delimited by an exclamation point (!) character.

The first field contains the letter F, which indicates the beginning of a NEONFormatter component definition. F must appear at the beginning of every line in the file, with the exception of comment lines.

The second field indicates the release number of the defined component. Using version numbers to define components enables NNFIE to support several revisions of export files.

The third field contains an integer that identifies each valid component in an export format. The following table lists valid format components and corresponding integer values used in the export files.

Integer	Format Component
1	Format
2	Input Control
3	Output Control (included for backward compatibility; use Output Master Control)
4	Delimiter (included for backward compatibility; use Literal)
5	Field
6	User Defined Type
7	Literal
8	Output Master Control
9	Default Control
10	User Exit Control
11	Fix Control
12	Length Control
13	Math Expression Control

Integer	Format Component
14	Substitute Control
15	Substring Control
16	Trim Control
17	Collection Control

Defining Format Components

The string data type must be 32 characters or less when used as a field type.

The encoded hex field type can be up to 254 characters. Valid characters in this field are 0x[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 Formatter Management APIs to populate the database with Formatter components. For a detailed explanation of field values, refer to the structures defined in ***Programming Reference for NEONFormatter***. NNFIE uses the information in the export file to populate the NNFMgr<Component Type>Info structures.

The following sample NNFIE component definitions use these conventions:

! indicates a delimiter between format components.

/ / indicates comments.

(…)+ indicates items within parentheses exist one or more times.

(…)* indicates items within parentheses exist zero or more times.

Flat Input Format Example

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

Components

```
F!<Version No – number>!<Format – integer>!
```

```
// NNFMgrFormatInfo structure
<Format Name – string>,
1,                                  // Input Indicator
0!                                  // Compound Indicator

// NNFMgrFlatFormatInfo structure
<Decomposition ID – int>,
<Length ID – int>,
<Termination ID – int>,
<Delimiter Name – string>!
<Number of Input Field/Control Pairs – integer>!

// NNFMgrInFieldInfo structure
(<Format Name – string>,
<Field Name – string>,
<Control Name – string>!)+
```

Flat Output Format 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!
```

Components

```
F!<Version No – number>!<Format – integer>!

// NNFMgrFormatInfo structure
<Format Name – string>,
0,                                  // Input Indicator
0!                                  // Compound Indicator

// NNFMgrFlatFormatInfo structure
<Decomposition ID –int>,
<Length ID – int>,
<Termination ID – int>,
<Delimiter Name – string>!
<Number of Output Field/Control Pairs – integer>!

// NNFMgrOutFieldInfo structure
(<Format Name – string>,
<Field Name – string>,
```

```
<Control Name – string>,
<Access Mode – int>,
<Subscript – integer>,
<Infield Name – string>!)+
```

Compound Format Example

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

Components

```
F!<Version No – number>!<Format – integer>!

// NNFMgrFormatInfo structure
<Format Name – string>,
<Input Indicator ID –int>,
1!                                  // Compound Indicator
<Number of Child Formats – integer>!

// 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>!)+
```

Input Control 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!
```

Components

```
F!<Version No – number>!<Input Control – integer>!

// 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>m
<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>!)+
```

Field Example

```
F!<Version No – number>!5!numeric,Numeric field!
```

Components

```
F!<Version No – number>!<Field – integer>!

// NNFMgrFieldInfo structure
<Field Name – string >,
<Comment – string >!
```

User-defined Type Example

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

Components

```
F!<Version No – number>!<User-defined Type – integer>!

// NNFMgrUserDefTypeInfo structure
```

```
<Type Name – string >,
<Native Type – string >,
<Validation Routine Name – string>!
```

Literal Example

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

Components

```
F!<Version No – number>!<Literal – integer>!

// NNFMgrLiteralInfo structure
<Literal Name – string >,
<Value - ASCII – encoded hex >,
<Value Length – integer>!
```

Output Master Control Example

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

Components

```
F!<Version No – number>!<Output Master Control – integer>!

// 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>!
```

Default Control Example

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

Components

```
F!<Version No – number>!<Default Control – integer>!

// NNFMgrDefaultCntlInfo structure
<Control Name – string>,
<Literal Name – string>,
<Value – ASCII-encoded hex>,
<Value Length – integer>!
```

User Exit Control Example

```
F!4.0!10!Sample_UserExitCntl,ExitRoutineName!
```

Components

```
F!<Version No – number>!<User Exit Control – integer>!

// NNFMgrUserExitCntlInfo structure
<Control Name – string>,
<Exit Routine Name – string>!
```

Fix Control Example

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

Components

```
F!4.0!<PrePostFix Control – integer>!

// NNFMgrPrePostFixCntlInfo structure
<Control Name – string>,
<Literal Name – string>,
<Value – ASCII-encoded hex>,
<Value Length – integer>,
<Place ID – enum NNFPrePostFix>,
<NULL Action Indicator – int>!
```

Length Control Example

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

Components

```
F!<Version No – number>!<Length Control – integer>!

// NNFMgrLengthCntlInfo structure
<Control Name – string>,
<Pad Literal Name – string>,
<Pad Value – ASCII – encoded hex>,
<Value Length – integer>!
```

Math Expression Control

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

Components

```
F!<Version No – number>!<Math Expression Control – integer>!

// NNFMgrMathExpCntlInfo structure
<Control Name – string>,
<Decimal Precision – integer>,
<Rounding Mode ID –int>!
<Math Segment Count – integer>!
(<Expression – string>!)+
```

Substitute Control

```
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!
```

Components

```
F!<Version No – number>!<Substitute Control – integer>!

// 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>!)*
```

Substring Control

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

Components

```
F!<Version No – number>!<Substring Control – integer>!

// NNFMgrSubstringCntlInfo structure
<Control Name – string>,
<Start – integer>,
<Length – integer>,
<Pad Literal Name – string>,
<Pad Value – ASCII-encoded hex>,
<Pad Value Length – integer>!
```

Trim Control

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

Components

```
F!<Version No – number>!<Trim Control – integer>!

// 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>!
```

Collection Control Example

The following *Sample Data* illustrates a Collection Control in an NNFIE file.

```
F!4.0!17!Sample_CollectionCntl,2!Sample_UserExitCntl,7!
CENTER_JUSTIFY,10!
```

Components

```
F!<Version No – number>!<Collection Control – integer>!\

// NNFMgrCollectionCntlInfo Structure
<Control Name – string>,
<Collection Count – integer>!
(<Child Control Name – string>,
<Child Control Type – enum NNCntlType>!)+
```

Sample Data

```
F!4.0!5!alpha,!
F!4.0!5!numeric,!
F!4.0!7!=,0x3D000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000,1!
F!4.0!8!alpha_OC,1,1,String,,0,0,NONE,0x00000000000000000000000000000000
0 0000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000,0,0,0,0,NONE,0x0000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
000000000,0,NONE,0!
F!4.0!8!numeric_OC,1,1,String,,0,0,NONE,0x00000000000000000000000000000000
0 0000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000,0,0,0,0,NONE,0x0000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
```

```
0000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000,0,NONE,0!
F!4.0.1!2!alpha_IC,0,1,String,0,,2,NONE,6,0,0,0,NONE,0x000000EFFFE8600
0000000000A95DC00101E20000000000006B54C00000001EFFFF01400000101EFFFF01
4EF3C7101000000010000000100101E20EFFFF0DCEFFFF0DCEFFFE850000B2E60EF07D
6D800101E20EF3C717800000000EFFFE854000FA841EFFFF088EFFFF088EFFFF014000
00000EFFFF014EFFFF0DC0000000000,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
6000000000000A95DC00101E20000000000006B54C00000001EFFFF01400000101EFFF
F014EF3C7101000000010000000100101E20EFFFF0DCEFFFF0DCEFFFE850000B2E60EF
07D6D800101E20EF3C717800000000EFFFE854000FA841EFFFF088EFFFF088EFFFF014
00000000EFFFF014EFFFF0DC0000000000,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 Messages

Code	Error Name	Error Message	Explanation	Response
-4001	NNFIEE_FILE_ EXISTS	Given file already exists so will not replace it	The specified export file name already exists.	Remove the file or specify a different export file name.
-4002	NNFIEE_NO_ IMPORT_FILE	No import files by the given name exist	The specified import file name already exists.	Check for the existence of the file.
-4003	NNFIEE_FAILED_TO _READ_FROM_ IMPORT_FILE	Failed to read from the import file	The file cannot be read.	Check for the existence of the file or possible access problems.
-4004	NNFIEE_FAILED_TO _SEPARATE_INPUT_ DATA	Failed to separate and fetch a piece of the input data	The import file has been corrupted.	Restore or recreate the file.

Code	Error Name	Error Message	Explanation	Response
-4005	NNFIEE_BAD_FILE_ STREAM	Bad file stream	Unable to obtain the required file stream.	Check for the existence of the import or export file
-4006	NNFIEE_NAME_ PROPERTY_ CONFLICT	Conflict with the existing Formatter element with the same name	A format component being imported conflicts with an existing component of the same name.	To import into a populated format database, rename the existing component and re-import, or rename the incoming component in the source database and re-export.
-4007	NNFIEE_INVALID_ IE_MODE	Invalid import/ export mode. Valid modes are: EXPORT_BY_NAME EXPORT_ALL IMPORT	Invalid mode specified on the command line or in the command file.	Check the accuracy of arguments passed to NNFIE.
-4008	NNFIEE_ ATTEMPTING_TO_ REEXPORT	Attempting to re-export an element that has been exported	Component defined that references itself.	Remove the circular reference to this component.
-4009	NNFIEE_FAILED_TO _IMPORT_ COMPONENTS	Components have not been imported	During import, one or more of the components required did not import. All components that use the failed component will not import.	Determine why the component did not import correctly.

Code	Error Name	Error Message	Explanation	Response
-4010	NNFIEE_INVALID_ FORMATTER_ ELEMENT	Invalid Formatter element type	Unknown format component found. File 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	Specified file incompatible.	Recreate or recover the export file.
-4012	NNFIEE_INVALID_ VERSION_NO	Invalid NNFIE version number	Version number found in the file 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 unable to register the component as exported or imported.	Rerun the import or export.
-4014	NNFIEE_NO_ FORMATS_TO_ EXPORT	No formats to export	Format database does not contain valid formats to export.	Create valid formats.
-4015	NNFIEE_NOTHING_ TO_IMPORT	Nothing to import	Import file does not contain format information.	Create export file from database that contains formats.
-4016	NNFIEE_FAILED_TO _ENCRYPT	Encryption failed	NNFIE unable to encrypt the export data successfully.	Rerun the export.

Code	Error Name	Error Message	Explanation	Response
-4017	NNFIEE_FAILED_TO_DECRYPT	Decryption failed	NNFIE unable to decrypt the import file. This is caused by file corruption.	Recreate of recover the export file.
-4018	NNFIEE_NNFIEERR_ALREADY_EXISTS	NNFIE.err already exists	Error file NNFIE.err exists.	Remove the NNFIE.err file and rerun.
-4019	NNFIEE_IE_FILE_ALREADY_EXISTS	I/E file already exists	Specified output file already exists.	Use a new export file name; move or rename existing export file.
-4020	NNFIEE_FAILED_TO_OPEN_DBMS_SESSION	Failed to open DBMS session	NNFIE unable to connect to the database.	Verify accuracy of NNFIE entry or session name specified with -s option in SQLSVSES file.
-4021	NNFIEE_FAILED_TO_OPEN_FMGR	Failed to initialize Formatter Manager	NNFIE unable to use the Format Manager library.	Verify MQSeries Integrator installed correctly.
-4022	NNFIEE_INVALID_CNTL_TYPE	Invalid control type	Unknown format control found; file exported from an unsupported version of MQSeries Integrator, or file is corrupt.	Verify version of MQSeries Integrator on source machine. Recover or recreate the export file.

NNFIE Format Error Messages

Code	Error Name	Error Message	Explanation	Response*
-4201	NNFIEE_GetFormat	GetFormat failed	Flat or compound format is not accessible in the database using NNFMgrGet Format.	Use the secondary Format Management API error code to resolve the problem.
-4202	NNFIEE_GetFirst Format	GetFirstFormat failed	First flat or compound format is not accessible in the database using NNFMgrGetFirst Format.	Use the secondary Format Management API error code to resolve the problem.
-4203	NNFIEE_GetNext Format	GetNextFormat failed	Next flat or compound format is not accessible in the database using NNFMgrGetNext Format.	Use the secondary Format Management API error code to resolve the problem.
-4204	NNFIEE_GetFirst FieldFromInput Format	GetFirstFieldFrom InputFormat failed	First field associated with a flat input format is not accessible in the database using NNFMgrGetFirst FieldFromInput Format.	Use the secondary Format Management API error code to resolve the problem.
-4205	NNFIEE_GetNext FieldFromInput Format	GetNextFieldFrom InputFormat failed	Next field associated with a flat input format is not accessible in the database using NNFMgrGetNext FieldFromInput Format.	Use the secondary Format Management API error code to resolve the problem.

Code	Error Name	Error Message	Explanation	Response*
-4206	NNFIEE_GetFirst FieldFromOutput Format	GetFirstFieldFrom OutputFormat failed	First field associated with a flat output format is not accessible in the database using NNFMgrGetFirst FieldFromOutput Format.	Use the secondary Format Management API error code to resolve the problem.
-4207	NNFIEE_GetNext FieldFromOutput Format	GetNextFieldFrom OutputFormat failed	Next field associated with a flat output format is not accessible in the database using NNFMgrGetNext FieldFromOutput Format.	Use the secondary Format Management API error code to resolve the problem.
-4208	NNFIEE_GetFirst ChildFormat	GetFirstChildForm at failed	First child format of a compound format is not accessible in the database using NNFMgrGetFirst ChildFormat.	Use the secondary Format Management API error code to resolve the problem.
-4209	NNFIEE_GetNext ChildFormat	GetNextChildForm at failed	Next child format of a compound format is not accessible in the database using NNFMgrGetNext ChildFormat.	Use the secondary Format Management API error code to resolve the problem.
-4210	NNFIEE_GetOutput Control	GetOutputControl failed	Specified output control is not accessible in the database using NNFMgrGet OutputControl.	Use the secondary Format Management API error code to resolve the problem.

Code	Error Name	Error Message	Explanation	Response*
-4211	NNFIEE_GetFirst OutputControl	GetFirstOutput Control failed	First output control is not accessible in the database using the Formatter Management API NNFMgrGetFirst OutputControl.	Use the secondary Format Management API error code to resolve the problem.
-4212	NNFIEE_GetNext OutputControl	GetNextOutput Control failed	Next output control is not accessible in the database using NNFMgrGetNext OutputControl.	Use the secondary Format Management API error code to resolve the problem.
-4213	NNFIEE_GetParse Control	GetParseControl failed	Specified parse or input control is not accessible in the database using NNFMgrGetParse Control.	Use the secondary Format Management API error code to resolve the problem.
-4214	NNFIEE_GetFirst ParseControl	GetFirstParse Control failed	First parse or input control is not accessible in the database using NNFMgrGetFirst ParseControl.	Use the secondary Format Management API error code to resolve the problem.
-4215	NNFIEE_GetNext ParseControl	GetNextParse Control failed	Next parse or input control is not accessible in the database using NNFMgrGetNext ParseControl.	Use the secondary Format Management API error code to resolve the problem.

Code	Error Name	Error Message	Explanation	Response*
-4216	NNFIEE_Get Delimiter	GetDelimiter failed	Specified delimiter is not accessible in the database using NNFMgrGet Delimiter.	Use the secondary Format Management API error code to resolve the problem.
-4217	NNFIEE_GetFirst Delimiter	GetFirstDelimiter failed	First delimiter is not accessible in the database using NNFMgrGetFirst Delimiter.	Use the secondary Format Management API error code to resolve the problem.
-4218	NNFIEE_GetNext Delimiter	GetNextDelimiter failed	Next delimiter is not accessible in the database using NNFMgrGetNext Delimiter.	Use the secondary Format Management API error code to resolve the problem.
-4219	NNFIEE_GetField	GetField failed	Specified field is not accessible in the database using NNFMgrGetField.	Use the secondary Format Management API error code to resolve the problem.
-4220	NNFIEE_GetFirst Field	GetFirstField failed	First field is not accessible in the database using NNFMgrGetFirst Field.	Use the secondary Format Management API error code to resolve the problem.
-4221	NNFIEE_GetNext Field	GetNextField failed	Next field is not accessible in the database using NNFMgrGetNext Field.	Use the secondary Format Management API error code to resolve the problem.

Code	Error Name	Error Message	Explanation	Response*
-4222	NNFIEE_Append FormatToFormat	AppendFormatTo Format failed	Attempt to append one flat or compound format into a compound format failed using NNFMgrAppend FormatToFormat.	Use the secondary Format Management API error code to resolve the problem.
-4223	NNFIEE_Append FieldToInputFormat	AppendFieldTo InputFormat failed	Attempt to append a field to a flat input format failed using NNFMgrAppend FieldToInput Format.	Use the secondary Format Management API error code to resolve the problem.
-4224	NNFIEE_Append FieldToOutput Format	AppendFieldTo OutputFormat failed	Attempt to append a field to a flat output format failed using NNFMgrAppend FieldToOutput Format.	Use the secondary Format Management API error code to resolve the problem.
-4225	NNFIEE_Append MathExpression	AppendMath Expression failed	Attempt to append a math expression detail entry to an existing math expression control failed using NNFMgrAppendM athExpression.	Use the secondary Format Management API error code to resolve the problem.
-4226	NNFIEE_Append LookupEntry	AppendLookup Entry failed	Attempt to append a lookup detail entry to an existing lookup control failed using NNFMgrAppend LookupEntry.	Use the secondary Format Management API error code to resolve the problem.

Code	Error Name	Error Message	Explanation	Response*
-4227	NNFIEE_Create Format	CreateFormat failed	Attempt to create a new input/output flat or compound format failed using NNFMgrCreateFormat.	Use the secondary Format Management API error code to resolve the problem.
-4228	NNFIEE_Create ParseControl	CreateParseControl failed	Attempt to create a new parse/input control failed using NNFMgrCreate ParseControl.	Use the secondary Format Management API error code to resolve the problem.
-4229	NNFIEE_Create OutputControl	CreateOutput Control failed	Attempt to create a new output control failed using NNFMgrCreate OutputControl.	Use the secondary Format Management API error code to resolve the problem.
-4230	NNFIEE_Create Delimiter	CreateDelimiter failed	Attempt to create a new delimiter failed using NNFMgrCreate Delimiter.	Use the secondary Format Management API error code to resolve the problem.
-4231	NNFIEE_CreateField	CreateField failed	Attempt to create a new field failed using NNFMgrCreate Field.	Use the secondary Format Management API error code to resolve the problem.
-4232	NNFIEE_SERIOUS_ ERROR_POSSIBLY_ DB_RELATED	GetErrorNo returned serious error number	General database error encountered using the Format Management APIs.	See Format Management API error code -2604.

Code	Error Name	Error Message	Explanation	Response*
-4233	NNFIEE_GetData TypeName	GetDataTypename failed	Attempt to retrieve the formal name for data type code failed due to an invalid data type code associated control.	Run the Formatter Consistency Checker to verify data type codes.
-4234	NNFIEE_GetData Type	GetDataType failed	Attempt to retrieve the data type code associated with the formal data type name failed.	NNFIE import file does not contain correct formal data type names. The NNFIE import file is corrupt or has been exported from a damaged database.
-4235	NNFIEE_GetFirst UserDefinedType	GetFirstUser DefinedType failed	First user-defined type is not accessible in the database using NNFMgrGetFirst UserDefinedType.	Use the secondary Format Management API error code to resolve the problem.
-4236	NNFIEE_GetNext UserDefinedType	GetNextUser DefinedType failed	Next user-defined type is not accessible in the database using NNFMgrGetNext UserDefinedType.	Use the secondary Format Management API error code to resolve the problem.
-4237	NNFIEE_CreateUser DefinedType	CreateUserDefined Type failed	Attempt to create a new user-defined type failed using NNFMgrCreate UserDefinedType.	Use the secondary Format Management API error code to resolve the problem.

Code	Error Name	Error Message	Explanation	Response*
-4238	NNFIEE_GetFirst Literal	GetFirstLiteral failed	First literal is not accessible in the database using NNFMgrGetFirst Literal.	Use the secondary Format Management API error code to resolve the problem.
-4239	NNFIEE_GetNext Literal	GetNextLiteral failed	Next literal is not accessible in the database using NNFMgrGetNext Literal.	Use the secondary Format Management API error code to resolve the problem.
-4240	NNFIEE_GetLiteral	GetLiteral failed	Specified literal is not accessible in the database using NNFMgrGet Literal.	Use the secondary Format Management API error code to resolve the problem.
-4241	NNFIEE_GetFirst OutMstrCntl	GetFirstOutMstr Cntl failed	First output master control is not accessible in the database using NNFMgrGetFirst OutMstrCntl.	Use the secondary Format Management API error code to resolve the problem.
-4242	NNFIEE_GetFirst DefaultCntl	GetFirstDefaultCntl failed	First default control is not accessible in the database using NNFMgrGetFirst DefaultCntl.	Use the secondary Format Management API error code to resolve the problem.
-4243	NNFIEE_GetFirst UserExitCntl	GetFirstUserExit Cntl failed	First user exit control is not accessible in the database using NNFMgrGetFirst UserExitCntl.	Use the secondary Format Management API error code to resolve the problem.

Code	Error Name	Error Message	Explanation	Response*
-4244	NNFIEE_GetFirst PrePostFixCntl	GetFirstPrePostFix Cntl failed	First pre / postfix control is not accessible in the database using NNFMgrGetFirst PrePostFixCntl.	Use the secondary Format Management API error code to resolve the problem.
-4245	NNFIEE_GetFirst SegmentFromMath ExpCntl	GetFirstSegment FromMathExpCntl failed	First segment of math expression detail control is not accessible in the database using NNFMgrGetFirst SegmentFromMath ExpCntl.	Use the secondary Format Management API error code to resolve the problem.
-4246	NNFIEE_Append SegmentToMathExp Cntl	AppendSegmentTo MathExpCntl failed	Attempt to append a math expression detail entry to an existing math expression failed using NNFMgrAppend SegmentMathExp Cntl.	Use the secondary Format Management API error code to resolve the problem.
-4247	NNFIEE_GetFirst SubstituteCntl	GetFirstSubstitute Cntl failed	First substitute control is not accessible in the database using NNFMgrGetFirst SubstituteCntl.	Use the secondary Format Management API error code to resolve the problem.
-4248	NNFIEE_GetFirst SubStringCntl	GetFirstSubString Cntl failed	First substring control is not accessible in the database using NNFMgrGetFirst SubStringCntl.	Use the secondary Format Management API error code to resolve the problem.

Code	Error Name	Error Message	Explanation	Response*
-4249	NNFIEE_GetFirst TrimCntl	GetFirstTrimCntl failed	First trim control is not accessible in the database using NNFMgrGetFirst TrimCntl.	Use the secondary Format Management API error code to resolve the problem.
-4250	NNFIEE_GetFirst CollectionCntl	GetFirstCollection Cntl failed	First output collection control is not accessible in the database using NNFMgrGetFirst CollectionCntl.	Use the secondary Format Management API error code to resolve the problem.
-4251	NNFIEE_Append CntlToCollection Cntl	AppendCntlTo CollectionCntl failed	Attempt to append an output operation to an output operation control failed using NNFMgrAppend CntlToCollection Cntl.	Use the secondary Format Management API error code to resolve the problem.
-4252	NNFIEE_GetFirst CntlFromCollection	GetFirstCntlFrom Collection failed	First output operation collection control is not accessible in the database using NNFMgrGetFirst CntlFrom Collection.	Use the secondary Format Management API error code to resolve the problem.
-4253	NNFIEE_GetFirst LengthCntl	GetFirstLengthCntl failed	First length control is not accessible in the database using NNFMgrGetFirst LengthCntl.	Use the secondary Format Management API error code to resolve the problem.

Code	Error Name	Error Message	Explanation	Response*
-4254	NNFIEE_GetFirst MathExpCntl	GetFirstMathExp Cntl failed	First math expression control is not accessible in the database using NNFMgrGetFirst MathExpCntl.	Use the secondary Format Management API error code to resolve the problem.
-4255	NNFIEE_GetNext OutMstrCntl	GetNextOutMstr Cntl failed	Next output master control is not accessible in the database using NNFMgrGetNext OutMstrCntl.	Use the secondary Format Management API error code to resolve the problem.
-4256	NNFIEE_GetOut MstrCntl	GetOutMstrCntl failed	Specified output master control is not accessible in the database using NNFMgrGetOut MstrCntl.	Use the secondary Format Management API error code to resolve the problem.
-4257	NNFIEE_GetNext DefaultCntl	GetNextDefault Cntl failed	Next default control is not accessible in the database using NNFMgrGetNext DefaultCntl.	Use the secondary Format Management API error code to resolve the problem.
-4258	NNFIEE_GetDefault Cntl	GetDefaultCntl failed	Specified default control is not accessible in the database using NNFMgrGet DefaultCntl.	Use the secondary Format Management API error code to resolve the problem.
-4259	NNFIEE_GetNext UserExitCntl	GetNextUserExit Cntl failed	Next user exit control is not accessible in the database using NNFMgrGetNext UserExitCntl.	Use the secondary Format Management API error code to resolve the problem.

Code	Error Name	Error Message	Explanation	Response*
-4260	NNFIEE_GetUser ExitCntl	GetUserExitCntl failed	Specified user exit control is not accessible in the database using NNFMgrGetUser ExitCntl.	Use the secondary Format Management API error code to resolve the problem.
-4261	NNFIEE_GetNext PrePostFixCntl	GetNextPrePostFix Cntl failed	Next pre / postfix control is not accessible in the database using NNFMgrGetNext PrePostFixCntl.	Use the secondary Format Management API error code to resolve the problem.
-4262	NNFIEE_GetPrePost FixCntl	GetPrePostFixCntl failed	Specified prefix / postfix control is not accessible in the database using NNFMgrGetPre PostFixCntl.	Use the secondary Format Management API error code to resolve the problem.
-4263	NNFIEE_GetNext SegmentFromMath ExpCntl	GetNextSegment FromMathExpCntl failed	Next segment of the math expression detail controls is not accessible in the database using NNFMgrGetNext SegmentFromMath ExpCntl.	Use the secondary Format Management API error code to resolve the problem.
-4264	NNFIEE_GetNext SubstituteCntl	GetNextSubstitute Cntl failed	Next substitute control is not accessible in the database using NNFMgrGet SubstituteCntl.	Use the secondary Format Management API error code to resolve the problem.

Code	Error Name	Error Message	Explanation	Response*
-4265	NNFIEE_GetSubstituteCntl	GetSubstituteCntl failed	Specified substitute control is not accessible in the database using NNFMgrGetSubstituteCntl.	Use the secondary Format Management API error code to resolve the problem.
-4266	NNFIEE_GetNextSubStringCntl	GetNextSubStringCntl failed	Next substring control is not accessible in the database using NNFMgrGetNextSubStringCntl.	Use the secondary Format Management API error code to resolve the problem.
-4267	NNFIEE_GetSubStringCntl	GetSubStringCntl failed	Specified substring control is not accessible in the database using NNFMgrGetSubStringCntl.	Use the secondary Format Management API error code to resolve the problem.
-4268	NNFIEE_GetNextTrimCntl	GetNextTrimCntl failed	Next trim control is not accessible in the database using NNFMgrGetNextTrimCntl.	Use the secondary Format Management API error code to resolve the problem.
-4269	NNFIEE_GetTrimCntl	GetTrimCntl failed	Specified trim control is not accessible in the database using NNFMgrGetTrimCntl.	Use the secondary Format Management API error code to resolve the problem.

Code	Error Name	Error Message	Explanation	Response*
-4270	NNFIEE_GetNext CntlFromCollection	GetNextCntlFrom Collection failed	Next output operation collection control is not accessible in the database using NNFMgrGetCntl FromCollection.	Use the secondary Format Management API error code to resolve the problem.
-4271	NNFIEE_GetNext LengthCntl	GetNextLengthCntl failed	Next length control is not accessible in the database using NNFMgrGetNext LengthCntl.	Use the secondary Format Management API error code to resolve the problem.
-4272	NNFIEE_GetLength Cntl	GetLengthCntl failed	Specified length control is not accessible in the database using NNFMgrGet LengthCntl.	Use the secondary Format Management API error code to resolve the problem.
-4273	NNFIEE_GetNext MathExpCntl	GetNextMathExp Cntl failed	Next math expression control is not accessible in the database using NNFMgrGetNextM athExpCntl.	Use the secondary Format Management API error code to resolve the problem.
-4274	NNFIEE_GetMath ExpCntl	GetMathExpCntl failed	Specified math expression control is not accessible in the database using NNFMgrGetMath ExpCntl.	Use the secondary Format Management API error code to resolve the problem.

Code	Error Name	Error Message	Explanation	Response*
-4275	NNFIEE_GetNext CollectionCntl	GetNextCollection Cntl failed	Next output collection control is not accessible in the database using NNFMgrGetNext CollectionCntl.	Use the secondary Format Management API error code to resolve the problem.
-4276	NNFIEE_Get CollectionCntl	GetCollectionCntl failed	Specified output collection control is not accessible in the database using NNFMgrGet CollectionCntl.	Use the secondary Format Management API error code to resolve the problem.
-4277	NNFIEE_GetUser DefinedType	GetUserDefined Type failed	Specified user-defined type is not accessible in the database using NNFMgrGetUser DefinedType.	Use the secondary Format Management API error code to resolve the problem.
-4278	NNFIEE_GetNext MathExpression	GetNextMath Expression failed	Next math expression is not accessible in the database using NNFMgrGetNext MathExpression.	Use the secondary Format Management API error code to resolve the problem.
-4279	NNFIEE_GetNext LookupEntry	GetNextLookup Entry failed	Next lookup entry control is not accessible in the database using NNFMgrGetNext LookupEntry.	Use the secondary Format Management API error code to resolve the problem.

Code	Error Name	Error Message	Explanation	Response*
-4280	NNFIEE_GetNext EntryFromSubstitute Cntl	GetNextEntryFrom SubstituteCntl failed	Next substitute field segment from the substitute control is not accessible in the database using NNFMgrGetNext EntryFrom SubstituteCntl.	Use the secondary Format Management API error code to resolve the problem.
-4281	NNFIEE_Append EntryToSubstitute Cntl	AppendEntryTo SubstituteCntl failed	Attempt to create a substitute field segment for the substitute control failed using NNFMgrAppend EntryToSubstitute Cntl.	Use the secondary Format Management API error code to resolve the problem.
-4500		Fatal internal error	Processing could not continue.	See previous error messages for further information.

*For additional Response information, see Formatter Management API error messages in the ***Programming Reference for Formatter APIs***.

Migrating Rules

Before you migrate rules, run the MQIntegrator r. 3.2 Consistency Checker program on the MQIntegrator r. 3.2 database to check for database errors. Fix any problems found with the Consistency Checker in the MQIntegrator r. 3.2 database before you proceed. See *Consistency Checker* on page 179.

Verify that the target database has enough space for the information to be migrated. Use NNRIE to export existing rules from a MQIntegrator 3.2 database and import to a MQSeries Integrator 1.1 database. Run the NNRIE executable to export rules from the MQIntegrator 3.2 database. NNRIE creates a text-based export file that can be interchanged between platforms. All application groups and their associated message types and rules must be exported.

The exported file should then be imported. Run the NNRIE executable again to import rules to the MQSeries Integrator 1.1 database.

WARNING!

If you are using a case-insensitive database, you cannot name components the same with only a change in case to identify them. For example, you cannot name one rule “r1” and another rule “R1”. In a case-insensitive environment, you must make each item unique using something other than case differences.

If you import components exported from a case-sensitive database into a case-insensitive database, NNRIE may fail during import if a conflict arises between two components named the same with only case differences.

Character Sort Order

Rules using string comparisons can evaluate differently on an EBCDIC machine and on an ASCII machine. The user must be careful when importing a rule set from an ASCII machine to an EBCDIC machine (or vice versa).

In ASCII, the order of characters is 0 - 9 < A - Z < a - z.

For EBCDIC, the order of characters is a - z < A - Z < 0 - 9.

For NEONRules, this difference in character sort has the following consequences:

Rule Argument	Results
int or float comparison	no difference
string comparison (=)	no difference
string comparison with only numeric characters	no difference
string comparison with only alphabetic characters	no difference
case-sensitive string comparison with only uppercase or lowercase alphabetic characters	no difference
string comparison (<, <=, >, >=) with alphabetic and numeric characters	possibly different
case-sensitive string comparison (<, <=, >, >=) with mixed case alphabetic characters	possibly different

Importing and Exporting Rules

NNRIE is used to export rule definitions and subscriptions not associated with a rule from a database to a file, and to import the exported file into a database.

Subscriptions are added to an Application Group / Message Type (Rule Set), and 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.

NNRIE allows the user to export rule definitions from a database to a file and to import the exported file into a database. With the NNRIE program, you can export subscriptions, single rules, rulesets, messages types, and application groups.

Using NNRIE to export rules requires the following preparation:

- DB2 must be installed.
- The operating system supports standard input, standard output, and standard error stream sources and sinks (SYSIN, SYSPRINT, SYSOUT).
- The Rules database schema and the Formatter database schema must be created.
- The Rules data in the database must be created using the Rules GUI or the Rules Management APIs.
- The target database has enough disk space allocated to hold the output file.

Note:
The user must unencrypt an NNRIE export file created on an ASCII platform prior to running NNRIE on OS/390. For more information, see *File Encryption* and *NNCRYPT*.

The SQLSVSES DD-name must reference a valid dataset containing valid SQLSVSES entries, or the application fails to connect to DB2 and terminates. When importing rules on OS/390, the owner of the rules is set to the userID of the person submitting the NNRIE import job, for example, the DB2 special register USER.

When exporting, the DCB attributes of the export files should be set to DSORG=PS, RECFM=VB, LRECL=32756, BLKSIZE=32760. The export records may be very large.

Note:
The WORKFILE, FAILFILE, IMPORTFL, and any other non-print class files should be allocated with the same DCB attributes before the job is executed.

NNRIE

The following sample job control language (JCL) is provided to illustrate how to run the NNRIE job in batch and pass startup parameters to it. The JCL at

your site will be different. See *Tailoring Jobs for Your Site* on page 17 for information about the symbolic parameters in this sample.

```
//*  <insert a valid jobcard here >
//*
//*********************************************************************
//*                                                                   *
//* Licensed Materials - Property of New Era of Networks, Inc.        *
//* Copyright (c) 1998-1999, New Era of Networks, Inc.                *
//* All Rights Reserved.                                              *
//*                                                                   *
//* Release 4.1.1                                                     *
//*********************************************************************
//*********************************************************************
//*                                                                   *
//*NNRIE: Rules Import/Export Utility                                 *
//*                                                                   *
//*********************************************************************
//NNRIE    PROC PRM=('-export DD:EXPORTFL -o -v -v -v '),
//         SMPHLQ='<smphlq>',      HLQ for NEONet distrib libs
//         MQSHLQ='<mqshlq>',      HLQ for MQS runtime libs
//         CEEHLQ='<lehlq>',       HLQ for Lang Envir libs
//         CSSHLQ='SYS1',          HLQ for Callable Sys Svcs (CSS-)Lib
//         SQLMEM='SQLSVSES',      MEMbername for SQLSVSES cntl cards
//         INIMEM='CLIINI',        MEMbername for CLIINI cntl cards
//         OPCLAS='*'              SYSOUT CLASS
//*
//STP0101  EXEC PGM=NNRIE,
//   PARM=&PRM
//*
//*   <tailor the member STEPLIB and copy it here>
//*
//SQLSVSES DD  DSN=&SMPHLQ..SNEOCNTL(&SQLMEM),DISP=SHR
//DSNAOINI DD  DSN=&SMPHLQ..SNEOCNTL(&INIMEM),DISP=SHR
//SYSOUT   DD  SYSOUT=&OPCLAS
//SYSPRINT DD  SYSOUT=&OPCLAS
//STATLOG  DD  SYSOUT=&OPCLAS
//CLITRACE DD  SYSOUT=&OPCLAS  used for DB2 CLI high-level tracing
//SYSIN    DD  DUMMY
//         PEND
//*
//* All datasets used by MQSeries Integrator must be preallocated and
//* cataloged prior to running any MQSeries Integrator jobs. The
```

```
//* recommended DCB attributes are:
//* DSORG=PS,RECFM=VB,LRECL=32756,BLKSIZE=32760
//*
//*
//*
//*SAMPLE EXPORT JCL
//*
//*NNRIE    EXEC NNRIE,
//*   PRM=('-export DD:EXPORTFL -o ')
//*//FAILFILE DD  DISP=SHR,DSN=<your-fail-file>
//*//RIEWRKFL DD  DISP=SHR,DSN=<your-work-file>
//*//EXPORTFL DD  DISP=SHR,DSN=<your-rules-export-file>
//*//INVTRYFL DD  DISP=SHR,DSN=<your-rules-export-file>
//*//ALTCMD   DD  DUMMY
//*
//*SAMPLE IMPORT JCL
//*
//NNRIE    EXEC NNRIE,
//   PRM=('-import DD:IMPORTFL -o ')
//FAILFILE DD  DISP=SHR,DSN=<your-fail-file>
//RIEWRKFL DD  DISP=SHR,DSN=<your-work-file>
//IMPORTFL DD  DISP=SHR,DSN=<your-rules-import-file>
//NNRIET   DD  DISP=SHR,DSN=<your-rules-export-file>
//CNFLCTFL DD  DISP=SHR,DSN=<your-rules-export-file>
//ALTCMD   DD  DUMMY
```

Syntax

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

Parameters

Name	Mandatory/ Optional	Description
-C [<command filename>]	Optional	Alternate command file. The default is DD:ALTCMD. If -C is provided, NNRIE reads command line options from the specified file instead of the command line.
-V (version)	Optional	Shows program version information only and does no processing.
-i | -import [<import filename>]	Mandatory	Required parameter to import data; mutually exclusive from -e. This parameter can include the name of a file that contains the import data. The default is DD:IMPORTFL. The referenced file must have been created with the NNRIE -e option.
-e | -export [<export filename>]	Mandatory	Required parameter to export data; mutually exclusive from -i. This parameter can include the name of a file that contains the export data. The default is DD:IMPORTFL.
-s [<session name>]	Optional	Name of session name in SQLSVSES; defaults to NNRIE.
-c <configuration filename>	Optional	Indicates the name of the configuration file. The default configuration file is SQLSVSES.

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 is displayed. If no export options are provided (-a, -m, -r, or -S), the entire database is exported.

Importing Rules

Syntax

```
NNRIE (-i | -import [<import filename>]
      [-T [<trace filename>]]
      [-o|-O|-l [<inventory conflict report filename>]|-g|-n]
      [-f [<failure filename>]]
      [-s [<session name>]])
```

Parameters

Name	Mandatory/ Optional	Description
-i | -import [<import filename>]	Mandatory	Required parameter to import data; mutually exclusive from -e. This parameter can include the name of a file that contains the import data. The default is DD:IMPORTFL.
-T [<trace filename>]	Optional	Specifies the name of the trace file. The default is DD:NNRIET.
-O	Optional	Overwrites imported message types. The default behavior is off (do not overwrite).
-o (overwrite flag)	Optional	The default behavior is off (do not overwrite). If this parameter is set during import, and a rule or subscription defined in the import file exists in the importing database, the old rule is overwritten with the new definition if you have update permission. If the user does 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 and is different in expression or subscription links in the importing database is noted.

Name	Mandatory/ Optional	Description
-l [<conflict report filename>]	Optional	Reports on any import conflicts, but does not import data. The default behavior is off. The default is DD:CNFLCTFL.
-f [<failure filename>]	Optional	Specifies the failure file that contains lines not imported. The default is FAILFILE. This file can be used as an import file after the issues causing the failure are addressed.

Use the following syntax to overwrite component by component:

```
NNRIE -i <filename> -o
```

Use the following syntax to import and completely overwrite the application group / message type pair in the database:

```
NNRIE –i DD:IMPORTFL –O
```

Tracing Import Progress

The user can track the progress of the import using the trace option. Use the following syntax to trace the command and save to a log file.

To track the progress on an import, enter the following:

```
NNRIE –i DD:NNRIET –T
```

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 alphabetic characters define import and export components that have been processed by NNRIE. These characters are displayed during import and export as progress indicators:

Character	Description
A	Application Group
M	Message Type
R	Rule
e	Rule expression written to file or added to database
n	Permission (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 rule in database

Resolving Component Conflicts

A conflict occurs when an imported rule or subscription does not match an existing component of the same name and type in the database. The user can overwrite the component definition, ignore or skip the component, or rename the imported component.

When a component is overwritten, the component definition within the export file is imported into the database.When a component is ignored or skipped, the component in the export file is not imported. Skipping or overwriting components does not affect the integrity of the database, however, a cleanup utility might have to be implemented. If the user renames a component, all references to that component in the export file are updated.

The user can choose to resolve conflicts in interactive mode or batch mode. Interactive conflict resolution is the default option. All conflicts and resolutions are reported to the NNRIE log file.

Use the following syntax to produce a report of import conflicts. Unless the user specifies a filename, the report is written to the NNRIE log file.

```
NNRIE -i <import filename> -l <conflict report filename>
```

If rule or 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.

Note:
Interactive mode is only available for use on a Windows NT workstation or UNIX-based workstation. It is not available on the OS/390 platform.

Resolving Conflicts in Batch Mode

Overwrite and Ignore/Skip options are available for resolving conflicts in batch mode. The selected option is used to resolve all conflicts.

Use the following code to implement batch Overwrite conflict resolution:

```
NNRIE -i <filename> -o
```

Use the following code to implement batch Ignore/Skip conflict resolution:

```
NNRIE -i <filename> -g
```

Use the following syntax to replace an entire application group/message type pair:

```
NNRIE –i <filename> –O
```

This command deletes each message type 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.

Resolving Conflicts in Interactive Mode

Overwrite, Ignore/Skip, and Rename options are available for resolving conflicts in interactive mode on an Windows NT workstation or UNIX-based workstation. If the user implements interactive conflict resolution, descriptions of the existing components and the import components are displayed.

Use the following syntax to implement interactive conflict resolution:

```
NNRIE -i <filename> -n
```

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

Troubleshooting Import Problems

If NNRIE is unable to import an application group, message type, rule or subscription, the error information is written to the DD:FAILFILE file. The reason for the component's failure to import is written to the DD:RIEWRKFL file.

Refer to the documentation on the import file format for instructions on editing the DD:FAILFILE file if it is version 1.1.

Note:
The information in DD:FAILFILE 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.

All conflicts and resolutions are reported to the DD:RIEWRKFL file. The following code illustrates a sample DD:RIEWRKFL 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
```

Exporting Rules

Syntax

```
NNRIE (-e | -export [<export filename>]
      [-t [<inventory report filename>]]
      [-a <appname>]
      [-m <msgname>]
      [-r <rulename>] | [-S <subsname>]
      [-o]
      [-q <comments in double quotes>]
      [-Q <comments filename>]
      [-s [<session name>])
```

Parameters

Name	Mandatory/ Optional	Description
-e | -export [<export filename>]	Mandatory	Required parameter to export data; mutually exclusive from -i. This parameter can include the name of a file that contains the export data. The default is DD:IMPORTFL.
-a [<appname>]	Optional	Identifies an application group to export. If a value for this parameter is not defined, all application groups are exported. This parameter can be repeated to define multiple application groups to export.
-m [<msgtype>]	Optional	Specifies the message type to export. This parameter requires the -a parameter. Default is export all message types within the specified application group. This parameter can be repeated to define multiple message types within the same application group.

Name	Mandatory/ Optional	Description
-S [<subsname>]	Optional	Specifies the name of the subscription to export. This parameter requires the -a and -m parameters. This parameter can be repeated to export multiple subscriptions.
-r [<rulename>]	Optional	Specifies the name of the rule to export. This parameter requires the -a and -m parameters. Default is export all rules within the specified application group and message type. This parameter can be repeated to define multiple rules within the same application group and message type.
-t [<inventory report filename>]	Optional	Creates an inventory of an export file in DD:INVTRYFL (does no processing).
-o (overwrite flag)	Mandatory for OS/390 only	The default behavior is off (do not overwrite). If this parameter is set during export, the export file is overwritten.
-q <comments in double quotes>	Optional	Adds comments enclosed in quotation marks to beginning of export file SQLSVSES.
-Q <comments filename>	Optional	Adds contents of <comments filename> to beginning of export file.

Exporting an Entire Database

Use the following syntax to export an entire database:

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

Exporting a Single Application Group

```
NNRIE -e [-a <appname>]
```

The application group name is exported, and then each message type within the application group is exported. The message type export includes all subscriptions and rules in the specified application group / message type.

Exporting a Message Type for an Application Group

```
NNRIE -e [-a <appname>] [-m <msgname>]
```

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

Exporting a Single Rule

```
NNRIE -e [-a <appname>] [-m <msgname>] [-r <rulename>]
```

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

Exporting a single subscription

```
NNRIE -e [-a <appname>] [-m <msgname>] [-S <subsname>]
```

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

Producing an Inventory Export File

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

```
NNRIE –t DD:INVTRYFL
```

The following code illustrates a sample inventory report for NNRIE export file named DD:INVTRYFL:

```
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<
 19990601120000 & 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 Readable Files

Earlier versions of NNRIE exported and imported encrypted files. In earlier releases, the only access to the Rules database configurations was through the Rules GUI. With the export files in a readable form, the user can write or modify scripts that create NEONRules components.

WARNING!
Use the Rules GUI to modify rules and subscriptions. The following information is provided for users who are experienced with importing and exporting rules.

NNRIE Header

When an NNFIE export file is created, a header file can be added to the beginning of the file. This file logs information about the data source and creation of the export file.

Use the following code to add a header file:

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

The header file contains the following information:

1. Time of creation (using Greenwich Mean Time)
2. Version of NNRIE
3. Database logon information
4. Database server version

5. Operating system version

The first character in the header file is a pound sign (#), which indicates that the comments should be ignored by NNRIE during import. The user can specify additional comments using the following export option:

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

Formatting NNRie Export Data

Each NEONRules component definition begins with a five-digit number that is the rule component type code defining the layout for the line, for example, 10001. The following structural concepts can help the user navigate through a typical NNRIE export file.

- The first line contains only an R (for Rules).
- The second line indicates the version number of the export file, 1.1.
- Commas are the field delimiters.

  Do not put spaces around commas.
- A comma used within a field must be preceded by a backslash (\).
- Components of an application group / message type must be listed in the following order:
  - application group
  - message type
  - subscription definitions
  - rules definitions
- Subscriptions must be listed before rules in the file.

The following pseudocode illustrates the structure of an NNRIE export file:

```
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)
}
```

Rule Components

The following code illustrates a sample component inventory file. Each of the components is described. For more information about the components, see the *Rules* chapter.

```
R
10001,4.1
10002,sja
10003,sja,InFlat,NEONET_FORMATTER
10007,sja,InFlat,,s1,,1998/07/14-09:44:43.0,1998/07/1409:44:43.0,1
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,
      1998/07/1708:59:19.0
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,
      1998/07/1708:59:20.0
10013,sja,InFlat,r2,s1
10013,sja,InFlat,r2,s2
```

(10001) Import/Export Version

```
10001,1.1
```

1.1 is the version number.

(10002) Application Group

```
10002,sja
```

sja is the application group.

(10003) Message

```
10003,sja,InFlat,NEONET_FORMATTER
```

InFlat is the import format name.

NEONET_FORMATTER is the evaluation type. This message type refers to an input format. For 1.1, this is the only valid evaluation type.

(10004) Rule

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

Component	Description
r1	Rule name.
1	Number of arguments.

Component	Description
0,0	Not used; ignore these values.
1	Active flag;1 is active, 0 is inactive.

(10007) Subscription

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

Component	Description
s2	Subscription name; preceded and followed by NULL values, delimited by commas.
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

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

Component	Description
s2	Subscription name; preceded by a NULL value, delimited by commas.
putqueue	Subscription action.
1	Action sequence number.

(10009) Option

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

Component	Description
OPT_TARGET_QUEUE	Option name.
1	Option sequence number.
HitQ	Option value.

(10010) Rule Permission

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

Component	Description
r1	Rule name.
PUBLIC	Permission group.
Update	Permission assigned to PUBLIC for this rule.
Granted	Permission assigned to PUBLIC for this rule.

(10011) Rule Expressions

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

Component	Description
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

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

Component	Description
s2	Subscription name; preceded by a NULL value.
RUL40RUTH	User name.
Update	Permissions assigned to RUL40RUTH.
Granted	Permissions assigned to RUL40RUTH.

(10013) Rule – Subscription Association

```
10013,sja,InFlat,r1,s1
```

Component	Description
r1	Rule name.
s1	Links the subscription name to the rule name.

Chapter 4
Formatter

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

What is Formatter?

NEONFormatter has two main functions: parsing and reformatting.

- Parse separates an input message into individual fields.
- Reformat converts an input message 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 Formatter GUI tool or the Formatter Management API functions.

The NEONFormatter GUI tool is a program with a graphical user interface that lets users enter format definition data and this information is then stored in a relational database.

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

APITEST and MSGTEST are two modules that show how to invoke the public functions and serve as tools for validating format definitions. The APITEST

module parses an input message and displays a hierarchical representation of the parse tree. The MSGTEST module reformats an input message into an output message.

The Consistency Checker verifies the integrity of the format definition data in the relational database. The Consistency Checker should be run periodically when format definition data is being built or maintained to insure the integrity of the data.

NNFIE 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 MQIntegrator r.3.2 export file into an MQSeries Integrator 1.1 database. NNFIE 1.1 exports data from an MQSeries Integrator 1.1 database only.

The NEONFormatter GUI tool has its own import and export function as well. This function uses an export file with a format different from the format 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. Fields are defined by a unique name and input control information used to define their beginning and end. Fields are cohesive parts of a message representing some type of information.

Each field has an associated parse control describing how to identify the field in the message. Input control information includes the data type for the field, tags preceding and 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 appears in a message.

NEONFormatter supports several data types including String, Numeric, and Binary. See *Data Type Descriptions* on page 215 for a complete list of supported data types.

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 can only contain 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 string field data in input fields. Instead of searching for a defined literal, use an RE to search for complex string patterns in field data. String-matching capabilities implemented comply with the POSIX 1003.2 standard for regular expressions.

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

Output Controls

Each field in an input message that appears in an output message or is used to affect a resulting field in an output message 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 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.

Both flat and compound input formats contain fields and parse controls, and are used to parse messages so they can be reformatted according to flat or compound output formats.

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

Using the reformat() API, 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 the parse() API to divide 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.

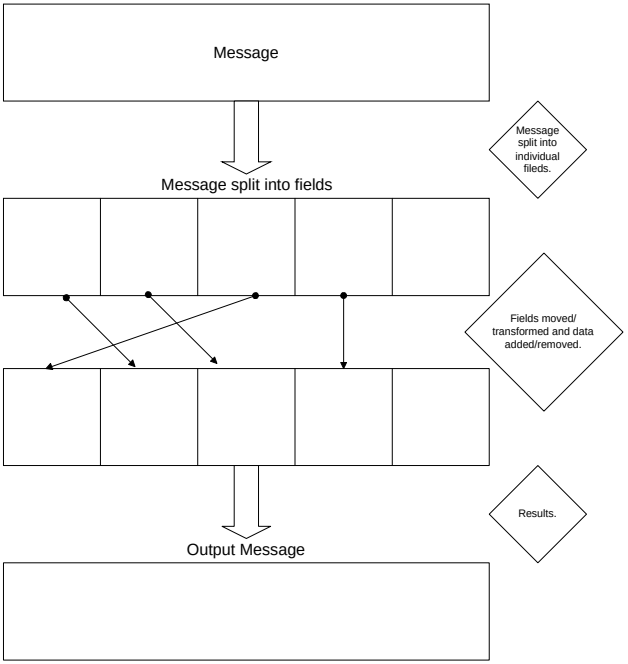
Possible transformations NEONFormatter can handle include:

- Adding, removing, or rearranging data, literals, tags, and delimiters.
- 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 translation functions.
- Executing user-written callback functions for user-defined type input field validation.

Parsing and Reformatting

NEONFormatter parses a message, using the parse() API to divide a message into the individual fields specified in its input control. When a message is parsed, the intermediate field results can be used.

The parsed message can then be reformatted using the reformat() API in a specified output message format. If the message provided to reformat() has not been parsed using parse(), reformat() calls parse() before reformatting the message.



Message Formatting

Automatic Format Conversion

Higher-level APIs can request NEONFormatter to reformat messages just before delivery to the receiving application by invoking dynamic formatting as a get option. Reformatting locations can vary, depending on the location of resources, such as source data, necessary to format the new message.

Testing Formats

Two test programs are provided with Formatter: MSGTEST and APITEST. The APITEST program parses a message and outputs the message structure and contents; it does not reformat a message. The MSGTEST program provides testing for the entire range of Formatter functionality.

Before running the Formatter test programs, you must verify that the SQLSVSES file includes the relevant information to be used to execute this program. The session name for the Formatter modules is new_format_demo. The syntax is:

```
new_format_demo:DB2PROD:USER001:XXXXX:USR001:
```

Run MSGTEST to test input and output formats. Run APITEST to validate input formats and to view how Formatter interpreted a message. The source code for MSGTEST and APITEST is included in the ***Programming Reference for NEONFormatter***. See MSGTEST and APITEST members in the <smphlq>.SNEOCPP library. Refer to this source code for using the Formatter API functions.

Note:

With the addition of Formatter debug functions, both MSGTEST and APITEST have an additional command line parameter:

```
msgtest ... -d [filename]
apitest ... -d [filename]
```

This function sets debugging mode to parse for this run of MSGTEST and APITEST. [filename] specifies an optional file where debug information will be written. If [filename] is not specified, debug information is written to the standard output stream (STDOUT) SYSPRINT. The default DD-name in the distribution JCL is DD:DBGLOG.

APITEST

The APITEST module outputs the structure and contents of a message parsed by Formatter. The input parameters (input filename and input format name) are provided in the SYSIN data stream:

The APITEST module calls the following APIs:

- Formatter::GetParsedInMsgCount
- Formatter::GetParsedInMsg
- ParsedMessage::GetMsgComp
- ParsedMessage::GetInfo
- ParsedMessage::GetCompCount
- ParsedMessage::GetFieldComp
- ParsedField::GetInfo
- ParsedField::GetStringValue
- ParsedField::GetValue

Compiling and Linking APITEST

The following sample job control language (JCL) is used to compile and link APITEST:

```
//* <insert a valid JOBCARD here>
//*
//************************************************************
//*                                                          *
//* Licensed Materials - Property of New Era of Networks, Inc.*
//* Copyright (c) 1998-1999, New Era of Networks, Inc.       *
//* All Rights Reserved.                                     *
//*                                                          *
//* Release 4.1.1                                            *
//************************************************************
//PROCLIST JCLLIB ORDER=(<smphlq>.SNEOPROC,SYS1.PROCLIB)
//COMPILE   EXEC  CBCCL15,
//    INFILE='<smphlq>.SNEOCPP',
//    INCFILE='<smphlq>.SNEOH',
//    OUTFILE='<smphlq>.SNEOLOAD',
//    MEMBER='APITEST'
//LKED.SYSIN DD *
  NAME APITEST(R)
```

```
//
```

APITEST

The following sample job control language (JCL) is provided to illustrate how to run the APITEST job in batch and pass startup parameters. The JCL at your site will be different. See *Tailoring Jobs for Your Site* on page 17 for information about the symbolic parameters in these samples.

```
//*  <insert a valid jobcard here >
//*
//*********************************************************************
//*                                                                   *
//* Licensed Materials - Property of New Era of Networks, Inc.        *
//* Copyright (c) 1998-1999, New Era of Networks, Inc.                *
//* All Rights Reserved.                                              *
//*                                                                   *
//* Release 4.1.1                                                     *
//*********************************************************************
//*********************************************************************
//*                                                                   *
//* APITEST: Test Formatter parse function                            *
//*                                                                   *
//*********************************************************************
//APITEST PROC PRM=('-d DD:DBGLOG'), run-time parms
//        SMPHLQ='<smphlq>',      HLQ for NEONet distrib libs
//        MQSHLQ='<mqshlq>',      HLQ for MQS run-time libs
//        CEEHLQ='<lehlq>',       HLQ for Lang Envir libs
//        CSSHLQ='SYS1',          HLQ for Callable Sys Svcs (CSS-) Lib
//        SQLMEM='SQLSVSES',      MEMbername for SQLSVSES cntl cards
//        INIMEM='CLIINI',        MEMbername for CLIINI cntl cards
//        OPCLAS='*'              DEFAULT SYSOUT CLASS
//*
//STP0101  EXEC PGM=APITEST,PARM=&PRM
//*
//*   <tailor the member STEPLIB for your site and copy it here>
//*
//SQLSVSES DD  DSN=&SMPHLQ..SNEOCNTL(&SQLMEM),DISP=SHR
//DSNAOINI DD  DSN=&SMPHLQ..SNEOCNTL(&INIMEM),DISP=SHR
//SYSUDUMP DD  SYSOUT=&OPCLAS
//SYSOUT   DD  SYSOUT=&OPCLAS
//SYSPRINT DD  SYSOUT=&OPCLAS
```

```
//STATLOG  DD  SYSOUT=&OPCLAS
//CLITRACE DD  SYSOUT=&OPCLAS  used for DB2 v5 CLI high-level tracing
//         PEND
//*
//* All datasets used by MQSeries Integrator must be preallocated and
//* cataloged prior to running any MQSeries Integrator jobs. The
//* recommended DCB attributes are:
//*    DSORG=PS,RECFM=VB,LRECL=32756,BLKSIZE=32760
//*
//APITEST  EXEC APITEST,PRM=(' -d DD:DBGLOG')
//DBGLOG   DD  SYSOUT=*
//APIIN    DD  DISP=SHR,DSN=<your-message-file>
//SYSIN    DD  *
DD:APIIN
<your-format-name>
/*
//
```

MSGTEST

The MSGTEST module adds an input message and an output format to Formatter, reformats the message text provided in an input file, and outputs the results to an output file. The input and output parameters are provided in the SYSIN data stream:

- input file name that contains the message text
- output file name
- input format name
- output format name

The MSGTEST module calls the following APIs:

- Formatter::AddInputMessage
- Formatter::AddOutputFormat
- Formatter::Reformat
- Formatter::GetOutMsgGroup
- OutMsgGroup::GetMsgCount
- OutMsgGroup::GetMsg

- OutMsg::GetMsgBuffer
- OutMsg::GetMsgLength

Compiling and Linking MSGTEST

The following sample job control language (JCL) compiles and links MSGTEST:

```
//* <insert a valid jobcard here>
//*
//*************************************************************
//*                                                           *
//* Licensed Materials - Property of New Era of Networks, Inc.*
//* Copyright (c) 1998-1999, New Era of Networks, Inc.        *
//* All Rights Reserved.                                      *
//*                                                           *
//* Release 4.1.1                                             *
//*************************************************************
//PROCLIST JCLLIB ORDER=(<smphlq>.SNEOPROC,SYS1.PROCLIB)
//COMPILE   EXEC  CBCCL15,
//    INFILE='<smphlq>.SNEOCPP',
//    INCFILE='<smphlq>.SNEOH',
//    OUTFILE='<smphlq>.SNEOLOAD',
//    MEMBER='MSGTEST'
//LKED.SYSIN DD *
  NAME MSGTEST(R)
//
```

MSGTEST

The following sample job control language (JCL) illustrates how to run the MSGTEST job in batch and pass startup parameters. The JCL at your site will be different. See *Tailoring Jobs for Your Site* on page 17 for information about the symbolic parameters in these samples.

```
//*  <insert a valid jobcard here >
//*
//*********************************************************************
//*                                                                   *
//* Licensed Materials - Property of New Era of Networks, Inc.        *
//* Copyright (c) 1998-1999, New Era of Networks, Inc.                *
//* All Rights Reserved.                                              *
//*                                                                   *
//* Release 4.1.1                                                     *
//*********************************************************************
//*********************************************************************
//*                                                                  *
//* MSGTEST: Test Formatter parse and reformat function              *
//*                                                                  *
//*********************************************************************
//MSGTEST  PROC PRM=('-d DD:DBGLOG'), run-time parms
//         SMPHLQ='<smphlq>',     HLQ for NEONet distrib libs
//         MQSHLQ='<mqshlq>',     HLQ for MQS run-time libs
//         CEEHLQ='<lehlq>',      HLQ for Lang Envir libs
//         CSSHLQ='SYS1',         HLQ for Callable Sys Svcs (CSS-) Lib
//         SQLMEM='SQLSVSES',     MEMbername for SQLSVSES cntl cards
//         INIMEM='CLIINI',       MEMbername for CLI INI cntl cards
//*        OPCLAS='*',            SYSOUT CLASS
//*
//STP0101  EXEC PGM=MSGTEST,
//   PARM=&PRM
//*
//*   <tailor the member STEPLIB and copy it here>
//*
//SQLSVSES DD  DSN=&SMPHLQ..SNEOCNTL(&SQLMEM),DISP=SHR
//DSNAOINI DD  DSN=&SMPHLQ..SNEOCNTL(&INIMEM),DISP=SHR
//SYSOUT   DD  SYSOUT=&OPCLAS
//SYSPRINT DD  SYSOUT=&OPCLAS
//STATLOG  DD  SYSOUT=&OPCLAS
//CLITRACE DD  SYSOUT=&OPCLAS  used for DB2 v5 CLI high-level tracing
```

```
//         PEND
//*
//*
//* All datasets used by MQSeries Integrator must be preallocated and
//* cataloged prior to running any MQSeries Integrator jobs. The
//* recommended DCB attributes are:
//*    DSORG=PS,RECFM=VB,LRECL=32756,BLKSIZE=32760
//*
//* All datasets used by NEONet must be preallocated and cataloged
//* prior to running any NEONet jobs. The recommended DCB attributes
//* are:   DSORG=PS,RECFM=VB,LRECL=32756,BLKSIZE=32760
//*
//MSGTEST  EXEC MSGTEST,
//     PRM=('-d DD:DBGLOG')
//DBGLOG   DD  SYSOUT=*
//INFILE   DD  DISP=SHR,DSN=<your-input-message-file>
//OUTFILE  DD  DISP=SHR,DSN=<your-output-message-file>
//SYSIN    DD  *
DD:INFILE
DD:OUTFILE
<your-input-format-name>
<your-output-format-name>
/*
//
```

Creating Formatter User Exits

A NEONFormatter user exit allows the user to externally customize NEONFormatter. A user exit should be created when the type of reformatting available in the standard Formatter product does not meet the user's requirements. The user-defined exit function replaces the lookup stub NNUESTUB.cpp. See *NNUESTUB* on page 122.

NEONFormatter calls the application to provide the function address. The user-defined exit code shares a location with NEONFormatter. The <smphlq>.SNEOULOD library is provided to hold user-defined user exits. NEONFormatter takes a field from a parsed format and passes the field to the user exit. The value changes as part of the reformat() function, and the new value is passed back to the field. NEONFormatter APIs include C++ user exit APIs. See ***Programming Reference for NEONFormatter APIs***.

Building and Installing a C++ User Exit

On OS/390, C++ user exits are written the same as C++ user exits are written on other platforms. However, to run C++ user exits on OS/390, you construct a DLL that contains all C++ user exits, instead of relinking the program that executes NEONFormatter.

One program contains all the user exits. It is compiled and linked into a DLL. NEONFormatter expects the DLL to be called NNUSER. The default NNUSER DLL is in the <smphlq>.SNEODLL library. The user should link the replacement NNUSER DLL into the <smphlq>.SNEOULOD library. This library is concatenated ahead of the SNEODLL library, so the user-defined version is executed instead of the default NNUSER DLL.

Note:
Do not overwrite the NNUSER member in the SNEODLL library. Save the NNUSER member using a different file name. This NNUSER member has no user exits.

User Exit Summary

The following steps are used to create and use C++ user exit functions:

1. Create user exit and user exit Cleanup functions. Functions must conform to the NN_EXIT_FUNC_t and NN_EXIT_CLEANUP_FUNC_t types defined in the nnexit.h header file.
2. Create a routine named NNGetUserExitFuncPtrs() so that a Formatter instance can look up the function pointers for the user exit and user exit Cleanup functions given an exit function name.
3. Build a DLL called NNUSER, exporting NNGetUserExitFuncPtrs().
4. Set the STEPLIB so that your NNUSER DLL is found before the stub version in the MQSeries Integrator library.
5. In the Formatter GUI, specify the name of the exit routine in the Exit Routine field on the Field Format Output Control Tool window.

The following pseudo-code describes the behavior of a NEONFormatter instance when it encounters a user exit as part of the reformat process:

```
user calls Formatter::Reformat()
formatter detects user exit is present and should be used as
part of output format control
formatter checks STEPLIB to determine if already cached
IF not in STEPLIB THEN
   call NNGetUserExitFuncPtrs()
   IF exit function pointer is not NULL THEN
      exit function and exit clean up function pointers added
      to registry
   ENDIF
ENDIF
IF exit function pointer is not NULL THEN
   call user exit
   IF user exit returns NN_ERSTATUS_OK error status THEN
      IF user exit cleanup defined THEN
      call user exit cleanup function
      IF user exit cleanup fails THEN
         set nonfatal error condition
      ENDIF
      ENDIF
```

```
ELSE
     set fatal error condition
END
```

Note:
A user exit cleanup failure does not cause the Formatter reformat process to fail.

Compiling and Linking UETEST

The following sample job control language (JCL) is used to compile and link MSGTEST as an module called UETEST:

```
//* <insert a valid jobcard here>
//*
//*************************************************************
//*                                                           *
//* Licensed Materials - Property of New Era of Networks, Inc.*
//* Copyright (c) 1998-1999, New Era of Networks, Inc.        *
//* All Rights Reserved.                                      *
//*                                                           *
//* Release 4.1.1                                             *
//*************************************************************
//*
//PROCLIST JCLLIB ORDER=(<smphlq>.SNEOPROC,SYS1.PROCLIB)
//COMPILE   EXEC  CBCCL15,
//    INFILE='<smphlq>.SNEOCPP',
//    INCFILE='<smphlq>.SNEOH',
//    OUTFILE='<smphlq>.SNEOLOAD',
//    MEMBER='MSGTEST'
//LKED.SYSIN DD *
  NAME UETEST(R)
/*
//
```

UETEST

The following code runs the UETEST module built in the previous section.

```
//*  <insert a valid jobcard here>
//*
//*****************************************************************
//*                                                               *
```

```
//* Licensed Materials - Property of New Era of Networks, Inc.       *
//* Copyright (c) 1998-1999, New Era of Networks, Inc.               *
//* All Rights Reserved.                                             *
//*                                                                  *
//* Release 4.1.1                                                    *
//********************************************************************
//********************************************************************
//*                                                                 *
//* UETEST: Test Formatter User Exit processing                     *
//*                                                                 *
//*******************************************************************
//UETEST   PROC PRM=('-d DD:DBGLOG'), run-time PARMs
//         SMPHLQ='<smphlq>',     HLQ for MQI distrib libs
//         MQSHLQ='<mqshlq>',     HLQ for MQS run-time libs
//         CEEHLQ='<lehlq>',      HLQ for Lang Envir  libs
//         CSSHLQ='SYS1',         HLQ for Callable Sys Svcs (CSS-) Lib
//         SQLMEM='SQLSVSES',     MEMbername for SQLSVSES cntl cards
//         INIMEM='CLIINI',       MEMbername for CLI INI cntl cards
//         OPCLAS='*'             SYSOUT CLASS
//*
//STP0101  EXEC PGM=UETEST,PARM=&PRM
//*
//*   <tailor the member STEPLIB and copy it here>
//*
//SQLSVSES DD  DSN=&SMPHLQ..SNEOCNTL(&SQLMEM),DISP=SHR
//DSNAOINI DD  DSN=&SMPHLQ..SNEOCNTL(&INIMEM),DISP=SHR
//SYSUDUMP DD  SYSOUT=&OPCLAS
//SYSOUT   DD  SYSOUT=&OPCLAS
//SYSPRINT DD  SYSOUT=&OPCLAS
//STATLOG  DD  SYSOUT=&OPCLAS
//CLITRACE DD  SYSOUT=&OPCLAS  used by DB2 CLI high-level tracing
//         PEND
//*
//*
//* All datasets used by MQSeries Integrator must be preallocated and
//* cataloged prior to running any MQSeries Integrator jobs. The
//* recommended DCB attributes are:
//*    DSORG=PS,RECFM=VB,LRECL=32756,BLKSIZE=32760
//*
//UETEST   EXEC UETEST,PARM=('-d DD:DBGLOG')
//DBGLOG   DD  SYSOUT=*
//MSGIN    DD DISP=SHR,DSN=<your-input-message-file>
```

```
//MSGOUT   DD DISP=SHR,DSN=<your-output-message-file>
//SYSIN    DD *
DD:MSGIN
DD:MSGOUT
<your-input-format>
<your-output-format>
/*
//
```

Compiling and Linking a Standard User Exit

The following sample Job Control Language (JCL) is used to compile and link the NNUSER DLL into the <smphlq>.SNEOULOD library:

```
//*  <insert a valid jobcard here >
//*
//*************************************************************
//*                                                           *
//* Licensed Materials - Property of New Era of Networks, Inc.*
//*                                                           *
//* Copyright (C) 1998, 1999, New Era of Networks             *
//* Version 4 Release 03                                      *
//*************************************************************
//* This JCL will compile the UETEST functions and create     *
//* a replacement NNUSER DLL in the <smphlq>.SNEOULOD library.*
//*                                                           *
//* The NNUSER DLL is loaded. The functions are called when   *
//* an Output Control of type NewUserExit_1 is created, and   *
//* a message is reformatted using that output control.       *
//*************************************************************
//PROCLIST JCLLIB ORDER=(<smphlq>.SNEOPROC,SYS1.PROCLIB)
//COMPILE   EXEC  CBCCL15,
//    INFILE='<smphlq>.SNEOCPP',
//    INCFILE='<smphlq>.SNEOH',
//    OUTFILE='<smphlq>.SNEOULOD',
//    MEMBER='UETEST',
//    PARM.PLKED=(DLLNAME(NNUSER),NOER),
//    LPARM=(AMODE=31,RMODE=ANY,DYNAM=NO,CALL=YES)
//PLKED.SYSDEFSD DD DISP=SHR,DSN=<smphlq>.SNEOUEXP(NNUSER)
//LKED.SYSIN DD *
  NAME NNUSER(R)
//
```

This compile and link JCL calls a cataloged procedure. This cataloged procedure contains symbolic parameters. You may have to modify the default symbolic parameters for your site. See *Tailoring Jobs for Your Site* on page 17.

```
//*********************************************************************
//*  MQI COMPILE PRELINK AND LINK A C++ PROGRAM                       *
//*  VERSION 4.03                                                     *
//*                                                                   *
//*********************************************************************
//*
//CBCCL PROC INFILE=,             INPUT ... REQUIRED
//   OUTFILE=,                    OUTPUT ... REQUIRED
//   MEMBER=,                     SOURCE MEMBER NAME...REQUIRED
//   INCFILE=,                    SOURCE .H LIBRARY ...REQUIRED
//   CREGSIZ='60M',               COMPILER REGION SIZE
//   LIBPRFX='<ceehlq>',          PRFX LE REQUIRED FOR COMPILER 1.8
//   LEPRFX='<lehlq>',            PRFX LE 1.5 LIBS
//   SMPPRFX='<smphlq>',          PRFX FOR NEON LIBS
//   TCPIP='<tcphlq>',            PRFX FOR TCP/IP LIBS
//   CLBPRFX='<c++hlq>',          PRFX FOR C++ LIBS
//   MQSHLQ='<mqshlq>',           MQ SERIES HLQ FOR LOADLIB
//   DB2HLQ='<db2hlq>',           DB2 HLQ FOR LOADLIB
//   CLANG='EDCMSGE',<NOT USED IN THIS RELEASE. KEPT FOR COMPATIBILITY
// CXXLANG='CBCMSGE',<NOT USED IN THIS RELEASE. KEPT FOR COMPATIBILITY
//   PLANG='EDCPMSGE',               PRE-LINKER MESSAGE NAME
//   PREGSIZ='2048K',                PRE-LINKER REGION SIZE
//   LPARM='AMODE=31,MAP,RENT',      LINKAGE EDITOR OPTIONS
//   TUNIT='SYSALLDA',               UNIT FOR TEMPORARY FILES
//   OPCLAS='*'                      SYSOUT OUTPUT CLASS
//*--------------------------------------------------------------------
//*  COMPILE STEP:
//*--------------------------------------------------------------------
//COMPILE EXEC PGM=CBCDRVR,REGION=&CREGSIZ,
//        PARM=('CXX OPTFILE(DD:OPTION)')
//STEPLIB  DD  DSN=&LIBPRFX..SCEERUN,DISP=SHR
//         DD  DSN=&CLBPRFX..SCBCCMP,DISP=SHR
//OPTION   DD  DSN=&SMPPRFX..SNEOJCL(OPTNOOE),DISP=SHR
//SYSMSGS  DD  DUMMY,DSN=&CLBPRFX..SCBC3MSG(&CLANG),DISP=SHR
//SYSXMSGS DD  DUMMY,DSN=&CLBPRFX..SCBC3MSG(&CXXLANG),DISP=SHR
//SYSIN    DD  DSN=&INFILE(&MEMBER),DISP=SHR
```

```
//USERLIB  DD  DSN=&INCFILE,DISP=SHR,DCB=(RECFM=VB,LRECL=255)
//SYSLIB   DD  DSN=&LEPRFX..SCEEH.H,DISP=SHR
//         DD  DSN=&LEPRFX..SCEEH.SYS.H,DISP=SHR
//         DD  DSN=&LEPRFX..SCEEH.ARPA.H,DISP=SHR
//         DD  DSN=&LEPRFX..SCEEH.NET.H,DISP=SHR
//         DD  DSN=&LEPRFX..SCEEH.NETINET.H,DISP=SHR
//         DD  DSN=&CLBPRFX..SCLBH.H,DISP=SHR
//         DD  DSN=&TCPIP..SEZACMAC,DISP=SHR
//         DD  DSN=&TCPIP..SEZAINST,DISP=SHR
//         DD  DSN=MQSHLQ..SCSQC370,DISP=SHR
//SYSLIN   DD  DSN=&SMPPRFX..SNEOOBJ(&MEMBER),DISP=SHR
//SYSPRINT DD  SYSOUT=&OPCLAS
//SYSOUT   DD  SYSOUT=&OPCLAS
//SYSCPRT  DD  SYSOUT=&OPCLAS
//SYSUT1   DD  UNIT=&TUNIT.,SPACE=(32000,(30,30)),
//             DCB=(RECFM=FB,LRECL=80,BLKSIZE=3200)
//SYSUT4   DD  UNIT=&TUNIT.,SPACE=(32000,(30,30)),
//             DCB=(RECFM=FB,LRECL=80,BLKSIZE=3200)
//SYSUT5   DD  UNIT=&TUNIT.,SPACE=(32000,(30,30)),
//             DCB=(RECFM=FB,LRECL=3200,BLKSIZE=12800)
//SYSUT6   DD  UNIT=&TUNIT.,SPACE=(32000,(30,30)),
//             DCB=(RECFM=FB,LRECL=3200,BLKSIZE=12800)
//SYSUT7   DD  UNIT=&TUNIT.,SPACE=(32000,(30,30)),
//             DCB=(RECFM=FB,LRECL=3200,BLKSIZE=12800)
//SYSUT8   DD  UNIT=&TUNIT.,SPACE=(32000,(30,30)),
//             DCB=(RECFM=FB,LRECL=3200,BLKSIZE=12800)
//SYSUT9   DD  UNIT=&TUNIT.,SPACE=(32000,(30,30)),
//             DCB=(RECFM=VB,LRECL=137,BLKSIZE=882)
//SYSUT10  DD  SYSOUT=&OPCLAS
//SYSUT14  DD  UNIT=&TUNIT.,SPACE=(32000,(30,30)),
//             DCB=(RECFM=FB,LRECL=3200,BLKSIZE=12800)
//SYSUT15  DD  SYSOUT=&OPCLAS
//*
//*--------------------------------------------------------------------
//* PRE-LINKEDIT STEP:
//*--------------------------------------------------------------------
//PLKED  EXEC  PGM=EDCPRLK,REGION=&PREGSIZ,COND=(8,LE,COMPILE),
//       PARM=(DLLNAME(&MEMBER),NOER)
//STEPLIB  DD  DSN=&LIBPRFX..SCEERUN,DISP=SHR
//SYSMSGS  DD  DSN=&LEPRFX..SCEEMSGP(&PLANG),DISP=SHR
//SYSLIB   DD  DSN=&LEPRFX..SCEECPP,DISP=SHR
//SYSIN    DD  DSN=&SMPPRFX..SNEOOBJ(&MEMBER),DISP=SHR
```

```
//         DD  DSN=&CLBPRFX..SCLBSID(IOSTREAM),DISP=SHR
//         DD  DSN=&CLBPRFX..SCLBSID(COMPLEX),DISP=SHR
//         DD  DSN=&CLBPRFX..SCLBSID(ASCCOLL),DISP=SHR
//         DD  DSN=&SMPPRFX..SNEOEXP(FMGR),DISP=SHR
//         DD  DSN=&SMPPRFX..SNEOEXP(MQI),DISP=SHR
//         DD  DSN=&SMPPRFX..SNEOEXP(NNMQS),DISP=SHR
//         DD  DSN=&SMPPRFX..SNEOEXP(NNSQLOBJ),DISP=SHR
//         DD  DSN=&SMPPRFX..SNEOEXP(NRULEFMT),DISP=SHR
//         DD  DSN=&SMPPRFX..SNEOEXP(RMGR),DISP=SHR
//         DD  DSN=&SMPPRFX..SNEOEXP(SQLOBJ),DISP=SHR
//         DD  DSN=&SMPPRFX..SNEOEXP(NNUSER),DISP=SHR
//         DD  DSN=&SMPPRFX..SNEOUEXP(GETVAL),DISP=SHR
//         DD  DSN=&SMPPRFX..SNEOUEXP(OPENFORM),DISP=SHR
//         DD  DSN=&SMPPRFX..SNEOUEXP(RULES),DISP=SHR
//         DD  DDNAME=SYSIN2
//SYSMOD   DD  DSN=&SMPPRFX..SNEOPOBJ(&MEMBER),DISP=SHR
//SYSDEFSD DD  DUMMY
//SYSOUT   DD  SYSOUT=*OPCLAS
//SYSPRINT DD  SYSOUT=*OPCLAS
//SYSIN2   DD  DUMMY
//*
//*--------------------------------------------------------------------
//* LINKEDIT STEP:
//*--------------------------------------------------------------------
//LKED    EXEC PGM=HEWL,REGION=1024K,
//             COND=((8,LT,COMPILE),(8,LE,PLKED)),
//      PARM='&LPARM'
//SYSLIN   DD  DSN=&SMPPRFX..SNEOPOBJ(&MEMBER),DISP=SHR
//         DD  DDNAME=SYSIN
//SYSLMOD  DD  DSN=&OUTFILE(&MEMBER),DISP=SHR
//MQSLOAD  DD  DSN=&MQSHLQ..SCSQLOAD,DISP=SHR
//DB2LOAD  DD  DSN=&DB2HLQ..SDSNLOAD,DISP=SHR
//SYSUT1   DD  UNIT=&TUNIT.,SPACE=(32000,(30,30))
//SYSUT2   DD  UNIT=&TUNIT.,SPACE=(32000,(30,30))
//SYSUT3   DD  UNIT=&TUNIT.,SPACE=(32000,(30,30))
//SYSUT4   DD  UNIT=&TUNIT.,SPACE=(32000,(30,30))
//SYSPRINT DD  SYSOUT=&OPCLAS
```

Parameter	Description	Values for Your Site
<ceehlq>	Language Prefix	

Parameter	Description	Values for Your Site
<lehlq>	Language Environment High Level Qualifier	
<tcphlq>	TCPIP High Level Qualifier	
<mqshlq>	MQSeries Load Library	
<db2hlq>	DB2 Load Library	
<smphlq>	MQSeries Integrator Export Library High Level Qualifiers	
<c++hlq>	C++ run-time and Link Edit Libraries	

Compiling and Linking a C/C++ User Exit Containing Embedded SQL

A sample user exit containing embedded DB2 SQL is provided in the SNEOSRCE library (member UETEST1). This is the same user exit as UETEST in the SNEOCPP library with the following exceptions:

1. The lines are reorganized so that no line extends beyond column 72. The DB2 precompiler requires its input files to be RECFM=F,FB and LRECL=80.
2. Several additional lines of SQL illustrate how to add DB2 SQL statements to your user exits. If a non-zero SQLCODE is returned from the SQL call, it is displayed. If the SQLCODE is zero, then the number of formats in the MQSeries Integrator NNF_FMT table is displayed.

Since NEONFormatter is designed to dynamically load the NNUSER DLL and call the functions contained therein, your user exit should be compiled and linked as a DLL. This DLL should replace any existing NNUSER DLL in the <smphlq>.SNEOULOD library.

A JCL member is provided (USREXIT2) to run the DB2 precompiler against your C/C++ source code, followed by a C/C++ compile, Prelink, and Link. The output of the Link stage is a new NNUSER DLL in the <smphlq>.SNEOULOD library. This JCL uses the CBCDB2CL cataloged

procedure in the <smphlq>.SNEOPROC library. The JCL for USREXIT2 is shown below:

```
//*  <insert a valid jobcard here >
//*********************************************************************
//*                                                                   *
//* Licensed Materials - Property of New Era of Networks, Inc         *
//*                                                                   *
//* Copyright (C) 1998, 1999, New Era of Networks                     *
//* Version 4 Release 03                                              *
//*********************************************************************
//* This JCL will compile the UETEST1 functions and create
//* a replacement NNUSER DLL in the <smphlq>.SNEOULOD library.
//*
//* The NNUSER DLL is loaded and the functions are called when
//* an Output Control of type NewUserExit_1 is created and
//* a message is reformatted using that output control.
//*
//* NOTE that the uetest1.cpp function is in the <smphlq>.SNEOSRCE
//* library because the DB2 precompiler requires its input files
//* to be RECFM=F,FB and LRECL=80.
//*
//* NOTE that you must rebind your plan to include all the
//* DBRMs for the IBM Call Level Interface as well as your
//* own application and user exit DBRMs. See JCL member
//* BINDUSRX for control statements to do this.
//*********************************************************************
//PROCLIST JCLLIB ORDER=(<smphlq>.SNEOPROC,SYS1.PROCLIB)
//COMPILE   EXEC  CBCDB2CL,
//    INFILE='<smphlq>.SNEOSRCE',
//    INCFILE='<smphlq>.SNEOH',
//    OUTFILE='<smphlq>.SNEOULOD',
//    DBRMLIB='<your-dbrm-library-here>',
//    MEMBER='UETEST1',
//    PARM.PLKED=(DLLNAME(NNUSER),NOER),
//    LPARM=(AMODE=31,RMODE=ANY,DYNAM=NO,CALL=YES)
//PLKED.SYSDEFSD DD DISP=SHR,DSN=<smphlq>.SNEOUEXP(NNUSER)
//LKED.SYSIN DD *
  INCLUDE DB2LIB(DSNALI)
  NAME NNUSER(R)
//
```

Binding a PLAN

Because MQSeries Integrator uses the IBM Call Level Interface (CLI) to access DB2 and your User Exit contains embedded SQL, there is an additional step required before you can use your User Exit. You must bind a plan that contains all the DBRMs for the IBM CLI as well as your own application and User Exit DBRMs. The JCL member BINDUSRX is designed to bind the sample user exit DBRM with the IBM packages that were bound when the CLI was first installed to create a new plan.

Prior to running MQSeries Integrator and your user exit, you must grant execute authority on the plan to the users who will be executing it. The PLANNAME=DSNACLI line in the CLIINI file (<smphlq>.SNEOCNTL library) must be changed to use your new plan name. The BINDUSRX JCL is shown below:

```
//*  <insert a valid jobcard here >
//*
//*********************************************************************
//*  This JCL is distributed for r.4.1.1 customers who may
//*  have user exits that contain embedded DB2 SQL statements.
//*
//*  The DBRMs created by precompiling your applications or
//*  user exits must be bound into the plan that also contains
//*  the IBM DBRMs for the Call Level Interface. It is assumed that
//*  the Call Level Interface is installed and the package binds
//*  for it have already been performed. See JCL member BINDCLI for
//*  more information on binding the CLI DBRMs.
//*
//*********************************************************************
//JOBLIB   DD  DISP=SHR,DSN=<db2hlq>.SDSNLOAD
//*
//BINDCLI  EXEC PGM=IKJEFT01,DYNAMNBR=20
//DBRMLIB  DD DISP=SHR,DSN=<db2hlq>.SDSNDBRM
//SYSTSPRT DD  SYSOUT=*
//SYSPRINT DD  SYSOUT=*
//SYSUDUMP DD  SYSOUT=*
//SYSTSIN  DD  *
 DSN SYSTEM(<db2-ssid>)

 BIND PACKAGE(USEREXIT)                  -
      MEMBER(UETEST1)                    -
```

```
      ACTION(REPLACE)                    -
      ISOLATION(CS)                      -
      DYNAMICRULES(RUN)                  -
      LIBRARY('<your-dbrm-library-here>')

 BIND PLAN(<your-plan-name>)          -
  PKLIST(DSNAOCLI.DSNCLICS            -
         DSNAOCLI.DSNCLINC            -
         DSNAOCLI.DSNCLIRR            -
         DSNAOCLI.DSNCLIRS            -
         DSNAOCLI.DSNCLIUR            -
         DSNAOCLI.DSNCLIC1            -
         DSNAOCLI.DSNCLIC2            -
         DSNAOCLI.DSNCLIF4            -
         DSNAOCLI.DSNCLIMS            -
         DSNAOCLI.DSNCLIQR            -
         USEREXIT.UETEST1             )   -
       OWNER(<plan-owner>)                 -
       QUALIFIER(<table-qualifier>)        -
       RETAIN                              -
       ISOLATION(CS) ACTION(REPLACE)

 END
/*
//
```

The NNUSER DLL can be compiled and linked using the CBCDB2CL cataloged procedure. This cataloged procedure contains symbolic parameters. You may have to modify the default symbolic parameters for your site. See *Tailoring Jobs for Your Site* on page 17.

```
//*********************************************************************
//*                                                                   *
//*  NEONet COMPILE PRELINK AND LINK A C++ PROGRAM WITH EMBEDDED SQL  *
//*  VERSION 4.03 - SOURCE CONTROL VERSION                            *
//*                                                                   *
//*********************************************************************
//*
//CBCDB2CL PROC INFILE=,             INPUT ... REQUIRED
//   OUTFILE=,                       OUTPUT .... REQUIRED
//   MEMBER=,                        SOURCE MEMBER NAME...REQUIRED
//   INCFILE=,                       SOURCE .H LIBRARY ...REQUIRED
//   DBRMLIB=,                       DB2 DBRM LIBRARY  ...REQUIRED
```

```
//   CREGSIZ='60M',                  COMPILER REGION SIZE
//   LIBPRFX='<ceehlq>',             PRFX LE REQUIRED FOR COMPILER 1.8
//   LEPRFX='<lehlq>',               PRFX LE 1.5 LIB(S)
//   SMPPRFX='<smphlq>',             PRFX FOR NEON LIB(S)
//   TCPIP='<tcphlq>',               PRFX FOR TCP/IP LIB(S)
//   CLBPRFX='<c++hlq>',             PRFX FOR C++ LIB(S)
//   MQSHLQ='<mqshlq>',              MQ SERIES HLQ FOR LOADLIB
//   DB2HLQ='<db2hlq>',              DB2 HLQ FOR LOADLIB
//   CLANG='EDCMSGE', NOT USED IN THIS RELEASE. KEPT FOR COMPATIBILITY
// CXXLANG='CBCMSGE', NOT USED IN THIS RELEASE. KEPT FOR COMPATIBILITY
//   PLANG='EDCPMSGE',               PRE-LINKER MESSAGE NAME
//   PREGSIZ='2048K',                PRE-LINKER REGION SIZE
//   LPARM='AMODE=31,MAP,RENT',      LINKAGE EDITOR OPTIONS
//   TUNIT='SYSALLDA',               UNIT FOR TEMPORARY FILES
//   OPCLAS='*'                      SYSOUT OUTPUT CLASS
//*
//*        DB2 PRECOMPILE THE C/C++ PROGRAM
//*
//PC      EXEC PGM=DSNHPC,REGION=4096K,
//   PARM=(TW,STDSQL(NO),HOST(CPP),MAR(1,80),NOOPTN,FLAG(I))
//STEPLIB  DD DISP=SHR,DSN=&DB2HLQ..SDSNEXIT
//         DD DISP=SHR,DSN=&DB2HLQ..SDSNLOAD
//SYSLIB   DD DISP=SHR,DSN=&INCFILE,DCB=(RECFM=VB,LRECL=255)
//SYSIN    DD DISP=SHR,DSN=&INFILE(&MEMBER)
//DBRMLIB  DD DISP=SHR,DSN=&DBRMLIB(&MEMBER)
//SYSCIN   DD DSN=&&DSNHOUT,DISP=(NEW,PASS),UNIT=SYSDA,
//             SPACE=(800,(500,500)),
//             DCB=(RECFM=VB,LRECL=255,BLKSIZE=2550)
//SYSPRINT DD SYSOUT=*
//SYSTERM  DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSUT1   DD SPACE=(800,(500,500),,,ROUND),UNIT=SYSDA
//SYSUT2   DD SPACE=(800,(500,500),,,ROUND),UNIT=SYSDA
//SYSUT3   DD SPACE=(800,(500,500),,,ROUND),UNIT=SYSDA
//SYSUT4   DD SPACE=(800,(500,500),,,ROUND),UNIT=SYSDA
//*
//*  COMPILE STEP:
//*
//COMPILE EXEC PGM=CBCDRVR,REGION=&CREGSIZ,
//  PARM=('CXX OPTFILE(DD:OPTION)')
//STEPLIB  DD  DSN=&LIBPRFX..SCEERUN,DISP=SHR
//         DD  DSN=&CLBPRFX..SCBCCMP,DISP=SHR
```

```
//OPTION   DD  DSN=&SMPPRFX..SNEOJCL(OPTNOOE),DISP=SHR
//SYSMSGS  DD  DUMMY,DSN=&CLBPRFX..SCBC3MSG(&CLANG),DISP=SHR
//SYSXMSGS DD  DUMMY,DSN=&CLBPRFX..SCBC3MSG(&CXXLANG),DISP=SHR
//SYSIN    DD  DSN=&&DSNHOUT,DISP=(OLD,PASS)
//USERLIB  DD  DSN=&INCFILE,DISP=SHR,DCB=(RECFM=VB,LRECL=255)
//SYSLIB   DD  DSN=&LEPRFX..SCEEH.H,DISP=SHR
//         DD  DSN=&LEPRFX..SCEEH.SYS.H,DISP=SHR
//         DD  DSN=&LEPRFX..SCEEH.ARPA.H,DISP=SHR
//         DD  DSN=&LEPRFX..SCEEH.NET.H,DISP=SHR
//         DD  DSN=&LEPRFX..SCEEH.NETINET.H,DISP=SHR
//         DD  DSN=&DB2HLQ..SDSNC.H,DISP=SHR
//         DD  DSN=&CLBPRFX..SCLBH.H,DISP=SHR
//         DD  DSN=&TCPIP..SEZACMAC,DISP=SHR
//         DD  DSN=&TCPIP..SEZAINST,DISP=SHR
//SYSLIN   DD  DSN=&&LOADSET,UNIT=&TUNIT.,
//             DISP=(MOD,PASS),SPACE=(512,(500,200)),
//             DCB=(RECFM=FB,LRECL=80,BLKSIZE=3200)
//SYSPRINT DD  SYSOUT=&OPCLAS
//SYSOUT   DD  SYSOUT=&OPCLAS
//SYSCPRT  DD  SYSOUT=&OPCLAS
//SYSUT1   DD  UNIT=&TUNIT.,SPACE=(32000,(30,30)),
//             DCB=(RECFM=FB,LRECL=80,BLKSIZE=3200)
//SYSUT4   DD  UNIT=&TUNIT.,SPACE=(32000,(30,30)),
//             DCB=(RECFM=FB,LRECL=80,BLKSIZE=3200)
//SYSUT5   DD  UNIT=&TUNIT.,SPACE=(32000,(30,30)),
//             DCB=(RECFM=FB,LRECL=3200,BLKSIZE=12800)
//SYSUT6   DD  UNIT=&TUNIT.,SPACE=(32000,(30,30)),
//             DCB=(RECFM=FB,LRECL=3200,BLKSIZE=12800)
//SYSUT7   DD  UNIT=&TUNIT.,SPACE=(32000,(30,30)),
//             DCB=(RECFM=FB,LRECL=3200,BLKSIZE=12800)
//SYSUT8   DD  UNIT=&TUNIT.,SPACE=(32000,(30,30)),
//             DCB=(RECFM=FB,LRECL=3200,BLKSIZE=12800)
//SYSUT9   DD  UNIT=&TUNIT.,SPACE=(32000,(30,30)),
//             DCB=(RECFM=VB,LRECL=137,BLKSIZE=882)
//SYSUT10  DD  SYSOUT=&OPCLAS
//SYSUT14  DD  UNIT=&TUNIT.,SPACE=(32000,(30,30)),
//             DCB=(RECFM=FB,LRECL=3200,BLKSIZE=12800)
//SYSUT15  DD  SYSOUT=&OPCLAS
//*
//*
//* PRE-LINKEDIT STEP:
//*
```

```
//PLKED   EXEC PGM=EDCPRLK,REGION=&PREGSIZ,COND=(8,LE,COMPILE),
//       PARM=(DLLNAME(&MEMBER),NOER)
//STEPLIB  DD  DSN=&LIBPRFX..SCEERUN,DISP=SHR
//SYSMSGS  DD  DSN=&LEPRFX..SCEEMSGP(&PLANG),DISP=SHR
//SYSLIB   DD  DSN=&LEPRFX..SCEECPP,DISP=SHR
//SYSIN    DD  DSN=*.COMPILE.SYSLIN,DISP=(MOD,PASS)
//         DD  DSN=&CLBPRFX..SCLBSID(IOSTREAM),DISP=SHR
//         DD  DSN=&CLBPRFX..SCLBSID(COMPLEX),DISP=SHR
//         DD  DSN=&CLBPRFX..SCLBSID(ASCCOLL),DISP=SHR
//         DD  DSN=&DB2HLQ..SDSNMACS(DSNAOCLI),DISP=SHR
//         DD  DSN=&SMPPRFX..SNEOEXP(FMGR),DISP=SHR
//         DD  DSN=&SMPPRFX..SNEOEXP(NNMQS),DISP=SHR
//         DD  DSN=&SMPPRFX..SNEOEXP(NNSQLOBJ),DISP=SHR
//         DD  DSN=&SMPPRFX..SNEOEXP(NRULEFMT),DISP=SHR
//         DD  DSN=&SMPPRFX..SNEOEXP(RMGR),DISP=SHR
//         DD  DSN=&SMPPRFX..SNEOEXP(SQLOBJ),DISP=SHR
//         DD  DSN=&SMPPRFX..SNEOUEXP(GETVAL),DISP=SHR
//         DD  DSN=&SMPPRFX..SNEOUEXP(OPENFORM),DISP=SHR
//         DD  DSN=&SMPPRFX..SNEOUEXP(RULES),DISP=SHR
//         DD  DDNAME=SYSIN2
//SYSMOD   DD  DSN=&&PLKSET,UNIT=&TUNIT.,DISP=(NEW,PASS),
//             SPACE=(32000,(30,30)),
//             DCB=(RECFM=FB,LRECL=80,BLKSIZE=3200)
//SYSDEFSD DD  DUMMY
//SYSOUT   DD  SYSOUT=&OPCLAS
//SYSPRINT DD  SYSOUT=&OPCLAS
//SYSIN2   DD  DUMMY
//*
//*
//* LINKEDIT STEP:
//*
//LKED   EXEC
PGM=HEWL,REGION=1024K,COND=((8,LT,COMPILE),(8,LE,PLKED)),
//       PARM='&LPARM'
//SYSLIB   DD  DSN=&LEPRFX..SCEELKED,DISP=SHR
//SYSOBJ   DD  DSN=&&OBJLIB,UNIT=&TUNIT.,DISP=(NEW,PASS),
//             SPACE=(32000,(30,30)),
//             DCB=(RECFM=FB,LRECL=80,BLKSIZE=3200)
//SYSLIN   DD  DSN=*.PLKED.SYSMOD,DISP=(OLD,DELETE)
//         DD  DDNAME=SYSIN
//SYSLMOD  DD  DSN=&OUTFILE(&MEMBER),DISP=SHR
//MQSLOAD  DD  DSN=&MQSHLQ..SCSQLOAD,DISP=SHR
```

```
//DB2LIB   DD  DSN=&DB2HLQ..SDSNLOAD,DISP=SHR
//SYSUT1   DD  UNIT=&TUNIT.,SPACE=(32000,(30,30))
//SYSUT2   DD  UNIT=&TUNIT.,SPACE=(32000,(30,30))
//SYSUT3   DD  UNIT=&TUNIT.,SPACE=(32000,(30,30))
//SYSUT4   DD  UNIT=&TUNIT.,SPACE=(32000,(30,30))
//SYSPRINT DD  SYSOUT=&OPCLAS
//SYSIN    DD  DUMMY
```

NNUESTUB

When a user exit is encountered as part of the reformatting process, NEONFormatter tries to resolve the exit name to a callable function address. Since the function name and address are developed outside the scope of NEONFormatter, the user application is called to provide the function address.

NNUESTUB is a stub lookup function that is replaceable by a user application. The following C++ user exit code must be modified to implement a user exit in C++:

```
static const char _uetest_cpp_id_[] = "$Header: /source/aig/formatter/
test/usere
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <iostream.h>

#include "nnexit.h"
#include "neodefs.h"
#include "dbtypes.h"
#include "interface.h"
#include "sqlapi.h"
#include "vqueue.h"
#include "qapi.h"
#include "formatter.h"

int rules(void);
int openformatdb(void);

NNExitRet
CountFields(const DbmsSession &rSession, const NNParsedFields
&rFields){
        NNExitRet oER;
        const char* pFldVal;
```

```
        int     counter = 0;
        char    counterString[32];

         pFldVal = rFields.GetFieldAscii("UvtFLDF2",0);
         while (pFldVal) {
                 counter++;
                 pFldVal = rFields.GetFieldAscii("UvtFLDF2",counter);
        }
         sprintf(counterString, "%d", counter);
         oER.SetByteArrayValue(counterString, strlen(counterString));
        return oER;
}

NNExitRet
InputNoMsgField(const DbmsSession &rSession,
                const NNParsedFields&rFields) {
         NNExitRet oER;
         const char* pFldVal = rFields.GetFieldAscii("JBLInFld!",0};
         if (!pFldVal) {
                 oER.SetError(NN_ERSTATUS_ERROR,"JBLInFld1 not
                 found!");
         }
        else {
               oER.SetByteArrayValue(pFldVal, strlen(pFldVal));
        }
        return oER;
}

NNExitRet
InputMsgFieldTest(const DbmsSession &rSession,
                  const NNParsedFields &rFields) {
         NNExitRet oER;
         char newtext[255];
        char *addtext = "Added by ue function";
        const char* pFldVal = rFields.GetFieldAscii("F4",0);

               if (!pFldVal) {
                         oER.SetError(NN_ERSTATUS_ERROR,"F4 not
found!");
               }
              else {
for (int i=0;*pFldVal !='\0';i++,pFldVal++)
```

```
newtext[i]=*pFldVal;

for (;*addtext!='\0';i++,addtext++)
newtext[i]=*addtext;

newtext[i++]='\0';

oER.SetByteArrayValue(newtext, strlen(newtext));
              }
              return oER;
}

NNExitRet
DataBaseCallFunc(const DbmsSession &rSession,
                 const NNParsedFields &rFields) {
        if (openformatdb()){
        return(NNExitRet((long)500,NN_ERSTATUS_OK));
        }
        else {
             cerr << "Could connect with the format database"<< end1;
        }
        NNExitRet oER;
        return oER;
}

NNExitRet
NoTypeTest(const DbmsSession &rSession, const
           const NNParsedFields &rFields) {
       NNExitRet oER;
       return oER;
}

NNExitRet
LongTest1(const DbmsSession &rSession,
          const NNParsedFields &rFields) {
      long l = 57;
      return(NNExitRet(l,NN_ERSTATUS_OK));
}

NNExitRet
CurrentFieldTest(const DbmsSession &rSession,
                 const NNParsedFields &rFields) {
```

```
        char buffer[256];
        int i;
        char *pBuffer;

        const char * pInFieldName = rFields.GetCurrInFldName();
        const char * pOutFieldName = rFields.GetCurrOutFldName();
        const char * pAsciiData = rFields.GetCurrInFldAsciiData();
        const char * pRawData = rFields.GetCurrInFldData();
        const int rawDataLength = rFields.GetCurrInFldLength();
        const int inFieldDataType = rFields.GetCurrInFldType();

               cerr << "Input field name = " << "'" << pInFieldName
                    << "'" << endl;
               cerr << "Output field name = " << "'" << pOutFieldName
                    << "'" << endl;
               cerr << "Input field ascii data = " << "'"
                    << pAsciiData << "'" << endl;
               pBuffer = buffer;
               for (i = 0; i < rawDataLength; i++, pRawData++) {
                      pBuffer += sprintf(pBuffer, "%c", *pRawData);
               }
              cerr << "Raw data = " << "'" << buffer << "'" << endl;
               cerr << "Input field data type = " << "'"
                    << inFieldDataType << "'" << endl;

               // kbae:        4.16.1997
               // Testing rFields.GetUserExitRoutineName...
               const char* pUserExitRoutineName =
   rFields.GetUserExitRoutineName();
               cerr << "User exit routine name = '" <<
   pUserExitRoutineName << "'" << endl;
              NNExitRet oER;
               oER.SetByteArrayValue("User exit called",
                                     strlen("User exit called"));
               return oER;
}

NNExitRet
RulesTest(const DbmsSession &rSession, const NNParsedFields &rFields)
{
       if (rules()){
             return(NNExitRet((long)100,NN_ERSTATUS_OK));
```

```
        }
        else {
              cerr << "Could connect with the rules engine"<< end1;
        }
        NNExitRet oER;
        return oER;
}

NNExitRet
LongTest2(const DbmsSession &rSession, const NNParsedFields &rFields)
{
        NNExitRet oER;
        oER = (long)1000;
        return oER;
}

long
Long2Cleanup(void) {
        // quick and dirty stub for sample cleanup routine
        return NN_ERSTATUS_OK;
}

NNExitRet
BadCleanup(const DbmsSession &rSession, const NNParsedFields &rFields)
{
        NNExitRet oER;
        oER = (long)130;
        return oER;
}

long
CleanBadCleanup(void) { // <out> error, no cleaning done {
        return -1;
}


NNExitRet
DoubleTest1(const DbmsSession &rSession,
             const NNParsedFields &rFields) {
        double d = 8.87;
        return(NNExitRet(d,NN_ERSTATUS_OK));
}
```

```
NNExitRet
DoubleTest2(const DbmsSession &rSession,
            const NNParsedFields &rFields) {
       NNExitRet oER;
       oER = 3.2715;
       return oER;
}

NNExitRet
ByteArrayTest1(const DbmsSession &rSession,
               const NNParsedFields &rFields) {
       char* acTestStr = "byte array test";
       return(NNExitRet(acTestStr, strlen(acTestStr),
NN_ERSTATUS_OK));
}

NNExitRet
ByteArrayTest2(const DbmsSession &rSession, const
               const NNParsedFields &rFields) {
        NNExitRet oER;
        char* acTestStr = "byte array test2";
        oER.SetByteArrayValue(acTestStr, strlen(acTestStr));
        return oER;
}

extern "C" void
NNGetUserExitFuncPtrs(
        char* acFuncName,
        NN_EXIT_FUNC_t&rUEptr,
        NN_EXIT_CLEANUP_FUNC_t&rUEClUpPtr) {

    if(strcmp(acFuncName, "NoTypeTest") == 0) {
                rUEptr = NoTypeTest;
                rUEClUpPtr = NULL;
   }
    else if(strcmp(acFuncName, "CountFields") == 0) {
                rUEptr = CountFields;
                rUEClUpPtr = NULL;
   }
    else if(strcmp(acFuncName, "LongTest1") == 0) {
                rUEptr = LongTest1;
```

```
                rUEClUpPtr = NULL;
    }
    else if(strcmp(acFuncName, "LongTest2") == 0) {
                rUEptr = LongTest2;
                rUEClUpPtr = Long2Cleanup;
    }
    else if(strcmp(acFuncName, "DoubleTest1") == 0) {
                rUEptr = DoubleTest1;
                rUEClUpPtr = NULL;
    }
    else if(strcmp(acFuncName, "DoubleTest2") == 0) {
                rUEptr = DoubleTest2;
                rUEClUpPtr = NULL;
    }
    else if(strcmp(acFuncName, "ByteArrayTest1") == 0) {
                rUEptr = ByteArrayTest1;
                rUEClUpPtr = NULL;
    }
    else if(strcmp(acFuncName, "ByteArrayTest2") == 0) {
                rUEptr = ByteArrayTest2;
                rUEClUpPtr = NULL;
    }
    else if(strcmp(acFuncName, "BadCleanup") == 0) {
                rUEptr = BadCleanup;
                rUEClUpPtr = CleanBadCleanup;
    }
    else if(strcmp(acFuncName, "InputMsgFieldTest") == 0) {
                rUEptr = InputMsgFieldTest;
                rUEClUpPtr = NULL;
    }
    else if(strcmp(acFuncName, "DataBaseCallFunc") == 0) {
                rUEptr = DataBaseCallFunc;
                rUEClUpPtr = NULL;
    }
    else if(strcmp(acFuncName, "RulesTest") == 0) {
                rUEptr = RulesTest;
                rUEClUpPtr = NULL;
    }
    else if(strcmp(acFuncName, "CurrentFieldTest") == 0) {
                rUEptr = CurrentFieldTest;
                rUEClUpPtr = NULL;
    }
```

```
    else if(strcmp(acFuncName, "InputNoMsgField") == 0) {
                rUEptr = InputNoMsgField;

rUEClUpPtr = NULL;
   }
                  else {
                         rUEptr = NULL;
                         rUEClUpPtr = NULL;
}
}
```

Chapter 5
Rules

NEONRules has two main functions: evaluating messages against a set of rules and reacting to the evaluation results.

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

NEONRules uses rules definitions that 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 actions to perform if the rule hits. Rules definition data resides in a relational database. Users build and modify rule definitions using one of two methods: the NEONRules graphical user interface (GUI) or NEONRules Management API functions.

Rules Components

The NEONRules GUI allows the user to populate screens with rule definition data and store the information in a relational database.

NEONRules 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 Rules daemon reads messages off a queue, evaluates the messages, and based on the results, performs the required reformatting and routing.

The following test modules are delivered with NEONRules:

- MQIPUTDA places a message on a queue with the required queue options for the MQSeries Integrator Rules daemon.
- MQIGETDA 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.
- RULETEST reads a message from a file and evaluates the message.
- RULOWNER allows the administrator to determine which rules are owned by a specific user and to change rule ownership.

The Rules Consistency Checker utility checks the correctness of the rule definition data in the relational database. As rule definition data is built and maintained, users should run the consistency checker periodically to insure data integrity.

The Permissions Consistency Checker verifies rule ownership, Read, Update, and PUBLIC permissions.

The NEOMQCC Consistency Checker verifies that queue names referenced in subscriptions exist.

The NNRIE tool allows the user to export rule definitions from a database to a file, and to import the exported file into a database. NNRIE can import from a MQIntegrator r. 3.2 export file into a MQSeries Integrator 1.1 database.

Application Groups

Application groups are logical divisions of rule sets for different business needs. You can define as many application groups as you need. For example, you might want rules for the accounting department and the application development department separated into two groups. You can 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 strings of data. Each application group can contain several message types, and a message type can be used with more than one application group. Message types are defined by the user. When

using Formatter, a message type is the same as an input format name. This format name is used by Formatter to parse input messages for rules evaluation.

Rules

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

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

Expressions, Arguments, Boolean Operators, and Rules Operators

An expression, or evaluation criteria, consists of a list of fields, associated operators, and associated comparison data connected with Boolean operators. An argument consists of the combination of a field name, Rules comparison operator, and static value or other field name. Field names depend on the message type, or input format name. The input format name is defined using NEONFormatter. Rules comparison operators are already defined within NEONRules. 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 APIs***.

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, NEONRules lets you retrieve associated subscriptions to be taken by the application. These subscriptions are the actions or commands, and the associated parameters or options to execute them.

Subscriptions are lists of actions to take when a message evaluates to true. Each rule must have at least one associated subscription. Subscriptions are uniquely identified within an application group / message type pair by a user-

defined subscription name. Each action within a subscription is defined by action name and need not be unique, since all actions are intended to be executed in sequence. A single subscription can be shared by multiple rules when the same subscription is associated with each of the rules. The shared subscription is retrieved only once no matter how many of its associated rules hit.

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

Rules and Subscription Permissions

Rule and subscription permissions restrict user access to individual rules, subscriptions, or their components in the NEONRules database. A 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, or 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 actions and options. Permissions only apply to managing rule and subscription contents, not rule evaluation. Permissions must be defined for subscriptions in the same way they are for rules.

The rule 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. Owners can change their own permissions 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. All other users are grouped into a public user group named PUBLIC and given Read permissions by default.

APIs

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

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

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

Rules Permissions

Permissions for Rules should be managed through the NEONRules GUI or through the NEONRules Management APIs. The RULOWNER utility allows the administrator to determine which rules are owned by a specific user and to change ownership of rules. To use the RULOWNER utility, you must edit the SQLSVSES file to include "rules" as the session-name parameter so the utility can connect to the Rules database. See *Editing the SQLSVSES File* on page 19.

Rule Ownership

The ownership of any rules owned by a specific user can be changed to another user. When rule ownership is changed, the permissions for the rules are transferred to the new owner and previous permissions are overwritten. The rule permissions are transferred when ownership is transferred from the previous owner to the new owner. The new owner is given the same permissions as the previous owner.

RULOWNER

The following sample job control language (JCL) illustrates how to run the RULOWNER job in batch and pass startup parameters to it.The JCL at your site will be different. See *Tailoring Jobs for Your Site* on page 17 for information about the symbolic parameters in this sample.

```
//*  <insert a valid jobcard here >
//*
```

```
//*******************************************************************
//*                                                                 *
//* Licensed Materials - Property of New Era of Networks, Inc.      *
//* Copyright (c) 1998-1999, New Era of Networks, Inc.              *
//* All Rights Reserved.                                            *
//*                                                                 *
//* Release 4.1.1                                                   *
//*******************************************************************
//*******************************************************************
//*                                                                 *
//*RULOWNER: Changes ownership of rules                             *
//*                                                                 *
//*******************************************************************
//RULOWNER PROC PRM=('-d DD:DBLOG'), run-time PARMs
//         SMPHLQ='<smphql>',     HLQ for MQSI distrib libs
//         MQSHLQ='<mqshql>',     HLQ for MQS run-time libs
//         CEEHLQ='<lehlq>',      HLQ for Lang Envir libs
//         CSSHLQ='SYS1',         HLQ for Callable Sys Svcs (CSS-) Lib
//         SQLMEM='SQLSVSES',     MEMbername for SQLSVSES cntl cards
//         INIMEM='CLIINI',       MEMbername for CLI INI  cntl cards
//         OPCLAS='*'             SYSOUT CLASS
//*
//STP0101  EXEC PGM=RULOWNER,
//   PARM=&PRM
//*
//*   <tailor the member STEPLIB and copy it here>
//*
//SQLSVSES DD  DSN=&SMPHLQ..SNEOCNTL(&SQLMEM),DISP=SHR
//DSNAOINI DD  DSN=&SMPHLQ..SNEOCNTL(&INIMEM),DISP=SHR
//DBLOG    DD  SYSOUT=&OPCLAS
//SYSOUT   DD  SYSOUT=&OPCLAS
//SYSPRINT DD  SYSOUT=&OPCLAS
//STATLOG  DD  SYSOUT=&OPCLAS
//CLITRACE DD  SYSOUT=&OPCLAS  used for DB2 CLI high-level tracing
//         PEND
//*
//*
//* All datasets used by MQSeries Integrator must be preallocated and
//* cataloged prior to running any MQSeries Integrator jobs. The
//* recommended DCB attributes are:
//*    DSORG=PS,RECFM=VB,LRECL=32756,BLKSIZE=32760
//*
```

```
//RULOWNER EXEC RULOWNER,PRM=('-d DD:DBLOG')
//DBLOG    DD  SYSOUT=*
//SYSIN    DD  *
<function: 1=list; 2=reassign>>
<if 1 or 2: current-rule-owner>
<if 2: new-rule-owner>
99                                  == required end-of-file indicator
/*
//
```

Verifying and Changing Rule Ownership

The RULOWNER utility writes prompts to the standard output stream (STDOUT) SYSPRINT. The responses are read from the standard input stream (STDIN) SYSIN:

```
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
99 Quit
```

To list rules owned by a certain owner, enter 1 as the first line in SYSIN, and then enter the id of the owner of the rules on the second line of SYSIN.

To change rule ownership, enter 2 in the first line of SYSIN. The second prompt written to SYSPRINT is:

```
User Name for Current Owner of Rules
```

Enter the current owner's name as the second line of SYSIN. An additional prompt is written to SYSPRINT requesting the new id assigned to the rules:

```
User Name for New Owner of Rules
```

Enter the id for the new owner of the rules as the third line of SYSIN.

If you select 1 at the prompt, then RULOWNER lists the application group name, message type name, and rule name of all rules owned by the specified user. If you select 2 at the prompt, then RULOWNER does not display this rule information.

To list the subscriptions owned by a certain user, type 3 at the prompt.

```
User Name for Owner of Subscriptions
```

The Application Group, Message Type, and Subscription Name for all the subscriptions owned by the specified user are listed.

To change subscription ownership, type 4 at the prompt.

```
User Name of Current Owner of Subscription
User Name for New Owner of Subscription
```

The owner of the subscription is changed.

Note:
The last line for SYSIN must be gg to end the program.

Examples

The following examples demonstrate how to use RULOWNER. The JCL at your site can be different. See *Tailoring Jobs for Your Site* on page 17 for information about the symbolic parameters in this sample.

Listing all rules owned by R40NEON

```
//SSYIN    DD *
1
R40NEON
99
/*
```

Listing all rules owned by R40TEST (not a valid user)

```
//SYSIN     DD *
1
R40TEST
99
/*

Expected result (in SYSOUT):
Error No: -5509
Error Msg: Unable to find user in database
```

Listing all rules owned by R40USR2 (no rules owned by user)

```
//SYSIN     DD *
1
R40USR2
99
/*

Expected result (in SYSOUT):
Error No: -5514
Error Msg: Unable to read permission
```

Changing all rules owned by R40NEON to R40USR2

```
//SYSIN     DD *
2
R40NEON
R40USR2
99
/*
```

Error Conditions

The error codes for other errors related to reading rules are listed in the ***Programming Reference for NEONRules***.

No Rules for Owner:

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

Error No: -5514
Error Msg: Unable to read permission
```

Invalid User:

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

The Rules Daemon

The 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 Rules daemon 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, explained in the ***User's Guide***, or by using the Management APIs. Application programmers can use the NEONRules APIs to interface database calls to execute rules. These functions are explained in the ***Programming Reference for NEONRules***.

Configuring MQSeries Integrator

To successfully execute the Rules daemon, a complete and valid installation of MQseries and MQSeries Integrator must exist prior to using Rules. All queues, rules, and formats must be created and saved before using the Rules daemon.

Queues

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

To have a message successfully evaluated by the MQSeries Integrator Rules daemon process, the input message must have the OPT_APP_GRP and OPT_MSG_TYPE options set. These options overrides the default option settings.

If these options are not set, the MQSeries Integrator Rules daemon assigns defaults. The defaults come from the RULENG MPF file. The MPF file is a parameter file that contains required and optional parameters. See *RULENGP* on page 147.

OPT_APP_GRP assigns the message to an application group and must match the application group name in the NEONRules GUI. The OPT_MSG_TYPE option must match the message type in rule definitions and the input format name in the format definitions. These two message options allow the MQSeries Integrator Rules daemon to evaluate the message against its rules

and only its rules. If the options are not set, the evaluation cannot occur, and failure processing will occur.

Rules

Unless the OPT_RELOAD_RULE_SET=TRUE option is used, the MQSeries Integrator Rules daemon is not dynamic with respect to rule definition and subscription definition. Only rules defined prior to starting up the MQSeries Integrator Rules daemon are used. Any rules added or modified after the Rules daemons starts up are not used until the OPT_RELOAD_RULE_SET=TRUE option is processed.

Formats

All formats associated with any message put to an input queue must be entered and saved prior to putting that message to the input queue. All formats used during a Reformat action must be entered and saved prior to starting the Rules daemon.

For information about entering rules and formats, refer to the ***User's Guide*** and the ***Programming Reference*** documents.

MQSeries Integrator Rules Daemon Processing

The MQSeries Integrator Rules daemon is built on top of the NEONRules APIs and performs these 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. NEONFormatter is called to parse the input message into fields. NEONRules then evaluates these fields. If a message is successfully evaluated, then subscriptions will be executed.

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

Subscription Execution

After a message is successfully evaluated against its rules, all subscriptions associated with those rules that evaluate to true are executed. If a message is successfully evaluated and no subscriptions are executed (no rules evaluate to true), 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 the output queue, 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 process on hold so that 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.

Note:
When you run the Rules daemon, no other process should drain the input queues. This will destroy guaranteed delivery and guaranteed sequence, and may cause long waits and possible deadlocks.

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 process is unable to put to the failure queue, the process rolls back, and the Rules daemon stays on that message until it can write to the failure queue.

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. The Rules daemon process 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. 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 process to wait, the OPT_NO_WAIT option must be set to FALSE.

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 Rules daemon are Reformat and Put Queue. Other actions defined require the user to write a custom Rules daemon program to process those actions. The Rules daemon does not execute generic actions.

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 an input format and an output format as options. Formatter APIs are called to perform the reformat of messages.

Put Queue

The Put Queue action takes a message, puts it on a specified destination queue, and sets the message type option as the message format type specified. The Put Queue action requires a destination queue name and a message format type as options. Both the queue name and message format type must exist in the database. MQSeries Queuing APIs are called to perform the Put Queue operation. The Put Queue action does not perform formatting.

If you use NEONRules Management APIs to add the Put Queue action, the action name is putqueue, and the option names are OPT_TARGET_QUEUE and OPT_MSG_TYPE.

WARNING!
If a subscription does not include the Put Queue action, messages are not put to a queue and can be lost. Run the Rules Consistency Checker to determine which subscriptions do not have a Put Queue action.

While the Reformat and Put Queue subscription options are the only actions that can be performed by the Rules daemon, the NEONRules APIs allow any number of actions and associated options. An application programmer can use the APIs and independently generated code to execute other types of

actions.The size of your database and performance requirements are the only limitations on the NEONRules APIs.

Failure Processing

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

The Rules daemon can be configured to set OPT_ERR_CODE and OPT_ERR_MSG queue options each time a message is put to the failure queue. Both options are intended to help users determine why the Rules daemon sent the message to the failure queue.

The OPT_ERR_CODE option value indicates which subsystem, Formatter or Rules, encountered the failure and provides the error code number. For a complete listing of the MQSeries Integrator error codes, see Error Messages in the ***Programming References***. The NNF prefix indicates a Formatter failure and NNR indicates a Rules failure.

The OPT_ERR_MSG queue option value provides the error message that corresponds to the error code.

Rules Daemon Error Messages

Code	Error Name	Explanation	Response
-10000	RULENG_INVALID_ PUT_QUEUE_ACTION _ERR	Putqueue action contains invalid or missing OPT_TARGET_QUEUE option name or value.	Correct the options in the Put Queue action.
-9999	QUEUE_CREATION_ FAILURE_ERR	Failure opening or connecting to queue specified in the OPT_TARGET_QUEUE option.	Correct the options in the Put Queue action.

Code	Error Name	Explanation	Response
-9998	QUEUE_ INITIALIZATION_ FAILURE_ERR	Failure to initialize queue specified in the OPT_TARGET_QUEUE option.	Verify that the specified queue exists.
-9997	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 daemon 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 from a Put Queue action.

- If no subscription actions are successfully executed, then 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 written to the log file only if logging is specified.
- The Rules daemon process waits if the output queue or No Hit queue is full or disabled, unless the incoming message has the OPT_NO_WAIT option set to TRUE.

Connecting to DB2 and MQSeries

To connect to DB2 and MQSeries, MQIRULEN, MQIGETDA, and MQIPUTDA require a parameter file for input. These parameters control various aspects of the product's behavior. The parameter filename DD:NAME is specified in the PARM= field. The default name is DD:MPF.

Using the MQSeries Integrator Rules Daemon

Because OS/390 limits the size of the PARM string that can be specified, the MQIRULEN executable accepts its parameters from the standard input stream (STDIN) SYSIN, while writing prompts for each parameter to the standard output stream (STDOUT) SYSPRINT.

MQIRULEN

The following sample job control language (JCL) illustrates how to run the MQIRULEN job in batch and pass startup parameters to it. The JCL at your site will be different. See *Tailoring Jobs for Your Site* on page 17 for information about the symbolic parameters in this sample.

```
//*  <insert a valid jobcard here>
//*
//*******************************************************************
//*                                                                 *
//* Licensed Materials - Property of IBM                            *
//*                                                                 *
//* (C) Copyright IBM Corp. 1998,1999                               *
//*******************************************************************
//*******************************************************************
//*                                                                 *
//* RULENG - Run the MQSI Rules Engine                              *
//*                                                                 *
//*******************************************************************
//RULENG   PROC PRM=(' '),         run-time parameters
//         SMPHLQ='<smphlq>',      HLQ for MQI distrib libs
//         MQSHLQ='<mqshlq>',      HLQ for MQS run-time libs
//         CEEHLQ='<lehlq>',       HLQ for Lang Envir libs
//         CSSHLQ='SYS1',          HLQ for Callable Sys Svcs (CSS-)Lib
//         SQLMEM='SQLSVSES',      MEMbername for SQLSVSES cntl cards
//         INIMEM='CLIINI',        MEMbername for CLI INI cntl cards
//         MPF='RULENGP',          MEMbername for MPF parameters
//         OPCLAS='*'              SYSOUT CLASS
//*
//STP0101  EXEC PGM=MQIRULEN,
//     PARM=&PRM
//*
//*   <tailor the member STEPLIB and copy it here>
```

```
//*
//SQLSVSES DD  DSN=&SMPHLQ..SNEOCNTL(&SQLMEM),DISP=SHR
//DSNAOINI DD  DSN=&SMPHLQ..SNEOCNTL(&INIMEM),DIPS=SHR
//MPF      DD  DSN=&SMPHLQ..SNEOMPF(&MPF), DISP=SHR
//SYSUDUMP DD  SYSOUT=&OPCLAS
//SYSPRINT DD  SYSOUT=&OPCLAS
//SYSOUT   DD  SYSOUT=&OPCLAS
//STATLOG  DD  SYSOUT=&OPCLAS
//CLITRACE DD  SYSOUT=&OPCLAS  used for DB2 v5 CLI high-level tracing
//         PEND
//*
//*
//* All datasets used by MQSeries Integrator must be preallocated and
//* cataloged prior to running any MQSeries Integrator jobs. The
//* recommended DCB attributes are:
//*    DSORG=PS,RECFM=VB,LRECL=32756,BLKSIZE=32760
//*
//*
//RULENG   EXEC RULENG
//     PRM=(' -p DD:MPF')
//ENGLOG   DD  SYSOUT=*            additional log file
//LOGFILE  DD  SYSOUT=*            log file specified in MPF
//
```

RULENGP

The MPF file defines parameters for rule processing. The following is a sample parameter file for the MQSeries Integrator Rules daemon:

```
##############################################################################
#
# This is the parameter file for MQSeries Integrator Rules daemon.
#
# Comments must have a # in the first column.
# Names must be separated from the equals sign by whitespace, and the
# value also must be separated with whitespace. No whitespace 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 daemon
# control

# Alternate User Authority Flag
CredentialsEnabled = 0

# MQSeries queue manager name...defaults to default queue manager
QueueManagerName = CSQ1

# number of replay/retry attempts before message is sent to failure
# queue (zero indicates no replays allowed)
MaxBackoutCount = 0

# these three queue names are mandatory!
InputQueueName   = RulesIn
NoHitQueueName   = RulesNoHit
FailureQueueName = RulesFail

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

[Logging]
# Log file control..."LogFileName" is the file specification for the
# log. Valid "LogLevel" settings are:
#      3 - log only fatal errors
#      2 - log errors and fatal errors
#      1 - log warnings, errors, and fatal errors
#      0 - log information, warnings, errors, and fatal errors
LogFileName = DD:LOGFILE
LogLevel    = 0

[Rules Database Connection]
#
# Rules and Formatter database connection information
# (mandatory)
#
ServerName         = <db2-ssid>
UserId             = XXXXX
Password           = XXXXX
```

```
DatabaseInstance   = <database-qualifier>
#
# DatabaseType is a numeric with these values:
# SYBASE CT    1
# SYBASE DB    2
# MSSQL        4
# DB2          5
# ODBC         6  includes db2 v5.1 using Call Level Interface (CLI)
# MQSERIES     7
# ORACLE 7     8
# ORACLE 8     9
#
DatabaseType = 6

#
# end of file!
#
```

Parameters

The following parameters are used in the RULENGP MPF file. The parameters are tunable, so their values can be adjusted to customize control and performance to your environment.

The parameters are organized 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 ([ ]). Because brackets may not display on OS/390, the left bracket ([) is represented as x'AD', and the right bracket (]) is represented as x'BD'.

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*)

Name	Mandatory/ Optional	Description
MaxBufferSize	Optional	Hard limit on growth of memory block = 1048576 bytes (by default*).
LoadImmediate	Optional	Value =1.

* The default values for these parameters are recommended for most environments.

Logging

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

Queues

Name	Mandatory/ Optional	Description
CredentialsEnabled	Optional	Value determines whether messages are put with alternate user authority. The default value is zero (0) or off, indicating that messages are put using daemon authority. When turned on (value is 1), messages are put using the publisher's authority.
QueueManagerName	Optional	Name of the local MQSeries Queue Manager. If not specified, the default MQSeries 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, indicating that no replay is allowed.
InputQueueName	Mandatory	Name of queue used by the MQSeries Integrator Rules daemon to process inbound or input messages. When multiple input queues are specified, use first,second,third..., where first, second, and third are the names of the queues. No whitespaces are used in the queue list.

Name	Mandatory/ Optional	Description
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.
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 1 second.
NoHitQueueName	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.
FailureQueueName	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.

Name	Mandatory/ Optional	Description
DefaultAppGroup	Mandatory	Indicates the default application group used for messages without a MQSI header. DefaultAppGroup value must be supplied by the user; no default value. Multiple default values for multiple input queues must be mapped 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
DefaultMsgType	Mandatory	Indicates the message type value used for messages without a MQSI header. DefaultMsgType value must be supplied by the user; no default value. The daemon uses the following MQSeries Message Descriptor (MQMD) values at run-time: "$MQMD.Format", "$MQMD.PutApplName", or "$MQMD.ApplIdentityData" Multiple default values for multiple input queues are mapped 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 the maximum number of handles is stored in the cache, the next attempt to insert a queue handle into the cache initiates a cache purge operation. The default value is OFF.
PurgeInterval	Optional	Integer value indicating the 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 default value = 1 second.

These Queue Handle Cache parameters are not required to run the MQSI Rules daemon. However, using these tunable parameters optimizes performance. When MaxHandles and PurgeInterval are not specified, the default is to disable the daemon 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
ServerName	Mandatory	The name of the server you want to connect to. For Oracle, this is optional. For DB2, enter the SubsystemID .
UserID	Mandatory	Your UserID; can be encrypted using the MQSIencrypt utility. UserID is not used in the OS/390 version.

Name	Mandatory/ Optional	Description
Password	Mandatory	Your password. Can be encrypted using the MQSIencrypt utility. Password is not used in the OS/390 version.
DatabaseInstance	Mandatory	The name of the database that you want to connect to. Leave as "???" for Oracle. For DB2 on OS/390, this value is the SQLID and must be a valid primary or secondary Auth-ID.
DatabaseType	Mandatory	Integer indicating the number of the database type: 1 = SYBASE with CTLIB 2 = SYBASE with DBLIB 4 = MSSQL 5 = DB2 6 = ODBC 7 = MQSERIES 8 = ORACLE 7 9 = ORACLE 8 For OS/390 on DB2 version 5.1 and later, MQSeries Integrator uses the Call Level Interface (CLI). You must specify DatabaseType 6 for the CLI.

Improving Performance

The wait time on the input queue defaults to three (3) seconds if no time is specified.

Place Control messages, such as cache-reload and shutdown on the Rule daemon primary input queue instead of on the reload queue to improve daemon performance.

MQIRULEC

The following sample JCL is used to compile and link MQIRULEN:

```
//*  <insert a valid jobcard here >
//*
```

```
//*************************************************************
//*                                                           *
//* Licensed Materials - Property of IBM                      *
//*                                                           *
//* (c) Copyright IBM Corp. 1988, 1999                        *
//*                                                           *
//*************************************************************
//PROCLIST JCLLIB ORDER=<smphlq>.SNEOPROC,SYS1.PROCLIB
//COMPILE   EXEC  CBCCL15,
//    INFILE='<smphlq>.SNEOCPP',
//    INCFILE='<smphlq>.SNEOH',
//    OUTFILE='<smphlq>.SNEOLOAD',
//    MEMBER='MQIRULEN'
//LKED.SYSIN DD *
  INCLUDE MQSLOAD(CSQBSTUB)
  NAME MQIRULEN(R)
/*
```

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, if they want to use the changes.

MQIRULEN checks the input and can be configured to check for notification messages. Notification messages are typically empty and have the following options:

- OPT_APP_GRP set to the application group.
- OPT_MSG_TYPE set to the message type.

  The application group and message type indicate which rule set to reload.
- OPT_RELOAD_RULE_SET set to TRUE indicating to the Rules daemon to reload the specified rule set.

You must decide whether to create a new queue for notification notices or use an existing queue, such as the queue used by the Rules daemon to get messages.

Sending a Reload Message

To send a reload message, you must modify the MQIPUTDA parameter file. Under Put Options, set OPT_RELOAD_RULE_SET = TRUE.

Shutting Down the Rules Daemon

To shut down an MQSeries Integrator Rules daemon process, run MQIPUTDA with OPT_SHUTDOWN = SHUTDOWN. PUTDATA is a member of the SNEOMPF library.

Testing Rules

Rules Test Programs

The MQIPUTDA, MQIGETDA, and RULETEST programs are provided for testing Rules. The NNRTRACE program is supplied to provide a debugging utility for NEONRules. These test programs are explained in this section.

The MQIPUTDA program can be used to put data to a Rules Daemon process queue in such a way that the process can evaluate the message. The MQIGETDA program can be used to get (or retrieve) messages from a Rules Daemon process output queue, or any other queue.

MQIPUTDA

The MQIPUTDA program reads a message from a file and puts the message on a queue with the OPT_APP_GRP, the OPT_MSG_TYPE, and possibly the OPT_NO_WAIT options set. The RulesIn queue is a possible input queue for the Rules daemon program and should be specified as such in the RULENG SYSIN stream. Setting OPT_NO_WAIT causes the Rules Daemon process to put messages on the Failure queue if the output queue is full or disabled.

This program sets the two options on the message that the Rules Daemon expects, specifically the application group and message type.

MQIPUTDA requires a connection to a database containing queuing data. The MQIPUTDA program expects that a queue name defined in the

command line exists, is enabled, and is defined in the input stream (SYSIN) of a running RULENG process.

Before running MQIPUTDA, you must verify that the SQLSVSES file includes the relevant information used to execute this program.

The session name in the SQLSVSES file is used by the Rules daemon to locate the appropriate line from which to retrieve connection data. The MQIPUTDA program expects to have a session name of input. The MQIGETDA program expects to have a session name of output. Using this connection data, the Rules daemon test programs are able to make a connection to the appropriate database.

Sample JCL for MQIPUTDA

The following sample job control language (JCL) illustrates how to run the MQIPUTDA job in batch and pass startup parameters to it. The JCL at your site will be different. See *Tailoring Jobs for Your Site* on page 17 for information about the symbolic parameters in this sample.

```
//*  <insert a valid jobcard here >
//*
//*********************************************************************
//*                                                                   *
//* Licensed Materials - Property of IBM                              *
//*                                                                   *
//* (C) Copyright IBM Corp. 1998,1999                                 *
//*                                                                   *
//*********************************************************************
//*********************************************************************
//*                                                                   *
//* PUTDATA: Put data on a queue                                      *
//*                                                                   *
//*********************************************************************
//PUTDATA  PROC PRM=(' '),          run-time parameters
//         SMPHLQ='<smphlq>',       HLQ for MQI distrib libs
//         MQSHLQ='<mqshlq>',       HLQ for MQS run-time libs
//         CEEHLQ='<lehlq>',        HLQ for Lang Envir libs
//         CSSHLQ='SYS1',           HLQ for Callable Sys Svcs (CSS-)Lib
//         MPF='PUTDATA',           Member Name for MPF parameters
//         OPCLAS='*'               SYSOUT CLASS
//*
//STP0101  EXEC PGM=MQIPUTDA,
```

```
//   PARM=&PRM
//*
//*   <tailor the member STEPLIB and copy it here>
//*
//MPF      DD  DSN=&SMPHLQ..SNEOMPF(&MPF),DISP=SHR
//SYSUDUMP DD  SYSOUT=&OPCLAS
//SYSOUT   DD  SYSOUT=&OPCLAS
//SYSPRINT DD  SYSOUT=&OPCLAS
//STATLOG  DD  SYSOUT=&OPCLAS
//         PEND
//*
//*
//* All datasets used by MQSeries Integrator must be preallocated and
//* cataloged prior to running any MQSeries Integrator jobs. The
//* recommended DCB attributes are:
//*    DSORG=PS,RECFM=VB,LRECL=32756,BLKSIZE=32760
//*
//PUTDATA  EXEC PUTDATA,
//  PRM=(' -p DD:MPF')
//INPUT   DD  DISP=SHR,DSN=<your-message-file>
```

MQIPUTDA Parameters

The parameters for the MQIPUTDA test program are stored in the SNEOMPF library. See the ***Application Programming Guide*** for additional information about the MQRFH structure.

```
[Put Control]
    #Name of the file that contains the message data
    #If not specified, an empty or null data file is assumed
    inputFileName                   =      DD:INPUT

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

    #Name of the queue manager that owns the queue
    queueManagerName                =      CSQ1

    #Name of the queue where the reply message will be sent.
    #Comment the following line if the replyToQ is the same as
    #queueName
    #replyToQ                        =      <your reply to q>
```

```
#Name of the queue manager that owns the replyToQ.
#Comment the following line if the replyToQmgr is the same as
#queueManagerN
#replyToQmgr                    =      CSQ1

#Name of the log file. Comment the following line if the
#error or warning information is to be logged to stderr.
#logFileName                    =      DD:LOGFILE

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

#Maximum permissable record size in case of variable length
#records.
#Record size in case of fixed length records.
maxUserDataLength              =      10000

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

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

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

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

#Record separator character. 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 to output statistics information
    #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_QDEF       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 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 the includeHeader == 1.
    dataFormat                      =      MQSTR

[Put Options]
    #This group defines the options that are attached to the message
    #before it is sent. The parameters in this group only take effect
    #if includeHeader  == 1.
    OPT_APP_GROUP                   =      mqsiAG
    OPT_MSG_TYPE                    =      mqsiIF

    #The OPT_SHUTDOWN option will cause the MQI Rules daemon to
    #terminate after committing the current unit-of-work.
#   OPT_SHUTDOWN                    =      SHUTDOWN

    #The OPT_RELOAD_RULE_SET option will cause the Rules daemon to
    #delete its cached values and read fresh rules definitions from
    #the database.
#   OPT_RELOAD_RULE_SET             =       TRUE
```

Parameters

The following parameters are used in the MQIPUTDA file. The parameters are used to define Put control, the Put message, and Put options.

Note:
The default values for MQIPUTDA should be the same as the MQMD deaults in MQSeries.

Put Control

Name	Mandatory/ Optional	Description
inputFileName	Optional	Name of file that contains the message data. Default filename is DD:INPUT.
queueName	Mandatory	Name of queue where the message is put.
queueManagerName	Optional	Name of the local MQSeries Queue Manager. If not specified, the default MQSeries queue manager is used.
replyToQ	Optional	Name of the queue where the reply message is sent.
replyToQmgr	Optional	Name of the queue manager that owns the replyToQ.
logFileName	Optional	Name of the log file. Default filename is DD:LOGFILE.
logLevel	Mandatory	Defines message logging to the file. Valid settings are: 3 = log only fatal errors 2 = log errors and fatal errors 1 = log warnings, errors, and fatal errors 0 = log informationals, warnings, errors, and fatal errors
maxUserDataLength	Mandatory	Maximum permissible record size in variable length records; must be greater than longest text between two recordSeparators.
messageCount	Mandatory	Number of times each message is put to the queue; must specify an integer value.

Name	Mandatory/ Optional	Description
transCommitSize	Mandatory	Transaction commit size. Zero (0) indicates commit all records once.
variableRecordLength	Mandatory	Indicates whether variable length records are present in the file. YES indicates variable records are present; NO indicates fixed length records are present.
segmentationAllowed	Mandatory	YES indicates segmentation is allowed; NO indicates segmentation is not allowed.
recordSeparator	Mandatory	Record separator character used with variable length records; indicates end of record. If variableRecordLength is YES, recordSeparator value must be specified.
numRecordsToRead	Mandatory	Indicates number of records to load from the file. Value is ALL if all records are loaded.
showStatistics	Mandatory	Binary value indicating whether to output statsitics information. Value of 1 indicates yes; zero (0) indicates no.

Put Message

Name	Mandatory/ Optional	Description
format	Mandatory	Value used to populate the format field of the message descriptor.
applIdentityData	Optional	Value used to populate the applIdentityData field of the message descriptor.

Name	Mandatory/ Optional	Description
putApplName	Optional	Value used to populate the putApplName field of the message descriptor.
applOriginData	Optional	Value used to populate the applOriginData field of the message descriptor.
expiry	Mandatory	Value used to populate the expiry field of the message descriptor. The value cannot be zero (0).
persistence	Mandatory	Value used to populate the persistence field of the message descriptor. Valid values are: 0 = MQPER_NOT_PERSISTENT 1 = MQPER_PERSISTENT 2 = MQPER_PERSISTENCE_AS _QDEF
messageType	Mandatory	Value used to populate the message type field of the message descriptor. Valid values are: 1 = MQMT_REQUEST 2 = MQMT_REPLY 4 = MQMT_REPORT 8 = MQMT_DATAGRAM
includeHeader	Mandatory	Specifies whether to include the RF header with the inbound message. Value of 1 indicates yes; zero (0) indicates no.
dataFormat	Optional	Specifies how to populate the MQRFH.Format field. This parameter only takes effect if includeHeader == 1.

Put Options

Name	Mandatory/ Optional	Description
OPT_APP_GROUP	Mandatory	Option attached to the message before it is sent. The application group and message type indicate which rule set to reload.
OPT_MSG_TYPE	Mandatory	Option attached to the message before it is sent
OPT_SHUTDOWN	Optional	Causes the Rules daemon to terminate after committing the current unit of work.
OPT_RELOAD _RULE_SET	Optional	Causes the Rules daemon to delete its cached values and read fresh rules from the database. Valid values are: TRUE and FALSE.

MQIPUTDC

The following JCL is used to compile MQIPUTDA:

```
//*  <insert a valid jobcard here >
//*
//
*************************************************************
//*                                                         *
//* Licensed Materials - Property of IBM                    *
//*                                                         *
//* (C) Copyright IBM Corp. 1998,1999                       *
//*                                                         *
//*************************************************************
//PROCLIST JCLLIB ORDER=(<smphlq>.SNEOPROC,SYS1.PROCLIB)
//COMPILE   EXEC  CBCCL15,
//    INFILE='<smphlq>.SNEOCPP',
//    INCFILE='<smphlq>.SNEOH',
//    OUTFILE='<smphlq>.SNEOLOAD',
//    MEMBER='MQIPUTDA'
//LKED.SYSIN DD *
```

```
INCLUDE MQSLOAD(CSQSTUB)
NAME MQIPUTDA(R)
```

MQIGETDA

The MQIGETDA process reads a queue, retrieving messages one at a time, and writing each message to the output file until the queue is empty. The MQIGETDA program expects that the queue name defined in the command line exists, it is enabled, and it has messages on it.

Before running MQIGETDA, you must verify that the SQLSVSES file includes the relevant information used to execute this program.

The session name in the SQLSVSES file is used by the Rules daemon to locate the appropriate line from which to retrieve connection data. The MQIGETDA program expects to have a session name of output. Using this connection data, the Rules daemon test programs are able to make a connection to the appropriate database.

Sample JCL for MQIGETDA

The following sample job control language (JCL) illustrates how to run the MQIGETDA job in batch and pass startup parameters to it. The JCL at your site will be different. See *Tailoring Jobs for Your Site* on page 17 for information about the symbolic parameters in this sample.

```
//*  <insert a valid jobcard here >
//*
//*********************************************************************
//*                                                                   *
//* Licensed Materials - Property of IBM                              *
//*                                                                   *
//* (C) Copyright IBM Corp. 1998,1999                                 *
//*                                                                   *
//*********************************************************************
//*********************************************************************
//*                                                                  *
//* GETDATA: Read data from a Queue                                   *
//*                                                                   *
//*********************************************************************
//GETDATA  PROC PRM=(' '),
//         SMPHLQ='<smphlq>',      HLQ for MQI distrib libs
//         MQSHLQ='<mqshlq>',      HLQ for MQS run-time libs
```

```
//         CEEHLQ='<lehlq>',       HLQ for Lang Envir libs
//         CSSHLQ='SYS1',          HLQ for Callable Sys Svcs (CSS-)Lib
//         MPF='GETDATA',          Member Name of MPF file
//         OPCLAS='*'              SYSOUT CLASS
//*
//STP0101 EXEC PGM=MQIGETDA,
//   PARM=&PRM
//*
//*   <tailor the member STEPLIB and copy it here>
//*
//MPF      DD  DSN=&SMPHLQ..SNEOMPF(&MPF), DISP=SHR
//SYSUDUMP DD  SYSOUT=&OPCLAS
//SYSPRINT DD  SYSOUT=&OPCLAS
//SYSOUT   DD  SYSOUT=&OPCLAS
//STATLOG  DD  SYSOUT=&OPCLAS
//SYSIN    DD  DUMMY
//         PEND
//*
//* All datasets used by MQSeries Integrator must be preallocated and
//* cataloged prior to running any MQSeries Integrator jobs. The
//* recommended DCB attributes are:
//*    DSORG=PS,RECFM=VB,LRECL=32756,BLKSIZE=32760
//*
//DEL     EXEC PGM=IEFBR14
//OUTFILE DD  DSN=<your-data-file-here>,
//            UNIT=SYSALLDA,
//            SPACE=<space-attributes-here>,
//            DISP=(MOD,DELETE)
//*
//GETDATA EXEC GETDATA,
```

```
       //     PRM=(' -p DD:MPF')
//OUTPUT  DD  DISP=SHR,DSN=<your-data-file-here>
//            DISP=(MOD,CATLG,KEEP),
//            UNIT=SYSALLDA,
//            SPACE=<space-attributes-here>,
//            DCB=<dcb-attributes-here>
```

MQIGETDA Parameters

The parameters for the MQIGETDA test program are stored in the SNEOMPF library.

```
[Get Control]
    #Name of the file to put the message data
    outputFileName                 =      DD:OUTPUT

    #Name of the queue to get the message from
    queueName                      =      RulesIn

    #Name of the queue manager that owns the queue
    queueManagerName               =      CSQ1

    #Maximum message size
    maxUserDataLength              =      10000

    #Name of the log file. Comment the following line if the
    #error/warning information is to be logged into SYSOUT.
    #logFileName                   =    DD:LOGFILE

    #"logLevel" used to control message logging to the file.
    #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
    #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          =          414D51205141514D20202020202020202034EA
```

```
#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                  =      1000

#Transaction commit size, ) 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. In MQSeries
#v.5.0 and later, 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

# Should the messages be rolled back after the get operation
rollback                      =      0
```

Parameters

The following parameters are used in the MQIGETDA file. The parameters are used to define Get control.

Note:
The default values for MQIGETDA should be the same as the MQMD deaults in MQSeries.

Get Control

Name	Mandatory/ Optional	Description
outputFileName	Optional	Name of file to put the message data. Default filename is DD:OUTPUT.
queueName	Mandatory	Name of queue to get the message from.
queueManagerName	Optional	Name of the local MQSeries Queue Manager. If not specified, the default MQSeries queue manager is used.
maxUserDataLength	Optional	Maximum message size in bytes.
logFileName	Optional	Name of the log file. Default filename is DD:LOGFILE.
logLevel	Mandatory	Defines message logging to the file. Valid settings are: 3 = log only fatal errors 2 = log errors and fatal errors 1 = log warnings, errors, and fatal errors 0 = log informationals, warnings, errors, and fatal errors

Name	Mandatory/ Optional	Description
messageID	Optional	ID of the message to get. If message ID and correlID are not defined, the application gets the next available message from the queue. The messageID field uses an encoded hex representation.
correlID	Optional	Correlation ID of the message to get. If message ID and correlID 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 messages to get. The application runs until <messageCount> messages are dequeued, or until the queue is empty.
commitSize	Mandatory	Transaction commit size;) indicates commit all records once.
getTimeout	Mandatory	Maximum length of time to wait for a message to arrive before the application reports a queue empty and exits.
showStatistics	Mandatory	Binary value indicating whether to output statistics information about dequeued messages. Value of 1 indicates yes; zero (0) indicates no.
outputToFile	Mandatory	Binary value indicating whether to output to a file. Value of 1 indicates yes; zero (0) indicates output should be sent to stderr.

Name	Mandatory/ Optional	Description
showDescriptor	Mandatory	Binary value indicating whether to output message descriptor data. Value of 1 indicates yes; zero (0) indicates no.
showData	Mandatory	Binary value indicating whether to output message data. Value of 1 indicates yes; zero (0) indicates no.
rollback	Mandatory	Binary value indicating whether to rollback messages after the get operation. Value of 1 indicates yes; zero (0) indicates no.

MQIGETDC

The following JCL compiles and links MQIGETDA:

```
//*  <insert a valid jobcard here >
//*
//*************************************************************
//*                                                           *
//* Licensed Materials - Property of IBM                      *
//*                                                           *
//* (c) Copyright IBM Corp. 1988, 1999                        *
//*                                                           *
//*************************************************************
//PROCLIST JCLLIB ORDER=(<smphlq>.SNEOPROC,SYS1.PROCLIB)
//COMPILE   EXEC  CBCCL15,
//    INFILE='<smphlq>.SNEOCPP',
//    INCFILE='<smphlq>.SNEOH',
//    OUTFILE='<smphlq>.SNEOLOAD',
//    MEMBER='MQIGETDA'
//LKED.SYSIN DD *
  INCLUDE MQSLOAD(CSQSTUB)
  NAME MQIGETDA(R)
```

RULETEST

The RULETEST program reads a message from a file and evaluates the message, using the application group / message type defined in the standard input stream (STDIN) SYSIN. The RULETEST program uses Formatter to evaluate messages only; the RULETEST program does not execute actions. After evaluation, subscriptions are retrieved as usual and output to the standard output stream (STDOUT) SYSPRINT, but not executed. The RULETEST program uses Rules for evaluating and retrieving subscriptions. This program does not execute subscriptions using Formatter.

The RULETEST program requires a connection to a database containing both NEONRules and NEONFormatter data, and this data must reside within the same database.

Sample JCL for RULETEST

The following sample job control language (JCL) is provided to illustrate how to run the RULETEST job in batch and pass startup parameters to it. The JCL at your site will be different. See *Tailoring Jobs for Your Site* on page 17 for information about the symbolic parameters in this sample.

```
//*  <insert a valid jobcard here >
//*
//*********************************************************************
//*                                                                   *
//* Licensed Materials - Property of New Era of Networks, Inc.        *
//* Copyright (c) 1998-1999, New Era of Networks, Inc.                *
//* All Rights Reserved.                                              *
//*                                                                   *
//* Release 4.1.1                                                     *
//*********************************************************************
//*********************************************************************
//*                                                                   *
//* RULETEST: Test Rule Evaluation                                    *
//*                                                                   *
//*********************************************************************
//RULETEST PROC SMPHLQ=<smphlq>',  HLQ for NEONetMQI distrib libs
//         MQSHLQ='<mqshlq>',      HLQ for MQS run-time libs
//         CEEHLQ='<lehlq>',       HLQ for Lang Envir  libs
//         CSSHLQ='SYS1',          HLQ for Callable Sys Svcs (CSS-)Lib
//        SQLMEM='SQLSVSES',      MEMbername for SQLSVSES cntl cards
```

```
//         INIMEM='CLIINI',        MEMbername for CLI INI cntl cards
//         OPCLAS='*'              SYSOUT CLASS
//*
//STP0101  EXEC PGM=RULETEST
//*
//*   <tailor the member STEPLIB and copy it here>
//*
//SQLSVSES DD  DSN=&SMPHLQ..SNEOCNTL(&SQLMEM),DISP=SHR
//DSNAOINI DD  DSN=&SMPHLQ..SNEOCNTL(&INIMEM),DISP=SHR
//SYSOUT   DD  SYSOUT=&OPCLAS
//SYSPRINT DD  SYSOUT=&OPCLAS
//STATLOG  DD  SYSOUT=&OPCLAS
//CLITRACE DD  SYSOUT=&OPCLAS   used for DB2 v5 CLI high-level tracing
//         PEND
//*
//* All datasets used by MQSeries Integrator must be preallocated and
//* cataloged prior to running any MQSeries Integrator jobs. The
//* recommended DCB attributes are:
//*    DSORG=PS,RECFM=VB,LRECL=32756,BLKSIZE=32760
//*
//* All datasets used by NEONet must be preallocated and cataloged
//* prior to running any NEONet jobs. The recommended DCB attributes
//* are:   DSORG=PS,RECFM=VB,LRECL=32756,BLKSIZE=32760
//*
//RULETEST EXEC RULETEST
//PUTDIN   DD  DISP=SHR,DSN=<your-test-message-file>
//SYSIN    DD  *
rules
DD:PUTDIN
<your-AppGrp>
<your-MessageType>
<verbose-y/n>
<reload-rules-cache-y/n>
/*
//
```

RULETSTC

The following JCL is used to compile and link RULETEST:

```
//*  <insert a valid jobcard here >
//*
//************************************************************
```

```
//*                                                                *
//* Licensed Materials - Property of New Era of Networks, Inc.*
//* Copyright (c) 1998-1999, New Era of Networks, Inc.        *
//* All Rights Reserved.                                      *
//*                                                           *
//* Release 4.1.1                                             *
//*************************************************************
//PROCLIST JCLLIB ORDER=(<smphlq>.SNEOPROC,SYS1.PROCLIB)
//COMPILE   EXEC  CBCCL15,
//    INFILE='<smphlq>.SNEOCPP',
//    INCFILE='<smphlq>.SNEOH',
//    OUTFILE='<smphlq>.SNEOLOAD',
//    MEMBER='RULETEST'
//LKED.SYSIN DD *
  NAME RULETEST(R)
```

Rules Debugging Utility

The NNRTRACE program is a debugging utility for testing rules. NNRTRACE evaluates a rule and messages associated with the rule to determine whether the rule will hit. A hit indicates that this message would cause the rule to hit. If the rule hits, the actions that can be performed by the rule are displayed. If no actions exist, the process fails while evaluating the message.

To use the NNRTRACE program, create an input file for the test procedure or use the MQIGETDA program to retrieve the messages to be tested from a queue. If you use MQIGETDA to retrieve messages, you might have to replay the queue to put the messages back on the queue because MQIGETDA will take a message off the queue.

Before running this executable, you must verify that the SQLSVSES file includes the relevant information to execute this program.

NNRTRACE

The following sample job control language (JCL) illustrates how to run the NNRTRACE job in batch and pass startup parameters to it. The JCL at your site will be different. See *Tailoring Jobs for Your Site* on page 17 for information about the symbolic parameters in this sample.

Sample JCL for NNRTRACE

```
//*  <insert a valid jobcard here >
//*
//*******************************************************************
//*                                                                 *
//* Licensed Materials - Property of New Era of Networks, Inc.      *
//* Copyright (c) 1998-1999, New Era of Networks, Inc.              *
//* All Rights Reserved.                                            *
//*                                                                 *
//* Release 4.1.1                                                   *
//*******************************************************************
//*******************************************************************
//*                                                                 *
//* NNRTRACE: Trace Rule Processing - used for debugging            *
//*                                                                 *
//*******************************************************************
//NNRTRACE PROC SMPHLQ=<smphlq>',  HLQ for MQI distrib libs
//         MQSHLQ='<mqshlq>',      HLQ for MQS run-time libs
//         CEEHLQ='<lehlq>',       HLQ for Lang Envir  libs
//         CSSHLQ='SYS1',          HLQ for Callable Sys Svcs (CSS-)Lib
//         SQLMEM='SQLSVSES',      MEMbername for SQLSVSES cntl cards
//         INIMEM='CLIINI',        MEMbername for CLI INI  cntl cards
//         OPCLAS='*'              SYSOUT CLASS
//*
//STP0101  EXEC PGM=NNRTRACE
//*
//*   <tailor the member STEPLIB and copy it here>
//*
//SQLSVSES DD  DSN=&SMPHLQ..SNEOCNTL(&SQLMEM),DISP=SHR
//DSNAOINI DD  DSN=&SMPHLQ..SNEOCNTL(&INIMEM),DISP=SHR
//SYSOUT   DD  SYSOUT=&OPCLAS
//SYSPRINT DD  SYSOUT=&OPCLAS
//STATLOG  DD  SYSOUT=&OPCLAS
//CLITRACE DD  SYSOUT=&OPCLAS   used for DB2 v5 CLI high-level tracing
//         PEND
//*
//* All datasets used by MQSeries Integrator must be preallocated and
//* cataloged prior to running any MQseries Integrator jobs. The
//* recommended  DCB attributes are:
//*   DSORG=PS,RECFM=VB,LRECL=32756,BLKSIZE=32760
//*
```

```
//NNRTRACE EXEC NNRTRACE
//INPUT   DD  DISP=SHR,DSN=<your-test-message-file>
//SYSIN    DD  *
rules
DD:INPUT
<your-AppGrp>
<your-MessageType>
<your-Rule-to-trace>
<session-name>
<verbose-y/n>
/*
//
```

Chapter 6
Consistency Checker

The Consistency Checker is a utility to check the consistency of MQSeries Integrator components. The Consistency Checker lists objects as invalid that are out of synchronization due to a recovery, bad migration, or some other reason.

Most of the items verify the internal structure of the rules to confirm that they are properly created. Some checks verify that user-defined data is correctly entered and whether records have corresponding features in the database. All formats and rules in an inconsistent state generate a report indicating the problem.

Starting the Consistency Checker

The following scripts constitute the Consistency Checker. Prior to contacting technical support, you should execute these scripts, and then examine the reports to determine if the database is corrupted. If any report produced by these scripts contains data, a data inconsistency exists that must be repaired.

- FORMATCC

  Checks logical consistency of the Formatter tables.

  To run the Consistency Checker for Formatter, execute the FORMATCC SQL script using SPUFI.

- RULECC

  Checks logical consistency of the Rules tables.

  To run the Consistency Checker for Rules, execute the RULECC SQL script using SPUFI.

- PERMCC

  Checks logical consistency of the Rules permissions.

  To run the Consistency Checker for Rules permissions, execute the PERMCC SQL script using SPUFI.

- NEOMQCC

  Scans the Rules database for putq Subscription actions, and determines whether a queue exists. NEOMQCC outputs a list of any missing queues. If all queues are valid, the following message is displayed:

  ```
  All queues exist. Put Messages are valid.
  ```

  Use the following sample JCL to run NEOMQCC:

```
//* <tailor member JOBCARD and insert here>
//*
//*********************************************************************
//*                                                                   *
//* Licensed Materials - Property of New Era of Networks, Inc.        *
//* Copyright (c) 1998-1999, New Era of Networks, Inc.                *
//* All Rights Reserved.                                              *
//*                                                                   *
//* Release 4.1.1                                                     *
//*********************************************************************
//*********************************************************************
//*                                                                   *
//* NEOMQCC: MQSeries Consistency Checker Application                 *
//*                                                                   *
//*********************************************************************
//NEOMQCC  PROC PRM=(' rules CSQ1'),
//         SMPHLQ='<smphlq>',      HLQ for NEONet distrib libs
//         MQSHLQ='<mqshlq>',      HLQ for MQS run-time libs
//         CEEHLQ='<ceehlq>',      HLQ for Lang Envir libs
//         CSSHLQ='SYS1',          HLQ for Callable Sys Svcs (CSS-)Lib
//         SQLMEM='SQLSVSES',      MEMbername for SQLSVSES cntl cards
//         INIMEM='CLIINI',        MEMbername for CLI INI  cntl cards
//         OPCLAS='*'              SYSOUT CLASS
//*
//STP0101  EXEC PGM=MQRPUTQC,
//   PARM=&PRM
```

```
//*
//* <tailor member STEPLIB and insert here>
//*
//SQLSVSES DD  DSN=&SMPHLQ..SNEOCNTL(&SQLMEM),DISP=SHR
//DSNAOINI DD  DSN=&SMPHLQ..SNEOCNTL(&INIMEM),DISP=SHR
//SYSOUT   DD  SYSOUT=&OPCLAS
//SYSPRINT DD  SYSOUT=&OPCLAS
//STATLOG  DD  SYSOUT=&OPCLAS
//CLITRACE DD  SYSOUT=&OPCLAS used for DB2 v5 CLI high-level tracing
//SYSIN    DD  DUMMY
//         PEND
//*
//*
//* All datasets used by MQSeries Integrator must be preallocated and
//* cataloged prior to running any MQSeries Integrator jobs. The
//* recommended DCB attributes are:
//*    DSORG=PS,RECFM=VB,LRECL=32756,BLKSIZE=32760
//*
//* All datasets used by NEONet must be preallocated and cataloged
//* prior to running any NEONet jobs. The recommended DCB attributes
//* are:   DSORG=PS,RECFM=VB,LRECL=32756,BLKSIZE=32760
//*
//NEOMQCC  EXEC NEOMQCC,PRM=(' <rules-session> <queue-manager>')
//
```

Consistency Checker Report: Formatter

The Formatter Consistency Checker report provides the following information:

Reported Inconsistency	Explanation
Case operations that refer to nonexistent case choices	Use 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.

Reported Inconsistency	Explanation
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.
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.

Reported Inconsistency	Explanation
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 or input control NONE	Choose fields other than NONE to insert into input flat format. Choose input controls other than NONE for input fields.
Incomplete output format fields that refer to field NONE 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.
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.

Reported Inconsistency	Explanation
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.
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.

Reported Inconsistency	Explanation
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.
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.

Reported Inconsistency	Explanation
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.
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.

Reported Inconsistency	Explanation
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.
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.

Reported Inconsistency	Explanation
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.
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.

Reported Inconsistency	Explanation
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.
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.

Reported Inconsistency	Explanation
User-defined data types with invalid data type identifiers	Extraneous data in the database. Database integrity may be compromised.

Consistency Checker Report: Rules

The Rules Consistency Checker report provides the following information:

Reported Inconsistency	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 (0)	A Boolean operator must always have at least two (2) child arguments or Boolean operators. This may cause load or evaluation failure.
Boolean operators that recurse more than five (5) times and maybe infinitely	This expression has several nested expressions. 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	Child 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.

Reported Inconsistency	Explanation
Field Name2 (Comparison Value) in arguments that refer to nonexistent fields in NEONFormatter	A field name was entered in an argument as a comparison value. The field name is not a valid field in the NEONFormatter. Evaluation may fail or not hit.
Field Name2 (Comparison Value) in arguments that refer to nonexistent flat fields in NEONFormatter	A field name was entered in an argument as a comparison value. 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. Currently, the NEONRules Consistency Checker does not check fields in compound formats.
Field names in arguments that refer to nonexistent flat fields in NEONFormatter	A field name was entered in an argument. 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. Currently, the NEONRules Consistency Checker does not check fields in compound formats.
Fields names in arguments that refer to nonexistent fields in NEONFormatter	A field name was entered in an argument, and the field name is not a valid field in the NEONFormatter. Evaluation may fail or not hit.
Message types in Rules that do not match a format in NEONFormatter	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.
Message types that refer to nonexistent application groups	The application group does not exist for the 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 requires the same number of children arguments or Boolean operators as is indicated. This will cause evaluation to work incorrectly.

Reported Inconsistency	Explanation
Number of arguments in a Boolean OR term is incorrect	If the expression uses OR, it should have a specific argument count of 1. A Boolean OR operator requires a certain number of children arguments 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 or operation. You cannot access these rules.
Rules that have argument count of zero (0)	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.
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 no active subscriptions	All rules must have at least one subscription. This report displays rules with no subscriptions. This may cause evaluation failure.
Subscription action (Put Message) message type does not exist in the NEONFormatter	The message type does not exist in the NEONFormatter.

Reported Inconsistency	Explanation
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 or Put Message) has less than two options	
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.
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.
WARNING! Rules that may not put messages on a queue	

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 necessary if users provide their own daemon and generic actions.

Consistency Checker Report: Permissions

The Rules Permissions Consistency Checker report provides the following information:

Reported Inconsistency	Explanation
Hierarchy definitions that are not complete for rule or subscription permissions	The hierarchy definitions must be complete during the installation of NEONRules with permissions.
Permission access 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 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.
Permissions granted to nonexistent Item	Rule or subscription permissions must refer to a valid item name.
Permissions granted to nonexistent subscriptions	Subscription permissions must refer to a valid subscription name.
Permissions granted to nonexistent Users	Rules permissions need both a valid user and rule subscription to be complete.
Permissions that are not complete	Rule or subscription permissions must include node, application group, message type, and rule or subscription name to be complete.
Permissions that do not exist in the hierarchy	Rule or subscription permissions must refer to valid hierarchy information.
Permissions that refer to nonexistent application groups	Rule or subscription permissions must refer to a valid application group.

Reported Inconsistency	Explanation
Permissions that refer to nonexistent message types	Rule or subscription permissions must refer to a valid message type/format name.
Permissions that refer to nonexistent nodes	Rule or subscription permissions must refer to the current node.
Permissions that refer to nonexistent Rules	Rule permissions must refer to a valid rule name.
Rules with multiple owners	Rules can only have one owner.
Rules with no owners	Each rule must have a single owner (Owner and Granted)
Subscriptions with multiple owners	Subscriptions can only have one owner.
Subscriptions with no owners	Each subscription can only have one owner (Owner and Granted).
Unique sequence generator invalid for permission users	A new Participant Id is derived by incrementing the SEQ_NUM for Participant and using this as the Id.
Unique sequence generator invalid for rule or subscription permission	A new Permission Grant Id is derived by incrementing the SEQ_NUM for Component and using this as the id for Component Grants. The Component Grant Id should never be greater than SEQ_NUM.

Appendix A

ASCII Extended Character Set

Decimal Value	Hex Value	Extended Character Set
000	00	NUL
001	01	SCH
002	02	STX
003	03	ETX
004	04	EOT
005	05	ENO
006	06	ACK
007	07	BEL
008	08	BS
009	09	HT
010	0A	LF
011	0B	VT
012	0C	FF
013	0D	CR
014	0E	SO
015	0F	SI

Decimal Value	Hex Value	Extended Character Set
016	10	DLE
017	11	DC1
018	12	DC2
019	13	DC3
020	14	DC4
021	15	NAK
022	16	SYN
023	17	ETB
024	18	CAN
025	19	EM
026	1A	SUB
027	1B	ESCAPE
028	1C	FS
029	1D	GS
030	1E	RS
031	1F	US

Decimal Value	Hex Value	Extended Character Set
032	20	SPACE
033	21	!
034	22	“
035	23	#
036	24	$
037	25	%
038	26	&
039	27	‘
040	28	(
041	29	)
042	2A	*
043	2B	+
044	2C	,
045	2D	-
046	2E	.
047	2F	/
048	30	0
049	31	1
050	32	2
051	33	3
052	34	4
053	35	5
054	36	6

Decimal Value	Hex Value	Extended Character Set
055	37	7
056	38	8
057	39	9
058	3A	:
059	3B	;
060	3C	<
061	3D	=
062	3E	>
063	3F	?
064	40	@
065	41	A
066	42	B
067	43	C
068	44	D
069	45	E
070	46	F
071	47	G
072	48	H
073	49	I
074	4A	J
075	4B	K
076	4C	L
077	4D	M

Decimal Value	Hex Value	Extended Character Set
078	4E	N
079	4F	O
080	50	P
081	51	Q
082	52	R
083	53	S
084	54	T
085	55	U
086	56	V
087	57	W
088	58	X
089	59	Y
090	5A	Z
091	5B	[
092	5C	\
093	5D	]
094	5E	^
095	5F	_
096	60	‘
097	61	a
098	62	b
099	63	c
100	64	d

Decimal Value	Hex Value	Extended Character Set
101	65	e
102	66	f
103	67	g
104	68	h
105	69	i
106	6A	j
107	6B	k
108	6C	l
109	6D	m
110	6E	n
111	6F	o
112	70	p
113	71	q
114	72	r
115	73	s
116	74	t
117	75	u
118	76	v
119	77	w
120	78	x
121	79	y
122	7A	z
123	7B	{

Decimal Value	Hex Value	Extended Character Set
124	7C	|
125	7D	}
126	7E	~
127	7F	DEL
128	80	€
129	81	unused
130	82	‚
131	83	*f*
132	84	„
133	85	…
134	86	†
135	87	‡
136	88	ˆ
137	89	‰
138	8A	Š
139	8B	‹
140	8C	Œ
141	8D	unused
142	8E	unused
143	8F	unused
144	90	unused
145	91	‘
146	92	‘

Decimal Value	Hex Value	Extended Character Set
147	93	“
148	94	”
149	95	•
150	96	–
151	97	—
152	98	˜
153	99	™
154	9A	š
155	9B	›
156	9C	œ
157	9D	unused
158	9E	unused
159	9F	Ÿ
160	A0	nonbreaking space
161	A1	¡
162	A2	¢
163	A3	£
164	A4	¤
165	A5	¥
166	A6	¦
167	A7	§
168	A8	¨

Decimal Value	Hex Value	Extended Character Set
169	A9	©
170	AA	ª
171	AB	«
172	AC	¬
173	AD	-
174	AE	®
175	AF	¯
176	B0	°
177	B1	±
178	B2	²
179	B3	³
180	B4	´
181	B5	µ
182	B6	¶
183	B7	·
184	B8	¸
185	B9	¹
186	BA	º
187	BB	»
188	BC	¼
189	BD	½
190	BE	¾
191	BF	¿
192	C0	À
193	C1	Á
194	C2	Â
195	C3	Ã
196	C4	Ä
197	C5	Å
198	C6	Æ
199	C7	Ç
200	C8	È
201	C9	É
202	CA	Ê
203	CB	Ë
204	CC	Ì
205	CD	Í
206	CE	Î
207	CF	Ï
208	D0	Ð
209	D1	Ñ
210	D2	Ò
211	D3	Ó
212	D4	Ô
213	D5	Õ
214	D6	Ö

Decimal Value	Hex Value	Extended Character Set
215	D7	×
216	D8	Ø
217	D9	Ù
218	DA	Ú
219	DB	Û
220	DC	Ü
221	DD	Ý
222	DE	Þ
223	DF	ß
224	E0	à
225	E1	á
226	E2	â
227	E3	ã
228	E4	ä
229	E5	å
230	E6	æ
231	E7	ç
232	E8	è
233	E9	é
234	EA	ê
235	EB	ë
236	EC	ì
237	ED	í
238	EE	î
239	EF	ï
240	F0	ð
241	F1	ñ
242	F2	ò
243	F3	ó
244	F4	ô
245	F5	õ
246	F6	ö
247	F7	÷
248	F8	ø
249	F9	ù
250	FA	ú
251	FB	û
252	FC	ü
253	FD	ý
254	FE	þ
255	FF	ÿ

Appendix B

EBCDIC Character Set

Decimal Value	Hex Value	EBCDIC Value*	Description	Binary
000	00	NUL	Null	0000 0000
001	01	SOH	Start of Heading	0000 0001
002	02	STX	Start of Text	0000 0010
003	03	ETX	End of Text	0000 0011
004	04	SEL	Select	0000 0100
005	05	HT	Horizontal Tab	0000 0101
006	06	RNL	Required New Line	0000 0110
007	07	DEL	Delete	0000 0111
008	08	GE	Graphic Escape	0000 1000
009	09	SPS	Superscript	0000 1001
010	0A	RPT	Repeat	0000 1010
011	0B	VT	Vertical Tab	0000 1011
012	0C	FF	Form Feed	0000 1100
013	0D	CR	Carriage Return	0000 1101
014	0E	SO	Shift Out	0000 1110
015	0F	SI	Shift In	0000 1111
016	10	DLE	Data Link Escape	0001 0000

Decimal Value	Hex Value	EBCDIC Value*	Description	Binary
017	11	DC1	Device Control 1	0001 0001
018	12	DC2	Device Control 2	0001 0010
019	13	DC3	Device Control 3	0001 0011
020	14	RES / ENP	Restore / Enable Presentation	0001 0100
021	15	NL	New Line	0001 0101
022	16	BS	Backspace	0001 0110
023	17	POC	Program-Operator Communication	0001 0111
024	18	CAN	Cancel	0001 1000
025	19	EM	End of Medium	0001 1001
026	1A	UBS	Unit Backspace	0001 1010
027	1B	CU1	Customer Use 1	0001 1011
028	1C	IFS	Interchange File Separator	0001 1100
029	1D	IGS	Interchange Group Separator	0001 1101
030	1E	IRS	Interchange Record Separator	0001 1110
031	1F	IBT / IUS	Intermediate Transmission Block / Interchange Unit Separator	0001 1111
032	20	DS	Digit Select	0010 0000
033	21	SOS	Start of Significance	0010 0001
034	22	FS	Field Separator	0010 0010
035	23	WUS	Word Underscore	0010 0011
036	24	BYP / INP	Bypass / Inhibit Presentation	0010 0100
037	25	LF	Line Feed	0010 0101

Decimal Value	Hex Value	EBCDIC Value*	Description	Binary
038	26	ETB	End of Transmission Block	0010 0110
039	27	ESC	Escape	0010 0111
040	28	SA	Set Attribute	0010 1000
041	29	SFE	Start Field Extended	0010 1001
042	2A	SM/SW	Set Mode/Switch	0010 1010
043	2B	CSP	Control Sequence Prefix	0010 1011
044	2C	MFA	Modify Field Attribute	0010 1100
045	2D	ENQ	Enquiry	0010 1101
046	2E	ACK	Acknowledge	0010 1110
047	2F	BEL	Bell	0010 1111
048	30			0011 0000
049	31			0011 0001
050	32	SYN	Synchronous Idle	0011 0010
051	33	IR	Index Return	0011 0011
052	34	PP	Presentation Position	0011 0100
053	35	TRN	Transparent	0011 0101
054	36	NBS	Numeric Backspace	0011 0110
055	37	EOT	End of Transmission	0011 0111
056	38	SBS	Subscript	0011 1000
057	39	IT	Indent Tab	0011 1001
058	3A	RFF	Required Form Feed	0011 1010
059	3B	CU3	Customer Use 3	0011 1011

Decimal Value	Hex Value	EBCDIC Value*	Description	Binary
060	3C	DC4	Device Control 4	0011 1100
061	3D	NAK	Negative Acknowledge	0011 1101
062	3E			0011 1110
063	3F	SUB	Substitute	0011 1111
064	40	SP	Space	0100 0000
065	41	RSP		0100 0001
066	42			0100 0010
067	43			0100 0011
068	44			0100 0100
069	45			0100 0101
070	46			0100 0110
071	47			0100 0111
072	48			0100 1000
073	49			0100 1001
074	4A	¢		0100 1010
075	4B	.		0100 1011
076	4C	<		0100 1100
077	4D	(		0100 1101
078	4E	+		0100 1110
079	4F	|		0100 1111
080	50	&		0101 0000
081	51			0101 0001

Decimal Value	Hex Value	EBCDIC Value*	Description	Binary
082	52			0101 0010
083	53			0101 0011
084	54			0101 0100
085	55			0101 0101
086	56			0101 0110
087	57			0101 0111
088	58			0101 1000
089	59			0101 1001
090	5A	!		0101 1010
091	5B	$		0101 1011
092	5C	*		0101 1100
093	5D	)		0101 1101
094	5E	;		0101 1110
095	5F	¬		0110 1111
096	60	-		0110 0000
097	61	/		0110 0001
098	62			0110 0010
099	63			0110 0011
100	64			0110 0100
101	65			0110 0101
102	66			0110 0110
103	67			0110 0111

Decimal Value	Hex Value	EBCDIC Value*	Description	Binary
104	68			0110 1000
105	69			0110 1001
106	6A	|		0110 1010
107	6B	,		0110 1011
108	6C	%		0110 1100
109	6D	_		0110 1101
110	6E	>		0110 1110
111	6F	?		0110 1111
112	70			0111 0000
113	71			0111 0001
114	72			0111 0010
115	73			0111 0011
116	74			0111 0100
117	75			0111 0101
118	76			0111 0110
119	77			0111 0111
120	78			0111 1000
121	79			0111 1001
122	7A	:		0111 1010
123	7B	#		0111 1011
124	7C	@		0111 1100
125	7D	’		0111 1101

Decimal Value	Hex Value	EBCDIC Value*	Description	Binary
126	7E	=		0111 1110
127	7F	"		0111 1111
128	80	€		1000 0000
129	81	a		1000 0001
130	82	b		1000 0010
131	83	c		1000 0011
132	84	d		1000 0100
133	85	e		1000 0101
134	86	f		1000 0110
135	87	g		1000 0111
136	88	h		1000 1000
137	89	i		1000 1001
138	8A			1000 1010
139	8B			1000 1011
140	8C			1000 1100
141	8D			1000 1101
142	8E			1000 1110
143	8F			1000 1111
144	90			1001 0000
145	91	j		1001 0001
146	92	k		1001 0010
147	93	l		1001 0011

Decimal Value	Hex Value	EBCDIC Value*	Description	Binary
148	94	m		1001 0100
149	95	n		1001 0101
150	96	o		1001 0110
151	97	p		1001 0111
152	98	q		1001 1000
153	99	r		1001 1001
154	9A			1001 1010
155	9B			1001 1011
156	9C			1001 1100
157	9D			1001 1101
158	9E			1001 1110
159	9F			1001 1111
160	A0			1010 0000
161	A1	~		1010 0001
162	A2	s		1010 0010
163	A3	t		1010 0011
164	A4	u		1010 0100
165	A5	v		1010 0101
166	A6	w		1010 0110
167	A7	x		1010 0111
168	A8	y		1010 1000
169	A9	z		1010 1001

Decimal Value	Hex Value	EBCDIC Value*	Description	Binary
170	AA			1010 1010
171	AB			1010 1011
172	AC			1010 1100
173	AD			1010 1101
174	AE			1010 1110
175	AF			1010 1111
176	B0			1011 0000
177	B1			1011 0001
178	B2			1011 0010
179	B3			1011 0011
180	B4			1011 0100
181	B5			1011 0101
182	B6			1011 0110
183	B7			1011 0111
184	B8			1011 1000
185	B9			1011 1001
186	BA			1011 1010
187	BB			1011 1011
188	BC			1011 1100
189	BD			1011 1101
190	BE			1011 1110
191	BF			1011 1111

Decimal Value	Hex Value	EBCDIC Value*	Description	Binary
192	C0	{		1100 0000
193	C1	A		1100 0001
194	C2	B		1100 0010
195	C3	C		1100 0011
196	C4	D		1100 0100
197	C5	E		1100 0101
198	C6	F		1100 0110
199	C7	G		1100 0111
200	C8	H		1100 1000
201	C9	I		1100 1001
202	CA	SHY		1100 1010
203	CB			1100 1011
204	CC			1100 1100
205	CD			1100 1101
206	CE			1100 1110
207	CF			1100 1111
208	D0	}		1101 0000
209	D1	J		1101 0001
210	D2	K		1101 0010
211	D3	L		1101 0011
212	D4	M		1101 0100
213	D5	N		1101 0101

Decimal Value	Hex Value	EBCDIC Value*	Description	Binary
214	D6	O		1101 0110
215	D7	P		1101 0111
216	D8	Q		1101 1000
217	D9	R		1101 1001
218	DA			1101 1010
219	DB			1101 1011
220	DC			1101 1100
221	DD			1101 1101
222	DE			1101 1110
223	DF			1101 1111
224	E0	\		1110 0000
225	E1			1110 0001
226	E2	S		1110 0010
227	E3	T		1110 0011
228	E4	U		1110 0100
229	E5	V		1110 0101
230	E6	W		1110 0110
231	E7	X		1110 0111
232	E8	Y		1110 1000
233	E9	Z		1110 1001
234	EA			1110 1010
235	EB			1110 1011

Decimal Value	Hex Value	EBCDIC Value*	Description	Binary
236	EC			1110 1100
237	ED			1110 1101
238	EE			1110 1110
239	EF			1110 1111
240	F0	0		1111 0000
241	F1	1		1111 0001
242	F2	2		1111 0010
243	F3	3		1111 0011
244	F4	4		1111 0100
245	F5	5		1111 0101
246	F6	6		1111 0110
247	F7	7		1111 0111
248	F8	8		1111 1000
249	F9	9		1111 1001
250	FA			1111 1010
251	FB			1111 1011
252	FC			1111 1100
253	FD			1111 1101
254	FE			1111 1110
255	FF	EO	Eight Ones	1111 1111

* In the IBM-DOS Character Set, the nonprinting characters may be displayed as figures, for example, (x03) ETX is shown as a heart, and (x0D) CR is shown as a musical note.

Appendix C

Data Type Descriptions

Supported Data Types

Data Type Field Values	Data Type #Define	Description
Not Applicable	DATA_TYPE_Not_ Applicable	No data type is assumed.
ASCII String	DATA_TYPE_ASCII_ String	A string of standard ASCII characters. Non-printable characters are valid if they are in the ASCII character set. EBCDIC characters outside the valid ASCII String range are not valid ASCII String characters. If a character being converted from ASCII to EBCDIC is not in the EBCDIC character set, the conversion results in a EBCDIC space (hexadecimal 40).
ASCII Numeric	DATA_TYPE_ASCII_ Numeric	A string of standard ASCII numeric characters. The hyphen (-) and period (.) characters are not valid ASCII numeric characters.

Data Type Field Values	Data Type #Define	Description
Binary	DATA_TYPE_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 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_TYPE_ EBCDIC_Data	A string of characters encoded using the EBCDIC (Extended Binary Coded Decimal Interchange Code) encoding used on larger IBM computers. If a character being converted from EBCDIC to ASCII is not in the EBCDIC character set, the conversion results in a space (hexadecimal 20).

Data Type Field Values	Data Type #Define	Description
IBM Packed Decimal	DATA_TYPE_IBM_ Packed_Decimal	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.
IBM Signed Packed Decimal	DATA_TYPE_IBM_ Signed_Packed_ Decimal	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. For example, the number 1234 is stored as a 3-byte value: 01 23 4C (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 5D.
IBM Zoned Decimal	DATA_TYPE_IBM_ Zoned_Decimal	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).

Data Type Field Values	Data Type #Define	Description
IBM Signed Zoned Decimal	DATA_TYPE_IBM_ Signed_Zoned_ Decimal	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, 123 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	DATA_TYPE_Little_ Endian2	Two-byte integer where the bytes are ordered with the rightmost byte being the high order or most significant byte. For example, the hexadecimal number 0x0102 is stored as 02 01 (where the number pairs show the hexadecimal values of the nibbles of a byte).
Little Swap Endian 2	DATA_TYPE_Little_ Swap_Endian2	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	DATA_TYPE_Little_ Endian4	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	DATA_TYPE_Little_ Swap_Endian4	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.

Data Type Field Values	Data Type #Define	Description
Big Endian 2	DATA_TYPE_Big_ Endian2	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	DATA_TYPE_Big_ Swap_Endian2	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	DATA_TYPE_Big_ Endian4	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	DATA_TYPE_Big_ Swap_Endian4	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_ 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 datatypes can contain negative values.

Data Type Field Values	Data Type #Define	Description
Decimal, U.S.	DATA_TYPE_ Decimal_US	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 datatypes can contain negative values.
Unsigned Little Endian 2	DATA_TYPE_ Unsigned_ LittleEndian2	Like Little Endian 2, except that the value is interpreted as an unsigned value.
Unsigned Little Swap Endian 2	DATA_TYPE_ Unsigned_ LittleSwapEndian2	Like Little Swap Endian 2, except that the value is interpreted as an unsigned value.
Unsigned Little Endian 4	DATA_TYPE_ Unsigned_ LittleEndian4	Like Little Endian 4, except that the value is interpreted as an unsigned value.
Unsigned Little Swap Endian 4	DATA_TYPE_ Unsigned_ LittleSwapEndian4	Like Little Swap Endian 4, except that the value is interpreted as an unsigned value.
Unsigned Big Endian 2	DATA_TYPE_ Unsigned_ BigEndian2	Like Big Endian 2, except that the value is interpreted as an unsigned value.
Unsigned Big Swap Endian 2	DATA_TYPE_ Unsigned_ BigSwapEndian2	Like Big Swap Endian 2, except that the value is interpreted as an unsigned value.
Unsigned Big Endian 4	DATA_TYPE_ Unsigned_ BigEndian4	Like Big Endian 4, except that the value is interpreted as an unsigned value
Unsigned Big Swap Endian 4	DATA_TYPE_ Unsigned_ BigSwapEndian4	Like Big Swap Endian 4, except that the value is interpreted as an unsigned value.

Data Type Field Values	Data Type #Define	Description
Date and Time		Based on the international ISO-8601:1988 standard datetime notation: YYYYMNDDhhmmss. See the first paragraph of the following Date and Time type descriptions for details on representing Date and Time components. Combined dates and times can be represented in any of the following data types. The list includes String, Numeric, and EBCDIC.
Time		Based on the international ISO-8601:1988 standard time notation: HHmmss where HH represents the number of complete hours that have passed since midnight (between 00 and 23), mm is the number of minutes passed since the start of the hour (between 00 and 59), and ss is the number of seconds since the start of the minute (between 00 and 59). Times are represented in 24-hour format. Times may be represented in any of the following list of data types. For some data types, a minimum of 4 bytes is required. The list includes: EBCDIC, String, and Numeric.
Date		Based on the international ISO-8601:1988 standard date notation: YYYYMNDD 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 data types. For some data types, a minimum of 4 bytes is required. The list includes: EBCDIC, String, and Numeric.

Data Type Field Values	Data Type #Define	Description
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). The format string for month numbers is MN. The format string of three letter abbreviations is MON. 3) Variations in the day of the month–use of a day of the month number (01-31). DD is the format string. 4) Variations in hour–12-hour or 24-hour representation, with or without a meridian indicator (AM or PM). Hours/minutes/seconds are represented as HHMMSS. 5) Valid data types include EBCDIC, String, and Numeric. For information on how to set the Year Cutoff value when you select Custom Date and Time, refer to the section *Specifying a Year Cutoff Value*.

Notes for Data Conversion

Formatter can convert data between any two supported types via an intermediary representation. The data conversion occurs when Formatter, during a reformat(), encounters an input field with one data type and a different data type for the output field.

Certain pairs of data conversions do not make much sense. For example, if you have a String in the input message with non-numeric data and the output format specifies that the data type for the field should be IBM packed decimal, that conversion cannot happen correctly. Formatter generates an error message indicating invalid data.

Formatter does not have a conversion function for every pair of native data types. Instead, Formatter converts data internally from the input data type to a String representation, and then from the String representation to the output data type. So instead of (K**2 - K) conversion functions, Formatter has (K*2 - 2) functions, where K is the number of native data types.

For example, to convert from IBM signed packed decimal to IBM packed decimal, if the input is:

12 34 56 7C (where each pair of numbers are the 2 nibbles of a byte of data)

The data length is 4 bytes and the data represents the number “+1234567.” The “C” is a sign nibble indicating the number is positive.

Formatter converts this to the String “+1234567,” then converts the String to IBM packed data:

12 34 56 7F

When binary data (DATA_TYPE_Binary_Data) is involved, it means that the bytes of data can have any value without restriction or interpretation. If you have a field in the input format that’s in binary and the corresponding field in the output format is also binary, what does Formatter do?

For example, if you have:

12 34 56 78 90 ab cd ef

where each pair of numbers are the 2 nibbles (in hexidecimal encoding) of a byte of data.

Formatter first converts this data to an ASCII string representation of the binary data:

“1234567890abcdef”

and then converts this ASCII string back to binary data:

12 34 56 78 90 ab cd ef

To convert between an ASCII string and binary, Formatter expects the ASCII string to be a proper representation of a binary value. If you have the input:

“Hello, world!”

and you want Formatter to generate a binary value on output.

Formatter issues an error because the ASCII string is not a proper string representation of a binary value. The string must be composed of the characters 0-9 and A-F.

The actual binary encoding of the ASCII string “Hello, world!” is:

48 65 6c 6c 6f 2c 20 77 6f 72 6c 64 21

Specify that the data type of the input is binary, not ASCII string. Formatter generates the internal ASCII string:

“48656c6c6f2c20776f726c6421”

and then converts this back to binary:

48 65 6c 6c 6f 2c 20 77 6f 72 6c 64 21

To construct a binary value that equates to the actual byte values of your input, specify that the input data type is also Binary. If you have an input data type other than Binary, Formatter attempts to interpret your input as the string representation of a binary value.

Appendix D
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
OS/390
AIX
DB2
IBM

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

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

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

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

Index

N

O

P

Q

R

S

T

U

Sending your comments to IBM
MQSeries Integrator for OS/390
System Management Guide
SC34-5748-00

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

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

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

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

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

- By 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 for OS/390
System Management Guide
SC34-5748-00

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

SC34-5748-00