Rules and Formatter Extension for IBM ® WebSphere Message Broker for Multiplatforms

System Management Guide

Version 6.0

Note: Before using this information, and the product it supports, be sure to read the general information under *Notices* on page 213**.**

First Edition (August 2005)
This edition applies to Rules and Formatter Extension for WebSphere [tm] Message Broker for Multiplatforms, Version 6.0, for IBM ® WebSphere Message Broker 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 1

About This Document 2
Internationalization 4
Supported Code Sets and Locales 4
Documentation Set 8
Document Conventions 8

Chapter 2: New Era of Networks Rules and Formatter Installation on z/OS 11

Installing the New Era of Networks Rules and Formatter Database 11
Executable and Catalogue File Location 15

Chapter 3: Configuration 17

Shared Libraries 17
Library Paths 19
Runtime Configuration 19
Sample nnsyreg.dat File 20
Locating the Example Folder 20
Maintaining the nnsyreg.dat File 21
Searching the System for Configuration Information 22
Configuring Runtime Parameters 23
Entering Session Information 25
Copying the Messages Properties File 36
Encrypting the Configuration File 37
New Era of Networks Rules and Formatter Permissions 38
Listing and Changing Permissions 39
Examples 42
Error Conditions 43
Message Reporting 44
NNSYmessageLog.nml 44

Chapter 4: Formats and Rules Database Migration 45

Process Overview 46
Preparing Formats and Rules for Upgrade 48
Determining Compatibility 48
Upgrading Formats 50
NNFie 50
Importing and Exporting Formats 50
Export Phase 51
Import Phase 51
Using NNFie with DB2 52
Preparing Formats for Upgrade 53
Exporting Formats 55
Examples 58
Troubleshooting NNFie Export Failures 58
Producing an Inventory Export File 58
Importing Formats 59
Examples 61
Resolving Component Conflicts 62
Troubleshooting Format Import Failures 63
NNFie Readable Files 65
NNFie Header 65
NNFie Export Data 66
NNFie Error Messages 85
Upgrading Rules 85
NNRie 85
Importing and Exporting Rules 86
Using NNRie with DB2 87
Preparing Rules for Upgrade 87
NNRie 88
Exporting Rules 89
Examples 92
Producing an Inventory Export File 93
Importing Rules 94
Examples 96
Tracking Import Progress 96
Resolving Component Conflicts 97
Troubleshooting NNRie Import Failures 99
NNRie Readable Files 100
Header 100
Export Data 101

NNRie Error Messages 106
Cross-Version and Cross-Platform Migration 107
Cross-Version Migration 107
Literals 107
Cross-Platform Migration 109
Migrating Between Versions and Platforms 109
Process Overview 110
Cross-Version and Cross-Platform Migration Procedures 111
Chapter 5: New Era of Networks Formatter ..119
Overview 119
New Era of Networks Formatter GUI 120
Fields and Input Controls 120
Output Controls 121
Formats 122
New Era of Networks Formatter Management APIs 123
Automatic Format Conversion 123
Reloading Formats 123
Testing Formats 124
Running apitest 125
Syntax 126
Running msgtest 127
Syntax 130
Removing Formats 132
Customizing New Era of Networks Formatter Using C++ 133
Creating C++ User Exits 134
Replacing the Lookup Stub Function 135
Building a C++ User Exit Library 136
C++ User Exit Example 138
Threading User Exits 139
Creating Custom Date and Time Formats 139
Chapter 6: New Era of Networks Rules 143
Overview 143
New Era of Networks Rules Components 144
Application Groups 144
Message Types 144
Rules 145
Expressions, Arguments, Boolean, and Rules Operators 145

Subscriptions, Actions, and Properties 145
Permissions 146
New Era of Networks Rules APIs 146
Reloading Rules 146
Testing Rules 147
ruletest 147
Syntax 148
Parameters 148
Remarks 149
Example ruletest calls 149
NNRtrace 150
Syntax 150
Parameters 151
Chapter 7: Consistency Checker 153
Overview 153
Running the Consistency Checker 153
Appendix A: Character Sets 157
Appendix B: Data Types 171
Data Type Conversion 180
Converting to String Representation 180
Data Type Conversion Constraints 185
Not Applicable 185
String 185
Numeric 187
Binary 189
EBCDIC 191
IBM Types 193
Endian 2 Types 195
Endian 4 Types 196
Decimal International and Decimal US 198
Date and Time 199
Appendix C: Viewing New Era of Networks Formats in Eclipse 203
Exporting Formats using the Command Line 203
Exporting Formats using the Formatter GUI 204
Migrating the Export File into Eclipse 205

Appendix D: Processing Messages in the NEONMSG Domain 207
Appendix E: Notices 213
Trademarks and Service Marks 215
Index 217

Chapter 1
Introduction

Rules and Formatter Extension for IBM ® WebSphere Message Broker for Multiplatforms provides the flexibility, scalability, and configurability necessary for complete application integration.

Rules and Formatter Extension for IBM ® WebSphere Message Broker for Multiplatforms:

- Evaluates, maps, transforms, and routes messages using content-based rules
- Provides a powerful data content-based, source-target mechanism with dynamic format parsing and conversion capability.
- Provides easy to use graphical user interfaces (GUIs) or command line application program interfaces (APIs) for the components, NEONRules and NEONFormatter.

Messages that are defined using NEONFormatter can be used with WebSphere Message Broker message flows.You can continue to use NEONFormatter to create new definitions of message formats. These definitions are not held in the WebSphere Message Broker message repository, but in a separate database set up specifically for this purpose and controlled by NEONRules and NEONFormatter.

When you want to use these message formats in the broker, you do not assign and deploy them through the Control Center, but must ensure that the broker has access to the database in which the definitions exist.

Primitive message processing nodes provide processing equivalent to MQSeries Integrator Version 1.1, plus extra functionality. New Era of Networks nodes cannot be used unless the NEONRules and NEONFormatter component is installed.

Node	Description
NeonFormatter	Transforms a message from one predefined New Era of Networks format to another.
NEONMap	Provides mapping function in addition to transforms provided by NEONFormatter.
NEONTransform	Provides richer transformation operations in addition to mapping to extend NEONFormatter processing.
NeonRules	Interprets message processing rules to write a message to the next node.
NEONRulesEvaluation	Extends the function of the NeonRules node by providing an additional routing option.

In this guide, the term z/OS means a z/OS operating system that is using the UNIX System Services (USS) component.

About This Document

The ***System Management Guide*** is designed for those who are responsible for NEONRules and NEONFormatter component administration. The system administrator should have an overall understanding of the Rules and Formatter Extension for IBM ® WebSphere Message Broker for Multiplatforms product and how it works. It is assumed that the system administrator is responsible for Rules and Formatter Extension for IBM ® WebSphere Message Broker for Multiplatforms setup, configuration, and testing. The system administrator should be supported by a database administrator, who administers the databases interacting with Rules and Formatter Extension for IBM ® WebSphere Message Broker for Multiplatforms, and a network administrator, who ensures that network communications are set up to work with Rules and Formatter Extension for IBM ® WebSphere Message Broker for Multiplatforms.

This guide is organized into the following chapters:

Chapter 1: Introduction provides a brief product overview and lists supported code sets and locales and the documentation set and conventions.

Chapter 2: NEONRules and NEONFormatter Installation on z/OS provides procedures for installing the NEONRules and NEONFormatter database, placing the executable and catalogue files, and setting the environment variables.

Chapter 3: Configuration provides information about operational differences, customizing jobs for your site, and file encryption.

Chapter 4: Formats and Rules Database Migration describes the use of the NNFie and NNRie utilities to export formats and rules from an existing New Era of Networks database and to import the formats and rules into an Rules and Formatter Extension for IBM ® WebSphere Message Broker for Multiplatforms 5.6 database.

Chapter 5: NEONFormatter provides an overview of the NEONFormatter product and describes procedures for testing formats and customizing NEONFormatter.

Chapter 6: NEONRules provides an overview of NEONRules components and describes using the ruletest, NNRTrace, putdata and getdata utilities to test rules.

Chapter 7: Consistency Checker describes the use of the Consistency Checker to verify that format, rules, and permissions components are created and configured correctly.

Appendix A: Character Sets identifies the ASCII, EBCDIC, and Hexadecimal conversion support in the GUIs and APIs.

Appendix B: Data Types identifies the supported data types and provides information on converting between different data types and on defining custom data types.

Appendix C, *Viewing New Era of Networks Formats in Eclipse*, describes how to export formats for viewing in Eclipse.

Appendix D, *Processing Messages in the NEONMSG Domain*, describes how to process messages in the NEONMSG domain and gives examples.

Appendix E, *Notices*, contains trademark and service mark information.

Internationalization

Rules and Formatter Extension for IBM ® WebSphere Message Broker for Multiplatforms is internationalized and provides broad locale and code set support. The following table lists the supported code sets and locales.

Supported Code Sets and Locales

Code Set Identifier	Code Set Name	Alias	Description
1208	IBM-1208	UTF8 utf-8 ibm-1208 cp1208	Unicode Transformation 8 bit
37	ibm-37	IBM037 ibm-037 cpibm37 ebcdic-cp-us ebcdic-cp-ca ebcdic-cp-wt ebcdic-cp-nl csIBM037 cp37 ebcdic-cp-us cp037	EBCDIC - US
273	ibm-273	IBM273 csIBM273 ebcdic-de cp273 cpibm273	EBCDIC - Germany

Code Set Identifier	Code Set Name	Alias	Description
280	ibm-280	IBM280 ebcdic-cp-it csIBM280 cp280 cpibm280	EBCDIC - Italy
284	ibm-284	IBM284 ebcdic-cp-es csIBM284 cp284 cpibm284	EBCDIC - Spain
285	ibm-285	IBM285 ebcdic-cp-gb csIBM285 ebcdic-gb cp285 cpibm285	EBCDIC - UK, Ireland
297	ibm-297	IBM297 ebcdic-cp-fr csIBM297 cp297 cpibm297	EBCDIC - France
819	ibm-819	LATIN_1 iso-8859-1 ibm-819 cp819 latin1 8859-1 csisolatin1 iso-ir-100 cp367 ISO_8859-1:1987	Latin 1
850	ibm-850	IBM850 cp850 850 csPC850Multilingual	PCLatin 1

Code Set Identifier	Code Set Name	Alias	Description
930	ibm-930	cp930 cpibm930	EBCDIC - Japan Mixed
935	ibm-935	cp935 cpibm935	EBCDIC - China Mixed
937	ibm-937	cp937 cpibm937	EBCDIC - Taiwan Mixed
942	ibm-942	shift_jis78 sjis78 ibm-932	SJIS 78
943	ibm-943_P130-2000	ibm-943_VASCII_VSUB_VPUA ibm-943	n/a
943	ibm-943_P14A-2000	ibm-943_VSUB_VPUA Shift_JIS csWindows31J sjis cp943 cp932 ms_kanji csshiftjis windows-31j x-sjis 943	SJIS
949	ibm-949	KS_C_5601-1987 iso-ir-149 KS_C_5601-1989 csKSC56011987 ksc-5601 johab ks_x_1001:1992	KSC-5601-1992 - Korea
950	ibm-950	n/a	Taiwan Big 5

Code Set Identifier	Code Set Name	Alias	Description
964	ibm-964	EUC-TW ibm-eucTW cns11643	Extended UNIX Code - Taiwan
970	ibm-970	EUC-KR ibm-eucKR csEUCKR	Extended UNIX Code - Korea
1252	ibm-1252	ibm-1004 cp1004	Windows Latin 1 without Euro Update
1383	ibm-1383	EUC-CN ibm-eucCN	Extended UNIX Code - China
5348	ibm-5348	windows-1252 cp1252	Windows Latin 1 with Euro Update
33722	ibm-33722	EUC-JP ibm-eucJP eucjs X-EUC-JP	Extended UNIX Code Japan

Language	Country	Variant	Locale Name
Chinese	China	n/a	zh_CN
Chinese	Taiwan	n/a	zh_TW
English	UK	n/a	en_GB
English	US	n/a	en_US
French	France	n/a	fr_FR
French	France	EURO	fr_FR_Euro

Language	Country	Variant	Locale Name
German	Germany	n/a	de_DE
German	Germany	EURO	de_DE_EURO
Italian	Italy	n/a	it_IT
Italian	Italy	EURO	it_IT_EURO
Japanese	Japan	n/a	ja_JP
Korean	Korea	n/a	ko_KR
Portuguese	Brazil	n/a	pt_BZ
Spanish	Spain	n/a	es_ES
Spanish	Spain	EURO	es_ES_EURO

Documentation Set

The Rules and Formatter Extension for IBM ® WebSphere Message Broker for Multiplatforms documentation set includes:

- ***System Management Guide***
- ***NEONFormatter Programming Reference***
- ***NEONRules Programming Reference***
- ***Application Development Guide***
- Rules, Formatter, and Visual Tester online help
- Installation Readme

Document Conventions

The following document conventions are used in this guide.

Text	Convention	Example
code	courier	`<user ID> <password>`
command line display	courier	`The message successfully parsed.`
command line entry	courier bold	**`NNFAD-t`**
command line prompt	courier	`Enter the input file name:`
path	regular	ora/bin (UNIX) ora\bin (NT)
book names	bold, italic	***Installation Guide***
chapter and section names	italic	*NT Installation*

Chapter 2

NEONRules and NEONFormatter Installation on z/OS

This chapter describes how to install Rules and Formatter Extension for IBM ® WebSphere Message Broker for Multiplatforms on the z/OS platform. It includes the following information:

- *Installing the NEONRules and NEONFormatter Database*
- *Executable and Catalogue File Location*

Note:
For information on how to install Rules and Formatter Support on the Windows and UNIX platforms, see the ***WebSphere MQ Integrator Administration Guide***.

Installing the NEONRules and NEONFormatter Database

This procedure describes how to install Rules and Formatter Extension for IBM ® WebSphere Message Broker for Multiplatforms.

1. To create two PDS data sets, do the following:

   PDSs should be FB 80. If you use half-track blocking, 120 blocks and 10 directory blocks are sufficient.

a. Create the same HighLevelQualifiers for both PDS data sets.

b. Create a LowLevelQualifier of SBIPJCL for one PDS.

c. Create another LowLevelQualifier of SBIPSQL for the other PDS.

For example, if you create a HighLevelQualifier of IBM.NEWERA, the LowLevelQualifiers would be IBM.NEWERA.SBIPJCL and IBM.NEWERA.SBIPSQL.

2. In the installation directory /usr/lpp/wmqi/nnsy/install.sql/scripts/db250, copy NnsyConfigFile.MSTR to a new data set named NnsyConfigFile.in.

3. In the NnsyConfigFile.in, locate the following tokens. Tokens are enclosed in angle brackets "<>":

Token	Description
<auth-id>	SQLID database owner.
<database-name>	Database in which all table spaces are created.
<storage-group-name>	DB2 storage group in which the databases are created.
<vcat-dataset-name-prefix>	Storage group IntegratedCatalogFacility.
<volume-list>	Storage group volumes.
<db2smprl>	Library that contains DB2 sample program.
<db2mpdl>	Library in which the sample program is located.
<db2inhlq>	High-level DB2 library qualifiers.
<db2ssysn>	DB2 subsystem name.
<db2locan>	DB2 location name.
<pgmname>	DB2 sample program name.

Token	Description
<planname>	DB2 sample plan name.
<NNHLQ>	Data set High Level Qualifiers created in step 1 of this procedure.

4. Replace all tokens with parameters and enclose them in single quotation marks.
5. From the installation directory, run the script NNInstallDriver.sh.

Notes:

If you install the NnsyConfigFile.in file in a directory other than /usr/lpp/wmqi/nnsy/install.sql/scripts/db250, you must specify the -c parameter followed by the directory which contains the NnsyConfigFile.in file to ensure that the install script searches the proper directory.

If you installed the product under a directory other than /usr/lpp/wmqi/nnsy/install.sql/scripts/db250, you must specify the -i parameter followed by the high-level directory of your installation to ensure that the install script searches the proper directory.

6. To set the environment variables, choose one of the following:
   - Add each variable to your profile using the command syntax in the following table.
   - Enter the command syntax on the command line each time you start an executable.

New Era of Networks Product Environment Variables

Variable	Command Syntax
DSNAOINI	export DSNAOINI=<path to dsnaoini file>
LIBPATH	export LIBPATH=/usr/lpp/wmqi/nnsy/lib:$LIBPATH

Variable	Command Syntax
NNSY_CATALOGUES	export NNSY_CATALOGUES=/usr/lpp/wmqi/nnsy/NNSYCatalogues/<locale name>
NNSY_ROOT	export NNSY_ROOT=/usr/lpp/wmqi/nnsy
PATH	export PATH=/usr/lpp/wmqi/nnsy/bin:$PATH

WARNING!

When you are making changes to the paths, type the following to prevent overwriting all data currently in the path:

For LIB_PATH:

export LIBPATH=$NNSY_ROOT/lib:$LIBPATH

For PATH:

export PATH=$NNSY_ROOT/bin:$PATH

7. From the JCL PDS created in step 1 of this procedure, run the following jobs:
   - CRSTOGRP
   - CRDB
   - CRTSPACE
   - CRTABS
   - GRANTS
   - META

   If these jobs run successfully, the NEONRules and NEONFormatter database is installed.

Executable and Catalogue File Location

To ensure the best performance, the executable and catalogue files of the New Era of Networks product should be placed on your local HFS drive.

If you place the product on a NFS drive, you must also mount these drives as binary. Failure to mount NFS drives as binary may result in unexpected product behavior as NFS attempts to translate the data.

Chapter 3
Configuration

This chapter describes the procedures for configuring Rules and Formatter Extension for IBM ® WebSphere Message Broker for Multiplatforms components. It includes the following information:

- *Shared Libraries*
- *Runtime Configuration*
- *Encrypting the Configuration File*
- *NEONRules and NEONFormatter Permissions*
- *Message Reporting*

Shared Libraries

Shared libraries are archived collections of object files. They are installed during the component installation process bin directory. The following is the path to the libraries that must be linked with the application object files:

- On Windows, the shared libraries and DLLs are in {installroot}\bin. The libraries needed to compile custom code are in {installroot}\lib. Libraries are referred to as Dynamic Link Libraries. You can identify shared libraries as files with a .dll extension.
- On UNIX, the libraries are in {installroot}/bin. Depending on the platform, you can identify shared libraries as files with an .so or .sl extension.
- On z/OS, DLL library files are in {installroot}/lib. You can identify shared libraries as files with a .dll or .a extension.

They provide the following benefits:

- You can easily perform code changes. Because binaries are no longer statically linked to other files during creation, the size of the binary at compilation is reduced.
- You are no longer required to recompile binaries for each software change you make. For example, when you receive code enhancements, you only need to replace the libraries because the code is brought in at run time.
- You can create and add User Exits to NEONFormatter without the need to link the executables again.

As a System Administrator, you need to know the following about the shared libraries.

- Shared libraries are stored in a specific location during installation. Do not move the libraries from this location. The executables search for them in a specific directory or folder. If you move or delete the libraries, the executables are rendered useless.
- Shared libraries in WebSphere Message Broker may not be compatible with prior releases. If library names and APIs are a different version than those used in the shared libraries, the shared libraries will not work.
- For Solaris, you must set the LD_LIBRARY_PATH environment variable to point to the NEONRules and NEONFormatter libraries.
- For z/OS, you must set the LIBPATH environment variable to point to the NEONRules and NEONFormatter libraries. For more information, see *NEONRules and NEONFormatter Installation on z/OS* on page 11.
- For more information on the public interface files that contain the library signatures against which the API function calls are built, see the ***NEONRules Programming Reference*** and the ***NEONFormatter Programming Reference***.

Library Paths

You can set and query library paths.

The folder or directory that contains the Rules and Formatter Extension for IBM ® WebSphere Message Broker for Multiplatforms shared libraries is contained in the environment variable information.

To set library paths:

Use the following environment variables.

Platform	Environment Variable
Windows	PATH
Linux	LD_LIBRARY_PATH
Solaris	LD_LIBRARY_PATH
HP-UX	SHLIB_PATH
AIX	LIBPATH
z/OS	LIBPATH

To query library paths on Windows:

Type the following and press ENTER:

```
set
```

Runtime Configuration

Runtime configuration is specified using the nnsyreg.dat file. You can specify the behavior for your databases, transports, and executables. In a typical file, configuration parameters appear as key-value pairs in a series of designated groups. By defining the key-value pairs for each group, you provide Rules and Formatter Extension for IBM ® WebSphere Message Broker for

Multiplatforms with runtime configuration information. At system startup, executables search for this information and use these specifications to operate Rules and Formatter Extension for IBM ® WebSphere Message Broker for Multiplatforms.

Sample nnsyreg.dat File

The following sample illustrates the hierarchical structure and information groups contained in the file.

```
#Session.new_format_demo
#NNOT_SHARED_LIBRARY=dvdb41sql
#NNOT_FACTORY_FUNCTION=NNSesMS6Factory
#NN_SES_SERVER=cesql_65
#NN_SES_USER_ID=NQEDB17
#NN_SES_PASSWORD=NQEDB17
#NN_SES_DB_NAME=NQEDB17

#Session.new_format_demo
#NNOT_SHARED_LIBRARY=dvdb41ora
#NNOT_FACTORY_FUNCTION=NNSesOra8Factory
#NN_SES_SERVER=ceorau_805
#NN_SES_USER_ID=NQEDB17
#NN_SES_PASSWORD=NQEDB17
#NN_SES_DB_NAME=NQEDB17

#Session.new_format_demo
#NNOT_SHARED_LIBRARY=dvdb41syb
#NNOT_FACTORY_FUNCTION=NNSesSybCTFactory
#NN_SES_SERVER=cesybu_1192
#NN_SES_USER_ID=NQEDB17
#NN_SES_PASSWORD=NQEDB17
#NN_SES_DB_NAME=NQEDB17

#Session.new_format_demo
#NNOT_SHARED_LIBRARY=dvdb41db2
#NNOT_FACTORY_FUNCTION=NNSesDB2Factory
#NN_SES_SERVER=NQEDB17
#NN_SES_USER_ID=nnsyrf
#NN_SES_PASSWORD=nnsyrf
```

Locating the Example Folder

An example nnsyreg.dat file is provided with the installation package of any New Era of Networks product. The example file is preset with the default parameters that are required for Rules and Formatter Extension for IBM ® WebSphere Message Broker for Multiplatforms and that load during product runtime.

To locate the nnsyreg.dat example file supplied with Rules and Formatter Extension for IBM ® WebSphere Message Broker for Multiplatforms:

- For Windows systems, go to the C:\Program Files\IBM\<WebSphere Message Broker installation directory>
- For HP-UX and Solaris, go to the /opt/<Websphere Message Broker installation directory>
- For AIX and z/OS, go to the /usr/opt/<Websphere Message Broker installation directory>

Maintaining the nnsyreg.dat File

If you have installed multiple New Era of Networks products, you will find that you have several copies of this file located throughout the nnsy sub-directories. To simplify the configuration and system search processes, it is recommended that you consolidate all of your configuration information into one single nnsyreg.dat copy and locate and maintain this file in the nnsy root directory. However, regardless of directory, Rules and Formatter Extension for IBM ® WebSphere Message Broker for Multiplatforms is capable of locating nnsyreg.dat configuration information at runtime by using a prioritized search sequence operation. For more information on the system search process, see *Searching the System for Configuration Information* on page 22.

An exception to maintaining a single nnsyreg.dat file would be when creating a separate test environment. In this case, you may want to place a copy of the nnsyreg.dat file in a separate test directory. After testing is complete, you can copy the nnsyreg.dat file or append any portion of the file to the original nnsyreg.dat file located in the runtime directory.

To maintain a single nnsyreg.dat file with single product installation:

This procedure explains how to maintain a single nnsyreg.dat file after you have installed only a single New Era of Networks product.

1. Locate the nnsyreg.dat file in the examples directory. For path information, see *Locating the Example Folder* on page 21.
2. Move the file to the product root directory.
3. Define all configuration parameters in this copy of the nnsyreg.dat.

To maintain multiple nnsyreg.dat files:

The following scenarios describe instances when you may want to maintain multiple copies of nnsyreg.dat:

- To separate test operations from your production environment, maintain a separate copy of the configuration file in your testing environment.
- To meet product requirements for a separate nnsyreg.dat file as specified in the product documentation.

Use this procedure to maintain multiple copies of the nnsyreg.dat file.

1. Locate all supplementary copies of the file in the product examples subdirectories.
2. To avoid confusion, rename each nnsyreg.dat file to reflect its purpose in the configuration process and enter the specific nnsyreg.dat file name for that product when starting up the engine for each product.
3. To avoid unnecessary configuration problems, become familiar with the system search sequence described in the following section.

Searching the System for Configuration Information

During runtime, Rules and Formatter Extension for IBM ® WebSphere Message Broker for Multiplatforms searches the system for configuration information using a prioritized search sequence. If you maintain more than one copy of the nnsyreg.dat file in the nnsy subdirectory, it is important to

become familiar with this sequence so that you avoid the incorrect loading of configuration parameters during runtime.

The following list describes the loading process:

- New Era of Networks product executables are initialized and configuration specifiers are added to a configuration search checklist.
- Rules and Formatter Extension for IBM ® WebSphere Message Broker for Multiplatforms searches for configuration information by proceeding through an ordered series of steps and loading the configuration data as it is located.
- If a duplicate parameter is found in multiple configuration files, the value with the highest priority in the search order is loaded and the others are ignored.
- After configuration information for a parameter is located and loaded, the Rules and Formatter Extension for IBM ® WebSphere Message Broker for Multiplatforms stops searching for that parameter.

The following table describes the automatic search order sequence.

Search Order Priority	Description
1	Searches for the parameter values that you specify using the command line.
2	Searches for the appname key values specified in the Windows registry for that application.
3	Searches for environment variables that are used for locating additional configuration data.
4	Searches for a file named nnsyreg.dat in the following directories: - current working directory - directory specified by the environment variable NN_CONFIG_FILE_PATH - directory specified by the environment variable NNSY_ROOT

Configuring Runtime Parameters

To ensure proper start-up and successful message processing, Rules and Formatter Extension for IBM ® WebSphere Message Broker for Multiplatforms component executables require that you define configuration parameters within the nnsyreg.dat file. For Rules and Formatter Extension for IBM ® WebSphere Message Broker for Multiplatforms and its components to operate properly, the configuration information must conform to the specifications outlined in this chapter.

As described earlier in this chapter, configuration parameters appear as key-value pairs contained in a series of designated groups. Each of these groups is called a configuration specifier. However, throughout the remainder of this procedure, the configuration specifier is referred to as a session entry. The session entries provide the runtime configuration instructions, eliminating the need for application code changes.

The following example illustrates a session entry with its associated <key>=<value> pairs.

```
Session.<Session Name>
     NNOT_SHARED_LIBRARY=<shared library filename>
     NNOT_FACTORY_FUNCTION=<factory function>
     NN_SES_SERVER=<my database server>
     NN_SES_DB_NAME=<my database name>
     NN_SES_USER_ID=<my user Id>
     NN_SES_PASSWORD=<my password>
#<comments>
```

Each session entry allows for optional and mandatory key value pairs. For an executable to start up properly, you must complete all mandatory key value pairs. When the product executables retrieve data from the configuration file, Rules and Formatter Extension for IBM ® WebSphere Message Broker for Multiplatforms verifies the names and values of the configuration parameter definitions. Any default parameters required for the product come preset in the example nnsyreg.dat file that is supplied with the product installation package.

For more information on the internal structure of the nnsyreg.dat file, see *Runtime Configuration* on page 19.

Guidelines for Creating Session Entries in nnsyreg.dat

The following guidelines can assist you in creating session entries or editing existing entries.

- Place the configuration specifier flush against the left margin of the text file and separate the configuration specifier and the session name with a period, as in the following example:

  ```
  session.<session entry name>
  ```

- Enter a descriptive session name, using the following criteria:
  - If the configuration is a program-specific setting, use the following model:.

    ```
    <component name>.<group name>.<identifier>
    ```

  - If the configuration is a universal setting, such as a database session, use the following model:

    ```
    <group name>.<identifier>
    ```

- Indent all key-value pairs.
- Separate key-value pairs with an equals symbol (=).
- To denote comments, use the symbol (#) as the first character on the line.
- Do not use spaces or extra characters at the end of a line.
- For key-value pairs that allow multiple values, use a comma-separated list with no spaces between values and commas. Delimit the list with parentheses.
- Do not create duplicate session entries within a single nnsyreg.dat file or throughout multiple file copies. As associated applications search the parameters in the configuration file, they use only the first instance of a valid value and ignore the remaining entries.

Entering Session Information

As part of the system configuration process, you must specify a session entry for each executable that is used during runtime in the nnsyreg.dat file. For

Rules and Formatter Extension for IBM ® WebSphere Message Broker for Multiplatforms, you must specify:

- Database session information
- Exporting and importing utility information
- Transport session information
- New Era of Networks node support information for the WebSphere Message Broker broker
- New Era of Networks Configuration Manager support information
- Runtime logging levels

The following generic format can serve as a model for entering session parameters. Use this model when configuring session information in the nnsyreg.dat file. This model contains an ordered format that is necessary to the configuration search process. As soon as a key is located, the value associated with the key is returned, and the search stops.

```
Session.<Session Name>
     NNOT_SHARED_LIBRARY=<shared library filename>
     NNOT_FACTORY_FUNCTION=<factory function>
     NN_SES_SERVER=<my database server>
     NN_SES_DB_NAME=<my database name>
     NN_SES_USER_ID=<my user Id>
     NN_SES_PASSWORD=<my password>
#<comments>
```

Specifying Database Session Information

To connect Rules and Formatter Extension for IBM ® WebSphere Message Broker for Multiplatforms to the NEONRules and the NEONFormatter databases, specify the database session information in the nnsyreg.dat file. You can define database sessions for the following DBMS types:

- Microsoft SQL server
- Oracle
- Sybase
- DB2

- XA

Note:
Using an XA database session supplied by the broker enables XA transaction support via New Era of Networks plug-ins.

To specify database session information:

1. Locate the nnsyreg.dat file.

   For more information, see *Locating the Example Folder* on page 21.

2. Using the following generic model, type a descriptive name for database session entry.

   Example:

   Session.SomeDatabaseSession

3. Populate the NNOT_SHARED_LIBRARY and NNOT_FACTORY_FUNCTION fields using the following table.

Database Type	NNOT_SHARED_LIBRARY	NNOT_FACTORY_FUNCTION
Microsoft SQL server	dvdb41sql	NNSesMS6Factory
Oracle	dvdb41ora	NNSesOra8Factory
Sybase CTLIB	dvdb41syb	NNSesSybCTFactory
DB2	dvdb41db2	NNSesDB2Factory
XA Windows	dvdb41msodbc	NNSesODBCFactory
XA UNIX	dvdb41mqi	NNSesMQIDBFactory

Note:
XA: When you use the above combination of shared library and factory function for XA, the New Era of Networks plug-ins request an

XA session from the broker. If an XA session is not available, a non-XA session is used.

4. Populate the following fields with your site-specific information:

   - NN_SES_SERVER

     Connects to this database server.

   - NN_SES_DB_NAME

     Connects to this database.

   - NN_SES_USER_ID

     Connects to the database as this user.

   - NN_SES_PASSWORD

     Connects to the database using this password.

Note:
When accessing the z/OS DB2 database from the mainframe, do not include values for the NN_SES_USER_ID or NN_SES_PASSWORD fields.

DB2 Example on Windows and UNIX

```
Session.DB2_DATABASE_SESSION_SAMPLE
    NNOT_SHARED_LIBRARY=dvdb41db2
    NNOT_FACTORY_FUNCTION=NNSesDB2Factory
    NN_SES_SERVER=db2serv
    NN_SES_USER_ID=<my user Id>
    NN_SES_PASSWORD=<my password>
    NN_SES_DB_NAME=NN52QA1
```

DB2 Example on z/OS

```
Session.DB2_DATABASE_SESSION_SAMPLE
    NNOT_SHARED_LIBRARY=dvdb41db2
    NNOT_FACTORY_FUNCTION=NNSesDB2Factory
    NN_SES_SERVER=db2serv
    NN_SES_DB_NAME=NN52QA1
```

Specifying Data Exporting and Importing Session Information

Before you can import and export rules and formats from your existing database to your target database using NEONFormatter (NNFie) and NEONRules (NNRie) import and export utilities, you must set up session entries for NNFie and NNRie in the nnsyreg.dat file.

To specify NNFie and NNRie session information:

1. Locate the nnsyreg.dat file.

   For more information, see *Locating the Example Folder* on page 21.

2. Using the generic model presented earlier, type the following session names.

   - For NNFie, type:

     ```
     Session.nnfie
     ```

   - For NNRie, type:

     ```
     Session.nnrmie
     ```

3. Populate the NNOT_SHARED_LIBRARY and NNOT_FACTORY_FUNCTION fields using the following table.

Database Type	NNOT_SHARED_LIBRARY	NNOT_FACTORY_FUNCTION
Microsoft SQL server	dvdb41sql	NNSesMS6Factory
Oracle	dvdb41ora	NNSesOra8Factory
Sybase CTLIB	dvdb41syb	NNSesSybCTFactory
DB2	dvdb41db2	NNSesDB2Factory

4. Populate the following fields with your own site-specific information:
   - NN_SES_SERVER

     Connects to this database server.
   - NN_SES_DB_NAME

     Connects to this database.
   - NN_SES_USER_ID

     Connects to the database as this user.
   - NN_SES_PASSWORD

     Connects to the database using this password.

     Note:
     When accessing the z/OS DB2 database from the mainframe, z/OS, do not include values for the NN_SES_USER_ID or NN_SES_PASSWORD fields.

NNFie Example on Windows and UNIX

```
Session.nnfie
      NNOT_SHARED_LIBRARY=dvdb41db2
      NNOT_FACTORY_FUNCTION=NNSesDB2Factory
      NN_SES_SERVER=<my server name>
      NN_SES_DB_NAME=<my database name>
      NN_SES_USER_ID=<my user Id>
      NN_SES_PASSWORD=<my password>
```

NNFie Example on z/OS

```
Session.nnfie
      NNOT_SHARED_LIBRARY=dvdb41db2
      NNOT_FACTORY_FUNCTION=NNSesDB2Factory
      NN_SES_SERVER=<my server name>
      NN_SES_DB_NAME=<my database name>
```

NNRie Example on Windows and UNIX

```
Session.nnrmie
```

```
NNOT_SHARED_LIBRARY=dvdb41db2
NNOT_FACTORY_FUNCTION=NNSesDB2Factory
NN_SES_SERVER=<my server name>
NN_SES_DB_NAME=<my database name>
NN_SES_USER_ID=<my user Id>
NN_SES_PASSWORD=<my password>
```

NNRie Example on z/OS

```
Session.nnrmie
      NNOT_SHARED_LIBRARY=dvdb41db2
      NNOT_FACTORY_FUNCTION= NNSesDB2Factory
      NN_SES_SERVER=<my server name>
      NN_SES_DB_NAME=<my database name>
```

Specifying Transport Session Information

You must specify transport session information in the nnsyreg.dat file to enable the Visual Tester Load feature in the NEONFormatter GUI.

To specify transport session information:

1. Locate the nnsyreg.dat file.

   For more information, see *Locating the Example Folder* on page 21.

2. Create an entry for MQSeries Session parameters.

   Example:

   ```
   Session.MQSeriesSession
        NNOT_SHARED_LIBRARY=dv41mqs
        NNOT_FACTORY_FUNCTION=NNMQSSessionFactory
        NNMQS_SES_OPEN_QMGR=<my queue manager name>
   ```

3. To set default parameters for MQSeries queuing, specify the following using the model in step 2:

   a. Set NNOT_SHARED_LIBRARY to dv41mqs.

   b. Set NNOT_FACTORY_FUNCTION to NNMQSSessionFactory.

   c. Set NNMQS_SES_OPEN_QMGR to your queue manager.

4. Create an entry for Transport parameters.

Example:

```
Transport.<my inbound queue name>
     NNOT_SHARED_LIBRARY=dv41mqs
     NNOT_FACTORY_FUNCTION=NNMQSQueueFactory
     NNOT_TIL_OPEN_TSI=<my inbound queue name>
     NNOT_TIL_OPEN_SESSION_ID=MQSeriesSession
```

5. To set default parameters for your inbound queue, specify the following:
   a. Set NNOT_FACTORY_FUNCTION to NNMQSQueueFactory.
   b. Set NNOT_SHARED_LIBRARY to dv41mqs.
   c. Set NNOT_TIL_OPEN_TSI to your inbound queue name.
   d. Set NNOT_TIL_OPEN_SESSION_ID to MQSeriesSession.

Specifying Node Support Information

You must specify New Era of Networks node support information for the Broker in the nnsyreg.dat file.

To specify node support session information:

1. Locate the nnsyreg.dat file.

   For more information, see *Locating the Example Folder* on page 21.

2. Using the generic model presented earlier, type the following session name:

   ```
   Session.MQSI_PLUGIN
   ```

3. Populate the NNOT_SHARED_LIBRARY and NNOT_FACTORY_FUNCTION fields using the following table.

Database Type	NNOT_SHARED_LIBRARY	NNOT_FACTORY_FUNCTION
Microsoft SQL server	dvdb41sql	NNSesMS6Factory

Database Type	NNOT_SHARED_LIBRARY	NNOT_FACTORY_FUNCTION
Oracle	dvdb41ora	NNSesOra8Factory
Sybase CTLIB	dvdb41syb	NNSesSybCTFactory
DB2	dvdb41db2	NNSesDB2Factory

4. Populate the following fields with your site-specific information:
   - NN_SES_SERVER

     Connects to this database server.
   - NN_SES_DB_NAME

     Connects to this database.
   - NN_SES_USER_ID

     Connects to the database as this user.
   - NN_SES_PASSWORD

     Connects to the database using this password.

     Note:
     When accessing the z/OS DB2 database from the mainframe, do not include values for the NN_SES_USER_ID or NN_SES_PASSWORD fields.

MQSI_PLUGIN Example on Windows and UNIX

```
Session.MQSI_PLUGIN
      NNOT_SHARED_LIBRARY=dvdb41db2
      NNOT_FACTORY_FUNCTION=NNSesDB2Factory
      NN_SES_SERVER=<my server name>
      NN_SES_DB_NAME=<my database name>
      NN_SES_USER_ID=<my user Id>
      NN_SES_PASSWORD=<my password>
```

MQSI_PLUGIN Example on z/OS

```
Session.MQSI_PLUGIN
      NNOT_SHARED_LIBRARY=dvdb41db2
      NNOT_FACTORY_FUNCTION= NNSesDB2Factory
      NN_SES_SERVER=<my server name>
      NN_SES_DB_NAME=<my database name>
```

Specifying New Era of Networks Configuration Manager Support Information

You must specify the support information for the New Era of Networks Configuration Manager in the nnsyreg.dat file.

To specify New Era of Networks Configuration Manager support information:

1. Locate the nnsyreg.dat file.

   For more information, see *Locating the Example Folder* on page 21.

2. Using the generic model presented earlier, type the following session name:

   ```
   Session.MQSI_CONFIG
   ```

3. Populate the NNOT_SHARED_LIBRARY and NNOT_FACTORY_FUNCTION fields using the following table.

Database Type	NNOT_SHARED_LIBRARY	NNOT_FACTORY_FUNCTION
Microsoft SQL server	dvdb41sql	NNSesMS6Factory
Oracle	dvdb41ora	NNSesOra8Factory
Sybase CTLIB	dvdb41syb	NNSesSybCTFactory
DB2	dvdb41db2	NNSesDB2Factory

4. Populate the following fields with your site-specific information:

   - NN_SES_SERVER

     Connects to this database server.

   - NN_SES_DB_NAME

     Connects to this database.

   - NN_SES_USER_ID

     Connects to the database as this user.

   - NN_SES_PASSWORD

     Connects to the database using this password.

> **Note:**
> When accessing the z/OS DB2 database from the mainframe, do not include values for the NN_SES_USER_ID or NN_SES_PASSWORD fields.

MQSI_CONFIG Example on Windows and UNIX

```
Session.MQSI_PLUGIN
      NNOT_SHARED_LIBRARY=dvdb41db2
      NNOT_FACTORY_FUNCTION=NNSesDB2Factory
      NN_SES_SERVER=<my server name>
      NN_SES_DB_NAME=<my database name>
      NN_SES_USER_ID=<my user Id>
      NN_SES_PASSWORD=<my password>
```

MQSI_CONFIG Example on z/OS

```
Session.MQSI_PLUGIN
      NNOT_SHARED_LIBRARY=dvdb41db2
      NNOT_FACTORY_FUNCTION= NNSesDB2Factory
      NN_SES_SERVER=<my server name>
      NN_SES_DB_NAME=<my database name>
```

Specifying Logging Levels

You can control the level of logging in the NNSYMessageLog.nml file. This file is used to track messages reported by the New Era of Networks supported nodes.

To specify logging levels:

1. Locate the nnsyreg.dat file.

   For more information, see *Locating the Example Folder* on page 21.

2. To set logging levels, type the following in the nnsyreg.dat file.:

```
Broker.Logging
       LogLevel=0
```

Logging Levels

Value	Description
0	Logs informational messages, warnings, errors, and fatal errors.
1	Logs warnings, errors, and fatal errors.
2	Logs errors and fatal errors.
3	Logs fatal errors only.

Copying the Messages Properties File

To deploy a message flow that includes any New Era of Networks node (NeonFormatter, NEONMap, NeonRules, NEONRulesEvaluation, or NEONTransform) to a broker, you must copy the New Era of Networks message properties file from the system on which you installed your broker to the system on which you are using the Control Center. This ensures that exceptions thrown by New Era of Networks nodes at deploy time are displayed in the Control Center log.

The properties file on the broker system is located in the directory Install_dir/messages. The file is named NEONMIF20.properties on UNIX and

MQSIv2NeonPlugins.properties on Windows. Transfer the file in ASCII mode to the directory install_dir/tool on your Control Center system.

Encrypting the Configuration File

To protect confidential information, such as user names and passwords, you can encrypt and decrypt the configuration file, running the NNCrypt utility against your configuration file.

Unlike previous releases, encrypted files no longer use a .crypt extension or take priority over decrypted files during configuration information searches.

Note:
On MQSeries Integrator for OS/390 version 1.1.1, NNCrypt was used for export files.

To encrypt your configuration file:

Type the following and press Enter:

```
NNCrypt (-encrypt ) -file <fileName>
```

To decrypt your configuration file:

Type the following and press Enter:

```
NNCrypt (-decrypt) -file <fileName>
```

Encryption Parameters

Parameter	Description
-encrypt	Encrypts the specified file. Returns an error if the file is already encrypted.
-decrypt	Decrypts the specified file. Returns an error if the file is already decrypted.

Parameter	Description
-file	Specifies the name of the file to encrypt or decrypt.

NEONRules and NEONFormatter Permissions

The permission utilities allows you to secure user access to rules and formats in NEONRules and NEONFormatter. With the use of a permission utility, PermUtil, you can assign permission to those users who have the authority to create or modify rules and formats.

You can assign permissions through either the GUIs or Management APIs.

The following types of permissions are available in NEONRules and NEONFormatter:

- Owner permission

  By default, the user who creates an object is the owner of the object. The owner is granted both Update and Read permission to the object and can grant or revoke Owner and Update permissions to other users. However, since only one user can have owner permission, by granting owner permission to another user, the current owner is revoking his own owner permission.

  When ownership is transferred, the permissions are transferred to the new owner and previous permissions are overwritten. The new owner is given the same permissions as the previous owner.

- Read permission

  By default, PUBLIC is given Read permission. Users with Read permission are allowed to view an object and all of its attributes. PUBLIC cannot be granted Owner permission for an object, and cannot be granted Update permission.

- Update permission

  Update permission must be granted by the owner of an object. The user with Update permission can modify an object name and any of its attributes, however, this user cannot delete an object.

  To delete an object, the user must have both Owner and Update permissions. An owner who revokes her own Update permission can no longer modify an object and cannot delete an object.

- Rules permissions

  In NEONRules, permissions only apply to managing rule and subscription contents, not rule evaluation. Rule and Subscription permissions assign ownership to complete rules or 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 all associated parts, including its expression (arguments) and subscriptions.

  A subscription is uniquely defined by its application group name, message type, and subscription name. A complete subscription includes all associated parts, including its actions and options.

- Format permissions

  NNFie treats permissions associated with a NEONFormatter component as a separate entity.

Listing and Changing Permissions

To list and change permissions in NEONRules or NEONFormatter, use the permission utility, PermUtil.

Before using PermUtil, you must:

- Edit the configuration file to contain a rules session entry so the utility can connect to the NEONRules database. For more information, see *Specifying Database Session Information* on page 26.
- Create database users before you grant permissions.

To run PermUtil on NEONRules and NEONFormatter:

1. At the command line, type:

```
PermUtil
```

   The following message appears:

```
*********************************
*  Application Group Selection  *
*********************************
Rules: R or r
Formatter: F or f
Any other key to exit application.
```

2. Type one of the following variables to select the component permissions to be queried:

   - For Rules, type R or r, press ENTER and continue with step 3.

     A list of NEONRules components is displayed.

   - For Formatter, type F or f, press ENTER.

     A list of NEONFormatter components is displayed.

3. Type the number corresponding to the component to which you want to change or verify permission.

   A list of functions available for the selected component is displayed.

4. Type the number of the function you want to perform.

   Rules example:

```
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
```

   Formatter example:

```
Function to Perform:
1  List Components of the selected Type Owned by a
```

```
    Certain User
2   Change All Components of the Selected Type owned by
    User A to be Owned By User B
```

To list rule or format ownership of components by user name:

Type 1, enter the id of the User Name for Owner and press ENTER. For Oracle, use all caps.

PermUtil displays a list of components owned by the specified user.

To change ownership of NEONRules or NEONFormatter components:

1. At the prompt, type 2 and press ENTER.
2. Type the user name for the Current Owner and press ENTER. For Oracle, use all caps.
3. Type the user name for the New Owner and press ENTER. For Oracle, use all caps.

   The ownership of specified components is changed.

To list the Rules subscription ownership by user name:

1. At the prompt, type 3 and press ENTER:

   ```
   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.
2. To change subscription ownership, type 4 and press ENTER:

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

   The owner of the subscription is changed.

Examples

The following examples demonstrate how to use PermUtil.

Listing all rules owned by R40NNSY

```
1
R40NNSY
```

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

```
1
R40TEST

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

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

```
1
R40USR2

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

Changing all rules owned by R40NNSY to R40USR2

```
2
R40NNSY
R40USR2
```

Listing all subscriptions owned by REL40USER

```
2
User Name for Owner of Subscriptions
>REL40USER

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

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

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

Changing all subscriptions owned by REL40USER1 to REL40USER2

```
2
User Name for Current Owner of Subscriptions
>REL40USER1
User Name for New Owner of Subscriptions
>REL40USER2

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

Error Conditions

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

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
```

Message Reporting

Log files and NNSYMessageLog.nml are used to track messages reported by New Era of Networks process applications.

You can control the volume of messages reported to the log by setting the LogLevel parameter in the configuration file. For more information on setting log levels, see *Specifying Logging Levels* on page 36.

To review the NNSYMessageLog.nml file:

Go to the current working directory.

Windows

<message broker installation directory>\bin

All Other Platforms

Current working directory

NNSYmessageLog.nml

The detail of the information in NNSYmessageLog.nml is reported using a standard message format:

```
CCYYMMDDhhmmssslllZ|<process name>|<process ID>|<thread ID>|
<correlation ID>|<class>|<code>|<severity>|<file>|<line>|
```

Chapter 4
Formats and Rules Database Migration

This chapter explains how to migrate data from an existing Rules and Formatter Extension for IBM ® WebSphere Message Broker for Multiplatforms database to a target database. It includes the following information:

- *Process Overview*
- *Preparing Formats and Rules for Upgrade*
- *Upgrading Formats*
- *Upgrading Rules*
- *Cross-Platform Migration*

The migration procedures described in this chapter assume that the database migration occurs within the same platform. To migrate data from one platform to another, see *Cross-Platform Migration* on page 107.

Terminology

Existing database

Contains the rules and formats for the currently installed release of NEONRules and NEONFormatter that are exported.

Target database

Contains the imported rules and formats for the newly installed release of NEONRules and NEONFormatter.

Process Overview

The following scenarios describe when data migration is needed:

- When upgrading to a new release of Rules and Formatter Extension for IBM ® WebSphere Message Broker for Multiplatforms, you must migrate data from your existing database to a new target database.
- When running multiple instances of Rules and Formatter Extension for IBM ® WebSphere Message Broker for Multiplatforms and sharing format and rule components between databases, you migrate data components from one database to another.
- When troubleshooting database or processing problems, you create data migration export and import files.

For Windows, Solaris, Linux, AIX, and HP-UX, the migration process comprises the following procedures:

- Setting up the existing database environment for export.
- Checking the consistency of your existing database.
- For upgrades from a New Era of Networks Rules, Formatter and previous versions of Messaging database, running the format clean-up script, checking the database consistency, and cleaning the database.
- Running NNFie to create an export file for formats.
- Running NNRie to create an export file for rules.
- Setting up the target database environment for import.
- Running NNFie to import the exported formats file into the target database.
- Running NNRie to import the exported rules file into the target database.
- Checking the consistency of the target database.

- Recompiling your applications, including user exits because the .h (include) files and .dll files in the upgraded release differ from previous releases.

On z/OS, the migration process comprises the following procedures:

- Creating a backup of your existing database.
- Verifying the consistency of your existing database by running the Consistency Checker which corresponds your existing database release.

  For more information, see *Consistency Checker* on page 153.
- Running NNFie to export formats to the export file.
- Running NNRie to export rules to the export file.
- Moving the exported files from the z/OS operating system to the USS system.
- Setting up the target database environment for import.
- Running NNFie to import the exported formats file into the target database.
- Running NNRie to import the exported rules file into the target database.
- Verifying the consistency of your target database by running the Consistency Checker which corresponds your database upgrade.

  For more information, see *Consistency Checker* on page 153.
- Recompiling your applications, including user exits. The .h (include) files and .a files in the upgraded release differ from previous releases.

Preparing Formats and Rules for Upgrade

Before upgrading formats and rules from an existing database to your target database, verify the following:

- Existing database contains valid rules and formats.
- Rules and Formatter Extension for IBM ® WebSphere Message Broker for Multiplatforms is installed.
- Target database is instantiated, contains no data, and allows sufficient space for rules and formats.

Note:
To upgrade to a target database that already contains data, see *NNFie* on page 50 and *NNRie* on page 88. These sections provide guidelines for resolving or preventing data conflicts.

- Sufficient disk space exists to hold the output file. This file can be re-directed to anywhere the system supports.

WARNING!
When upgrading formats and rules to a target database, you must have two separate database instances, your existing database and one for your target database. Do not attempt to rebuild your current database after exporting all the data.

Determining Compatibility

To maintain compatibility when running import/export utilities, do the following:

- Identify your product and component database in the *Compatibility Matrix for Data Migration* table.

- Identify the appropriate NNFie or NNRie executable to import or export formats or rules.
- Identify the appropriate Consistency Checker executable to test data consistency.
- Locate the executables in the product directory.

Use the following table to assist you in working through the procedures described in this chapter.

Compatibility Matrix for Data Migration

Product	Component Databases	Export / Import Utilities
WebSphere Message Broker v5.0	New Era of Networks Formatter 4.0 New Era of Networks Rules 4.0	NNFie4.0 NNRie4.0 consistency Checker 5.6
WebSphere MQ Integrator Version 2.1	NEONFormatter 5.6 NEONRules 5.6	NNFie5.6 NNRie5.6 Consistency Checker 5.6

Upgrading Formats

As part of the Rules and Formatter Extension for IBM ® WebSphere Message Broker for Multiplatforms installation, the data upgrade process uses NNFie to move formats from an existing database to the target database.

NNFie

Use the NEONFormatter Import and Export utility, NNFie, to export formats from an existing database and import them to a target database. During export, NNFie creates a flat file and reads the same file structure for import.

The NNFie utility allows you to export a complete set of formats and their associated components or select only specific formats and components for export.

WARNING!
You cannot simultaneously run NNFie and NNRie. Database problems may result.

Importing and Exporting Formats

To export or import formats, you must use the NNFie release that corresponds to your current New Era of Networks Rules and Formatter database. For more information on determining the correct release, see *Determining Compatibility* on page 48.

To export and import formats using the NEONFormatter GUI, see the NEONFormatter online help.

Note:
If your DB2 database resides on z/OS, you must import your data on the same platform from which you export it.

Export Phase

During the export phase, you can choose between exporting a group of specific formats or the entire set.

When NNFie creates an export file, it places a header with source and date information at the beginning of the file. You can also add comments to the header by preceding each comment line with a pound sign (#). The header and comments are ignored by NNFie during the import phase.

The NNFie export file contains the components defined by the NEONFormatter Management API structures. Often the component definition information occurs in numeric form rather than text form. Without having a firm understanding of NEONFormatter ordinal type values and specific component definitions, you may find the contents of the export file difficult to decipher. In addition, the NNFie export file contains an inventory option that generates a component inventory listing in the NNFie.log file.

During the export phase, you can specify a text comment to be included in the export file.

In earlier releases, the length of a NEONFormatter component record 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 you can specify the line length that you want.

Import Phase

During the import phase, all formats and associated controls in the import file are loaded. NNFie detects situations in which an existing component that is modified during an import can cause the import of that component to fail. If an existing component is overwritten and the component being imported is identical, then the import can succeed. Any format or control that contains a component that fails to import also fails.

When importing output operation collections from earlier releases, NNFie removes any data-using operations that exist before data-producing operations in the collection. Data-using operations include length, trim, uppercase, lowercase, substring, justify, and prefix/suffix. Data-producing operations include math expression, default, substitute, and user exits. NNFie generates a warning message whenever an output operation is removed from an output operation collection. For additional information on output operation collections, see the ***NEONFormatterProgramming Reference*** or the ***Rules and Formatter Extension for IBM ® WebSphere Message Broker for Multiplatforms User's Guide***.

In earlier releases of Rules and Formatter Extension for IBM ® WebSphere Message Broker for Multiplatforms, when a component conflict was detected in NNFie import files, a conflict was logged, and the component was not imported. This process of identifying import conflicts without importing data allowed you to verify the contents of export files with working databases.

The current release of NNFie provides you with greater flexibility in conflict management. With the current product release, new Overwrite, Ignore/Skip, and Rename options are now available for resolving conflicts in existing database components, and all decisions to resolve conflicts are reported to an NNFie.log file throughout the import process. If a component fails to import, the line containing an error from the export file is written directly to the NNFie.err file.

Using NNFie with DB2

When using NNFie with DB2, pre-existing user permissions are not maintained when importing your NEONFormatter database. To maintain existing permissions, you can update the Permissions table before or after you import your formats.

If a user permission is not associated with a format in the import file, the user performing the import becomes the owner of that format.

To add user permissions before importing formats:

1. Log into the database using the existing userid that you want to maintain during the import process.
2. Define a temporary field to add to the database.

   This action adds the associated userid to the Permissions table.
3. Delete the temporary field.

To add user permissions after importing formats:

Use PermUtil to update a userid.

For more information, see *NEONRules and NEONFormatter Permissions* on page 38.

Preparing Formats for Upgrade

Before upgrading formats to your target database, verify the following:

- A supported RDBMS system was installed.
- Target database was created and instantiated using the NEONRules and NEONFormatter database schema.

- Formats, rules, and related components exist in a valid NEONRules and NEONFormatter database.
- Target database has enough disk space allocated to hold the output file.
- A database session entry was defined in the configuration file for running NNFie.

  For more information, see *Specifying Data Exporting and Importing Session Information* on page 29.
- Data consistency was checked and any data problems were corrected. NNFie is not designed to import or export databases that are corrupted or have unresolved issues with the data.
  - Run the NEONFormatter Consistency Checker against your existing database.
  - Repair any inconsistencies using the NEONFormatter graphical user interface (GUI).
  - Run the NEONRules Consistency Checker against your existing database.
  - Repair any inconsistencies using the NEONRules GUI.

  For more information, see *Consistency Checker* on page 153.

  For more information on compatibility, see *Determining Compatibility* on page 48.

Note:
For Rules and Formatter Extension for IBM ® WebSphere Message Broker for Multiplatforms, delete the NNFie.err and NNFie.log files before running NNFie. Failure to follow this procedure results in an error message in the NNFie.log that states that NNFie.err already exists.

Case Sensitivity

It is recommended that you avoid using letter-case to distinguish component names because many databases are not case-sensitive. In the case of importing databases that are not case-sensitive, two components named

[item1] and [ITEM1] would be identified as being the same. The second occurrence of this pair of components would cause a conflict.

NNFie

NNFie commands and parameters must be entered in the following order:

Syntax

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

Exporting Formats

The following parameters are available for exporting formats:

```
NNFie ((-C <command filename>)
      (-e | -export <export filename>
                    [-m <format name>+]
                    [-q "comment"]
                    [-Q <Comment filename>]
                    [-w <number>]
                    [-s <session name>]
                    [-k <type> <name> <version>]
                    [-c [<database configuration filename>]])
      (-t <import filename> [-s <session name>]
      [-c [<database configuration filename>]])
```

```
(-I <import filename> [-s <session name>])
[-c [<database configuration filename>]]))
```

Export Parameters

Parameter	Mandatory / Optional	Description
-C <command filename>	Optional	Alternate command filename; default file is NNFie.cmd. 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 use quotation marks to enclose arguments that contain spaces. All spaces in arguments must be preceded by a backslash. If an argument contains a backslash, this backslash must be preceded by an additional backslash.
-e | -export <export filename>	Mandatory	Required parameter to export data from the named file; mutually exclusive from -i. The default file is NNFie.exp.
-m <format name>	Optional	Specifies the format name to export individual formats. The default is set to export all formats.
-q "comment"	Optional	Adds comments enclosed in quotation marks to beginning of the export file. If the comment does not contain spaces, quotation marks are not necessary. If a command file is used, refer to its documentation for information on how it handles spaces.
-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.
-s <session name>	Optional	Specifies a session name in the configuration directory. If not specified, the system defaults to nnfie.

Parameter	Mandatory / Optional	Description
-k	Optional	Exports component by name. You must enter an integer type, name, and version. The integer type can have the following values: -k 1 = format_name -k 2 = parse_cntl -k 5 = field -k 6 = user_defined_type -k 7 = literal -k 8 = output_master -k 9 = default_cntl -k 10 = user_exit -k 11 = fix_cntl -k 12 = length -k 13 = math_cntl -k 14 = substitute -k 15 = substring -k 16 = trim_cntl -k 17 = collection -k 18 = map_name
-c [<database configuration filename>]	Optional	Name of configuration file. Default configuration file is nnsyreg.dat.
-t <export filename>	Optional	Writes an inventory of the export file to NNFie.log.
-I <export filename>	Optional	Writes description of all conflicts in export file to NNFie.log.
Filenames for both import and export must be no longer than 255 characters.		

Examples

To export all the formats in the entire database:

Type the following and press ENTER:

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

To export a single format:

Type the following and press ENTER:

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

To export several formats:

Type the following and press ENTER:

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

Troubleshooting NNFie Export Failures

You can verify the contents of the export files from a working database before importing any data to facilitate the validation process for archiving.

Any export failures are recorded in the NNFie.log file. However, you may also want to review the NNSYmessageLog.nml for additional error information.

To view this conflict report:

Type the following and press ENTER:

```
NNFie -I <export filename>
```

Producing an Inventory Export File

When you set the inventory export file option, NNFie produces an export inventory in the NNFie.log file. This file provides you with a readable format to review the items contained in the export file before you begin the import process. In addition, this format allows you can write or modify scripts that

create NEONFormatter components. For more information, see *NNFie Readable Files* on page 65.

To produce an export inventory listing in the NNFie.log file:

Type the following and press ENTER:

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

Importing Formats

The following parameters are available for importing formats.

```
NNFie ((-C <command filename>)
      (-i | -import <import filename>
                  [-T] [-o|-l [<conflict report filename>]
                       |-g|-n|-4|-p]
                  [-s <session name>])
      (-t <import filename> [-s <session name>]
      [-c [<database configuration filename>]])
      (-I <import filename> [-s <session name>])
      [-c [<database configuration filename>]]))
```

Note:
On z/OS, the -p parameter is mandatory when importing a 4.x export file to into a NEONRules and NEONFormatter V5.6 database in DB2. This parameter imports all literals as string types.

Import Parameters

Parameter	Mandatory / Optional	Description
-C <command filename>	Optional	Alternate command filename; default file is NNFie.cmd. 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 use quotation marks to enclose arguments that contain spaces. All spaces in arguments must be preceded by a backslash. If an argument contains a backslash, this backslash must be preceded by an additional backslash.
-i | - import <import filename>	Mandatory	Required parameter to import data from the named file; mutually exclusive from -e. The default file is NNFie.exp.
-T [<transaction boundaries>]	Optional	Specifies the number of rules or subscriptions within a transaction. Allows the user to determine when to start and close a transaction. A parameter of 0 makes the import a single transaction.
-o	Optional	Overwrites all conflicts and replaces all components of same name with those in the export file.
-l [<conflict report filename>]	Optional	Reports import conflicts, but does not import data. Default is off.
-g	Optional	Ignore; do not import any conflicting formats.
-n	Optional	Implement interactive conflict resolution. NNFie defaults to -n if no options are selected.

Parameter	Mandatory / Optional	Description
-4	Optional	Use R4_0 conflict resolution if a component in the export file conflicts 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.
-p	Optional	Imports literals as necessary string types. On z/OS, this parameter becomes mandatory when importing a 4.x export file to into a NEONRules and NEONFormatter V5.6 database in DB2.
-s <session name>	Optional	Specifies a session name in the configuration directory. If not specified, the system defaults to NNFie.
-t <import filename>	Mandatory	Writes an inventory of the import file to NNFie.log.
-c [<database configuration filename>]	Optional	Name of configuration file. Default configuration file is nnsyreg.dat.
-I <import filename>	Mandatory	Writes description of all conflicts in import file to NNFie.log.
Filenames for both import and export must be no longer than 255 characters.		

Examples

The following procedures show how to import formats using the NNFie command line utility. You must import formats before you import rules.

To import formats:

Type the following and press ENTER:

```
NNFie -i [<import filename>] [-s <session name>]
```

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. These conflicts and their resolutions are reported to the NNFie.log file.

To view the NNFie.log file:

Type the following and press ENTER:

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

The following options can help you reduce the number of import conflicts; however, caution should be used when exercising these options:

- Overwrite

  When you overwrite a component, the component definition within the import file is imported into the database; however, supporting components may remain unused.

- Ignore

  When you ignore a component, supporting components that have been imported may remain unused.

  Note:
  When you use the Ignore option, the component in the import file is added to the internal inventory of imported components and is not imported.

- Rename

  When you rename a component, all references to that component in the import file are updated.

Resolving Import Conflicts

You can resolve conflicts in batch or interactive mode. Interactive mode is only available for use on a Windows workstation, a UNIX-based workstation, or a z/OS UNIX command shell.

Resolving Conflicts in Batch Mode

In the batch mode, you can use the Overwrite and Ignore/Skip options to resolve conflicts. The selected option is applied to all resolutions.

To select the batch Overwrite conflict resolution option:

Type the following and press ENTER:

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

To select the batch Ignore/Skip conflict resolution option:

Type the following and press ENTER:

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

Resolving Conflicts in Interactive Mode

In the interactive mode on a Windows, or on a UNIX-based workstation, you can use the Overwrite, Ignore/Skip, and Rename options to resolve conflicts. Descriptions of the existing components and the import components are displayed.

To select interactive conflict resolution:

Type the following and press ENTER:

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

Example

```
Literal: "MyLiteral" conflicts with an existing Formatter
element!
literalLength ( existing=2 | incoming=3 )
Overwrite, Ignore, or Rename component (OIR): R
Please enter new component name: MyLiteral_NewValue
```

Troubleshooting Format Import Failures

Import failures are recorded in the NNFie.log file. However, you may also want to review the NNSYmessageLog.nml for additional error information.

The following describe the types of import failure:

- Missing or Incorrect Data Error Message

  This error message does not provide specific component information.

  ERROR: <error message>

- Missing or Incorrect Data Error Message for a Specific Formatter Element

  This error message contains NEONFormatter 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>
  ```

- Upgrading Output Controls Using Math Expressions

  In NEONet 4.1.1, math expressions used the internal representation of mapped data during evaluation. The data parsed as Decimal International (14,95) was represented internally as Decimal U.S. (14.95). The math expression parser read the data with the U.S. decimal separator (period) with no problem. When this same output control is imported into a Rules and Formatter Extension for IBM ® WebSphere Message Broker for Multiplatforms database and read by NEONFormatter, the data type conversion is done immediately, and the current form of the data is Decimal International (14,95). Because the NEONFormatter math expression parser uses the current form of the data, it sees the International decimal separator, the comma, and produces the following error (1021):

  Mathematical expression parse failed: 'MEparse mask = 00000200'.

  To avoid this error, you must create the following output control collection:

  <Math expression> + <'Convert' data type output operation>

- If your DB2 database resides on z/OS, you must import your data on the same platform from which you export it.

- You may have created literal values in the NEONFormatter GUI that have corrupted the NNFie export files and generate an import error. If you have a string literal that has non-printable characters, it is written to the export file in a form that disrupts NNFie literal processing.

  For example, if a linefeed character (0x0A) is entered and saved as a string literal, the NNFie export file corrupts.

  For more information on entering literal values, see the ***Rules and Formatter Extension for IBM ® WebSphere Message Broker for Multiplatforms User's Guide***.

NNFie Readable Files

You can write or modify scripts that create NEONFormatter components. The following guidelines are provided for users who are experienced with modifying the file format.

NNFie Header

When an NNFie export file is created, you can add a header file to log the following information:

- Time of creation (using Greenwich Mean Time)
- NNFie version number
- Database logon information
- Database server version
- Operating system version

You can also add comments in the header file by beginning each comment line with a pound sign (#). These comments are ignored by NNFie during import.

To add a header file:

Type the following and press ENTER:

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

To specify comments in the header file:

Type the following and press ENTER:

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

NNFie Export Data

Within the export file, a continuation marker divides the format component definition into several lines. 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 length is 80 characters, but you can specify your own line length.

To specify the line length default:

Type the following and press ENTER:

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

For enumerated type, refer to the appropriate header files.

Component Definition Guidelines

NNFie uses the NEONFormatter Management APIs to populate the database with NEONFormatter component definitions.

The following structural concepts can help you read the component definitions a typical NNFie export file:

- ! delimits format components.
- / / begins a comment line.
- (…)+ indicates items within parentheses exist one or more times.
- (…)* indicates items within parentheses exist zero or more times.
- All string data types used as field types are 32 characters or less.
- Encoded hex field types are up to 254 characters. Valid field characters are 0x[0-9A-F].

  For more information on field values, see the ***NEONFormatter Programming Reference***.

- When an integer defines a code for an enumerated type, all definitions using enumerated type have the fixed type defined as enum. Valid entries appear in the fmtcodes.h file in the nnsy/rulfmt56/include/FMTR directory.
- Each component definition begins with three identification fields, delimited by an exclamation point (!) character.
  - First field

    Begins with the letter F, indicating the beginning of a NEONFormatter component definition.
  - Second field

    Contains the release number of the defined component. By using release numbers to define components, NNFie can support several revisions of export files. Release 4.0 is used for all controls except Release 5.0.1 is used for INPUT_CONTROL, COLL_C, MAP, MAP_LINK, FORMAT_GROUP, and PERMISSION. Release 4.0 is used for all other components.
  - Third field

    Contains an integer that indicates a valid component in an export format.
  - Fourth field

    Contains an integer that indicates a valid format component. The following table describes valid format components and corresponding values used in the export files.

Value	Format Component	Description
1	FORMAT	Format
2	INPUT_CONTROL	Input Control. Input Control uses release 5.x instead of 4.x due to the changes in Regular Expression structure.
3	OUTPUT_CONTROL	Output Control (included for backward compatibility; use Output Master Control)
4	NNF_DELIMITER	Delimiter (included for backward compatibility; use Literal)
5	FIELD	Field
6	USER_DEFINED_TYPE	User Defined Type
7	LITERAL	Literal
8	OUTPUT_MSTR	Output Master Control
9	DEFAULT_C	Default Control
10	EXIT_C	User Exit Control
11	FIX_C	PrePostFix Control
12	LENGTH_C	Length Control
13	MATH_C	Math Expression Control
14	SUBSTT_C	Substitute Control
15	SUBSTR_C	Substring Control
16	TRIM_C	Trim Control
17	COLL_C	Collection Control. Collection Control uses release 5.0.1 instead of 4.0 due to serialized controls.

Value	Format Component	Description
18	MAP	Format Map
19	MAP_LINK	Map Link (hierarchy)
20	FORMAT_GROUP	Format Group
21	PERMISSION	Permission
22	PLUG-IN	Plug-in
23	ENCODING	Code sets

- Permissions are listed in the export file in the form of a component and are supported on all components except the MAP_LINK and PERMISSIONS. User permissions include:

  O = Owner

  U = Update

  UO = Update & Owner

  For more information on user permissions, see the ***Rules and Formatter Extension for IBM ® WebSphere Message Broker for Multiplatforms User's Guide***.

Sample NNFie Component Import File

The following is a sample NNFie import file containing component definitions. For more information about NEONFormatter components, see the ***Rules and Formatter Extension for IBM ® WebSphere Message Broker for Multiplatforms User's Guide***.

Flat Input Format Example

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

Components

```
F!<NNFie_Version_for_Component>!
```

```
<NNFie_Component_Type — integer>!

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

// NNFMgrFlatFormatInfo structure
<Decomposition ID — integer>,
<Length ID — integer>,
<Termination ID — integer>,
<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!<NNFie_Version_for_Component>!
<NNFie_Component_Type — integer>!

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

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

// NNFMgrOutFieldInfo structure
```

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

Compound Format Example

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

Components

```
F!<NNFie_Version_for_Component>!
<NNFie_Component_Type – integer>!

// NNFMgrFormatInfo structure
<Format Name – string>,
<Input Indicator ID – integer>,
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 Count - integer>,
<Repeat Field Name – string>!)+
```

Input Control Example

```
F!5.6!2!ic,1,1,String,0,,1,NONE,scolon,0,1,0,0,0,1,NONE,0x00,\
NONE,0,0,0,1,NONE,0,101,!0!
```

Components

```
// NNFMgrParseControlInfo structure
<Control Name - string>,
<optionalInd - integer>,
<fieldType integer>,
```

```
<dataType integer>,
<dataTermination integer>,
<dataRegExp string>,
<dataDelimiter string>,
<dataLength integer>,
<dataLengthUnit integer>,
<tagType integer>,
<tagStorageType integer>,
<tagTermination integer>,
<tagLength integer>,
<tagLitrlName string>,
<tagValue integer>,
<tagDelimiter string>,
<tagLengthUnit integer>,
<lengthType integer>,
<lengthTermination integer>,
<lengthLength integer>,
<lengthDelimiter string>,
<lengthLengthUnit integer>,
<decimalLocation integer>,
<validationParamName string>,
<NameValuePair* userDefInValNameValuePairArray>,
<dataAttr string>,
<baseDataType integer>,
<yearCutoff integer>,
<useZeroYearCutoffInd integer>!
```

Field Example

```
F!4.0!5!numeric,Numeric field!
```

Components

```
F!<NNFie_Version_for_Component>!
<NNFie_Component_Type — integer>!

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

User-defined Type Example

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

```
UserDefinedTypeValidation!
```

Components

```
F!<NNFie_Version_for_Component>!
<NNFie_Component_Type — integer>!

// NNFMgrUserDefTypeInfo structure
<Type Name — string>,
<Native Type — string>,
<Validation Routine Name — string>!
```

Binary-type Literal Example

```
F!5.6!7!comma,0x2C,1!0!
```

Components

```
//NNFMgrLiteralInfo Structure
<Literal Name - string>,
<Value - ASCII - encoded hex>,
<Value Length - integer>,
<Data Type Id - integer >!
```

String-type Literal Example

```
F!5.6!7!pipe,|,1,1!
```

Components

```
//NNFMgrLiteralInfo Structure
<Literal Name - string>,
<Value - string>,
<Value Length - integer>,
<Data Type Id - 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!<NNFie_Version_for_Component>!
```

```
<NNFie_Component_Type — integer>!

// NNFMgrOutMstrCntlInfo structure
<Master Name — string>,
<Optional Indicator ID — integer>,
<Field Type ID — integer>,
<Data Type Name — string>,
<Data Attribute ID — integer>,
<Base Data Type ID — integer>,
<Tag Type ID — integer>,
<Tag Literal Name — string>,
<Tag Value — ASCII-encoded hex>,
<Tag Value Length — integer>,
<Tag-before-Length Indicator ID — integer>,
<Length Type ID — integer>,
<Operation Type ID — integer>,
<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!<NNFie_Version_for_Component>!
<NNFie_Component_Type — 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!<NNFie_Version_for_Component>!
<NNFie_Component_Type — integer>!

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

PrePostFix Control Example

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

Components

```
F!<NNFie_Version_for_Component>!
<NNFie_Component_Type — integer>!

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

Length Control Example

```
F!5.6!12!length_CN,4,0,pad_w,0x77,1!
```

Components

```
<Control Name - string>,
<Pad Literal Name - string>,
<padValue - ASCII - encoded hex>,
<padValueStorageType integer>,
<padValueLen integer>,
<padValueType integer>,
<dataLen integer>,
<dataLenUnit integer>!
```

Math Expression Control Example

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

Components

```
F!<NNFie_Version_for_Component>!
<NNFie_Component_Type — integer>!

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

Substitute Control Example

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

Components

```
F!<NNFie_Version_for_Component>!
<NNFie_Component_Type — 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 — integer>!
<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 — integer>!)*
```

Substring Control Example

```
F!5.6!15!substring_85,1,3,1,literal_77,0x23,1!
```

Components

```
<Control Name - string>,
<Start - integer>,
<Length - integer>,
<Pad Literal Name - string>,
<Pad Value - ASCII
<padValueStorageType - integer>
<padValueLen - integer>
<padValueType - integer><substringUnit - integer>!
```

Trim Control Example

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

Components

```
F!<NNFie_Version_for_Component>!
<NNFie_Component_Type — 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!5.0.1!17!Sample_CollectionCntl,2!Sample_UserExitCntl,7!
CENTER_JUSTIFY,10!
```

Components

```
F!<NNFie_Version_for_Component>!
<NNFie_Component_Type — integer>!

// NNFMgrCollectionCntlInfo Structure
```

```
<Control Name — string>,
<Collection Count — integer>!
(<Child Control Name — string>,
<Child Control Type — enum NNCntlType>!)+
```

Sample Data

```
F!5.0.1!5!alpha,!
F!5.0.1!5!numeric,!
F!5.0.1!7!=,0x3D000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000,1!
F!5.0.1!8!alpha_OC,1,1,String,,0,0,NONE,0x0000000000000000000
000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000,0,0,0,0,
NONE,0x000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000
000000000000000,0,NONE,0!
F!5.0.1!8!numeric_OC,1,1,String,,0,0,NONE,0x000000000000000000
000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000,0,0,0,
0,NONE,0x000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000
000000000000000000,0,NONE,0!
F!5.0.1!2!alpha_IC,0,1,String,0,,2,NONE,6,0,0,0,NONE,0x000000E
FFFE86000000000000A95DC00101E20000000000006B54C00000001EFFFF01
400000101EFFFF014EF3C710100000001000000100101E20EFFFF0DCEFFFF
0DCEFFFE850000B2E60EF07D6D800101E20EF3C7178000000000EFFFE854000
FA841EFFFF088EFFFF088EFFFF01400000000EFFFF014EFFFF0DC000000000
0,NONE,0,0,0,NO NE,0,101,!0!
F!5.0.1!2!numeric_IC,0,1,Numeric,0,,2,NONE,8,0,0,0,NONE,0x0000
00EFFFE86000000000000A95DC00101E20000000000006B54C00000001EFFF
F01400000101EFFFF014EF3C710100000001000000100101E20EFFFF0DCEF
```

```
FFF0DCEFFFE850000B2E60EF07D6D800101E20EF3C717800000000EFFFE854
000FA841EFFFF088EFFFF088EFFFF01400000000EFFFF014EFFFF0DC000000
0000,NONE,0,0, 0,NONE,0,101,!0!
F!5.0.1!Flat_IC,1,0!1,0,0,NONE!2!Flat_IC,alpha,alpha_IC!
Flat_IC,numeric,numeric_IC!
F!5.0.1!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!5.0.1!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!
```

Map Example

```
F!5.0.1!18!Vail,1, for the Vail_Inbound messages.!\
10!\
Aspen Area Visited,Aspen Area Visited,-1,-1,VailIdentifier,3,!\
Aspen Defined Skier Characteristic,Skier Name,1,-1,,0,!\
Aspen Defined Skier Number,Skier Identifier,-1,-1,,0,!\
Aspen Lift Location,Lift Number,-1,-1,,0,!\
Aspen Document Date,Ski Date,-1,-1,,0,!\
Aspen Lift Time,Ski Time,-1,-1,,0,!\
Boarding Time,Ski Time,-1,-1,,0,!\
Lift,Lift Number,-1,-1,,0,!\
Time,Boarding Time,-1,-1,,0,oc_skitime!\
Name,Skier Name,-1,-1,UPPER_CASE,6,!
```

Components

```
F!<NNFie_Version_for_Component>!18!
<Map Name - string>,
<Map version -integer>,
<Map description - string>!
<number of fields in map - integer>!

-------------------------------------------------------
NOTE 1: The portion of the map below refers to the
NNFMgrMapInfo structure, and will repeat according
to the number of fields in map, specified above.

NOTE 2: Using Default values
To use the (default) map information defined in the
output format, use the following values:
```

```
access mode: -1
subscript: -1
Child control name: (leave empty. note double commas
in example)
Child control type: 0 (only used when Child control
name is left empty)
Output master control name: (leave empty. note comma
followed immediately by '!' in most entries
             below.)
-----------------------------------------------------
(
    <Output field name - string>,
    <Input field name - string>,
    <access mode - integer>,
    <subscript - integer>,
    <Child control name - string>,
    <Child control type - integer>,
    <Output master control name -string>!
)+
```

Map Link Example

```
F!5.0.1!19!Newmap_1,1,NewMap_2,1!
```

Components

```
F!<NNFie_Version_for_Component>!
<NNFie_Component_Type — integer>!

// NNFMgrMapLink structure
<Child Map Name — string>,
<Child Map Version — integer>,
<Parent Map Name - string>,
<Parent Map Version — integer>,
<Level — integer>!
```

Format Group Example

```
F!5.0.1!20!Newcollection,7!0DupeNewFmt6!0NewFormat_6!\
NewFormat_1!NewFormat_2!NewFormat_3!bob.IC!NewFormat_4!
```

Components

```
F!<NNFie_Version_for_Component>!
<NNFie_Component_Type — integer>!

// NNFMgrFormatGroupInfo structure
<Format Collection Name - string>,
<Number of Formats in Collection - integer>,
<Format Name - string>,
<Format Name - string>,
(<Format Name - string>)+!
```

Permissions Example

```
F!5.0.1!21!Newcollection,0,20!bfitzpatrick,UO!
```

Components

```
F!<NNFie_Version_for_Component>!
<NNFie_Component_Type — integer>!

// Permissions
<Component Name - string>,
<Component Version - integer>,
<Component Group - integer>!
<User Name - string>,
<State - Update and Owner>!
```

Encoding Example

```
F!5.6!23!encode_type,ibm-1252!
```

Components

```
F!NNFie_Version_for_Component>!
<NNFie_Component_Type — integer>!

<"encode-type">,
<Encoding Type Name>!
```

The following example illustrates a Plugin that contains two instances and two parameters. It is divided to provide clarity.

Plugin Example

```
F!5.01!22!PluginName,PluginDescription,PluginLibraryName,\
PluginFactoryName,1000,100,0,
ParameterName1,DataValue,1,0,0,0,2002,0,
ParameterName2,DataValue,2,0,0,0,2002,0,
,,0,0,0,0,0,1,
InstanceName1,InstanceDataValue,0,0,0,500,2002,0,
ParameterName1,DataValue,1,0,0,500,2002,0,
ParameterName2,DataValue,2,0,0,500,2002,0,
,,0,0,0,0,0,1,
InstanceName2,InstanceDataValue,0,0,0,501,2002,0,
ParameterName1,DataValue,1,0,0,501,2002,0,
ParameterName2,DataValue,2,0,0,501,2002,0,
,,0,0,0,0,0,1,
,,0,0,0,0,0,1,
2002!
```

Buffer Example

```
F!<NNFie_Version_for_Component>!<NNFie_Component_Type -\
integer>!
```

Plugin Component

```
<Type name – string>,
<Type description – string>,
<Type libraryName – string>,
<Type factoryName – string>,
<Value appId – long>,
<Value subAppId – long>,
<Value version – long>,
```

Plugin Parameter Component

```
<Type name – string>,<Type dataValue – string>,
<Value seqNum – long>,
<Value dataLength – long>,
<Value dataType – long>,
<Value instanceId – long>,
<Value typeId – long>,
<Value flag – integer>,
```

Note:
The Plugin Parameter section is repeatable from zero to an unlimited number of times.

Plugin Parameter Ender Component

```
<Type name – string>,This must be empty.
<Type dataValue –string>, This must be empty.
<Value seqNum – long>,This must be zero.
<Value dataLength – long>,This must be zero.
<Value dataType – long>,This must be zero.
<Value instanceId – long>,This must be zero.
<Value typeId – long>,This must be zero.
<Value flag – integer>,This must be one.
```

Note:
The Plugin Parameter Ender section must happen exactly once after all of the previous sections.

Instance Component

```
<Type name – string>,
<Type dataValue – string>,
<Value seqNum –long>,
<Value dataLength – long>,
<Value dataType – long>,
<Value instanceId – long>,
<Value typeId – long>,
<Value flag – integer>,
```

Note:
The Instance section is repeatable from zero to an unlimited number of times.

Instance Parameter Component

```
<Type name – string>,<Type dataValue – string>,
<Value seqNum – long>,
<Value dataLength – long>,
<Value dataType – long>,
```

```
<Value instanceId – long>,
<Value typeId – long>,
<Value flag – integer>,
```

Note:
The Instance Parameter section is repeatable from zero to an unlimited number of times. However, this section is embedded within the outer section an unlimited number of times.

Instance Parameter Ender Component

```
<Type name – string>,This must be empty.
<Type dataValue – string>,This must be empty.
<Value seqNum – long>,This must be zero.
<Value dataLength – long>,This must be zero.
<Value dataType – long>,This must be zero.
<Value instanceId – long>,This must be zero.
<Value typeId – long>,This must be zero.
<Value flag – integer>,This must be one.
```

Note:
The Instance Parameter Ender section must happen exactly once after all of the previous sections.

Instance Ender Component

```
<Type name – string>,This must be empty.
<Type dataValue – string>,This must be empty.
<Value seqNum – long>,This must be zero.
<Value dataLength – long>,This must be zero.
<Value dataType – long>,This must be zero.
<Value instanceId – long>,This must be zero.
<Value typeId – long>,This must be zero.
<Value flag – integer>,This must be one.
```

Note:
The Instance Ender section must happen exactly once after all of the previous sections.

Component

```
<Value typeId – long>!
```

Note:
This section must happen exactly once after all of the previous sections.

NNFie Error Messages

For a complete listing of NNFie error messages, see the NEONFormatter ***Programming Reference***.

Also, check the NNSYmessageLog.nml for information about database-specific errors. In some instances, a database-specific error can be directly reported to the NNSYmessageLog without reporting the error as an NNFie failure.

Upgrading Rules

As part of the Rules and Formatter Extension for IBM ® WebSphere Message Broker for Multiplatforms installation, the data upgrade process uses NNRie to move rules from an existing database to the target database.

NNRie

You can use the NEONRules Import and Export utility, NNRie to upgrade rules data from one release to another release, or from one database to another database.

WARNING!

- You cannot simultaneously run NNFie and NNRie. Database problems may result.

Importing and Exporting Rules

NNRie creates a text-based export file. The exported file can be imported into another release of Rules and Formatter Extension for IBM ® WebSphere

Message Broker for Multiplatforms. Then you run NNRie again to import rules to the target Rules and Formatter Extension for IBM ® WebSphere Message Broker for Multiplatforms database.

Note:
If your DB2 database resides on z/OS, you must import your data on the same platform from which you export it.

You can select all rules components or specific components for export. If a specific component is selected, all of its subordinate components must also be exported. For example, if a specific rules is selected for export, all of its subscriptions and actions are selected for export by default. You also have the option to export subscriptions, rulesets, message types, or application groups by specifying the export and import parameters that are passed to NNRie.

When you add subscriptions to application group/message types (rulesets), you associate them with multiple rules in the same application group/message type. In this case, the rule name no longer identifies the subscriptions. During the export/import process, you will be prompted to create the new subscription names. If you export subscriptions by name only, no rule information is retrieved; however, if you export rules, all the associated subscriptions are exported at the beginning of the application group/message type.

In earlier releases, NNRie allowed you to set the database transaction boundaries for each rule and subscription. In the current release the transaction boundaries exist as a default and provide the following benefits:

- By defining variable transaction boundaries, you could determine how many rules and subscriptions to import with a transaction and use the number of rules and subscriptions to determine when it was time to close and begin a new transaction.
- If any portion of the transaction fails, the portion of the importing data at the time of the failure is written to a separate file for correction and re-importing.
- By narrowing the transaction boundaries, you can specify the Fail file, but as the transaction interval was increased, the database transaction log required more space.

During exporting, NNRie verifies the values for those parameters that do not have default values. If a required value is missing or incorrect, NNRie displays an error message and does not export the entire database.

Using NNRie with DB2

When using NNRie with DB2, pre-existing user permissions are not maintained when importing your NEONRules database. To maintain existing permissions, you can update the Permissions table before or after you import your rules.

If a user permission is not associated with a rule in the import file, the user performing the import becomes the owner of that rule.

To add user permissions before importing rules:

1. Log into the database using the existing userid that you want to maintain during the import process.
2. Define a temporary field to add to the database.

   This action adds the associated userid to the Permissions table.
3. Delete the temporary field.

To add user permissions after importing rules:

Use PermUtil to update a userid.

For more information, see *NEONRules and NEONFormatter Permissions* on page 38.

Preparing Rules for Upgrade

Before upgrading rules to your target database, verify the following:

- Data consistency was checked and any data problems were corrected. NNRie is not designed to import or export databases that are corrupted or have unresolved issues with the data.

  For more information, see *Consistency Checker* on page 153.
- A supported RDBMS system was installed.

- Target database was created and instantiated using the NEONRules and NEONFormatter database schema.
- Formats, rules, and related components exist in a valid NEONRules and NEONFormatter database.
- Target database has enough disk space allocated to hold the output file.
- A database session entry was defined in the nnsyreg.dat file for running NNRie.

  For more information, see *Specifying Data Exporting and Importing Session Information* on page 29.

Case Sensitivity

It is recommended that you avoid using letter-case to distinguish component names because many databases are not case-sensitive. In the case of importing databases that are not case-sensitive, two components named [item1] and [ITEM] would be identified as being the same. The second occurrence of this pair of components would fail to be imported.

NNRie

NNRie commands and parameters must be entered in the following order:

Syntax

```
NNRie (-C [<command filename>]) | -V |
(-i | -import [<import filename>]
              [-o|-O|-l [<conflict report filename>]
                 |-g|-n]
              [-T [<transaction boundaries>]]
              [-U]
              [-h [<history filename>]
              [-f [<failure filename>]]
              [-s [<session name>]]) |
(-e | -export [<export filename>]
            [[[-a <appname>] [...]]
              [-m <msgname>] [...]]
              [-r <rulename>] [...]] | [-S <subsname>] [...]]]
              [-t [<inventory report filename>]]
```

```
              [-w]
              [-s [<session name>]]
              [-c [<database configuration filename>]])
```

Remarks

NNRie displays a brief usage reminder if syntax is entered with no parameters. If the -V parameter is used, only release and copyright information is displayed. If no export options are provided (-a, -m, -r, or -S), the entire database is exported. These parameters must include values. If a required value is missing, NNRie displays an error and exits.

Exporting Rules

The following parameters are available for exporting rules:

```
NNRie (-C [<command filename>]) | -V |
(-e | -export [<export filename>]
              [-o|-O|-l [<conflict report filename>]
            [[[-a <appname> [...]]
              [-m <msgname>] [...]]
              [-r <rulename>] [...]] | [-S <subsname> [...]]]
              [-t [<inventory report filename>]]
              [-w]
              [-s [<session name>]]
              [-c [<database configuration filename>]])
```

Export Parameters

Name	Mandatory / Optional	Description
-C [<command filename>]	Optional	Alternate command file. If -C is provided, NNRie reads command line options from the specified file instead of the command line. The default filename is NNRie.cmd.
-V (version)	Optional	Returns version information only; performs no processing.
-e | -export [<export filename>]	Mandatory	Required parameter to export data; mutually exclusive from -i. Can include name of file that contains export data. Default file is NNRie.exp.
-a [<appname>]	Optional	Identifies 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. Requires the -a parameter. Default is to export all message types within the specified application group. This parameter can be repeated to define multiple message types within the same application group.
-r [<rulename>]	Optional	Specifies the name of the rule to export. Requires the -a and -m parameters. Default is to 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.

Name	Mandatory / Optional	Description
-S [<subsname>]	Optional	Specifies the name of the subscription to export. Requires the -a and -m parameters. This parameter can be repeated to export multiple subscriptions.
-t [<export filename>]	Optional	Creates an inventory of an export file; does no processing. For more information, see *Producing an Inventory Export File* on page 93.
-w	Optional	Export file line width; default is 80 characters.
-s <session name>	Optional	Specifies a session name in the configuration directory. If not specified, the system defaults to nnrmie.
-c [<database configuration filename>]	Optional	Name of configuration file. The default configuration file is nnsyreg.dat.

Examples

The following procedures show how to export rules using the NNRie command line utility. For more information on optional parameters and resolving NNRie conflicts, see*Resolving Component Conflicts* on page 62.

To export the entire NEONRules database:

Type the following and press ENTER:

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

To export a set of application groups:

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.

Type the following and press ENTER:

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

To export a set of message types for an application group:

The application group name and message type name are exported. The message type export includes all subscriptions and rules in the specified application group / message type. If multiple message type names are given, the subscriptions and rules for each message type are exported.

Type the following and press ENTER:

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

To export rules for application groups and message types:

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.

Type the following and press ENTER:

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

To export subscriptions for application groups and message types:

No rule information is exported. The application group and message type name information are exported, and then the subscription information is exported. If multiple subscriptions are given, each subscription is exported.

Type the following and press ENTER:

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

Producing an Inventory Export File

The NNRie inventory export file provides an option that lets you see which items are contained in an export file. When this option is set, NNRie produces an export inventory in the NNRie.log.

To produce an export inventory listing in the NNFie.log file:

Type the following and press ENTER:

```
NNRie –t <export filename to inventory>
```

Importing Rules

The following parameters are available for importing rules.

```
NNRie (-C [<command filename>]) | -V |
(-i | -import [<import filename>]
              [-o|-O|-l [<conflict report filename>]
                 |-g|-n]
              [-T [<transaction boundaries>]]
              [-U]
              [-h [<history filename>]
              [-f [<failure filename>]]
              [-s [<session name>]])
```

Import Parameters

Name	Mandatory / Optional	Description
-C [<command filename>]	Optional	Alternate command file. If -C is provided, NNRie reads command line options from the specified file instead of the command line. The default filename is NNRie.cmd.
-V (version)	Optional	Returns version information only; performs no processing.
-i |-import [<import filename>]	Mandatory	Required parameter to import data; mutually exclusive from -e. Can include name of file that contains the import data.Referenced file must have been created with the NNRie -e option. Default filename is NNRie.exp.

Name	Mandatory / Optional	Description
-o (overwrite data)	Optional	Default is off (do not overwrite). If a rule or subscription defined in the import file exists in the importing database, the existing data is overwritten with the new definition. 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.
-O	Optional	Completely overwrites all imported rulesets (application message type pairs). Default does not overwrite.
-l [<conflict report filename>]	Optional	Reports import conflicts, but does not import data. Default is off.
-g	Optional	Ignore; do not import any conflicting rules and subscriptions
-n	Optional	Implement interactive conflict resolution. Default is on.
-T [<transaction boundaries>]	Optional	Specifies the number of rules or subscriptions within a transaction. Allows the user to determine when to start and close a transaction. A parameter of 0 makes the import a single transaction.
-U [<update frequency>]	Optional	Update Statistics every x rules imported. Designed to work with Oracle 7.x and 8.x databases only. The update frequency value specifies the number of messages to import between statistic updates. Default is zero (0) or off.
-h [<history filename>]	Optional	Import history. If successful, the history file should look exactly like the import file. Default file is NNRieT.log.

Name	Mandatory / Optional	Description
-f [<failure filename>]	Optional	Specifies the failure file that contains lines not imported. This file can be used as an import file after the issues causing the failure are addressed. Default filename is NNRie.err.
-s <session name>	Optional	Specifies a session name in the configuration directory. If not specified, the default is nnrmie.

Examples

The following procedures show how to import rules using the NNRie command line utility. For more information on optional parameters and resolving NNRie conflicts, see *Resolving Component Conflicts* on page 62.

You must import formats before you import rules.

To import rules:

Type the following and press ENTER:

```
NNRie -i [<import filename>] [-s <session name>]
```

Tracking Import Progress

You can track the progress of the import. The following alphabetic characters define import and export components as they are processed by NNRie. These characters are displayed during import and export as progress indicators.

Trace Letters

Character	Description
A	Application Group
C	Action; written to file or added to database
M	Message Type

Character	Description
P	Option; written to file or added to database
R	Rule
S	Subscription; written to file or added to database
c	Action; read from file
e	Rule expression written to file or added to database
l	Subscription linked to rule in database
n	Permission (rule or subscription)
p	Option; read from file
s	Subscription; read from file

Resolving Component Conflicts

A conflict occurs when an imported rule or subscription does not match an existing imported rule or subscription of the same name and type in the database. These conflicts and their resolutions are reported to the NNRie.log file.

To view the NNRie.log file:

Type the following and press ENTER:

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

The following options can help you reduce the number of import conflicts. These options do not affect the integrity of the database.

- Overwrite

  When you overwrite a component, the component definition within the export file is imported into the database. In previous releases, NNRie checked for a conflict before overwriting a rule or subscription and recorded the conflict in the log file. In the current

release, NNRie no longer checks for conflicts when the -o parameter is used in importing.

- Ignore

  When you ignore a component, the component in the export file is not imported; however, supporting components that were imported may remain unused.

  Note:
  When you use the Ignore option, the component in the import file is added to the internal inventory of imported components and is not imported.

- Rename

  When you rename a component, all references to that component in the import file are updated.

Resolving Import Conflicts

You can resolve conflicts in batch or interactive mode. Interactive mode is only available for use on a Windows workstation, a UNIX-based workstation, or a z/OS UNIX command shell.

Resolving Conflicts in Batch Mode

In the batch mode, you can use the Overwrite and Ignore options to resolve conflicts. The selected option is applied to all resolutions.

To select the batch Overwrite conflict resolution option:

Type the following and press ENTER:

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

To select the batch Ignore conflict resolution option:

Type the following and press ENTER:

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

To replace an entire application group / message type pair:

Type the following and press ENTER:

```
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

Interactive conflict resolution is the default option. If rule or subscription conflicts exist, NNRie goes into interactive mode. During interactive mode, do not leave NNRie running unattended, unless you overwrite existing rules and subscriptions with -o or message types with -O.

In the interactive mode on a Windows or on a UNIX-based workstation, you can use the Overwrite, Ignore / Skip, and Rename options to resolve conflicts. Descriptions of the existing components and the import components are displayed.

To select the interactive conflict resolution option:

Type the following and press ENTER:

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

Troubleshooting NNRie Import Failures

All rules import conflicts and resolutions are reported to the NNRie.log file. To review import failures, see the NNRie.log file, and for additional error information, check the NNSYmessageLog.nml file.

The following is a sample NNRie.log file:

```
Conflict with Subscription: 'S3'
    App Name: 'MsgTest'
    Msg Name: 'MsgTest'
  Subs in import file:
    Owner: 'Public'
    Comment: 'New Checking'
  Subs in Database:
```

```
Owner: 'PUBLIC'
Comment: ''
Conflict Exists in : Comment
```

To edit the NNRie.log file for Rules and Formatter Extension for IBM ® WebSphere Message Broker for Multiplatforms, see the documentation on the import file format for instructions. For earlier versions of the export file, contact Technical Support to resolve the import problem.

NNRie Readable Files

You can write or modify scripts that create NEONRules components.

Header

When an NNRie export file is created, you can add a header file to the beginning of the file to log the following information:

- Time of creation (using Greenwich Mean Time)
- NNRie version number
- Database logon information
- Database server version
- Operating system version

You can also add comments in the header file by beginning each comment line with a pound sign (#). These comments are ignored by NNRie during import.

To add a header file:

Type the following and press ENTER:

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

To specify comments in the header file:

Type the following and press ENTER:

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

Export Data

You can use the NEONRules GUI to modify rules and subscriptions. The following guidelines are provided for users who are experienced with modifying the file format.

Component Definition Guidelines

NNRie uses the NEONRules Management APIs to populate the database with NEONRules component definitions.

The following structural concepts can help you read the component definitions a typical NNRie export file:

- Commas with no spaces delimit component field and a backslash (\) must precede a comma if a comma is used part of a field definition.
- First line contains an R, indicating the beginning of a NEONRules component.
- Second line contains a five-digit number for the rule component type code, such as 10001, and the release number of the export file, such as 5.6.
- Third line contains a rule component type code.
- The remaining lines list components of an application group / message type in the following required order:
  - application group
  - message type
  - subscription definitions
  - rules definitions

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)
}
```

Sample NNRie Component Import File

The following is a sample NNRie import file containing component definitions. For more information about NEONRules components, see the ***Rules and Formatter Extension for IBM ® WebSphere Message Broker for Multiplatforms User's Guide***.

```
R
10001,5.6
10002,sja
10003,sja,InFlat,NNSYRF_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

```
10001,5.6
```

5.6 is the version number.

(10002) Application Group

```
10002,sja
```

sja is the application group.

(10003) Message

```
10003,sja,InFlat,NNSYRF_FORMATTER
```

InFlat is the import format name.

NNSYRF_FORMATTER is the evaluation type. This message type refers to an input format.

(10004) Rule

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

Rule Component	Description
r1	Rule name.
1	Number of arguments.
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
```

Rule 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
```

Rule 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
```

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

(10010) Rule Permission

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

Rule 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
```

Rule 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
```

Rule 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
```

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

NNRie Error Messages

For a complete listing of NNRie error messages, see the ***NEONRules Programming Reference.***

Also, check the NNSYmessageLog.nml for information about database-specific errors. In some instances, a database-specific error can be directly reported to the NNSYmessageLog without reporting the error as an NNRie failure.

Cross-Platform Migration

This section describes how to maintain database integrity while migrating between product versions and platforms. Before migrating databases between versions or platforms, consider the following:

- The internal structures of your New Era of Networks Rules and Formatter databases differ depending on the product version.
- The locale of the platform in which these databases reside determine how the data is represented.
- The NNFie and NNRie import and export utilities are provided with the New Era of Networks Rules and Formatter to perform the migration.

Cross-Platform Migration

To migrate formats and rules between platforms, observe the following guidelines:

- Use the NNFie and NNRie utilities provided with New Era Support for WMQI to migrate rules and formats between platforms.
- Do not transfer export files that use different code pages between platforms by ftp.

  Characters that are in the syntactic character set, that is, characters whose code points are invariant between most code pages may not be converted correctly during an ftp transfer. For example, the exclamation mark (!) is at code point 0x5A in some EBCDIC code pages and can be converted as the pipe symbol (|).

  NNFie and NNRie use the ICU function calls to convert character data. These calls are more sensitive to variations in code points between code pages.

Migrating Between Versions and Platforms

The specific steps of the migration procedure depend on the version and platform of your existing and target databases. The following sections present a general overview of the process followed by specific procedures for each type of cross-platform and cross-version migration.

Process Overview

The migration process includes the following general steps:

- Creating export files from your existing database using the NNFie and NNRie import/export utilities.
  - If you are exporting from a Version 2.1/5.0 database, you have the option to run NNFie and NNRie locally or remotely. The export files are generated on the machine in which you run these utilities.
- Importing your formats and rules into a Version 6.0 database using NNFie and NNRie.

  To run the NNFie and NNRie utilities on a different machine than where your export files reside, do not use ftp to transfer the files. Instead, you should mount the directory on the remote machine containing the export files as a file system on the local machine, provided that you are migrating from an ASCII to an ASCII database or from an EBCDIC to an EBCDIC database. If you are migrating from an ASCII to an EBCDIC database or from an EBCDIC to an ASCII database, you must use the utilities supplied by Rules and Formatter Extension for WebSphere Message Broker 6.0.
  - If you are running NNFie and NNRie from the command line, you must define a sessionin the nnsyreg.dat file that describes the remote database and specify the session identifier using the -s option.
  - If you are running NNFie and NNRie from the GUI, the database context is defined by the current connection to the data source.

- Opening the New Era of Networks Formatter Version 6.0 GUI to do the following:

Cross-Version and Cross-Platform Migration Procedures

The following procedures provide specific details for cross-version and cross-platform migration.

Note:
Before attempting the migration procedures described in this section, you must obtain a copy of the WMQI V2.1 readme for instructions on completing the NNSY name change.

Existing Database on WMQI 2.1 or Websphere Message Broker 5.0 for any O/S Target Database on Websphere Message Broker V6.0 for any O/S

Note:
Since the meta data has not changed between versions, it is possible to use the exact same database for 6.0 that the 2.1/5.0 is currently using.

If you are changing the O/S as part of the migration, use the windows client version of NNRie and NNFie. If you are staying with the same O/S you can use the NNFie and NNRie on that O/S for export and import.

To complete this migration, you must use a 2.1/5.0 for export and also the 6.0 for import.

1. Run the NNFie and NNRie utilities on the source machine with WMQI 2.1 or Websphere Message Broker 5.0 for Windows to create the following export files:

nnfie.exp

nnrie.exp

2. On the machine running Websphere Message Broker v6.0 for Windows, map the directory containing these files as a network drive or copy the files to the 6.0 client machine.

3. To import the exported databases into the target into the target Websphere Message Broker v6.0 database, use the command line utilities NNFie and NNRie on Windows.

4. In the Websphere Message Broker v6.0 Formatter GUI, review your literal definitions to confirm that they are correctly defined if your target Broker will be running with a different codepage than the previous broker environment.

Chapter 5
NEONFormatter

This chapter describes NEONFormatter components and includes the following information:

- *Overview*
- *Testing Formats*
- *Removing Formats*
- *Customizing NEONFormatter Using C++*

Internationalization

Rules and Formatter Extension for IBM ® WebSphere Message Broker for Multiplatforms is an internationalized product. When testing formats, you have the option to specify the code set and locale of the input and output messages. For a complete listing of supported codes sets and locales, see *Internationalization* on page 4.

Overview

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

NEONFormatter parses input messages by analyzing input data that it extracts from a relational database and categorizes the data into defined

fields. In addition, NEONFormatter provides the functionality of allowing you to reformat message data to transform or enrich output messages.

You can create the format definitions that direct the parsing of input messages and the reformatting of output messages using the following:

- NEONFormatter graphical user interface (GUI)

  Provides you with ease of use as you populate screens with format definition data.

- NEONFormatter Management API functions

  Provide you with a set of C functions that allow you to write your own applications which call API functions for building format definitions.

To validate format definitions, two executables, apitest and msgtest, show how to invoke the public functions and serve as tools. apitest parses input messages and displays a hierarchical representation of the parse tree. msgtest reformats input messages into output messages.

To verify format definition integrity in the database, you regularly run a Consistency Checker as you build and maintain format definition data.

To export format definitions from a database to an export file and to import from the export file into a database, you can use the NEONFormatter export/import tool, NNFie. For more information, see *Formats and Rules Database Migration* on page 45.

NEONFormatter GUI

The NEONFormatter GUI uses the following components to parse and reformat messages.

Fields and Input Controls

Input controls are used to parse the individual fields of an input message into the following types of information:

- Tags

  Tags are sets of bits or characters explicitly defining a string of data that can proceed or follow a field. For example, <DATE> and

</DATE> might mark the beginning and end of a date field in a message.

- Field length
- Number of field group repetitions

  Repetition count indicates how many times a format or group of fields repeats within a message.

- Literals

  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

  Regular expressions (REs) are strings for defining 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, see the ***NEONFormatter Programming Reference***.

Output Controls

Each field in an input message that is used in an output message must have an associated output 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.

Like input controls, output control definitions contain essential format information, such as the type of math operation, prefix and suffix data, user exit routine, pad characters, and default value.

Formats

You define formats by grouping fields and associating the appropriate input and output controls with that format.

Messages are read by NEONFormatter using individual data fields; however, format definitions can contain several layers of complexity before the actual field values within a message are identified. For example, formats can be flat, containing fields and associated input or output controls, or compound, containing one or more flat or compound formats. Both flat and compound input format definitions contain fields and input controls and are used to parse messages which are reformatted according to flat or compound output format definitions.

You can define and configure formats choosing between two options:

- For ease of use, choose the NEONFormatter GUI.
- NEONFormatter Management APIs. Use Reformat() to translate data derived from input messages into output message guided by the definitions you create for input and output formats, and use Parse() to divide the message into individual fields during translation.

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.

NEONFormatter Management APIs

You can also use the NEONFormatter Management APIs to parse a message. Use the Parse() API to divide a message into the individual fields specified in its input control. After a message is parsed, the intermediate field results can be used for reformatting.

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.

Automatic Format Conversion

High-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.

Reloading Formats

Some versions of MQSeries Integrator allowed the Formatter database to be dynamically reloaded.

- MQSeries Integrator Version 1 performs this action on receiving a message with the OPT_RELOAD option set.
- MQSeries Integrator Version 2.0.1 contains the mqsinrfreload command.
- WebSphere MQ Integrator Version 2.1 contains the mqsinrfreload command. mqsinrfreload works with the NeonRules and NeonFormatter nodes and the NEON domain message parser. It does not work with NEONMap, NEONRulesEvalutaitn, and NEONTransform nodes and the NEONMSG domain message parser. If you make additions or changes to the database, you must stop and restart every broker that must access the changes.

- WebSphere Message Broker contains the mqsireload command. This command is documented in the WebSphere Message Broker online help.

Testing Formats

Two test programs are provided with NEONFormatter:

- apitest

  The apitest program asks the caller for an input file name and an input format name. It treats the contents of the input file as a single message with the input format specified, and then parses it and outputs the structure of the parsed message to standard out.

- msgtest

  The msgtest program asks the caller for an input file name, an output file name, an input format name and an output format name. It treats the contents of the input file as a single message with the input format specified, and then reformats it and writes the resulting output message to the output file.

To test formats:

1. To execute the testing programs, verify that the nnsyreg.dat file includes the following session:

```
Session.new_format_demo
   NNOT_SHARED_LIBRARY=dvdb41db2
   NNOT_FACTORY_FUNCTION=NNSesDB2Factory
   NN_SES_SERVER=<my server name>
   NN_SES_USER_ID=<my user Id>
   NN_SES_PASSWORD=<my password>
   NN_SES_DB_NAME=<my database name>
```

Note:
When accessing the z/OS DB2 database from the mainframe, do not include values for the NN_SES_USER_ID or NN_SES_PASSWORD fields.

For more information, see *Configuring Runtime Parameters* on page 24.

2. To test input and output formats, run msgtest.
3. To validate input formats and view how NEONFormatter interpreted a message, run apitest.

Note:
With the addition of NEONFormatter 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 is written. If [filename] is not specified, debug information is written to the standard output stream.

Running apitest

apitest calls the following APIs to output the structure and content of a input message parsed by NEONFormatter:

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

- ParsedField::GetValue

To run apitest:

1. At the command line prompt, type the following and press ENTER:

   ```
   apitest
   ```

2. At the prompt, type the name of the file in this directory that contains the message to be parsed and reformatted.

   You are prompted for the input file name. This file is in the current directory and contains one message to be parsed.

3. At the prompt, type the name of the input format to be read from the database identified in the nnsyreg.dat file.

Syntax

The syntax for apitest with optional parameters is:

```
apitest
      [-d <filename>]
      [-help]
      [-ics <codeset>]
      [-ilc <locale>]
      [-s <session>]
```

Parameters

Parameter	Mandatory / Optional	Description
-d <filename>	Optional	Specifies the file name to put the parsed fields in to. If not specified, the information is written to the screen.
-help	Optional	Displays the syntax for apitest and exits without parsing the file.

Parameter	Mandatory / Optional	Description
-ics <codeset>	Optional	Specifies the code set of the input message data., such as ibm-1252 or ibm-943, during parsing. If not specified, the system code set defaults. For more information on supported code sets, see *Internationalization* on page 4.
-ilc <locale>	Optional	Specifies the locale of the input message, such as en_US or ja_JP, during parsing. If not specified, the system locale defaults. For more information on supported locales, see *Internationalization* on page 4.
-s <session>	Optional	Specifies the database session. The default is new_format_demo.

Running msgtest

msgtest provides the following functionality:

- Reads the format definitions that you have created with the NEONFormatter GUI or the NEONFormatter Management APIs, such as input and output formats, delimiters, and other control information, from the database.
- Applies this information to an input message it reads from a file while parsing and reformatting that message.
- Adds an input message and an output format to NEONFormatter, reformats the message text provided in an input file, and outputs the results to an output file.

msgtest input and output parameters are:

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

- map name
- map version

msgtest calls the following APIs:

- Formatter::AddInputMessage
- Formatter::AddOutputFormat
- Formatter::Reformat
- Formatter::GetOutMsgGroup
- OutMsgGroup::GetName
- OutMsgGroup::GetMsgCount
- OutMsgGroup::GetMsg
- OutMsg::GetMsgBuffer
- OutMsg::GetMsgLength
- OutMsg::GetName
- OutMsg::GetSubscriptionList
- OutMsg::GetSubscriptionLength
- RulesSubscriptionList::getFirst
- RulesSubscriptionList::getNext
- RulesSubscription::getActionList
- RulesActionList::getFirst
- RulesActionList::getNext
- RulesAction::getOptionList
- RulesOptionList::getFirst
- RulesOptionList::getNext
- RulesOption::getName
- RulesOption::getValue

To run msgtest:

1. At the command line prompt, type the following and press ENTER:

   ```
   msgtest
   ```

2. At the prompt, *Enter the input file name:*, type the name of the file in this directory that contains the message to be parsed and reformatted.
3. At the prompt, *Enter the output file name:*, type the name of the file that will contain the reformatted message.
4. At the prompt, *Enter the input format name:*, type the name of the input format that will be read from the database identified in the nnsyreg.dat file.
5. At the prompt, *Enter the output format name:*, type the name of the output format that will be read from the database identified in the nnsyreg.dat file.

Tip:

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

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

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

The myFormatterTest.txt file contains:

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

Syntax

The syntax for msgtest with optional parameters is:

```
msgtest
      [-d <filename>]
      [-dcp]
      [-dcm]
      [-dco]
      [-help]
      [-ics <codeset>]
      [-ilc <locale>]
      [-li]
      [-lo]
      [-lf]
      [-map]
      [-nv]
      [-ocs <codeset>]
      [-olc <locale>]
      [-s <session>]
```

Parameters

Parameter	Mandatory / Optional	Description
-d <filename>	Optional	Specifies the file name to debug.
-dcp	Optional	Turns on debug parse for the specified file.
-dcm	Optional	Turns on debug map for the specified file.
-dco	Optional	Specifies the output file for debug information. If not specified, debug information is written to the screen (STDOUT).
-help	Optional	Displays a Help message and exits.
-ics <codeset>	Optional	Specifies the code set of the input message data., such as ibm-1252 or ibm-943, during parsing. If not specified, the system code set defaults. For more information on supported code sets, see *Internationalization* on page 4.

Parameter	Mandatory / Optional	Description
-ilc <locale>	Optional	Specifies the locale of the input message, such as en_US or ja_JP, during parsing. If not specified, the system locale defaults. For more information on supported locales, see *Internationalization* on page 4.
-li	Optional	Prints the additional information about how the message was parsed.
-lo	Optional	Prints additional information about how the input field data is mapped to the output field in the message during the reformat.
-lf	Optional	Prints additional information about the data transformations of each field in the message.
-map	Optional	Prompts for map name and version for reformatting.
-nv	Optional	Disables user type validation.
-ocs <codeset>	Optional	Specifies the code set of the output message data., such as ibm-1252 or ibm-943, during reformatting. If not specified, the system code set defaults. For more information on supported code sets, see *Internationalization* on page 4.
-olc <locale>	Optional	Specifies the locale of the output message, such as en_US or ja_JP, during reformatting. If not specified, the system locale defaults. For more information on supported locales, see *Internationalization* on page 4.
-s <session>	Optional	Specifies the database session. The default is new_format_demo.

Removing Formats

NEONFormatter provides the utility, NNFmtRmv, to remove unwanted input or output formats and their associated components, such as fields, parse controls, literals, and output controls. NNFmtRmv maintains referential integrity by verifying that components have no additional associations with other NEONFormatter components before removing them

You must have user permission to remove formats.

To remove formats:

1. Before removing formats, verify that the following session is in the nnsyreg.dat file:

```
Session.nnfmgr
   NNOT_SHARED_LIBRARY=dvdb41db2
   NNOT_FACTORY_FUNCTION=NNSesDB2Factory
   NN_SES_SERVER=<my server name>
   NN_SES_USER_ID=<my user Id>
   NN_SES_PASSWORD=<my password>
   NN_SES_DB_NAME=<my database name>
```

Note:
When accessing the z/OS DB2 database from the mainframe, do not include values for the NN_SES_USER_ID or NN_SES_PASSWORD fields.

For more information, see *Configuring Runtime Parameters* on page 24.

2. To remove formats, type the following and press ENTER at the command prompt:

```
NNFmtRmv remove[ -fN <format_name>]+
```

Examples

```
NNFmtRmv remove -fN flat_format_1
NNFmtRmv remove -fN flat_format -fN flat_format_2
NNFmtRmv remove -fN compound_format
```

Customizing NEONFormatter Using C++

You can customize NEONFormatter by creating user exits or creating your own custom date and time formats.

For example, you can create a user exit when the standard reformatting types of NEONFormatter do not meet your data needs. Use the user exit APIs provided with NEONFormatter to create your own user exits.

User exits function in the following manner:

- NEONFormatter calls a routine named NNGetUserExitFunctionPtrs () to provide the function address. The user-defined exit code shares a location with NEONFormatter.
- 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.

For more information on user exits, see the ***NEONFormatter Programming Reference***.

In addition, you may have special needs that require tailoring your message date and time format to meet specific requirements. In this case, use the AddDTfmt utility to add your custom date and time format string to the NEONFormatter database. You can also migrate any properly created custom date and time formats from one database to another NEONFormatter database.

Creating C++ User Exits

Creating a user exit includes the following steps:

- Creating user exit and user exit cleanup functions. Functions must conform to the following types:

  NN_EXIT_FUNC_t

NN_EXIT_CLEANUP_FUNC_t

- Modifying the routine, NNGetUserExitFunctionPtrs () so that a NEONFormatter instance can look up the function pointers for the user exit and user exit cleanup functions given an exit function name.
- Setting the shared library path environment variable so that the newly-created user exit library in the <customer>/lib directory is found before the stub version in the /usr/lpp/wmqi/Nnsy/lib:$LIBPATH directory.
- Building a user exit library.
- Specifying the name of the exit routine in the Exit Routine field on the Field Format Output Control Tool window of the NEONFormatter GUI.

To assist you in creating user exits, the $(INSTALL) /examples/NNSYrf directory provides the following files:

File	Description
userexit.cpp	This file contains sample user exit code.
userexit.exp	This is an NNFie export file that contains input and output format definitions that exercise the user exit.
userexit.txt	This file contains an input message that can be used to test the formats and user exit above.

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 /cust/lib to determine if already cached
IF user exit library not cached,
  THEN
     Formatter caches the user exit library, according to the
     shared library path environment variables
  ENDIF
Call NNGetUserexitFunction Ptrs() with the user exit routine
  name.
```

```
IF exit function pointer is not null, THEN
  call user exit function
  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 NEONFormatter reformat process to fail.

Replacing the Lookup Stub Function

The user-defined exit function replaces the lookup stub. However, a stub version of NNGetUserExitFunctionPtrs () is defined in the user exit library in the installation directory.

To override the stub function:

1. Create the replacement function, compile it, and then link it, using the example template that is provided with the product. A sample makefile is available in the /examples directory.
2. To ensure that the replacement user exit lookup function is used instead of the stub function, place the PATH environment variable that has the name of the directory containing your user exit ahead of the library directory.

Building a C++ User Exit Library

To successfully complete this procedure, observe the following platform-specific requirements.

For Windows:

- Use a version of Make for Windows (such as the MKS Toolkit Make).
- Only modify ROOT and map drives as specified in this procedure.
- Verify that the following drives are mapped. These drives are supplied in the $(INSTALL)/examples/Makefile.

  K:\ - The location of the appropriate database installation files.

  L:\ - The Microsoft Visual Studio .NET directory including vc7/bin and vc7/lib.

For HP-UX:

- Set the NLS_LANG environment variable to AMERICAN.
  Example:
  export NLS_LANG=AMERICAN

WARNING!
On z/OS, it is not recommended that you call DB2 from a user exit because you may receive unpredictable results.

To build a C++ User Exit library:

1. Modify the following values in the $(INSTALL)/examples/NNSYrf/Makefile:

Directory	Description
ROOT	The location of the installation $(INSTALL).
COMPILER.inc	The location of the appropriate compiler include files.
COMPILER.lib	The location of the appropriate compiler libraries.
DB_HOME	The location of the appropriate database installation files.

2. Put the userexit code in the examples directory in the file userexit.cpp.
3. Type the appropriate make command to create the user exit library:

Operating System	Command
Windows	make nnuserexit.dll
Solaris, Linux	make libnnuserexit.so
AIX	make libnnuserexit.so
HP-UX	make libnnuserexit.sl
z/OS	make nnuserexit.a

If you are using Solaris, Linux, AIX, or HP-UX, the commands in the previous table create the actual library with the name specified, as well as a symbolic link to that library without the .1 at the end.

4. Choose one of the following:

   - Modify the library path so that the loader picks up the new user exit library before the stub library that came with your installation.
   - Save the stub library in another location and put the new library in the directory that contains the NEONRules and NEONFormatter libraries.

Operating System	Environment Variable
Windows	PATH
Solaris, Linux	LD_LIBRARY_PATH
AIX	LIBPATH
HP-UX	SHLIB_PATH
z/OS	LIBPATH

C++ User Exit Example

You can use the code files provided with NEONFormatter as a user exit example. To locate the files, go to nnsy\rulfmt56\examples\NNSYrf directory.

Sample Filemame	Description
Makefile	"make" input file used to build the example executables and libraries.
userexit.cpp	C++ source file containing NNGetUserExitFuncPtrs lookup function and example User Exit functions.
userexit.exp	NNFie export file with formats required by the example.
msgtest.cpp	Test application to invoke NEONFormatter.
userexit.txt	Sample data file for input to msgtest.

The following example shows a simple conversion of English length measurements to Metric. Input messages are in the form:

<value> <space> <unit> <newline>

Sample Field	Description
<value>	numeric decimal number
<space>	single blank space (hex 20)
<unit>	literals defined as "inch", "foot", "yard", or "mile"
<newline>	line break

The input message, "37.35 inch", in userexit.txt is run using msgtest. Message processing directs the message to the example user exit that performs the conversion. The reformatted message should be returned as "94.869000 centimeter".

User Exit Data Conversion

After the user exit function returns its data, NEONFormatter converts the input data to output. You can choose from the following two options to convert your data:

- To base the output data type on the input data type, set the return data type in the user exit to be the same type as the input type.
- To use the return value of the user exit type on the output data, set the output control to data type *Not Applicable*.

Threading User Exits

When threading user exits on C++, use a re-entrant design to ensure upward compatibility.

Creating Custom Date and Time Formats

The NEONFormatter GUI provides an extensive list of commonly used date and time formats from which to choose. To view the packaged list, see *Data Types* on page 171.

If these packaged formats do not meet your specific message needs, you can use the AddDTfmt utility to create your own custom date and time format string and add it to the NEONFormatter database. You can migrate any properly created custom date and time format from one NEONFormatter database to another. However, once created you cannot delete or modify an existing format.

This AddDTfmt utility is installed in the product bin directory during the product installation.

To create a custom date and time format:

To create a custom format, you must create a string using any of the following New Era of Networks date and time format specifiers:

YY = 2-digit year

YYYY = 4-digit year

MON = 3-letter month name abbreviation

MN = 2-digit month (1-2)

DD = 2-digit day of the month (1-31)

HH = 2-digit hour (0-23)

MM = 2-digit minute (0-59)

SS = 2-digit second (0-59)

AM = AM indicator

PM = PM indicator

JJJ = 3-digit day of year

The format specification string cannot contain both 2-digit and 4-digit year specifiers or numeric months and abbreviated month names. The Julian date must also include either a 2-digit or 4-digit year specification.

For ASCII and EBCDIC underlying data types, the format specifiers can be in any order and can include any number of spaces or delimiters.

If you choose to enter spaces as separators, you must precede and end the entire format string with the quote (") character. For separators, you can use any characters, such as punctuation, that are not included in the list of New Era of Networks format specifiers.

1. To connect AddDTfmt to the NEONFormatter database, go to the examples\NNSYrf directory.
2. Open the nnsyreg.dat file, and do the following:
   a. Uncomment the set of session information that applies to the database you are using.
   b. Create a Date Format session using the following Oracle model:

```
Session.add_date_fmt
   NNOT_SHARED_LIBRARY=dvdb41ora
   NNOT_FACTORY_FUNCTION=NNSesOra8Factory
   NN_SES_SERVER=<server name>
   NN_SES_USER_ID=<user ID>
   NN_SES_PASSWORD=<password>
   NN_SES_DB_NAME=<database name>
```

   c. Create a Date Format session for DB2 using the following model:

```
Session.add_date_fmt
   NNOT_SHARED_LIBRARY=dvdb41db2
   NNOT_FACTORY_FUNCTION=NNSesDB2Factory
   NN_SES_SERVER=db2serv
   NN_SES_USER_ID=<user ID>
   NN_SES_PASSWORD=<password>
   NN_SES_DB_NAME=<database name>
```

3. Return to the command prompt, type the following and press ENTER:

   ```
   AddDTfmt <date format>
   ```

4. To verify that the new format was added to the NEONFormatter database, do the following:

   a. Open the NEONFormatter GUI.

   b. Display an Input or Output Controls Properties sheet.

   c. Scroll through the Data Type and Data Format drop-down list and verify that your custom format was added to the list of selections.

Note:
Use alpha characters and any other printable characters to populate your custom format. This selection is made when defining input or output formats in the NEONFormatter GUI.

Chapter 6
NEONRules

This chapter describes the NEONRules component. It includes the following information:

- *Overview*
- *Testing Rules*

Overview

NEONRules evaluates the contents of a message and uses the evaluation results to perform actions on the message. Users build and modify rule definitions using one of two methods:

- NEONRules graphical user interface (GUI)
- NEONRules Management API functions

The NEONRules GUI allows the user to define a rule 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 following test executables are delivered with NEONRules:

- 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.

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

and maintained, users should run the Consistency Checker periodically to ensure data integrity. For more information, see *Chapter 7: Consistency Checker* on page 153.

NNRie is a command line tool used to export rule definitions from a database to a file and to import the exported file into a database. NNRie can import from a Rules and Formatter Extension for IBM ® WebSphere Message Broker for Multiplatforms 5.6 export file into a Rules and Formatter Extension for IBM ® WebSphere Message Broker for Multiplatforms 5.6 database. NNRie version 1.1 only exports data from a 1.1 database. For more information, see *Chapter 4: Formats and Rules Database Migration* on page 45.

NEONRules Components

The NEONRules components are:

- Application groups
- Message types
- Rules

Application Groups

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

Message Types

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

Rules

A rule is an expression that tests the properties of the data in an input message. A rule contains specific actions to be processed by the application if the rule evaluates to true against a message. These actions can be thought of as computer commands and the associated parameters required to execute the rule. Each rule is uniquely identified by its application group / message type / rule name.

The following three items must be defined for each rule:

- Evaluation criteria
- Subscriptions
- Permissions

Expressions, Arguments, Boolean, and Rules Operators

An expression is a set of evaluation criteria and it contains a list of fields, associated operators, and associated comparison data (either static values or other fields) connected with Boolean operators. An argument contains the combination of a field name, Rules comparison operator, and static value or other field name when using NEONFormatter. Field names depend on the message type, which is the input format name. Field names are 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), | (OR), and parentheses can be used to set the evaluation priority. For more information on operators, see the ***NEONRules Programming Reference***.

Subscriptions, Actions, and Properties

When a rule evaluates to true, it is considered a hit. If the rule does not evaluate to true, it is considered a no-hit. When a rule hits, you can retrieve associated subscriptions. The subscriptions are the actions or commands and the associated parameters are the message properties to execute them. If the parse of the input data fails, the message is sent to the failure queue.

Subscriptions are lists of actions to take when a message evaluates to true. Each rule must have at least one associated subscription. Subscriptions are

uniquely identified within an application group / message type pair by a user-defined subscription name. Permissions must be defined for subscriptions in the same way they are for rules. You can define as many subscriptions as you need. Each action within a subscription is defined by an action name. The action does not need to be unique, since all actions are intended to be executed in sequence. A single subscription can be shared by multiple rules. In this case, the shared subscription would be retrieved only once no matter how many of its rules hit.

An action has a list of one or more associated message properties. A message property consists of a property name-value pair. The user defines all action names and property name-value pairs.

Permissions

When you add a rule, you are assigned rule ownership. You must assign permissions to the rule if you want other users to be able to update your rule. The default setting is Read-only permission.

For more information, see *Using NNRie with DB2* on page 87.

NEONRules APIs

The two types of NEONRules APIs are:

- NEONRules APIs — Evaluates rules and retrieves subscription, hit, and no-hit information. Before you evaluate a rule, the rule must exist and you must use CreateRulesEngine() to create a VRule object. After that, you can do as many evaluations and subscription retrievals as needed.
- NEONRules Management APIs — Maintains rule information. Add, Read, and Update APIs are available as well as APIs to delete an entire rule or subscription and all associated information.

Reloading Rules

Some versions of MQSeries Integrator allowed the Rules database to be dynamically reloaded.

- MQSeries Integrator Version 1 performs this action on receiving a message with the OPT_RELOAD option set.

- MQSeries Integrator Version 2.0.1 contains the mqsinrfreload command.
- WebSphere MQ Integrator Version 2.1 contains the mqsinrfreload command. mqsinrfreload works with the NeonRules and NeonFormatter nodes and the NEON domain message parser. It does not work with NEONMap, NEONRulesEvalutaitn, and NEONTransform nodes and the NEONMSG domain message parser. If you make additions or changes to the database, you must stop and restart every broker that must access the changes.
- WebSphere Message Broker contains the mqsireload command. This command is documented in the WebSphere Message Broker online help.

Testing Rules

Rules are tested using ruletest and NNRtrace. These utilities are described in the following sections.

ruletest

ruletest reads a message from a file and evaluates the message using the application group and message type defined either on the command line or, if ruletest is being used interactively, input by the user at the command line prompts. After evaluation, subscriptions are retrieved as they would normally be retrieved and output to the screen, but the subscriptions are not executed using queuing and NEONRules.

ruletest can be executed using two methods:

- ruletest evaluates the message using the specified application group and message type if the parameters were entered listed at command time.
- ruletest can be used interactively by providing no command line parameters. When ruletest is called with no command line parameters, the user is prompted for parameters and for the message codeset and locale. If you press ENTER at the message codeset and

message locale prompts, the codeset and locale system default is used. In interactive mode, ruletest loops through the prompt, optional reload, and evaluation steps so many evaluations can be done using the same session.

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

Syntax

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

Parameters

Name	Mandatory / Optional	Description
-i <input filename>	Mandatory	The input file from which ruletest reads. The file must reside in the directory that the process is run from or the fully qualified path must be provided.
-a <application group>	Mandatory	The ruletest program requires this parameter to evaluate rules.
-m <message type>	Mandatory	The ruletest program requires this parameter to evaluate rules.
-s <session name>	Optional	The session name tag from configuration file. The default session name is rules.
-v (verbose)	Optional	The ruletest program logs to the screen if this optional parameter is set. The process defaults to no logging.

Name	Mandatory / Optional	Description
-? (usage)	Optional	The ruletest program displays all usage parameters.

Remarks

- Before running ruletest, a complete and valid installation of NEONFormatter must exist and the database must be running in a stable state.
- The ruletest program requires a connection to a single database containing both NEONFormatter and NEONRules data. The database session information must be defined in the configuration file; for more information, see *Specifying Database Session Information* on page 26.
- The ruletest program uses NEONFormatter to evaluate messages only; the ruletest program does not execute actions.
- The ruletest program uses NEONRules for evaluating and retrieving subscriptions.

Example ruletest calls

Case 1

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

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

Case 2

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

Case 2 uses "test" session to connect to database. Evaluates message from inputfile.txt against rule set defined by TestApp and TestFmt. Does not perform any subscription actions, just lists them.

Case 3

```
ruletest
```

Case 3 prompts for the required input: session name, input filename, application group name, message type name, verbose, reload, codeset, and locale.

This continues to evaluate messages until an empty line is entered for the required input.

NNRtrace

NNRtrace is a rules debugging utility. This utility evaluates a rule and the message associated with the rule to determine whether the rule hits. 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 NNRtrace, create an input file for the test procedure. Verify that the configuration file includes the database name and server name information and that NNRtrace resides in the same directory as the configuration file.

Syntax

```
NNRtrace  -i <input file name>
          -a <application group>
          -m <message type>
          -r <rule name>
         [-s <session name>]
         [-o <output file name>]
         [-e <encoding>]
         [-l <locale>]
         [-v]
```

If command line parameters are not specified, the user is prompted for the required information.

Parameters

Name	Mandatory / Optional	Description
-i <input filename>	Mandatory	The input file. This file must reside in the directory that the process is run from or the fully qualified path must be provided.
-a <application group>	Mandatory	Required parameter to specify the rule.
-m <message type>	Mandatory	Required parameter to specify the rule.
-r <rule name>	Mandatory	Required parameter to specify the rule.
-s <session name>	Optional	Rules session name corresponding to the session name in the configuration file. The default session name is rules.
-o <output file name>	Optional	The output file to which results of the program are written. The process defaults to standard output if this parameter is not specified.
-e<encoding>	Optional	Specifies the codeset.
-l<locale>	Optional	Specifies the locale of the input message, such as en_US or ja_JP, during parsing. If not specified, the system locale defaults. For more information on supported locales, see *Internationalization* on page 4.
-v (verbose)	Optional	The program logs to the screen. The default parameter is no logging.

Chapter 7

Consistency Checker

This chapter describes the Consistency Checker utility. It includes the following information:

- *Overview*
- *Running the Consistency Checker*

Overview

The Consistency Checker is a utility that determines NEONRules and NEONFormatter database consistency by verifying internal structure of rules and formats, user-typed data quality, and whether its records have corresponding database features.

The verification process identifies asynchronous objects that became invalid as a result of a recovery or improper database upgrade.

Running the Consistency Checker

It is recommended that you verify database consistency before you begin the upgrade process. This includes running the Consistency Checker twice:

- Before exporting data from an existing database
- After importing data into a new database

To run the Consistency Checker on Windows or UNIX:

1. Determine the Consistency Checker version number that is compatible with your existing database. For more information, see *Determining Compatibility* on page 48.
2. From the root of your Rules and Formatter Extension for IBM ® WebSphere Message Broker for Multiplatforms software installation, type the following at the command prompt and press ENTER:

   ```
   cd install.sql/cc
   ```

3. Change to the subdirectory that corresponds with the database type and version that you are checking.
4. To run the Consistency Checker, choose one of the following:

   - To check the consistency of your formats, type the following and press ENTER:

     For Windows

     ```
     formatcc520.cmd <user id> <password> <server name>
       <database name> <locale name>
     ```

     For UNIX

     ```
     formatcc520.sh <user id> <password> <server name>
       <database name> <locale name>
     ```

   - To check the consistency of your rules, type the following and press ENTER:

     For Windows

     ```
     rulecc520.cmd <user id> <password> <server name>
       <database name> <locale name>
     ```

     For UNIX

     ```
     rulecc520.sh<user id> <password> <server name>
       <database name> <locale name>
     ```

   For more information on locale names, see *Internationalization* on page 4.

Note:
For DB2, do not include the <database name> parameter and enter your database name as the <server name> parameter.

5. To determine the consistency of your database, review the report generated in the log file.

To run the Consistency Checker on z/OS:

1. Run the Consistency Checker for formats by using MAINFCC in the SBIPJCL library.
2. Run the Consistency Checker for rules by using MAINRCC in the SBIPJCL library.
3. Run the Consistency Checker for rules permissions by using PERMCC in the SBIPJCL library.
4. Repair any data inconsistencies.

Note:
To run the Consistency Checker on databases previous to version 5.6, see the ***System Management Guide*** for that specific version.

Appendix A
Character Sets

For blank entries, see the relevant EBCDIC code pages.

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.

Decimal Value	Hex Value	ASCII Extended Value	EBCDIC Value	EBCDIC Description	Binary
000	00	NUL	NUL	Null	0000 0000
001	01	SCH	SOH	Start of Heading	0000 0001
002	02	STX	STX	Start of Text	0000 0010
003	03	ETX	ETX	End of Text	0000 0011
004	04	EOT	SEL	Select	0000 0100
005	05	ENO	HT	Horizontal Tab	0000 0101
006	06	ACK	RNL	Required New Line	0000 0110
007	07	BEL	DEL	Delete	0000 0111
008	08	BS	GE	Graphic Escape	0000 1000
009	09	HT	SPS	Superscript	0000 1001
010	0A	LF	RPT	Repeat	0000 1010
011	0B	VT	VT	Vertical Tab	0000 1011
012	0C	FF	FF	Form Feed	0000 1100

Decimal Value	Hex Value	ASCII Extended Value	EBCDIC Value	EBCDIC Description	Binary
013	0D	CR	CR	Carriage Return	0000 1101
014	0E	SO	SO	Shift Out	0000 1110
015	0F	SI	SI	Shift In	0000 1111
016	10	DLE	DLE	Data Link Escape	0001 0000
017	11	DC1	DC1	Device Control 1	0001 0001
018	12	DC2	DC2	Device Control 2	0001 0010
019	13	DC3	DC3	Device Control 3	0001 0011
020	14	DC4	RES/ENP	Restore/Enable Presentation	0001 0100
021	15	NAK	NL	New Line	0001 0101
022	16	SYN	BS	Backspace	0001 0110
023	17	ETB	POC	Program-Operator Communication	0001 0111
024	18	CAN	CAN	Cancel	0001 1000
025	19	EM	EM	End of Medium	0001 1001
026	1A	SUB	UBS	Unit Backspace	0001 1010
027	1B	ESCAPE	CU1	Customer Use 1	0001 1011
028	1C	FS	IFS	Interchange File Separator	0001 1100
029	1D	GS	IGS	Interchange Group Separator	0001 1101
030	1E	RS	IRS	Interchange Record Separator	0001 1110

Decimal Value	Hex Value	ASCII Extended Value	EBCDIC Value	EBCDIC Description	Binary
031	1F	US	IBT / IUS	Intermediate Transmission Block / Interchange Unit Separator	0001 1111
032	20	SPACE	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
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	0			0011 0000

Decimal Value	Hex Value	ASCII Extended Value	EBCDIC Value	EBCDIC Description	Binary
049	31	1			0011 0001
050	32	2	SYN	Synchronous Idle	0011 0010
051	33	3	IR	Index Return	0011 0011
052	34	4	PP	Presentation Position	0011 0100
053	35	5	TRN	Transparent	0011 0101
054	36	6	NBS	Numeric Backspace	0011 0110
055	37	7	EOT	End of Transmission	0011 0111
056	38	8	SBS	Subscript	0011 1000
057	39	9	IT	Indent Tab	0011 1001
058	3A	:	RFF	Required Form Feed	0011 1010
059	3B	;	CU3	Customer Use 3	0011 1011
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	A	RSP		0100 0001
066	42	B			0100 0010
067	43	C			0100 0011
068	44	D			0100 0100
069	45	E			0100 0101

Decimal Value	Hex Value	ASCII Extended Value	EBCDIC Value	EBCDIC Description	Binary
070	46	F			0100 0110
071	47	G			0100 0111
072	48	H			0100 1000
073	49	I			0100 1001
074	4A	J	¢		0100 1010
075	4B	K	.		0100 1011
076	4C	L	<		0100 1100
077	4D	M	(		0100 1101
078	4E	N	+		0100 1110
079	4F	O	|		0100 1111
080	50	P	&		0101 0000
081	51	Q			0101 0001
082	52	R			0101 0010
083	53	S			0101 0011
084	54	T			0101 0100
085	55	U			0101 0101
086	56	V			0101 0110
087	57	W			0101 0111
088	58	X			0101 1000
089	59	Y			0101 1001
090	5A	Z	!		0101 1010
091	5B	[	$		0101 1011

Decimal Value	Hex Value	ASCII Extended Value	EBCDIC Value	EBCDIC Description	Binary
092	5C	\	*		0101 1100
093	5D	]	)		0101 1101
094	5E	^	;		0101 1110
095	5F	-	¬		0110 1111
096	60	‘	-		0110 0000
097	61	a	/		0110 0001
098	62	b			0110 0010
099	63	c			0110 0011
100	64	d			0110 0100
101	65	e			0110 0101
102	66	f			0110 0110
103	67	g			0110 0111
104	68	h			0110 1000
105	69	i			0110 1001
106	6A	j	|		0110 1010
107	6B	k	,		0110 1011
108	6C	l	%		0110 1100
109	6D	m	_		0110 1101
110	6E	n	>		0110 1110
111	6F	o	?		0110 1111
112	70	p			0111 0000
113	71	q			0111 0001

Decimal Value	Hex Value	ASCII Extended Value	EBCDIC Value	EBCDIC Description	Binary
114	72	r			0111 0010
115	73	s			0111 0011
116	74	t			0111 0100
117	75	u			0111 0101
118	76	v			0111 0110
119	77	w			0111 0111
120	78	x			0111 1000
121	79	y			0111 1001
122	7A	z	:		0111 1010
123	7B	{	#		0111 1011
124	7C	|	@		0111 1100
125	7D	}	'		0111 1101
126	7E	~	=		0111 1110
127	7F	DEL	"		0111 1111
128	80	Ä	Ä		1000 0000
129	81	unused	a		1000 0001
130	82	,	b		1000 0010
131	83	*f*	c		1000 0011
132	84	„	d		1000 0100
133	85	…	e		1000 0101
134	86	†	f		1000 0110
135	87	‡	g		1000 0111

Decimal Value	Hex Value	ASCII Extended Value	EBCDIC Value	EBCDIC Description	Binary
136	88	ˆ	h		1000 1000
137	89	‰	i		1000 1001
138	8A	Š			1000 1010
139	8B	‹			1000 1011
140	8C	Œ			1000 1100
141	8D	unused			1000 1101
142	8E	unused			1000 1110
143	8F	unused			1000 1111
144	90	unused			1001 0000
145	91	‘	j		1001 0001
146	92	‘	k		1001 0010
147	93	“	l		1001 0011
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	š			1001 1010
155	9B	›			1001 1011
156	9C	œ			1001 1100
157	9D	unused			1001 1101

Decimal Value	Hex Value	ASCII Extended Value	EBCDIC Value	EBCDIC Description	Binary
158	9E	unused			1001 1110
159	9F	Ÿ			1001 1111
160	A0	nonbreaking space			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
170	AA	ª			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	²			1011 0010

Decimal Value	Hex Value	ASCII Extended Value	EBCDIC Value	EBCDIC Description	Binary
179	B3	³			1011 0011
180	B4	´			1011 0100
181	B5	µ			1011 0101
182	B6	¶			1011 0110
183	B7	·			1011 0111
184	B8	¸			1011 1000
185	B9	¹			1011 1001
186	BA	º			1011 1010
187	BB	»			1011 1011
188	BC	¼			1011 1100
189	BD	½			1011 1101
190	BE	¾			1011 1110
191	BF	¿			1011 1111
192	C0	À	{		1100 0000
193	C1	Á	A		1100 0001
194	C2	Â	B		1100 0010
195	C3	Ã	C		1100 0011
196	C4	Ä	D		1100 0100
197	C5	Å	E		1100 0101
198	C6	Æ	F		1100 0110
199	C7	Ç	G		1100 0111
200	C8	È	H		1100 1000

Decimal Value	Hex Value	ASCII Extended Value	EBCDIC Value	EBCDIC Description	Binary
201	C9	É	I		1100 1001
202	CA	Ê	SHY		1100 1010
203	CB	Ë			1100 1011
204	CC	Ì			1100 1100
205	CD	Í			1100 1101
206	CE	Î			1100 1110
207	CF	Ï			1100 1111
208	D0	Ð	}		1101 0000
209	D1	Ñ	J		1101 0001
210	D2	Ò	K		1101 0010
211	D3	Ó	L		1101 0011
212	D4	Ô	M		1101 0100
213	D5	Õ	N		1101 0101
214	D6	Ö	O		1101 0110
215	D7	×	P		1101 0111
216	D8	Ø	Q		1101 1000
217	D9	Ù	R		1101 1001
218	DA	Ú			1101 1010
219	DB	Û			1101 1011
220	DC	Ü			1101 1100
221	DD	Ý			1101 1101
222	DE	Þ			1101 1110

Decimal Value	Hex Value	ASCII Extended Value	EBCDIC Value	EBCDIC Description	Binary
223	DF	ß			1101 1111
224	E0	à	\		1110 0000
225	E1	á			1110 0001
226	E2	â	S		1110 0010
227	E3	ã	T		1110 0011
228	E4	ä	U		1110 0100
229	E5	å	V		1110 0101
230	E6	æ	W		1110 0110
231	E7	ç	X		1110 0111
232	E8	è	Y		1110 1000
233	E9	é	Z		1110 1001
234	EA	ê			1110 1010
235	EB	ë			1110 1011
236	EC	ì			1110 1100
237	ED	í			1110 1101
238	EE	î			1110 1110
239	EF	ï			1110 1111
240	F0	ð	0		1111 0000
241	F1	ñ	1		1111 0001
242	F2	ò	2		1111 0010
243	F3	ó	3		1111 0011
244	F4	ô	4		1111 0100

Decimal Value	Hex Value	ASCII Extended Value	EBCDIC Value	EBCDIC Description	Binary
245	F5	õ	5		1111 0101
246	F6	ö	6		1111 0110
247	F7	÷	7		1111 0111
248	F8	ø	8		1111 1000
249	F9	ù	9		1111 1001
250	FA	ú			1111 1010
251	FB	û			1111 1011
252	FC	ü			1111 1100
253	FD	ý			1111 1101
254	FE	þ			1111 1110
255	FF	ÿ	EO	Eight Ones	1111 1111

Appendix A

Appendix B
Data Types

Not Applicable

No data type is assumed.

String

A string of character data that is associated with a code set. For more information on conversion, see *Data Type Conversion* on page 180.

Numeric

A string of standard ASCII numeric characters.

Binary

The Binary data type is used to parse any value and transform that value to an ASCII representation of the value internally in NEONFormatter. 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

A string of characters encoded using the EBCDIC (Extended Binary Coded Decimal Interchange Code) encoding that is used on larger IBM computers. During a reformat from EBCDIC to ASCII, if a character being converted is not in the EBCDIC character set, the conversion results in a space (hexadecimal 20).

Note:
The EBCDIC data type should not be used for new applications. It exists only to support previous user formats. If data needs to be encoded in EBCDIC, you must implement one of the supported EBCDIC code sets. For more information on supported code sets, see ***Rules and Formatter Extension for IBM ® WebSphere Message Broker for Multiplatforms User's Guide***.

IBM Packed Integer

Data type on larger IBM computers used to represent integers in compact form. Each byte represents two decimal digits, one in each nibble of the byte. The final nibble is always a hexadecimal F. For example, the number 1234 is stored as a 3-byte value: 01 23 4F (the number pairs show the hexadecimal values of the nibbles of each byte). The number 12345 is stored as a 3-byte value: 12 34 5F. There is no accounting for the sign of a number; all numbers are assumed to be positive.

IBM Signed Packed Integer

Data type on larger IBM computers used to represent integers in compact form. This data type takes into account the sign (positive or negative) of a number. Each byte represents two decimal digits, one in each nibble of the byte. The final nibble is a hexadecimal C if the number is positive, and a hexadecimal D if the number is negative.

The following example illustrates how to generate a default value for an IBM Packed Integer:

The input data type is IBM Signed Packed Decimal, with a default ASCII value of -12345. The control is optional and there is no corresponding field in the input message, so NEONFormatter uses the default value, converts it to

IBM Signed Packed Decimal, and generates the following output: 12 34 5D. Each pair of numbers represents the two nibbles of a byte. The result is three bytes long.

IBM Zoned Integer

Data type on larger IBM computers used to represent integers. Each decimal digit is represented by a byte. The left nibble of the byte is a hexadecimal F. The right nibble is the hexadecimal value of the digit. For example, 1234 is represented as F1 F2 F3 F4 (the number pairs show the hexadecimal values of the nibbles of each byte).

IBM Signed Zoned Integer

Data type on larger IBM computers used to represent integers. Each decimal digit is represented by a byte. The left nibble of each byte, except the last byte, is a hexadecimal F. The left nibble of the last byte is a hexadecimal C if the number is positive, and a hexadecimal D if the number is negative. The right nibble of each byte is the hexadecimal value of the digit. For example, 1234 is represented as F1 F2 F3 C4 (the number pairs show the hexadecimal values of the nibbles of each byte). -1234 is represented as F1 F2 F3 D4.

Little Endian 2

Two-byte integer where the bytes are ordered with the rightmost byte being the high order or most significant byte. For example, the hexadecimal number 0x0102 is stored as 02 01 (where the number pairs show the hexadecimal values of the nibbles of a byte).

Little Swap Endian 2

Two-byte integer where the two bytes are swapped with respect to a Little Endian 2 value. For example, the hexadecimal number 0x0102 is stored as 01 02.

Little Endian 4

Four-byte integer where two bytes of each word are swapped with respect to Little Endian 4. For example, the hexadecimal number 0x01020304 is stored as 03 04 01 02.

Little Swap Endian 4

Four-byte integer where the bytes are ordered with the rightmost byte being the high order or more significant byte. For example, the hexadecimal number 0x0102 is stored as 02 01, where the number pairs show the hexadecimal values of the nibbles as a byte.

Big Endian 2

Two-byte integer where the bytes are ordered with the leftmost byte being the high order or most significant byte. For example, the hexadecimal number 0x0102 is stored as 01 02 (where the number pairs show the hexadecimal values of the nibbles of a byte).

Big Swap Endian 2

Two-byte integer where the two bytes are swapped with respect to a Big Endian 2 value. For example, the hexadecimal number 0x0102 is stored as 02 01.

Big Endian 4

Four-byte integer where the bytes are ordered with the leftmost byte being the high order or most significant byte. For example, the hexadecimal number 0x01020304 is stored as 01 02 03 04 (where the number pairs show the hexadecimal values of the nibbles of a byte).

Big Swap Endian 4

Four-byte integer where the two bytes of each word are swapped with respect to a Big Endian 4 value. For example, the hexadecimal number 0x01020304 is stored as 02 01 04 03.

Decimal, International

Data type where every third number left of the decimal point is preceded by a period. The decimal point is represented by a comma. Numbers right of the decimal point represent a fraction of one unit. For example, the number 12345.678 is represented as 12.345,678. Decimal international data types can contain negative values.

Decimal, U.S.

Data type where every third number left of the decimal point is preceded by a comma. The decimal point is represented by a period. Numbers right of the decimal point represent a fraction of one unit. For example, the number 12345.678 is represented as 12,345.678. Decimal US data types can contain negative values.

Unsigned Little Endian 2

Like Little Endian 2, except that the value is interpreted as an unsigned value.

Unsigned Little Swap Endian 2

Like Little Swap Endian 2, except that the value is interpreted as an unsigned value.

Unsigned Little Endian 4

Like Little Endian 4, except that the value is interpreted as an unsigned value.

Unsigned Little Swap Endian 4

Like Little Swap Endian 4, except that the value is interpreted as an unsigned value.

Unsigned Big Endian 2

Like Big Endian 2, except that the value is interpreted as an unsigned value.

Unsigned Big Swap Endian 2

Like Big Swap Endian 2, except that the value is interpreted as an unsigned value.

Unsigned Big Endian 4

Like Big Endian 4, except that the value is interpreted as an unsigned value.

Unsigned Big Swap Endian 4

Like Big Swap Endian 4, except that the value is interpreted as an unsigned value.

Date

Based on the international ISO-8601:1988 standard date notation: YYYYMMDD where YYYY represents the year in the 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 can be represented in Numeric, String, and EBCDIC base data types. For some data types, a minimum of four bytes is required.

Note:
The EBCDIC data type should not be used for new applications. It exists only to support previous user formats. If data needs to be encoded in EBCDIC, you must implement one of the supported EBCDIC code sets. For more information on supported code sets, see ***Rules and Formatter Extension for IBM ® WebSphere Message Broker for Multiplatforms User's Guide***.

Time

Based on the international ISO-8601:1988 standard time notation: HHMMSS, where HH represents the number of complete hours passed since midnight (00-23), MM is the number of minutes passed since the start of the hour (00-59), and SS is the number of seconds since the start of the minute (00-59). Times are represented in 24-hour format.

Times can be represented in Numeric, String, and EBCDIC base data types. For some data types, a minimum of four bytes is required.

Note:
The EBCDIC data type should not be used for new applications. It exists only to support previous user formats. If data needs to be encoded in EBCDIC, you must implement one of the supported EBCDIC code sets. For more information on supported code sets, see ***Rules and Formatter Extension for IBM ® WebSphere Message Broker for Multiplatforms User's Guide***.

Date and Time

Based on the international ISO-8601:1988 standard datetime notation: YYYYMMDDHHMMSS.

Combined dates and times can be represented in Numeric, String, and EBCDIC base data types. For some data types, a minimum of eight bytes is required.

Note:
The EBCDIC data type should not be used for new applications. It exists only to support previous user formats. If data needs to be encoded in EBCDIC, you must implement one of the supported EBCDIC code sets. For more information on supported code sets, see ***Rules and Formatter Extension for IBM ® WebSphere Message Broker for Multiplatforms User's Guide***.

Custom Date and Time

Custom Date and Time enables users to define formats for dates, times, and combined dates and times. Custom Date and Time can be represented in Numeric, String, and EBCDIC data types.

Note:
The EBCDIC data type should not be used for new applications. It exists only to support previous user formats. If data needs to be encoded in EBCDIC, you must implement one of the supported EBCDIC code sets. For more

information on supported code sets, see ***Rules and Formatter Extension for IBM ® WebSphere Message Broker for Multiplatforms User's Guide***.

Date/Time format specifications are strings made up of the following character combinations:

String	Value Range	Description
YY	00-99	2-digit year
YYYY	1900-2xxx	4-digit year
MON	JAN-DEC	3-letter abbreviation for month name
MN	01-12	2-digit month
DD	01-31	2-digit day of month
HH	01-23	2-digit hour of day
MM	01-59	2-digit minute of hour
SS	01-59	2-digit second of minute
AM	none	AM indicator
PM	none	PM indicator
JJJ	001-366	3-digit day-of-year; Julian date

The date/time format specifiers can be in any order. For String and EBCDIC underlying data types, the specifiers can include spaces and delimiters. A format specification string cannot include both 2-digit and 4-digit strings, or both numeric months and abbreviate month names.

NEONFormatter defines several different date and time representations that can be parsed and output. The following is a list of the custom formats supplied with NEONFormatter:

MM/DD/YY MN/DD/YY HH:MM MNDDYY

DD/MN/YY	DD/MN/YY HH:MM	MNDDYYYY
MN/DD/YYYY	MN/DD/YY HH:MM:SS PM	DDMONYY
DD/MN/YYYY	DD/MN/YY HH:MM:SS PM	DDMONYYYY
DD-MON-YY	MN/DD/YY HH:MM:SS	MONYY
DD-MON-YYYY	DD/MN/YYHH:MM:SS	MONYYYY
MON-YY	HH:MM PM	MONDDYYYY
MON-YYYY	HH:MM	MNDDYYHHMM
MN/DD/YY HH:MM PM	HH:MM:SS PM	MNDDYYHHMMSS
DD/MN/YY HH:MM PM	HH:MM:SS	HHMMSS

Data Type Conversion

Converting to String Representation

The following table describes the source values for each data type and how the data is converted to an intermediate String representation.

Data Type	Source Data Value Range	Intermediate String Representation
Not Applicable	Any value, any length.	Source value is unchanged.
String	A string of characters using one of the supported code sets that is any length. For more information on code sets, set the Rules and Formatter Extension for IBM ® WebSphere Message Broker for Multiplatforms ***Installation Guide***.	Source value is unchanged.
Numeric	A string of characters 0–9 (no '+' or '-') of any length, in the native character encoding for the local machine.	Source value is unchanged.
Binary	Any value, any length.	The binary string is converted to its string encoded hexadecimal form. For example, the binary string 01 23 AC (where each pair of numbers represents the 2 nibbles of a byte in hexadecimal) is converted to the string 0x0123AC, and the prefix 0x is prepended to the data.

Data Type	Source Data Value Range	Intermediate String Representation
EBCDIC	A string of characters of any length, encoded in EBCDIC.	Each character of the EBCDIC string is converted to its equivalent native encoded value. Characters in the EBCDIC code set that are not in the native code set are converted to a native encoded space character. NOTE: The EBCDIC data type should not be used for new applications. It exists only to support previous user formats. If data needs to be encoded in EBCDIC, you must implement one of the supported EBCDIC code sets. For more information on supported code sets, see ***Rules and Formatter Extension for IBM ® WebSphere Message Broker for Multiplatforms User's Guide***.
IBM Packed Integer	Maximum 16-byte value. Each nibble can contain the hexadecimal value 0-9. The last nibble contains a hexadecimal F.	String that represents the number. For example, the packed integer value 12 34 5F (where each pair of numbers represents the 2 nibbles of a byte in hexadecimal) becomes 12345.
IBM Signed Packed Integer	Maximum 16-byte value. Each nibble (except for the last nibble) can contain the hexadecimal value 0-9.	String that represents the number. The last nibble contains a hexadecimal C for positive numbers and a hexadecimal D for negative integers. For example, the signed packed integer value 12 34 5C (where each pair of numbers represents the 2 nibbles of a byte in hexadecimal) becomes +12345. The signed packed integer value 12 34 5D becomes -12345.

Data Type	Source Data Value Range	Intermediate String Representation
IBM Zoned Integer	The left nibble of each byte is a hexadecimal F. The right nibble of each byte is a hexadecimal 0-9.	String that represents the number. For example, the zoned integer value F1 F2 F3 F4 F5 becomes 12345.
IBM Signed Zoned Integer	The left nibble of each 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 a hexadecimal 0-9.	String that represents the number. For example, the signed zoned integer value F1 F2 F3 F4 C5 becomes +12345. The signed zoned integer value F1 F2 F3 F4 D5 becomes -12345.
Little Endian 2 Little Swap Endian 2 Big Endian 2 Big Swap Endian 2	A 2-byte integer in the range –32768 to 32767 (-(2**15) to (2**15) – 1)	String that represents the integer value (negative, positive, or 0).
Unsigned Little Endian 2 Unsigned Little Swap Endian 2 Unsigned Big Endian 2 Unsigned Big Swap Endian 2	A 2-byte integer in the range 0 to 65535 (0 to (2**16) – 1)	String that represents the integer value (>= 0).

Data Type	Source Data Value Range	Intermediate String Representation
Decimal, International	A character string representing a number, where every third digit left of the decimal point is preceded by a period (.). The decimal point is represented by a comma (,). The value can be preceded by + or -. For example, the number 12345.678 is represented as 12.345,678. Numbers that contain no separators or a subset of separators are considered valid input:12345-12345,123.456789,00	String that represents the number, without 3-digit separators, but with a decimal point. For example, 1.234,56 does not become 1,234.56, it becomes 1234.56. If the input value contains no decimal point, a decimal point is appended to the end of the value. For example, 12345 in Decimal International is converted to the string 12345.
Decimal, U.S.	A character string representing a number, where every third digit left of the decimal point is preceded by a comma(,). The decimal point is represented by a period (.). The value can be preceded by + or -. For example, the number 12345.678 is represented as 12,345.678. Numbers that contain no separators or a subset of separators are considered valid input:12345-12345.123,456789.00	String that represents the number, without 3-digit separators, but with a decimal point. For example, 1,234.56 becomes 1234.56. If the input value contains no decimal point, a decimal point is appended to the end of the value. For example, 12345 in Decimal, U.S. is converted to the string 12345.

Data Type	Source Data Value Range	Intermediate String Representation
Date and Time	A 14-byte numeric string. Each character is in the range 0-9 that represents a valid date in the international ISO-8601:1988 standard datetime notation: YYYYMNDDHHMMSS. The base data type is String, EBCDIC, or Numeric.	If the base data type is String or Numeric, the input value is unchanged. If the base data type is EBCDIC, each EBCDIC character is converted to its native encoding.
Time	A 6-byte numeric string. Each character is in the range 0-9 that represents a valid time in the international ISO-8601:1988 standard datetime notation: HHMMSS. The base data type can be String, EBCDIC or Numeric.The Time value is converted to the Date and Time format, with the date portion set to zeroes.	If the base data type is EBCDIC, each EBCDIC character is converted to its native encoding.
Date	An 8-byte numeric string. Each character is in the range 0-9 that represents a valid date in the international ISO-8601:1988 standard datetime notation: YYYYMNDD. The base data type is String, EBCDIC, or Numeric.	The Date value is converted to the Date and Time format, with the time portion set to zeroes. If the base data type is EBCDIC, each EBCDIC character is converted to its native encoding.

Data Type	Source Data Value Range	Intermediate String Representation
Custom Date and Time	A string in one of the custom date / time formats. Users cannot currently specify date / time formats. The base data type is String, EBCDIC, or Numeric.	The intermediate representation of the custom date / time format is in the default date / time format: Date and Time (14-byte numeric string in the form: YYYYMNDDHHMMSS). If the base data type is EBCDIC, each character value is converted to its native encoding.

Data Type Conversion Constraints

There are some pairs of data type conversions that are not sensible. For example, converting the string *good morning* to a number. This section discusses the constraints that exist for data conversion pairs.

Not Applicable

A data type of Not Applicable means that you do not want data type conversion to take place. The output data type should also be Not Applicable, so that NEONFormatter does not attempt to change the data between the input message and the output message.

String

A string of character data that is associated with a code set.

Output Data Type	Constraints
Not Applicable	The data remains unchanged between the input message and the output message.
String	The data type of the input message remains unchanged in the output message unless the two code sets differ. When the input code set is different than the output code set, a data conversion occurs.

Output Data Type	Constraints
Numeric	Valid only if each character of the field value is 0 to 9. In this case, the data remains unchanged between the input message and the output message.
Binary Data	Valid only if the string contains only the characters 0 to 9 and A to F, and the string contains an even number of characters. The string is interpreted as the string encoded hexadecimal form of a binary field. For example, the string0123AC is converted to the 3-byte binary value: 01 23 AC, where each pair of numbers represents the 2 nibbles of a byte in hexadecimal.
EBCDIC Data	This is a valid conversion. Values in the ASCII character set that do not have equivalent values in the EBCDIC character set are converted to an EBCDIC space character.
IBM Packed Integer IBM Zoned Integer	Valid only if the string is an integer with a value greater than or equal to zero within the allowed range for the IBM type.
IBM Signed Packed Integer IBM signed Zoned Integer	Valid only if the string is an integer with a value within the allowed range for the IBM type.
Little Endian 2 Little Swap Endian 2 Big Endian 2 Big Swap Endian 2	Valid only if the string represents an integer, and has a value within the range allowed for Endian 2 types.
Unsigned Little Endian 2 Unsigned Little Swap Endian 2 Unsigned Big Endian 2 Unsigned Big Swap Endian 2	Valid only if the string represents an integer and has a value within the range allowed for Unsigned Endian 2 types.
Little Endian 4 Little Swap Endian 4 Big Endian 4Big Swap Endian 4	Valid only if the string represents an integer and has a value that is within the range allowed for Endian 4 types.

Output Data Type	Constraints
Unsigned Little Endian 4 Unsigned Little Swap Endian 4 Unsigned Big Endian 4 Unsigned Big Swap Endian 4	Valid only if the string represents an integer and has a value that is within the range allowed for Unsigned Endian 4 types.
Decimal, International Decimal, U.S.	Valid only if the string represents an integer or floating point number.
Date and Time Custom Date and Time Time Date	Valid only if the string is in the form based on the international ISO-8601:1988 standard datetime notation: YYYYMNDDHHMMSS.

Numeric

A numeric string is a sequence of characters encoded in the native encoding for the machine on which NEONFormatter executes. A numeric string contains only the characters 0 to 9 (no + or -).

Output Data Type	Constraints
Not Applicable	The data remains unchanged between the input message and the output message.
String	The data remains unchanged between the input message and the output message.
Numeric	The data remains unchanged between the input message and the output message.
Binary Data	Valid only if the numeric string contains an even number of characters. This is because the string is interpreted as the string encoded hexadecimal form of a binary field. For example, the string 012345 is turned into the 3-byte binary value: 01 23 45 (where each pair of numbers represents the 2 nibbles of a byte in hexadecimal).

Output Data Type	Constraints
EBCDIC Data	This is a valid conversion. Each ASCII character is converted to its EBCDIC equivalent NOTE: The EBCDIC data type should not be used for new applications. It exists only to support previous user formats. If data needs to be encoded in EBCDIC, you must implement one of the supported EBCDIC code sets. For more information on supported code sets, see ***Rules and Formatter Extension for IBM ® WebSphere Message Broker for Multiplatforms User's Guide***.
IBM Packed Integer IBM Signed Packed Integer IBM Zoned Integer IBM Signed Zoned Integer	Valid only if the string represents a number with a value within the range allowed for the IBM types.
Little Endian 2 Little Swap Endian 2 Big Endian 2 Big Swap Endian 2	Valid only if the string represents a number with a value within the range allowed for Endian 2 types.
Unsigned Little Endian 2 Unsigned Little Swap Endian 2 Unsigned Big Endian 2 Unsigned Big Swap Endian 2	Valid only if the string represents a number with a value within the range allowed for Unsigned Endian 2 types.
Little Endian 4 Little Swap Endian 4 Big Endian 4 Big Swap Endian 4	Valid only if the string represents a number with a value within the range allowed for Endian 4 types.
Unsigned Little Endian 4 Unsigned Little Swap Endian 4 Unsigned Big Endian 4 Unsigned Big Swap Endian 4	Valid only if the string represents a number with a value within the range allowed for Unsigned Endian 4 types.

Output Data Type	Constraints
Decimal, International Decimal, U.S.	This is a valid conversion.
Date and Time Custom Date and Time Time Date	Valid only if the string is in the form based on the international ISO-8601:1988 standard datetime notation: YYYYMNDDHHMMSS.

Binary

Binary indicates a sequence of binary characters.

Output Data Type	Constraints
Not Applicable	Same behavior as String.
String	The data is converted to a string encoded hexadecimal format. For example, the binary string 01 23 AC (each pair of numbers represents the 2 nibbles of a byte in hexadecimal) is converted to the string 0x0123AC. The prefix 0x is prepended to the data.
Numeric	This is a valid conversion only if the string encoded hexadecimal form of the binary string contains only the numbers 0 to 9.
Binary Data	The data remains unchanged between the input message and the output message.

Output Data Type	Constraints
EBCDIC Data	Same behavior as String, except each character is an EBCDIC encoded character instead of a native encoded character. NOTE: The EBCDIC data type should not be used for new applications. It exists only to support previous user formats. If data needs to be encoded in EBCDIC, you must implement one of the supported EBCDIC code sets. For more information on supported code sets, see ***Rules and Formatter Extension for IBM ® WebSphere Message Broker for Multiplatforms User's Guide***.
IBM types	This is a valid conversion only if the string-encoded hexadecimal form of the binary string contains only the numbers 0 to 9, and has a value within the range allowed for the IBM types.
Endian types	This is a valid conversion only if the string-encoded hexadecimal form of the binary string contains only the numbers 0 to 9, and the number is within the range allowed for the various Endian types.
Decimal, International Decimal, U.S.	This is a valid conversion only if the string-encoded hexadecimal form of the binary string contains only the numbers 0 to 9.
Date and Time Custom Date and Time Date Time	Valid only if the string-encoded hexadecimal form of the binary value is in the form based on the international ISO-8601:1988 standard datetime notation: YYYYMNDDHHMMSS.

EBCDIC

A string of data encoded using the Extended Binary Coded Decimal Interchange Code (EBCDIC) used on larger IBM machines.

Note:
The EBCDIC data type should not be used for new applications. It exists only to support previous user formats. If data needs to be encoded in EBCDIC, you must implement one of the supported EBCDIC code sets. For more information on supported code sets, see ***Rules and Formatter Extension for IBM ® WebSphere Message Broker for Multiplatforms User's Guide***.

Output Data Type	Constraints
Not Applicable	Same behavior as String.
String	A valid conversion in which each EBCDIC character is converted to an ASCII equivalent takes place when the string data is compatible with the US EBCDIC code set. If a value in the US EBCDIC code set does not have an equivalent value in the native-encoded character set, it is converted to a native-encoded space character.
Numeric	Valid only if each character of the field value is EBCDIC 0 to 9.
Binary Data	Valid only if the string contains the EBCDIC characters 0 to 9 and A to F, and the string contains an even number of characters. This is because the string is interpreted as the string-encoded hexadecimal form of a binary field. For example, the string 0123AC is converted to the 3-byte binary value: 01 23 AC (where each pair of numbers represents the 2 nibbles of a byte in hexadecimal).

Output Data Type	Constraints
EBCDIC Data	The data remains unchanged between the input message and the output message. NOTE: The EBCDIC data type should not be used for new applications. It exists only to support previous user formats. If data needs to be encoded in EBCDIC, you must implement one of the supported EBCDIC code sets. For more information on supported code sets, see ***Rules and Formatter Extension for IBM ® WebSphere Message Broker for Multiplatforms User's Guide***.
IBM Packed Integer IBM Signed Packed Integer IBM Zoned Integer IBM Signed Zoned Integer	Valid only if the string represents an integer and has a value within the range allowed for the IBM types.
Little Endian 2 Little Swap Endian 2 Big Endian 2 Big Swap Endian 2	Valid only if the string represents an integer, and has a value within the range allowed for the Endian 2 types.
Unsigned Little Endian 2 Unsigned Little Swap Endian 2 Unsigned Big Endian 2 Unsigned Big Swap Endian 2	Valid only if the string represents an integer and has a value within the range allowed for the Unsigned Endian 2 types.
Little Endian 4 Little Swap Endian 4 Big Endian 4 Big Swap Endian 4	Valid only if the string represents an integer and has a value within the range allowed for the Endian 4 types.

Output Data Type	Constraints
Unsigned Little Endian 4 Unsigned Little Swap Endian 4 Unsigned Big Endian 4 Unsigned Big Swap Endian 4	Valid only if the string represents an integer and has a value within the range allowed for the Unsigned Endian 4 types.
Decimal, International Decimal, U.S.	Valid only if the string represents an integer or floating point number.
Date and Time Custom Date and Time Time Date	Valid only if the string is in the form based on the international ISO-8601:1988 standard datetime notation: YYYYMNDDHHMMSS.

IBM Types

This is a numeric type that includes IBM Packed, IBM Signed Packed, IBM Zoned, and IBM Signed Zoned.

Output Data Type	Constraints
Not Applicable	Same behavior as String.
String	This is a valid conversion. The value, incorporating the implied decimal point, is converted to a string representing the number.
Numeric	Valid only if the number is an integer greater than or equal to zero.
Binary Data	Valid only if the number is an integer greater than or equal to zero and has an even number of digits. The number is first converted to a string, and then the string is interpreted as the string-encoded hexadecimal form of the binary value.

Output Data Type	Constraints
EBCDIC Data	This is a valid conversion. The value, incorporating the implied decimal point, is converted to an EBCDIC-encoded string representing the number. NOTE: The EBCDIC data type should not be used for new applications. It exists only to support previous user formats. If data needs to be encoded in EBCDIC, you must implement one of the supported EBCDIC code sets. For more information on supported code sets, see ***Rules and Formatter Extension for IBM ® WebSphere Message Broker for Multiplatforms User's Guide***.
IBM Packed Integer IBM Signed Packed Integer IBM Zoned Integer IBM Signed Zoned Integer	This is a valid conversion for all pairs of data types, as long as values in the source data type are in the range allowed for the target data type.
Little Endian 2 Little Swap Endian 2 Big Endian 2 Big Swap Endian 2	Valid only if the number is an integer and has a value within the range allowed for the Endian 2 types.
Unsigned Little Endian 2 Unsigned Little Swap Endian 2 Unsigned Big Endian 2 Unsigned Big Swap Endian 2	Valid only if the number is an integer and has a value within the range allowed for the Unsigned Endian 2 types.
Little Endian 4 Little Swap Endian 4 Big Endian 4 Big Swap Endian 4	Valid only if number is an integer and has a value within the range allowed for the Endian 4 types.

Output Data Type	Constraints
Unsigned Little Endian 4 Unsigned Little Swap Endian 4 Unsigned Big Endian 4 Unsigned Big Swap Endian 4	Valid only if number an integer and has a value within the range allowed for the Unsigned Endian 4 types.
Decimal, International Decimal, U.S.	This is a valid conversion.
Date and Time Custom Date and Time Time Date	Valid only if the number converts to a string in the form based on the international ISO-8601:1988 standard datetime notation: YYYYMNDDHHMMSS.

Endian 2 Types

This is a 2-byte Numeric type.

Output Data Type	Constraints
Not Applicable	Same behavior as String.
String	This is a valid conversion. The value is converted to a string representing the integer.
Numeric	Valid only if the value of the number is an integer with a value greater than or equal to zero.
Binary Data	Valid only if the value of the number is an integer with a value greater than or equal to zero with an even number of digits.

Output Data Type	Constraints
EBCDIC Data	This is a valid conversion. The value is converted to an EBCDIC-encoded string representing the integer.
IBM Packed Integer IBM Signed Packed Integer IBM Zoned Integer IBM Signed Zoned Integer	This is a valid conversion, as long as values in the source data type are in the range allowed for the target data type.
Endian 2 types	This is a valid conversion, as long as values in the source data type are in the range allowed for the target data type.
Endian 4 types	Valid for conversions signed being to signed. Valid for conversions unsigned to unsigned. Valid for conversions unsigned being to signed. Valid for conversions signed to unsigned, if the number is greater than or equal to zero.
Decimal, International Decimal, U.S.	This is a valid conversion.
Date and Time Custom Date and Time Time Date	Not a valid conversion. Cannot generate enough digits to represent a date/time value.
IBM Packed Integer IBM Signed Packed Integer IBM Zoned Integer IBM Signed Zoned Integer	This is a valid conversion, as long as values in the source data type are in the range allowed for the target data type.

Endian 4 Types

This is a 4-byte numeric type that includes: Little Endian 4, Little Swap Endian 4, Big Endian 4, Big Swap Endian 4, Unsigned Little Endian 4,

Unsigned Little Swap Endian 4, Unsigned Big Endian 4, and Unsigned Big Swap Endian 4.

Output Data Type	Constraints
Not Applicable	Same behavior as String.
String	This is a valid conversion. The value is converted to a string representing the integer.
Numeric	Valid only if the value of the number is an integer with a value greater than or equal to zero.
Binary Data	Valid only if the value of the number is a positive integer with an even number of digits.
EBCDIC Data	This is a valid conversion. The value is converted to an EBCDIC-encoded string representing the integer. NOTE: The EBCDIC data type should not be used for new applications. It exists only to support previous user formats. If data needs to be encoded in EBCDIC, you must implement one of the supported EBCDIC code sets. For more information on supported code sets, see ***Rules and Formatter Extension for IBM ® WebSphere Message Broker for Multiplatforms User's Guide***.
IBM Packed Integer IBM Signed Packed Integer IBM Zoned Integer IBM Signed Zoned Integer	This is a valid conversion. For signed numbers being converted to unsigned numbers, only values greater than or equal to zero in the correct range are valid. For unsigned numbers being converted to signed numbers, only values in the correct range are valid.

Output Data Type	Constraints
Endian 2 types	Valid for conversions from signed to signed only if the Endian 4 number is in the range allowed for signed Endian 2 types. Valid for conversions from unsigned to unsigned only if the Endian 4 number is in the range allowed for unsigned Endian 2 types. Valid for conversions from signed to unsigned only if the Endian 4 number in the range allowed for unsigned Endian 2 types. Valid for conversions from unsigned to signed only if the Endian 4 number is in the range allowed for signed Endian 2 types.
Endian 4 types	Valid for conversions from signed to signed. Valid for conversions from unsigned to unsigned. Valid for conversions from unsigned to signed. Valid for conversions from signed to unsigned, only if the number is greater than or equal to zero.
Date and Time Custom Date and Time Time Date	Valid only if the number converts to a string in the form based on the international ISO-8601:1988 standard datetime notation: YYYYMNDDHHMMSS.

Decimal International and Decimal US

A Decimal International or Decimal US value is a string representing a number, where every third digit left of the decimal point is preceded by a comma (Decimal US) or a period (Decimal International). The decimal point is represented by a period (Decimal US) or a comma (Decimal International). A + or a - can precede the value. For example, the number 12345.678 is represented as 12,345.678 in Decimal US and 12.345,678 in Decimal International.

Output Data Type	Constraints
Not Applicable	The data remains unchanged between the input message and the output message.
String	The data remains unchanged between the input message and the output message.
Numeric	Invalid conversion. Numeric does not accept ., ,, + or -.
Binary Data	Invalid conversion. Binary does not accept ., ,, + or -.
EBCDIC Data	The data remains unchanged between the input message and the output message.
IBM Packed Integer IBM Signed Packed Integer IBM Zoned Integer IBM Signed Zoned Integer Endian types Date and Time Custom Date and Time Time Date	Invalid conversion.
Decimal, International Decimal, U.S.	This is a valid conversion. If going between International and US, . is changed to , and , is changed to ..

Date and Time

This includes Date and Time, Custom Date and Time, Date, and Time.

Output Data Type	Constraints
Not Applicable	Same behavior as String.
String	This is a valid conversion. The value is converted to a string representing the value of the date/time in its default format. Date and Time, Date, and Time are already in default format, so there is no change in data. Data in Custom Date and Time format is changed as described.
Numeric	Same behavior as String.
Binary Data	This is a valid conversion. For example, the date/time value 19560601190000 (June 1, 1956 at 7:00 PM) is converted to the binary string (each pair of numbers represents the 2 nibbles of a byte): 19 56 06 01 19 00 00.
EBCDIC Data	Same behavior as String, except that each string character is encoded as EBCDIC. NOTE: The EBCDIC data type should not be used for new applications. It exists only to support previous user formats. If data needs to be encoded in EBCDIC, you must implement one of the supported EBCDIC code sets. For more information on supported code sets, see ***Rules and Formatter Extension for IBM ® WebSphere Message Broker for Multiplatforms User's Guide***.
IBM Packed Integer IBM Signed Packed Integer IBM Zoned Integer IBM Signed Zoned Integer	This is a valid conversion if the integer number that represents the date/time is within the range allowed for the IBM types. The date/time is converted to its default integer format. That integer is then converted to the IBM type.
Endian 2 types	This is valid only if the integer number that represents the date/time is within the range allowed for Endian 2 types.
Endian 4 types	This is valid only if the integer number that represents the date/time is within the range allowed for Endian 4 types.

Output Data Type	Constraints
Decimal, International Decimal, U.S.	This is a valid conversion.
Date and Time Custom Date and Time	Date and Time and Custom Date and Time can be converted between each other, and to Time (drops the date portion) or to Date (drops the time portion).
Time	A Time can be converted to a Time, a Date (set to all zeroes), or Date and Time (date portion set to zeroes), or Custom Date and Time (date portion set to zeroes).
Date	A Date can be converted to a Date, a Time (set to all zeroes), or Date and Time (time portion set to zeroes), or Custom Date and Time (time portion set to zeroes).

Appendix C
Viewing New Era of Networks Formats in Eclipse

To view New Era of Networks Formatter message logical models in Eclipse, you must export formats using either a command line export tool or the New Era of Networks Formatter GUI. The formats are exported in the WMQI MRP form. The MRP export file is then migrated into Eclipse for viewing.

Exporting Formats using the Command Line

To use the command line interface for NNSY2MRP, type the following:

```
C:>NNSY2MRP
```

Option	Description
-s session name	The default is MQSI_PLUGIN.
-f export file name	The default is NNSY2MRP.MRP
-l <FormatName> 0	Use this to specify a specific format to export.
-20 <FormatCollectionName> 0	Use this to specify a format collection to export.

Note:
The entire contents of the Formatter database is exported if you do not use either the -1 or -20 option.

The local environment settings used for NNSY2MRP are identical to those used for NNFie. For information on these settings, see *Chapter 3: Configuration* on page 17.

Exporting Formats using the Formatter GUI

To use the Formatter GUI to export formats:

1. From the menu bar of the New Era of Networks Formatter window, select File > MRPExport.

   The MRPExport window opens.

2. Select the type of components for export from the Filter Component drop-down list.

   The Component List box displays all the components of that type.

3. To narrow your component display:

   a. Type an alpha or numeric entry in the Filter Text field, and press TAB.

   b. Select a filtering method from the Filter Type drop-down list, and click Apply Filter

      Components with names meeting the filter criteria are displayed in the Component List.

   c. From the Component List, select the individual components you want to export, or click Select All

   d. To move the selected components from the Component List to the Export List, click >>> or drag and drop each component to the Export List.

   e. To reverse the action and remove components from the Export List, select the components you want to remove, and click <<<.

4. To execute the export process, click Export.

   - If the export is successful, a message confirms the location of the export file.
   - If conflicts occur during the export, an Export Status Errors tab is enabled and the conflicts are displayed on this tab.

Migrating the Export File into Eclipse

To view the export file, use the MQSIMIGRATEMSGSETS tool to migrate the export file into an Eclipse project. For more information on MQSIMIGRATEMSGSETS, refer to the WebSphere Message Broker online help.

Appendix D
Processing Messages in the NEONMSG Domain

You can use the facilities provided by the message processing nodes and the brokers to transform an input MRM or XML message to an output NEONMSG message or to manipulate a message in the NEONMSG domain. However, be aware that the NEONMSG parser does not recognize repeating fields in these messages, unless you explicitly flag their containing structure (format) as repeating: if the input message includes repeating fields, and you do not flag them as such, the output message generated will not accurately reflect the input.

The only node that can determine your input message repeat structure by direct reference to the NNSY Formatter database is the MQInput node on the initial parse of a message in the NEONMSG domain. This is because the functionality provided in a message flow means you can potentially deviate from message structure encoded in the Formatter database.

In circumstances where you need to manipulate a message in the NEONMSG domain or transform your message to this domain, in WMQI 2.1.0, you can manually edit the message to insert the attribute NNSY-XMLrepeating="true" to each repeating collection. (In MQSI 2.0.2, you must use NEON-XML repeating="true".)

For example, an input message might contain:

<PurchaseOrder>

<CustomerName>IBM</CustomerName>

<OrderDetail>

<PartNum>1234-L</PartNum>

<PartDesc>Large Widget</PartDesc>

</OrderDetail>

<OrderDetail>
<PartNum>1234-M</PartNum>
<PartDesc>Medium Widget</PartDesc>
</OrderDetail>
<OrderDetail>
<PartNum>1234-S</PartNum>
<PartDesc>Small Widget</PartDesc>
</OrderDetail>
</PurchaseOrder>

This must be manually edited to contain:

<PurchaseOrder>
<CustomerName>IBM</CustomerName>
<OrderDetail NNSY-XML-repeating="true">
<PartNum>1234-L</PartNum>
<PartDesc>Large Widget</PartDesc>
</OrderDetail>
<OrderDetail NNSY-XML-repeating="true">
<PartNum>1234-M</PartNum>
<PartDesc>Medium Widget</PartDesc>
</OrderDetail>
<OrderDetail NNSY-XML-repeating="true">
<PartNum>1234-S</PartNum>
<PartDesc>Small Widget</PartDesc>
</OrderDetail>
</PurchaseOrder>

.

An example of the ESQL needed might be:

```
SET "OutputRoot"."Properties"."MessageSet" =
'NEONMESSAGESET';
SET "OutputRoot"."Properties"."MessageType" =
'PurchaseOrder_MT';
SET "OutputRoot"."MQMD" =
"InputRoot"."MQMD";
SET "OutputRoot"."MQMD"."Format" =
'MQSTR';
.
DECLARE K INT;
DECLARE N INT;
.
SET "OutputRoot"."NEONMSG"."PurchaseOrder_OC".
"CustomerName_OF"."CustomerName"=
"InputRoot"."XML"."PurchaseOrder"."CustomerName";
.
SET K = CARDINALITY,
("InputRoot"."XML"."PurchaseOrder"."OrderDetail"[]);
SET N = 0;
WHILE N < K DO
SET N=N+1;
SET "OutputRoot"."NEONMSG"."PurchaseOrder_OC".
"OrderDetail_OC"[N]."NNSY-XML-repeating" = 'true';
SET "OutputRoot"."NEONMSG"."PurchaseOrder_OC".
```

```
"OrderDetail_OC"[N]."OrderDetail_OF"."PartNum"=
"InputRoot"."XML"."PurchaseOrder"."OrderDetail"[N].
"PartNum";
SET "OutputRoot"."NEONMSG"."PurchaseOrder_OC".
"OrderDetail_OC"[N]."OrderDetail_OF"."PartDesc"=
"InputRoot"."XML"."PurchaseOrder"."OrderDetail"[N].
"PartDesc";
END WHILE;
```

An example of corresponding NNSY Output Formats:

```
Format/PurchaseOrder_MT/Ordinal
|
Format/PurchaseOrder_OC/Ordinal
|
Format/CustomerName_OF
| |
| Field/CustomerName/1/Normal Access/CustomerName
| |
| - Output Control/CustomerName/Data Field(Name Search)
|   |
|   - Data/String
|
Format/OrderDetail_OC/Ordinal/Repeating/3
|
Format/OrderDetail_OF
|
```

Field / PartNum / 1 / Controlling Field / PartNum

| |

| - Output Control / PartNum / Data Field(Name Search)

| |

| - Data / String

|

Field / PartDesc / 2 / Access Sibling Instance / PartDesc

|

- Output Control / PartDesc / Data Field(Name Search)

|

- Data / String

Notes:

In the last example, the tree structure of the formats must be adjusted, regardless of business logic, by comparison to the input message because formats that contain formats themselves (compound formats) cannot contain both field references and format references in the NNSY data model.

The use of the repeating flag is pertinent not only to cases where you are transforming from an XML or MRM domain but, also if you intend to manipulate the NEONMSG tree (even though you may not have explicitly left the NEONMSG domain). For example, if you process your message through a Compute Node, the tree will again have to have the repeating attribute flagged for all instances of repeats.

Appendix E
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.

Any performance data contained herein was determined in a controlled environment. therefore, the results obtained in other operating environments

may vary significantly. some measurements may have been made on develoment-level systems and there is no guarantee that these measurements will be the same on generally available systems. furthermore, some mesurements may hve been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information includes examples of data and reports used in daily business operations. To illustrate them as completley as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

this information contains sample application programs in source language, which illustrate programming techniques on varius 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. 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 IBM's application programming interfaces.

Trademarks

The following terms are trademarks of International Business Machines Corporation in the United States, other countries, or both:

AIX	CICS	DB2
DB2 Universal Database	developerWorks	
Everyplace	FFST	First Failure Support Technology
IBM	IMS	IMS/ESA
iSeries	Language Environment	MXSeries
MVS	NetView	OS/400
OS/390	pSeries	RACF
RETAIN	RS/6000	SupportPac
Tivoli	VisualAge	WebSphere
xSeries	z/OS	zSeries

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

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

Pentium is a registered trademark of Intel.

UNIX is a registered trademark in the United States andother countries licensed exclusively through The Open Group.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

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

Index

A

actions 145
AND operator 145
APIs
  for Rules 146
apitest 124, 126
application groups 144
arguments 145

B

batch mode
  resolving conflicts in New Era of Networks Formatter 62
  resolving conflicts in New Era of Networks Rules 97
Big Endian 2 data type 174
Big Endian 4 data type 174
Big Swap Endian 2 data type 174
Big Swap Endian 4 data type 174
Binary data type 171
Boolean operators
  AND 145
  OR 145
building libraries for user exits 136

C

C++ user exits
  creating 134
  threading 139
catalogue files 15
changing ownership of rules 39
code sets, supported 4
COLL_C 68
compatibility matrix 48
component inventory
  producing inventory export files for New Era of Networks Formatter 58
  producing inventory export files for New Era of Networks Rules 93
compound formats 71, 122
configuration
  defining database session information 26
  defining export/import utility information 29
  defining New Era of Networks Configuration Manager support information 34
  defining New Era of Networks node support information 32
  defining runtime parameters 19
  defining transport session information 31
  encrypting the configuration file 37
  setting runtime logging levels 36
  system search criteria 22
configuring
  nnsyreg.dat file 20
configuring centralized message reporting 44
Consistency Checker scripts
  MAINFCC 153
  MAINRCC 153
  PERMCC 153
controls
  output 121
converting data types 180, 185
converting formats 123
cross-platform migration
  procedures 111
  process overview 110
cross-version migration
  procedures 111
  process overview 110
Custom Date and Time data type 177
customizing New Era of Networks Formatter
  creating C++ user exits 134
  creating custom date/time formats 139

D

data types
- Big Endian 2 174
- Big Endian 4 174
- Big Swap Endian 2 174
- Big Swap Endian 4 174
- Binary 171
- converting 180, 185
- Custom Date and Time 177
- Date 176
- Date and Time 177
- Decimal, International 175
- Decimal,U.S. 175
- EBCDIC 172
- IBM Packed Integer 172
- IBM Signed Packed Integer 172
- IBM Signed Zoned Integer 173
- IBM Zoned Integer 173
- Little Endian 2 173
- Little Endian 4 174
- Little Swap Endian 2 173
- Little Swap Endian 4 174
- Not Applicable 171
- Numeric 171
- String 171
- Time 176
- Unsigned Big Endian 2 175
- Unsigned Big Endian 4 176
- Unsigned Big Swap Endian 2 176
- Unsigned Big Swap Endian 4 176
- Unsigned Little Endian 2 175
- Unsigned Little Endian 4 175
- Unsigned Little Swap Endian 2 175
- Unsigned Little Swap Endian 4 175
- value ranges 180

database
- installation 11

Date and Time data type 177
Date data type 176
DB2
- using NNFie 52
- using NNRie 87

Decimal, International data type 175
Decimal, U.S. data type 175
DEFAULT_C 68
defining formats 122

E

EBCDIC data type 172
encryption, configuration file 37
environment variables 19
error conditions 43
error messages
- for NNFie 85
- for NNRie 106

examples
- compound format 71
- field 72
- input control 71
- input format 69
- nnsyreg.dat file 20
- output control 73
- output format 70
- permissions 81

executable files 15
EXIT_C 68
exporting formats
- a single format 58
- from an entire database 58
- NNFie 50

exporting rules
- a single application group 92
- application groups types 92
- from an entire database 92
- message types 92
- NNRie 86

expressions 145

F

FIELD 68
files
- catalogue 15
- executable 15

FIX_C 68
flat formats 69, 122
FORMAT 68
FORMAT_GROUP 69
formats
- automatic conversion 123
- compound example 71
- defining 66, 122
- export options 55

flat examples 69
migrating with NNFie 50
removing 132
testing 124
Formatter
apitest executable 126
compound formats 122
defining formats 122
flat formats 122
msgtest executable 130
output controls 121
Formatter GUI
fields 120
input controls 120
literals 120
repetition count 120
tags 120

G

guidelines for entering session information 19

I

IBM Packed Integer data type 172
IBM Signed Packed Integer data type 172
IBM Signed Zoned Integer data type 173
IBM Zoned Integer data type 173
importing formats
commands 59
NNFie 50
optional parameters 59
importing rules
commands 94
NNRie 86
optional parameters 94
INPUT_CONTROL 68
installing the New Era of Networks Rules and Formatter database 11
interactive mode
resolving conflicts in New Era of Networks Formatter 62
resolving conflicts in New Era of Networks Rules 97
internationalization 4

L

LENGTH_C 68
linking
replacing Rules and Formatter library links 17
to Rules and Formatter libraries 17
LITERAL 68
literals
binary vs. string migration 107
Little Endian 2 data type 173
Little Endian 4 data type 174
Little Swap Endian 2 data type 173
Little Swap Endian 4 data type 174
locales, supported 4
lookup stub function, overriding 135

M

MAINFCC 153
MAINRCC 153
Management APIs
for Rules 146
MAP 69
MAP_LINK 69
MATH_C 68
Message Reporting
configuring centralized message reporting 44
message types 144
messages
parsing 123
reformatting 123
Messaging and Queuing
Message Reporting
configuring centralized message reporting 44
migrating
NNFie 50
NNRie 85
migration
binary vs. string literals 107
cross-platform 109
cross-version 107
msgtest 125, 130

N

New Era of Networks Rules and Formatter database installation 11

NN_DB_SES_DB_NAME 27, 30, 33, 35
NN_DB_SES_PASSWORD 28, 30, 33, 35
NN_DB_SES_SERVER 27, 30, 33, 35
NN_DB_SES_USER_ID 28, 30, 33, 35
NNFie 50
  error messages 85
  exporting formats 50
  importing formats 50
  optional parameters 54
  producing inventory export files 58
  readable files 65
  syntax 54
  troubleshooting export failures 58
  troubleshooting import failures 63
  upgrading formats 50
  using with DB2 52
NNFie.log file 58
NNFmtRmv utility 132
NNHGetMsg 123
NNHPutMsg 123
NNRie
  error messages 106
  exporting rules 86
  importing rules 86
  producing inventory export files 93
  readable files 100
  tracing import progress 96
  troubleshooting import failures 99
  upgrading rules 85
  using with DB2 87
NNRie.log file 93
NNRtrace 147, 150
NNSYMessageLog.nml file 44
nnsyreg.dat file
  defining database session information 26
  defining export/import utility information 29
  defining New Era of Networks Configuration Manager support information 34
  defining New Era of Networks node support information 32
  defining transport session information 31
  guidelines for creating session entries 24
  locating the Examples folder 20
  maintaining the file 21
  sample file 20
  setting runtime logging levels 36
Not Applicable data type 171
Numeric data type 171

O

optional parameters for
  exporting formats 55
  exporting rules 89
  importing formats 59
  importing rules 94
options for Rules 145
OR operator 145
output controls 121
OUTPUT_MSTR 68

P

parsing messages 123
PERMCC 153
PERMISSION 69
permissions
  example 81
  list/change 39
  rules and formatter users 38
PermUtil 39

R

readable files
  NNFie export data 65
  NNFie headers 65
  NNRie export data 100
  NNRie headers 100
reformatting messages 123
removing formats 132
reporting
  configuring centralized 44
resolving component conflicts
  in New Era of Networks Formatter 62
  in New Era of Networks Rules 97
Rules
  actions 145
  APIs 146
  application groups 144
  arguments 145
  associating 145
  Boolean operators 145
  changing ownership 39

Consistency Checker 144
error conditions 43
export options 89
expressions 145
Management APIs 146
message types 144
migrating with NNRie 85
naming rules 145
operators 145
options 145
ownership 39
rule names 145
subscriptions 145
test programs 147
transferring permissions 39
Rules and Formatter
linking to libraries 17
ruletest 147
RULOWNER utility 39
running the Consistency Checker 153
runtime parameter configuration 19

S

session entry guidelines 19
String data type 171
subscription ownership 41
subscriptions 92
SUBSTR_C 68
SUBSTT_C 68
supported code sets and locales 4
syntax
NNFie 54
NNRie 88

T

test executables
apitest 124
msgtest 124
testing formats 124
testing rules 147
NNRtrace 147
ruletest 147
threading C++ user exits 139
Time data type 176
tokens 11
tracing import progress
in New Era of Networks Rules 96
transferring
ownership 43
permissions 39
TRIM_C 68
troubleshooting
format export failures 58
format import failures 63
rules import failures 99

U

Unsigned Big Endian 2 data type 175
Unsigned Big Endian 4 data type 176
Unsigned Big Swap Endian 2 data type 176
Unsigned Big Swap Endian 4 data type 176
Unsigned Little Endian 2 data type 175
Unsigned Little Endian 4 data type 175
Unsigned Little Swap Endian 2 data type 175
Unsigned Little Swap Endian 4 data type 175
user exits
building libraries 136
creating for C++ 134
example 135, 138
overriding the stub function 135
threading 139
USER_DEFINED_TYPE 68
utilities
NNFie 50
NNFmtRmv 132
NNRie 85
NNRtrace 147
PermUtil 38, 39

V

value ranges 180

Sending your comments to IBM
Rules and Formatter Extension for IBM ® WebSphere Message Broker for Multiplatforms
System Management Guide

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:

  IBM United Kingdom Laboratories
  Hursley Park
  Winchester
  Hampshire
  SO21 2JN

- By fax:
  - From outside the U.K., use your international access code followed by 44 1962 870229
  - From within the U.K., use 01962 816151

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