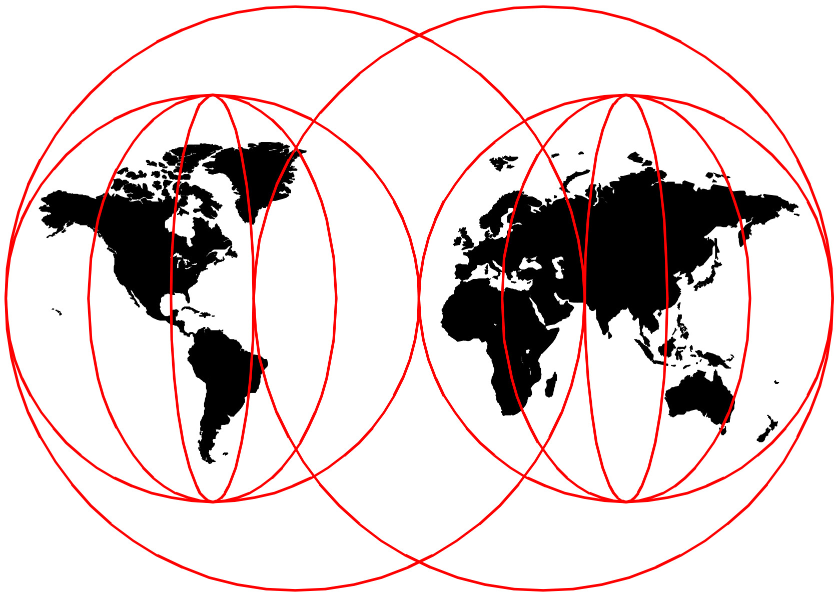
Using the MQSeries Integrator Version 1.0

Dieter Wackerow, Jorgen Becker-Hansen, Ken Palmer, Morton Saetra



International Technical Support Organization

http://www.redbooks.ibm.com

SG24-5386-00

SG24-5386-00

International Technical Support Organization

Using the MQSeries Integrator Version 1.0

May 1999

Take Note!

Before using this information and the product it supports, be sure to read the general information in Appendix B, “Special Notices” on page 175.

First Edition (May 1999)

This edition applies to Version 1.0 of IBM MQSeries Integrator, Program Number 5801-AAR.

Comments may be addressed to:
IBM Corporation, International Technical Support Organization
Dept. HZ8 Building 678
P.O. Box 12195
Research Triangle Park, NC 27709-2195

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

Contents

Figures

Tables

Preface

The application integration market is rapidly emerging, driven by organizations' business need and the availability of a new class of software, known as message brokers.

The MQSeries Integrator is the first product available in the MQSeries Integrator layer of IBM's Business Integration with MQSeries. It is message brokering software that ensures business-critical applications and processes can understand each other. Based on MQSeries' messaging and queuing capabilities, the MQSeries Integrator is a real-time, intelligent, rules-based message routing and dynamic message content transformation and formatting system that allows you to integrate all types of applications and systems into robust, flexible and scalable information networks.

This redbook explains how and for what you can use the MQSeries Integrator in your enterprise. It gives you a broad understanding of the product and its features that increase productivity in application integration. It describes:

- How to integrate applications where one program produces output in a different format from what the partner program needs as input. Examples that show how the MQ Series Integrator can add information to a message are also included.
- How to use the product for intelligent routing, that is, sending a message to one or more different programs according to information within the message. Examples explain how you can forward a complete message or different parts of a messge to several partners.
- How to use the graphical user interface provided with the MQSeries Integrator to perform integration tasks quickly and easily.

Persons who have no knowledge of the MQSeries Integrator can use this publication as a textbook.

The Team That Wrote This Redbook

This redbook was produced by a team of specialists from around the world working at the International Technical Support Organization, Raleigh Center.

Dieter Wackerow is the MQSeries expert at the International Technical Support Organization, Raleigh Center. His areas of expertise include application design and development for different industries, performance evaluations, capacity planning, and modelling of computer systems and networks. He writes extensively and teaches IBM classes worldwide on all areas of application development with MQSeries.

Morten Saetra is an IBM Certified MQSeries Specialist in Norway. He has five years of experience in MQSeries and 13 years with CICS. During this time he has advised and supported customers in various MQSeries and CICS projects. Besides that he also teaches MQSeries classes for IBM Learning Services.

Ken Palmer is a Certified I/T Specialist in the United States. He has 9 years of experience in the transaction system arena. He has worked at IBM for 12 years. His areas of expertise include MQSeries, which he implemented at the 1996 Atlanta Olympic Games.

Jorgen Becker-Hansen has worked with middleware for over six years, recently joining MQSeries Strategic Support as a consultant in the UK. He holds a Computer Science degree from the University of Westminster.

Thanks to the following people for their invaluable contributions to this project:

Bashar Kilani
Mark Swinson
IBM Hursley, England

Comments Welcome

Your comments are important to us!

We want our redbooks to be as helpful as possible. Please send us your comments about this or other redbooks in one of the following ways:

- Fax the evaluation form found in “ITSO Redbook Evaluation” on page 189 to the fax number shown on the form.
- Use the online evaluation form found at `http://www.redbooks.ibm.com`
- Send your comments in an internet note to `redbook@us.ibm.com`

Chapter 1. Overview

Conducting business globally and electronically becomes more and more complex. With the rapid growth of business on the Internet, the wave of mergers and acquisitions, and industry deregulation, companies can no longer afford isolated IT systems that can't talk to each other.

With MQSeries, IBM provides middleware that allows customers to reliably interconnect heterogeneous platforms and environments using asynchronous messaging. Customers have used the base MQSeries product to reliably interconnect their applications and systems within their enterprise, and also across enterprises.

Now IBM announced additional products and capabilities to address larger business integration needs. These new capabilities not only will provide value added services to assist in applications and information integration, but will also support integration of their business processes. Furthermore, customers will be able to do such integration within their enterprise and across their customer/supplier/third-party enterprises.

1.1 Business Integration

IBM provides a solution for business integration. IBM Business Integration with MQSeries consists of three offerings which can be used together or separately, in any combination:

- **MQSeries**, IBM's industry-leading, messaging-oriented middleware, enables diverse applications to communicate securely and reliably, with enterprise-level performance, over a wide range of platforms. MQSeries leads the market with over 4000 customer sites and has broad partner support.
- **MQSeries Integrator** (MQSI) software enables integration of applications and systems into robust, flexible and scalable information networks. Based on MQSeries messaging and queuing capabilities, the MQSeries Integrator is a real-time, intelligent rules-based message routing and dynamic message content transformation and formatting system. Along with this functionality, preconfigured templates for major packaged applications and e-business extensions will also be provided.
- **MQSeries Workflow** is a workflow management system that automates business processes involving applications and/or people to give enterprises more control of their business activities. It helps align and integrate resources, accelerate process flow, optimize costs, eliminate

errors and improve productivity, by capturing and automating a business's processes. MQSeries Workflow is the next generation (V3) of IBM's FlowMark product.

MQSeries Integrator and MQSeries Workflow rely on MQSeries. MQSeries Integrator intercepts MQSeries messages transforming and routing the message contents based upon information stored in a rules database. MQSeries Workflow uses MQSeries messaging for the communication between the MQSeries Workflow clients and servers.

This redbook provides detailed information on how to use the MQSeries Integrator for message reformatting and routing.

1.2 What the MQSeries Integrator Can Do for You

The MQSeries Integrator extends messaging capabilities into the business arena. The integration of new and existing applications is consuming more and more management time. Estimates show that 40-60% of IS development and maintenance costs go to integration and communication programming for heterogeneous systems and applications.

The MQSeries Integrator, is a real-time, application-to-application, message transformation and routing program that offers:

- Flexibility and ease-of-use for rapidly adding, extending, or replacing applications
- Intelligent routing for seamless integration of applications, databases, and networks according to conditions set by the business

Some highlights of the features of the MQSeries Integrator follow:

- It forms the first product in the message brokering layer of the IBM Business Integration framework.
- It makes adding, extending, or replacing applications in an MQSeries network simple and easy.
- it applies intelligent routing to seamlessly integrate applications, databases, and networks.
- It enables application-to-application message transformation.
- It supports custom built and predefined application libraries.
- It supports PeopleSoft GL, SAP R/3, and S.W.I.F.T. templates from New Era of Networks Inc. (NEON).

The MQSeries Integrator is optimized for high-volume, in-storage transformation of messages. It is very useful in environments where many applications exchange information, each requiring slightly different message formats.

The MQSI graphical user interface allows users to perform integration tasks quickly and easily. Preconfigured templates for major packaged applications and e-business extensions will also be available. This product is now available under IBM terms and conditions worldwide.

The MQSI is Year 2000 ready. When used in accordance with the associated documentation, it is capable of correctly processing, providing, and/or receiving date data within and between the twentieth and twenty-first centuries, provided that all products (for example, hardware, software, and firmware) used with the product properly exchange accurate date data.

MQSeries Integrator is message brokering software that ensures business-critical applications and processes can understand each other. Based on MQSeries' messaging and queuing capabilities, the MQSeries Integrator is a real-time, intelligent rules-based message routing and dynamic message content transformation and formatting system that allows you to integrate all types of applications and systems into robust, flexible and scalable information networks.

1.3 The MQSI Architecture

MQSI is an anonymous publish/subscribe architecture that essentially contains three parts:

- **Transport:**
  Message & Queuing middleware provides the heart of most enterprise's transport. With over 50 % market share, and the widest product platform coverage, MQSeries is the de facto standard for message oriented middleware. MQSeries has achieved popularity with customers because it enables you to develop applications that deal with messages and queues through a common API and it frees you from needing to worry about the underlying operating system or network. The MQSeries Integrator integrates to practically any type of transport.
- **Formatter:**
  The meta-data formatting engine allows the translation of "anything to anything." It allows for field-by-field translation of data and information. It is a dynamic translator. Once a set of transaction/file formats are defined,

they can be reused over and over again. As an adjunct to the formatter, we offer format libraries for applications like SAP and Peoplesoft.

- **Rules/Routing Engine:**
  The rules/routing engine allows for a huge number of business rules to be evaluated. In working with the formatting engine, we can translate at the message layer to move information from system to system, or from one system to multiple systems at the same time. What makes the rules processor so unique is that we can process the same number of messages against 1,000 rules, 10,000 rules, 100,000 rules or more with no degradation in performance. The rules and formatting engine has been benchmarked at over 1000 messages per second (medium sized Solaris server used for benchmark).

MQSI is the only rules processor in the industry that does not generate code in any way. It is a metadata-based engine; this is part of the reason it scales so well.

1.4 How to Use the MQSI

For the user, the main components of the MQSeries Integrator are:

- A database
- Formatter GUI
- Rules GUI
- The Rules Daemon (also called rules processor or rules engine)
- Configuration files
- Various tools/utilities
- Queues managed by MQSeries

These components and their functions and many examples on how to use them are described in this book.

Figure 1 on page 5 elucidates the components and when they are used. In the center is the MQSeries Integrator database. It contains input and output formats and rules on how they are used to map data from an incoming message to one or more outgoing messages.

At development time you define formats and rules. This is done with two programs (graphical user interfaces):

- Formatter GUI
- Rules GUI

Both programs require a database connection file (sqlsvses.cfg) to access the database. What this file contains is explained later.

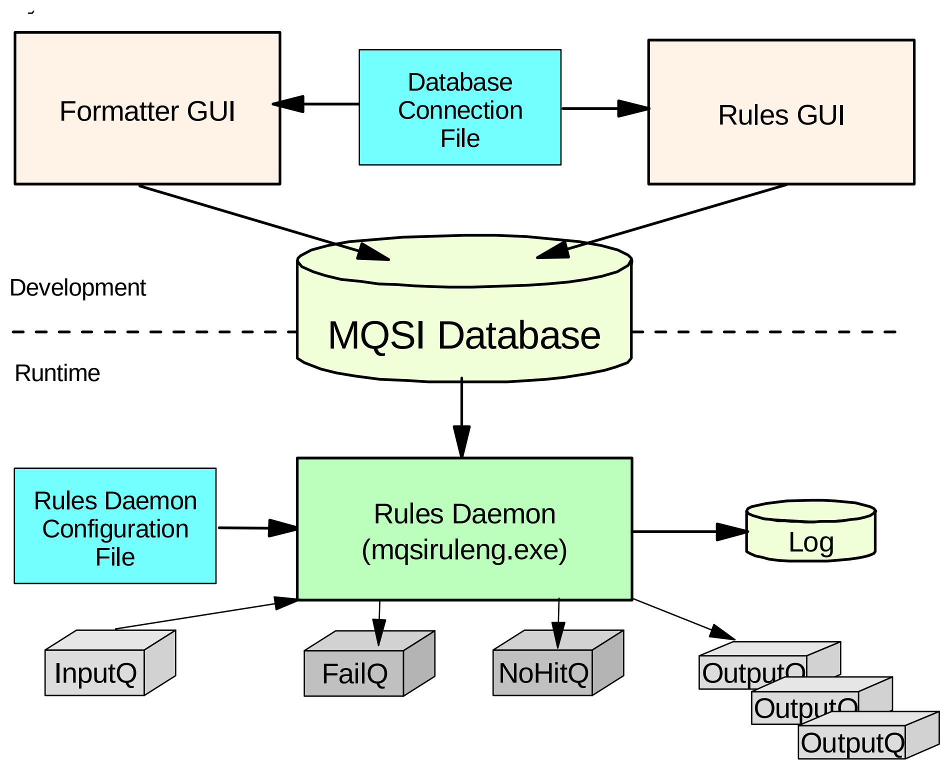


Figure 1. MQSeries Integrator Components

You use the Formatter GUI to define input and output formats, that is, how input and output messages are structured. Fields in a message can be fixed or variable in length, they can be separated with delimiters or contain the field length. Fields may also be tagged; that is, they can be identified by a unique field ID.

With the Rules GUI you define how input formats are mapped to output formats and to what queue they are put. You can specify many rules for a single message, and each rule can create a different message with different contents.

You can also add fields to a message using literal values or values you calculate from fields in the input message. It is also possible to substitute values in the input message with different ones in the output message.

At run time, when in production the NEON Rules Daemon, also known as rules processor, uses the information in the database to process incoming messages. The rules processor needs information provided in a rules processor configuration file. You can have several instances of the rules processor running, each using a different configuration file.

The rules processor configuration file is important. It contains, for example, the name of the default application group and message type. Notice that this is the MQSI message type and not the MQSeries message type in the message header MQMD.

An application group is a container for messages. For example, all messages regarding payroll applications can belong to one application group and messages concerning sales to another. Message types are input formats defined with the Formatter GUI.

Each instance of the rules processor works with one application group; and for each application group you can have one default message. Also, in Version 1.0, you can have only one input queue per rules processor instance.

How does the MQSI process (input) messages other than the default one?

The application group and the message type must be specified in the message. To do this the MQSI header is added to the message. That means, a message the rules processor processes, can have one of two formats:

- MQSeries Header + Data
- MQSeries Header + MQSI Header + Data

Of course, the program that sends the message must provide the MQSI header. How this header looks is described later in this book.

One more important fact has to be noted here. In Version 1.0, the MQSI can handle only datagram messages, no requests or replies. Information in ReplyToQ and ReplyToQMgr in the MQSeries message descriptor are lost.

Figure 1 on page 5 also shows some queues. This means of course, that you must have a queue manager running. MQSI relies on it services.

As said before, each rules processor instance can have only one input queue. However, there may be multiple output queues it routes messages to. Into what queue (and consequently to which application) a message is put depends on the rule(s) associated with the input format (message).

If the message does not match the input format it is put on the FailQ (you may choose a different name). This queue name and also the name of the queue the message is put when none of the rules matches (no hit) is specified in the rules processor configuration file. The reason for the exception is written to the rules processor's log file.

There are also several utilities provided that help with debugging of the definitions you enter into the database. The most valuable is the Visual Tester, an MQSeries SupportPac. This and other tools are described later, too.

In the next chapters of this book you will read:

- How to install and configure the MQSeries Integrator and related software, such as a DB2 database
- A tutorial that gets you started with the product
- Several examples on how to define formats and how to reformat messages
- How to define rules to map input to output messages
- An example that demonstrates the abilities of MQSI to perform as an intelligent router
- Some comments about security

1.5 Literature

Literature about the MQSeries Integrator is available in the form of PDF files on the product CD. The books are:

- *Application Development Guide*
- *Installation and Configuration Guide*
- *Programming Reference for NEON Formatter*
- *Programming Reference for NEON Rules*
- *System Management Guide*
- *User's Guide*

These books are not available as hardcopy.

Chapter 2. Installation and Setup

In this chapter we will explain how to install, configure and run the installation verification procedure (IVP) for MQSeries Integrator Version 1.0 (MQSI). We will also set up the database used by MQSI.

The following platforms will be covered: Windows NT and AIX. All platforms require that a database is installed before you can use the MQSI. We will use DB2; however, most of the popular databases are supported.

The administration graphical user interface (GUI) for the Formatter and Rules components will run on Windows NT only.

2.1 Windows NT Installation

MQSeries Integrator Version 1.0 is supported by MQSeries Version 5.0 or later. We recommend that you apply MQSeries CSD3 or higher. The MQSI product package comes with MQSeries V5.0 and CSD3. However, if Version 5.0 or 5.1 is already installed on your machine you will not need to remove it and install it again.

For our project, we are using IBM DB2 Universal Database (UDB) Version 5. Both DB2 UDB Personal Edition (PE) and the full server version can be used. Most installations will probably use the Server version. If DB2 UDB PE is chosen, you should also install DB2 Connect Personal Edition. The tools and utilities used to administer the database use the DB2 client which connects to the server with ODBC (open database connectivity). It is possible to connect directly to the database if the database is on the same machine as the GUI interface; however, in this case we have chosen to use the DB2 client with ODBC in all cases. After the database is installed we install the MQSI product and configure it.

The installation steps are:

1. Prepare for the installation.
2. Install in any order:
   - MQSeries
   - MQSeries Integrator
   - DB2 UDB
3. Set up the database.
4. Install the database schema.

5. Edit the database connection file.
6. Verify the installation.
7. Edit the Makefile (optional).

2.1.1 Preparing for the Installation

It is important that we collect and read all relevant information prior to starting the installation. In addition, the different software releases and levels have to be checked and possibly upgraded.

The *MQSeries Integrator Installation and Configuration Guide* is a good place to start. This document is available as softcopy. You find it on the CD in the directory \Books and the filename is mqi_in_40.pdf.

In order to create the MQSI database you should read the appropriate readme file for the database software you are using. You can find the readme files on the product CD in the directory \IVP. We used *readme.db2*. This file provides assistance in setting up the DB2 V5.0 environment for use with the MQI.

2.1.2 Installing MQSeries

For our project we installed the complete V5.0 product consisting of:

- MQSeries Server for Windows NT
- MQSeries Clients
- MQSeries Toolkit
- MQSeries Online Documentation
- MQSeries Bindings for Java
- MQSeries Internet Gateway

Simply place the MQSeries CD in the CD-ROM drive and follow the instructions. After the product is installed we applied CSD3. This is also self-starting. After installation you have an \mqm directory on the drive where you have chosen to install the product.

2.1.3 Installing DB2 Universal Database

If a database is not already available you can install DB2 on your own system. For this project, we installed:

- DB2 Universal Database Personal Edition V5.0
- DB2 Connect Personal Edition V5.0

You may select from three installation types: typical, compact and custom. We selected custom which uses the same disk space as the typical installation. We installed the following functions:

- Graphical tools
- DB2 ODBC driver
- Integrated SNA support
- Documentation

Furthermore, we chose to automatically start the DB2 instance at boot time.

Next you have to enter a user ID and password for the database. You may choose the same ID you use on your NT system. You will become the owner of the database. After installation you will see three new directories: \Db2Log, \IMNNQ_NT and \SQLLIB.

2.1.4 Installing MQSeries Integrator for DB2

Just follow the InstallShield Wizard. If you have MQSeries already installed on your machine, be sure to give the correct directory names in the installation screens. Read the ReadMQI.txt file for hints and take note about what database you are using. You find this file on the product CD in the directory \Books.

The installation program requests that you confirm three installation folders:

- MQI for the MQSeries Integrator
- MQM for the MQSeries program files
- MQM for the MQSeries data files

If you choose the custom installation you may select the following features:

1. Program files
2. SDK files
3. Online documentation and help files

The installation program will not overwrite the MQSeries version already installed on your system.

Note: You cannot install the MQSI on a FAT-formatted disk.

2.1.5 Setting Up the MQSI Database

There are three steps required in the configuration of DB2 before the MQSI can use it. These are:

1. Create a new database to contain the formats and rules.

2. Configure a client connection to the new database.
3. Create tablespace within the database for the MQSI tables.

2.1.5.1 Creating the MQSI Database

We have to create the MQSI database and an alias. We access the database from the MQSI GUI via the ODBC driver. For this we use the database alias.

We have chosen the MQSI database name to be MQSITDB and the database alias to be MQSITDBA. This leads to the naming convention shown in Table 1 on page 12.

Table 1. MQSI Database Naming Conventions

Application	Environment	System	Function	Comment
MQSI	T	DB		MQSI test database
MQSI	T	DB	A	Alias to MQSI test database

Notes: Environment: T = Test, P = Production, ...
System: DB = DB2, ...
Function: A = Alias

To create the MQSI database we need to bring up a DB2 Command Line Processor (DB2 CLP) window. From the start menu select:

Programs
  DB2 for Windows NT
    Command Line Processor

This window shows the DB2 => prompt.

You create the database with the following command:

```
create database MQSITDB
```

Note: To exit the DB2 command line program type quit at the DB2 => prompt.

2.1.5.2 Creating an Alias for the MQSI Database

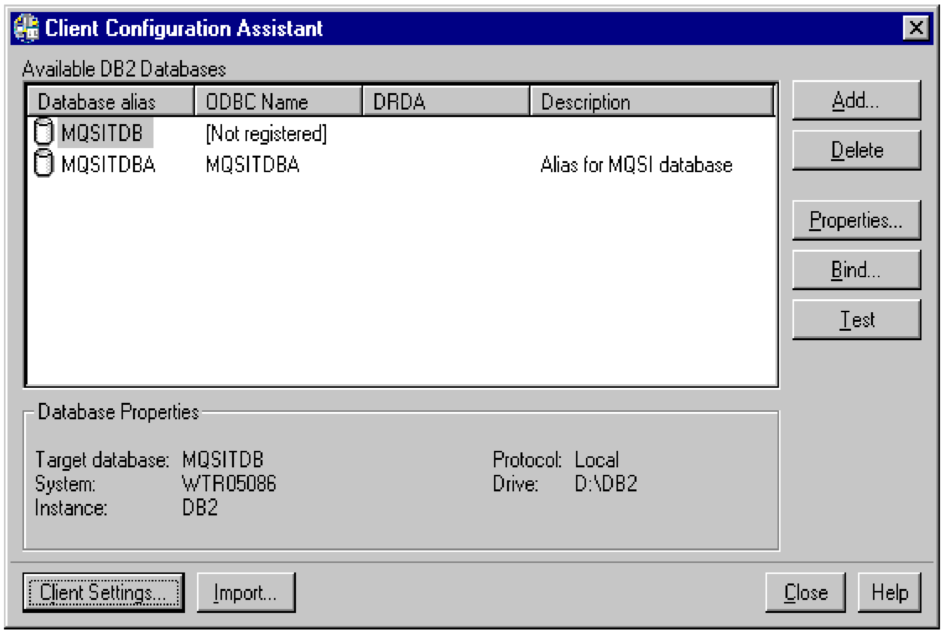
To connect to a DB2 database, there must be a database alias set up. DB2 automatically creates an alias to the database on the server when it creates the database. This alias has the same name as that of the database. To connect to the database via an NT client, an alias needs to be made on the client machine. If the server is already on the NT machine, a new database alias must still be made because the default one does not have an ODBC driver associated with it. An ODBC driver is required for connecting to the database using the MQSI GUI. This can be done using the DB2 program called Client Configuration Assistant.

To create the alias click **Start** and then select:

Programs
DB2 for Windows NT
Client Configuration Assistant

This brings up the window shown in Figure 2. The Client Configuration Assistant is a smartguide that helps you with your database configuration.

Note: In this example we use a local database.



Database alias	ODBC Name	DRDA	Description
MQSITDB	[Not registered]		
MQSITDBA	MQSITDBA		Alias for MQSI database

Target database: MQSITDB
System: WTR05086
Instance: DB2
Protocol: Local
Drive: D:\DB2

Figure 2. Client Configuration Assistant

- Click **Add**.
- Now a window with six tabs appears. The following list explains what you have to do. To switch from one tab to the next click **Next** or the next tab.

1. Source	Check the radio button to **Manually configure a connection to a DB2 database.**

2. Protocol	Select what type of connection you have, that is **Local** if the database is on the same machine.
3. Local	The radio button **Database in the same instance** is already marked.
4. Target Database	Type the name of the database, here MQSTIDB.
5. Alias	Specify the name of the database alias name, MQSITDBA.
6. ODBC	Check the box **Register this database for ODBC** and mark the radio button **As a system data source.** Then click the **Done** button.

- Next you will see a pop-up window that you can test. Click the **Test Connection** button to check if the database connection works. In the subsequent window you have to enter your user ID and password.
- When you go back into the CCA the result should be something similar to Figure 2.

2.1.5.3 Creating Table Space for the MQSI Database

For a DB2 database we now have to create the table spaces used by MQSI. The table spaces needed are:

- NNF for the NEON Formatter
- NNR for the NEON Rules
- NNP for the NEON Product

Remember to grant administrator (DBADM) privilege on the MQSI database to the user who will perform the installation.

Before you can work with the database you have to connect to it. If you work with a local database type the following command at the db2 => prompt:

```
connect to MQSITDB
```

If you work with a remote database use the database alias. Type the following command at the db2 => prompt:

```
connect to MQSITDBA user (username) using (password)
```

DB2 will respond with a message like the one below:

```
   Database Connection Information
 Database product       = DB2/NT 5.0.0
 SQL authorization ID   = USERABC
 Local database alias   = MQSITDB
```

The create table space commands have to be run from a DB2 prompt, that is, bring up the DB2 CLP window. The commands are:

```
create tablespace NNF managed by system using ('NNF')
create tablespace NNR managed by system using ('NNR')
create tablespace NNP managed by system using ('NNP')
```

2.1.6 Installing the Database Schema

Now the database is ready for the installation of the Formatter and Rules tables. You can find more information in the *Installation and Configuration Guide*.

The procedure inst_db.bat in the directory \mqi\install.sql installs the MQSeries Integrator database schema. This procedure will take between 10 and 45 minutes, depending on the server's processor speed and current activity.

You are still in the DB2 CLP window. Exit the CLP program by typing:

```
db2 => quit
```

Next change directories and run the bat file:

```
d:\SQLLIB\BIN\ cd \
d:\ cd mqi\install.sql
d:\MQI\install\ inst_db.bat userid password database
```

Use the database alias name, MQSITDBA, for the database. You will be prompted several times to verify information. You should also look for any error messages in the file c:\TEMP\inst_db2.log.

2.1.7 The Database Connection File

Once the Formatter and Rules tables are created, the MQSI GUIs can connect to the database using the ODBC option. Some MQSeries Integrator executables connect to the database using the database connection file *sqlsvses.cfg* in the directory \MQI\bin.

This file contains information for the database sessions that describe the server name, user ID, password, and database name that a particular session uses. Executables search the sqlsvses.cfg file for a given session name and attempt to connect to the MQSeries Integrator database.

You must edit the sample file and update it with your own installation specific information. For more information, refer to the *MQSeries Integrator System Management Guide*. Uncomment the section that applies to your database type. The changes for DB2 are shown in Figure 3 on page 16.

```
# Example SessionNames (uncomment the SessionNames needed):
#
#       DB2 database (final colon required on each line):
new_format_demo:MQSITDBA:Userid:Password:
rules:MQSITDBA:Userid:Password:
import:MQSITDBA:Userid:Password:
#       (change Database, Userid, Password above)
```

Figure 3. DB2-Related Information in the sqlsvses.cfg File

2.1.8 Verifying the Installation

The steps needed to run the IVP are:

1. Create a queue manager named QAQM with the command:
   `crtmqm QAQM`
2. Start the queue manager with the command
   `strmqm QAQM`
3. Use runmqsc to define the IVP queues. The define commands are in member mqs.txt in the CD-ROM \IVP directory.

   ```
   runmqsc QAQM < e:\ivp\mqs.txt
   ```

   You may copy the IVP directory from the installation CD-ROM to the hard disk for easier use later. The file contains the commands to create four queues:

   ```
   define qlocal('RulesIn') replace
   define qlocal('RulesNoHit') replace
   define qlocal('RulesFail') replace
   define qlocal('Output') replace
   ```

4. Edit the database connection file *sqlsvses.cfg* in \mqi\bin so that there is a session entry with the name "new_format_demo" as shown in Figure 3 on page 16. Make sure that the entries for the database name or alias, user name and password match your environment.

5. Import the test format and rules files. Use the following commands from the drive:\mqi\bin directory:

```
nnfie -i d:\IVP\formats.fie -s new_format_demo
nnrie -i d:\IVP\rules.rie -s new_format_demo
```

Important

You cannot import from the CD. The NNFIE and NNRIE programs try to write to the IVP directory. Therefore, copy the IVP directory from the CD on your hard drive, for example, d:\mqi\ivp.

6. Now start the Formatter GUI to see what is defined in the IVP sample. Do not change either format if you want to run the IVP successfully.

   To start the Formatter GUI, select from the Start button:

   Programs
   MQSeries Integrator
   Formatter

   In the logon window enter:

   - Your user ID and password
   - For DBMS select **ODBC-DB2 (ODBC)**
   - For driver select the database alias **MQSITDBA**

   After awhile the window shown in Figure 4 appears.

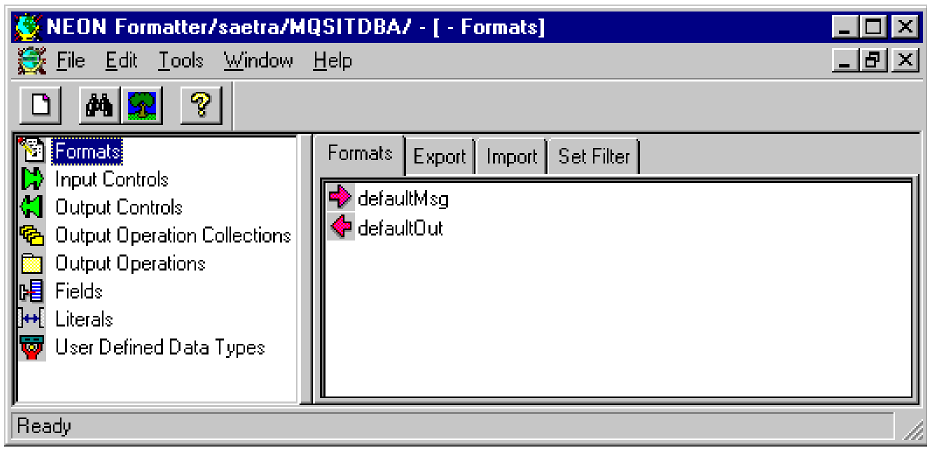


Figure 4. Formatter GUI for IVP

7. Now start the Rules GUI to see what rules the IVP sample uses. Again, do not change anything.

   To start the Rules GUI, select from the Start button:

   Programs

       MQSeries Integrator

           Rules

   In the Logon window enter:

   - Your user ID and password
   - For DBMS select **ODBC-DB2 (ODBC)**
   - For driver select the database alias **MQSITDBA**

   After a while the window shown in Figure 5 appears. Double-click **defaultApp** to expand the tree.

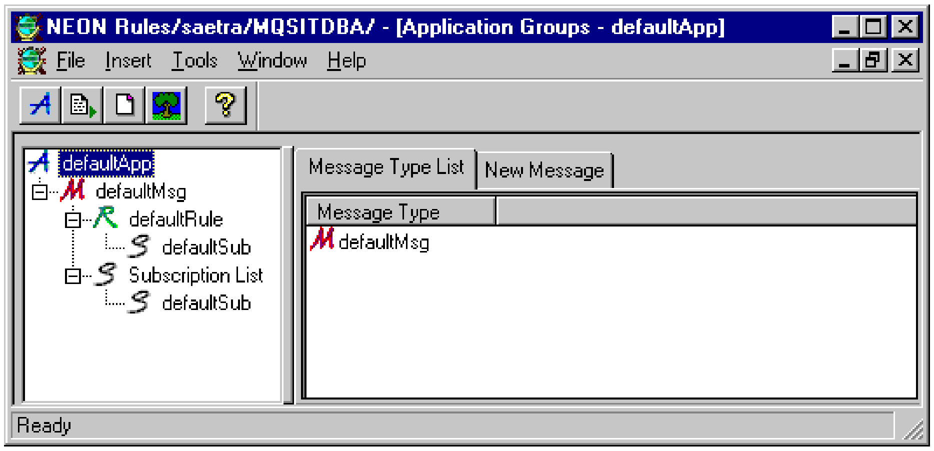


Figure 5. Rules GUI for IVP

8. Change the rules processor configuration file *mqsiruleng.mpf* in \mqi\bin directory to match your environment. The file verify.txt in the \ivp directory states what must be changed.

   Note: The field *Servername* is used by DB2. Other databases use the field *Databaseinstance*. The connection file also names queue manager and queues used by the IVP. There should be no need to change them.

   Figure 6 on page 19 shows the file mqsiruleng.mpf for the IVP.

```
[Queues]
# Parameters related to queues, MQSeries control, and rules engine control

# Alternate User Authority Flag
CredentialsEnabled = 0

# MQSeries queue manager name
QueueManagerName = QAQM

# retry limit
MaxBackoutCount= 0

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

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

[Logging]
# Log levels:
#3 - log only fatal errors
#2 - log errors, and fatal errors
#1 - log warnings, errors, and fatals
#0 - log informationals, warnings, errors, and fatals
LogFileName = mqsiruleng.log
LogLevel    = 0

[Rules Database Connection]
# all fields are mandatory)
ServerName       = MQSITDBA
UserId           = userabc
Password         = password
DatabaseInstance = MQSITDBA
#
# DatabaseType is a numeric with these values:
#       SYBASE = 1      MQSQL = 2      ORACLE = 3
#       DB2    = 4      ODBC  = 5
#
DatabaseType = 5
```

Figure 6. Rules Daemon Configuration File mqsiruleng.mpf for the IVP

9. Now start the rules processor, MQSI's run-time program. From a DOS prompt in the \mqi\bin directory issue the command:

   mqsiruleng -p mqsiruleng.mpf

   You may minimize the window.

10. Use the MQSeries sample program amqsput to put some messages on the queue RulesIn as shown in Figure 7.

    Note: The format of the message is "string;string;", such as hotdog;fries;.

```
C:\>amqsput RulesIn QAQM
Sample AMQSPUT0 start
target queue is RulesIn
hotdog;fries;
cat;mouse;

Sample AMQSPUT0 end

C:\>amqsget Output QAQM
Sample AMQSGET0 start
message < FRIES HOTDOG>
message < MOUSE CAT>
no more messages
Sample AMQSGET0 end

C:\>
```

Figure 7. Using the IVP

11. Get the messages from the Output queue after the rules processor has processed them. Use the MQSeries sample program amqsget to get the messages. You will notice in Figure 7 that the two input fields are reversed.

Note: If you made a typing error the rules processor will not find a matching rule and the message ends up in the RulesFail queue.

2.1.9 Editing the Makefile

This step is optional. An example Makefile, demonstrating how to rebuild the MQSeries Integrator executables is supplied in the \mqi\examples directory. Edit this Makefile before you use it.

2.2 AIX Installation

As with the Windows NT installation, MQSeries Integrator for AIX Version 1.0 is supported by MQSeries Version 5.0 or later. The product comes with MQSeries V5.0. However, if Version 5.0 or 5.1 is already installed you will not need to remove it and install it again.

For this project we used IBM DB2 UDB Enterprise Edition with the DB2 Client Application Enabler. We can also have the Windows NT DB2 Client Application Enabler installed on a Windows NT machine in order to administer the database via an ODBC connection to the server. This ODBC connection under Windows NT must be in place in order for us to use the Formatter and Rules GUI tools.

It is important to collect and read all the relevant information prior to starting the AIX installation.

The installation steps are:

1. Prepare for the AIX installation.
2. Install in any order:
   - MQSeries
   - MQSeries Integrator
   - DB2 UDB
3. Set up the database.
4. Install the database schema.
5. Edit the database connection file.
6. Verify the installation.
7. Edit the Makefile (optional).

2.2.1 Preparing for the Installation

This step involves collecting and reading all the relevant MQSI manuals and readme files. In addition to the *MQSeries Integrator Installation and Configuration Guide* you should look at the readmqi.txt in the root directory of the CD and the readme file for the database you will be using in the IVP directory. For DB2 this would be readme.db2. Additional release notes can be found in the READMES subdirectory. The *MQSeries Integrator Installation and Configuration Guide* and other documentation files can be found on the CD under the /Books directory in three subdirectories, HTML, PDF and PS in these three formats.

2.2.2 Installing MQSeries

To install MQSeries on AIX you should follow the instructions in the manual *MQSeries for AIX Version 5.0 Quick Beginnings,* GC33-1867. You have to create a user ID and make it a member of the mqm group before installing the product. The user ID will own the files and directories once installed. Even when installing MQSeries as part of the MQSeries Integrator installation this should be done first.

To mount the CD-ROM to an appropriate directory you should see the instructions in the chapter on AIX Installation of the *MQSeries Integrator Installation and Configuration Guide.* This tells you how to use the System Management Interface Tool (SMIT) GUI to do this. Alternatively, if familiar with command line commands you may want to simply issue some commands to do this, such as:

```
mkdir /cdrom
chmod 777 /cdrom
/etc/mount -rv cdrfs /dev/cd0 /cdrom
```

This would mount the CD-ROM to a directory called /cdrom and make it accessible to all users.

MQSeries will be installed in a directory /var/mqm. This cannot be altered. Read the preparation instructions in the *MQSeries for AIX Quick Beginnings Version 5.0,* GC33-1867 regarding disk space requirements and how to create new directories or file systems if required.

2.2.3 Installing the AIX DB2 Universal Database

If the database is not already installed you could choose to install DB2. For instructions on how to do this you may wish to read the chapter on installing the DB2 products in *DB2 UDB for UNIX Quick Beginnings V5R2,* S10J-8148. You will need to mount the DB2 for AIX CD-ROM as before and then run the command:

```
./db2setup
```

You should also read the readme.txt on this CD before installing.

From the menu presented when you run the dbsetup command we chose to install DB2 UDB Enterprise Edition and the DB2 Client Application Enabler. We used the defaults for the instance name (db2inst1) and password (ibmdb2) leaving the communications defaults also, using TCP/IP with a port number of 50000.

DB2 Version 5 by default will install all the software to a directory /usr/lpp/db2_05_00. It will also create the DB2 instance in a directory /home/{db2 instance name}. This would be /home/db2inst1 using the default instance name. The first step you will need to do once installed is to log on as the instance owner (db2inst1) and add the following to the profile:

$. ./home/{db2 instance name}/sqllib/db2profile

Once this .profile is run this sets up the path correctly to be able to run the db2 command and at the prompt create the database:

```
Db2=>create database MQSITDB
```

This is exactly as described in 2.1.5.1, “Creating the MQSI Database” on page 12 for the Windows NT installation, including the naming convention.

2.2.4 Installing MQSeries Integrator for DB2

To install MQSeries Integrator we can use the System Management Interface Tool (SMIT) GUI. From the System Management menu select:

Software Installation and Maintenance
Install and Update Software
Install and Update from LATEST Available Software.

You can then change the installation options to match those in Figure 8:

```
* INPUT device / directory for software            /cdrom
* SOFTWARE to install                              [_all_latest]          +
  PREVIEW only? (install operation will NOT occur)  no                    +
  COMMIT software updates?                          yes                   +
  SAVE replaced files?                              no                    +
  AUTOMATICALLY install requisite software?         yes                   +
  EXTEND file systems if space needed?              yes                   +
  OVERWRITE same or newer versions?                 no                    +
  VERIFY install and check file sizes?              no                    +
  Include corresponding LANGUAGE filesets?          yes                   +
  DETAILED output?                                  no                    +
  Process multiple volumes?                         yes                   +
```

Figure 8. AIX Installation

This will result in the MQSI software being installed to a directory /usr/lpp/mqi containing a number of subdirectories.

2.2.5 Creating the MQSI Database on AIX

Having already created the database on the AIX server machine from the db2 prompt as described in 2.2.3, “Installing the AIX DB2 Universal Database” on page 22, we can continue the process on the Windows NT machine using it as a client.

We use the Client Configuration Assistant described in 2.1.5.2, “Creating an Alias for the MQSI Database” on page 12 to manually configure a connection to the AIX DB2 database using the TCP/IP protocol rather than Local. We type in the host name or IP address of the AIX machine and the port number (default 50000).

We then continue creating the NNF, NNR and NNP table spaces as described in 2.1.5.3, “Creating Table Space for the MQSI Database” on page 14.

Next we install the database schema using the inst_db.bat batch file as shown in 2.1.6, “Installing the Database Schema” on page 15. Alternatively, you may create the table spaces and run the inst_db.sh script from the /usr/lpp/mqi/bin directory on the AIX machine.

2.2.6 Verifying the Installation

The installation verification process is similar to that described for Windows NT in 2.1.8, “Verifying the Installation” on page 16. The steps needed to run the Installation Verification Program (IVP) using AIX are:

1. Create a queue manager named QAQM on the server AIX machine.
2. Start the queue manager.
3. Use runmqsc to define the IVP queues. The define commands are in the member MQS.txt in the CD-ROM /IVP directory. You may wish to copy this directory to the hard disk for later use. For example, copy them to /usr/lpp/mqi/bin/IVP.
4. Edit the database connection file sqlsvses.cfg in /usr/lpp/mqi/bin so that there is a session entry with the name "new_format_demo". You will also need to change the entries for the database name or alias, user name and password to match your installation.
5. You can now import the test format and rules files. From /usr/lpp/mqi/bin type in the following, remembering that AIX commands are case-sensitive:

```
•$./NNFie -i /IVP/formats.fie -s new_format_demo
•$./NNRie -i /IVP/rules.rie -s new_format_demo
```

Note: Steps 4 and 5 could be done via the client ODBC connection from Windows NT.

6. If you now bring up the Formatter and Rules GUI tools on the Windows NT client machine you will be able to see the IVP sample.
7. Change the rules processor configuration file MQSIruleng.mpf in /usr/lpp/mqi/bin to reflect your environment. To determine what needs to be changed read the verify.txt file in the /IVP directory. For DB2 you will need to change the Servername parameter rather than the Databaseinstance parameter.
8. Make /usr/lpp/mqi/bin the current directory.
9. To start the rules processor issue the command:

   ```
   $./MQSIruleng -p MQSIruleng.mpf
   ```

10. Put the messages on the RulesIn queue with the sample program. Use amqsput RulesIn QAQM.
11. Note that the format of the messages is "string;string;"; for example, hotdog;fries; or cat;mouse;. Get the messages from the output queue after the rules processor has processed them. Use the sample program amqsget to retrieve the messages. You will see that the two fields are now in a reversed order.
12. Optional: An example Makefile, demonstrating how to rebuild the MQSeries Integrator executables, is supplied in /usr/lpp/mqi/examples. To use this you must edit it appropriately.

Chapter 3. Getting to Know the MQSI - A Tutorial

This chapter helps you to do your first steps with the MQSeries Integrator. We describe a short example that touches all important features of the product. In this walkthrough, we show how the MQSI takes an input message consisting of two fields, first and last name, and swaps the order of the fields along with converting any lower-case letters to upper-case. This is shown in Figure 9.

In this quick-start example we create definitions through the Formatter GUI and Rules GUI. We explain how the Visual Tester, a supporting utility, can be used to test message reformatting, and how the rules daemon, MQSI's run-time program, processes the messages in a real environment.

For this walkthrough we will choose the following order:

1. Create the definitions for the Formatter.
2. Define the Rules.
3. Test the formats and rules using the Visual Tester.
4. Run the Rules Daemon against the test data, formats and rules.

Note: We assumed that MQSI has been installed properly on a Windows NT platform as described in Chapter 2, "Installation and Setup" on page 9.

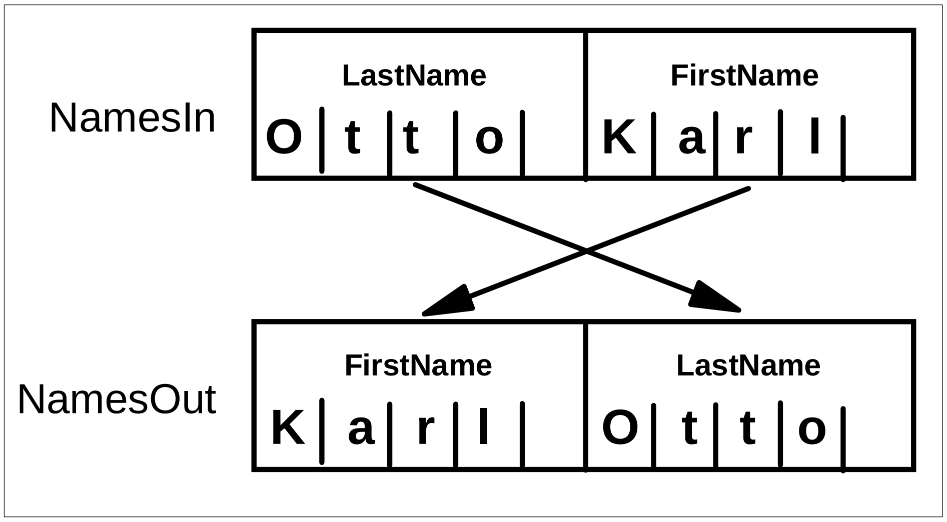


Figure 9. Tutorial: Input and Output Messages

3.1 Working with the MQSI Format Administrator

In this section we will create the appropriate definitions for our sample formats. First, let's bring up the Formatter GUI. Click the **Start** button and select:

Programs
  MQSeries Integrator
    Formatter

This will bring up the logon window where you have to enter the following:

User ID:	Your User ID
Password:	Your password
DBMS:	Select **ODBC - DB2(ODBC)** from the drop-down list
Driver:	Select the alias database name, here **MQSITDBA**
Qualifier:	For DB2, leave this field blank

Click **OK** to log on. Next, you will see the Formatter window:

.

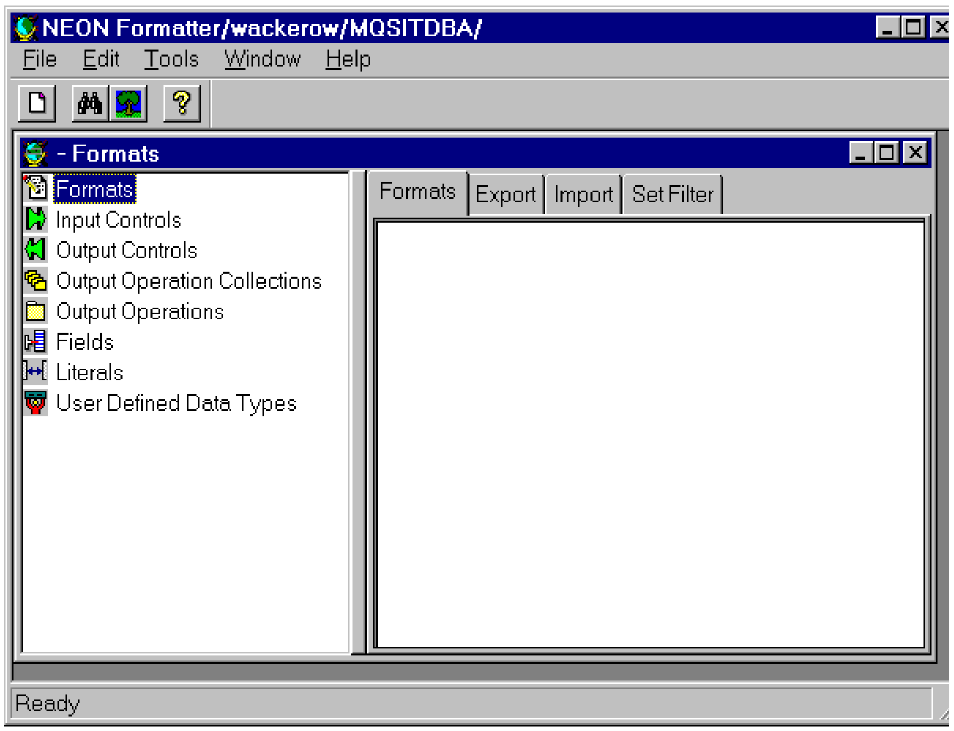


Figure 10. MQSI Formatter Window

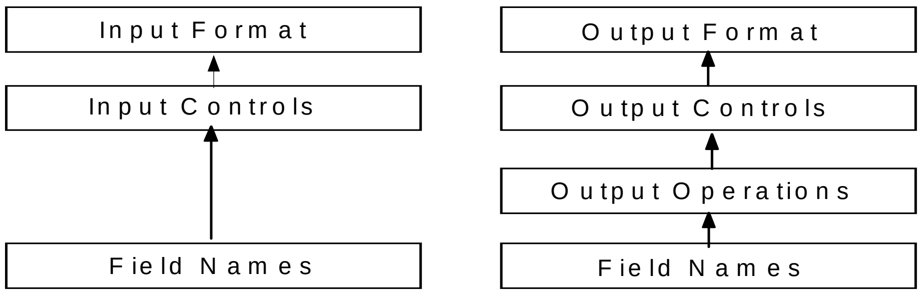
Note: The Formatter window shown in Figure 10 on page 28 does not contain any objects. All objects used by the IVP have been deleted.

Figure 9 on page 27 shows the formats for the input and output messages. Both messages contain the same fixed length fields, however, in a reversed order. Now we have to determine what objects we have to define. Table 2 lists the object names, their types and describes what they are used for.

Table 2. Tutorial: Formatter Objects to Define

Object Type	**Object Name**	**Description**
Input Format	NamesIn	Name of input format
Output Format	NamesOut	Name of output format
Input Controls	String5	Used to parse the input fields; here both fields are five-byte long strings
Output Controls	String5	Used to format the output fields; both fields are fixed five bytes
Output Operation Collections	-	Not used
Output Operations	UPPER_CASE	Used to make the output fields upper-case; referenced by the output controls
Fields	FirstName LastName	Field names used in both formats
Literals	-	Not used
User Defined Data Types	-	Not used

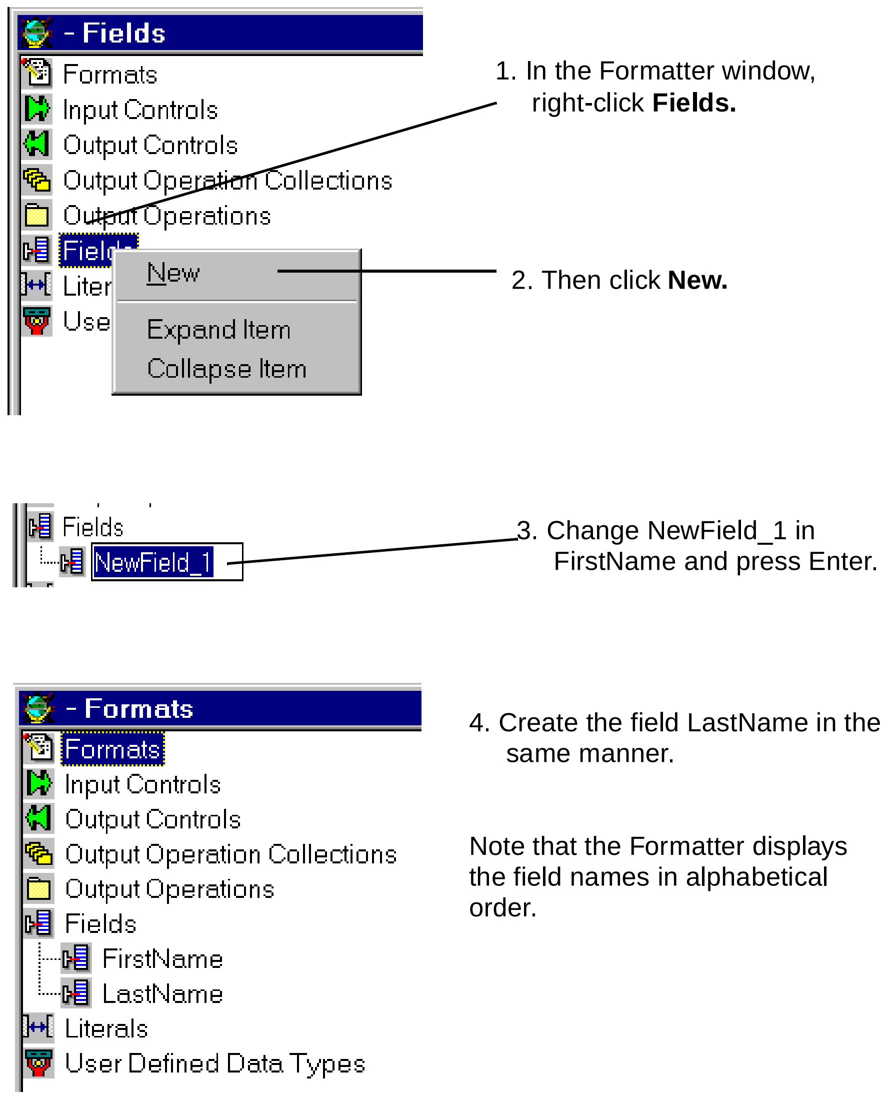
The MQSI is an object-oriented product. We create the objects from the bottom up starting with the fields in this order:



3.1.1 How to Define a Field

We create two fields, FirstName and LastName. Both names are used in the input format and in the output format. Names are case-sensitive.

Note: Here you specify only the field names and not any of their attributes. You do this in the input and output controls.



1. In the Formatter window, right-click **Fields.**

2. Then click **New.**

3. Change NewField_1 in FirstName and press Enter.

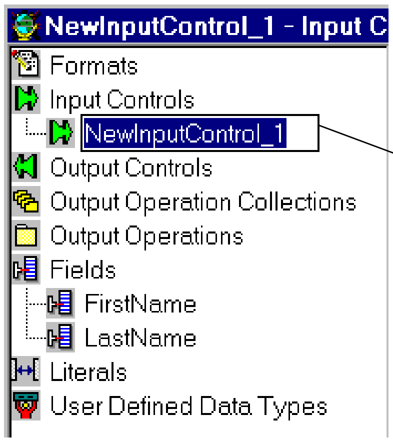
4. Create the field LastName in the same manner.

Note that the Formatter displays the field names in alphabetical order.

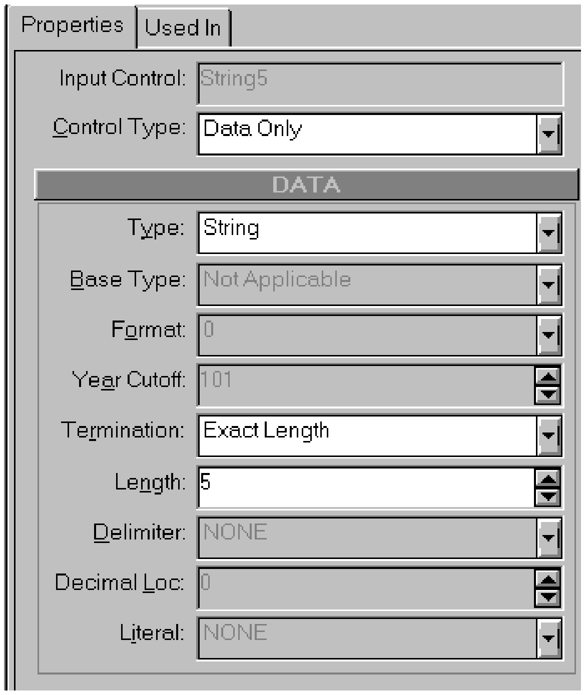
Figure 11. Tutorial: Creating Input Fields

3.1.2 How to Define an Input Control

Since all fields are fixed five characters long we need only one input control to tell the Formatter how to parse the input record. We will give this input control the generic name String5. You can use the same input control for several input fields.



1. Right-click **Input Controls.**
2. Click **New** in the pop-up.
3. Change the name of the input control to String5.
4. Press Enter.
5. On the right side of the window, fill in the properties as shown below.



Input Control:	String5
Control Type:	Data Only
Type:	String
Base Type:	Not Applicable
Format:	0
Year Cutoff:	101
Termination:	Exact Length
Length:	5
Delimiter:	NONE
Decimal Loc:	0
Literal:	NONE

6. The fields contains only data; no delimiter or length field.
7. The field contains only characters.
8. The field length is fixed.
9. The length is 5.
10. Click the **Apply** button at the bottom right in the window.

Figure 12. Tutorial: Creating an Input Control

3.1.3 How to Define an Input Format

For the input we use a flat format, that is, a format that contains fields only. The figure below shows another choice, a compound format. Such a format contains other formats. Right-click **Formats** and select the choices shown below.

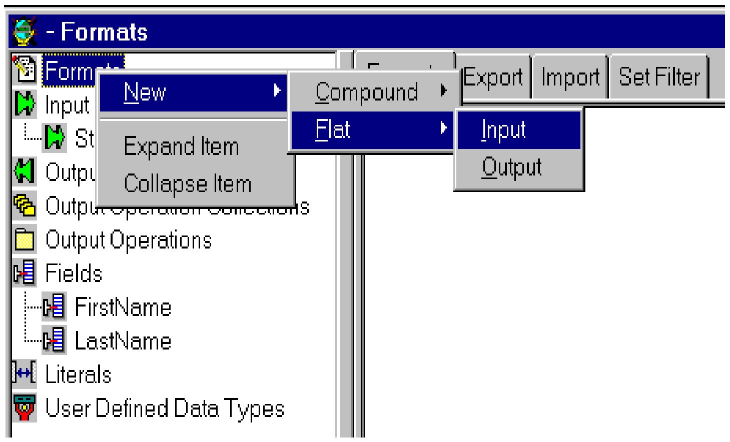
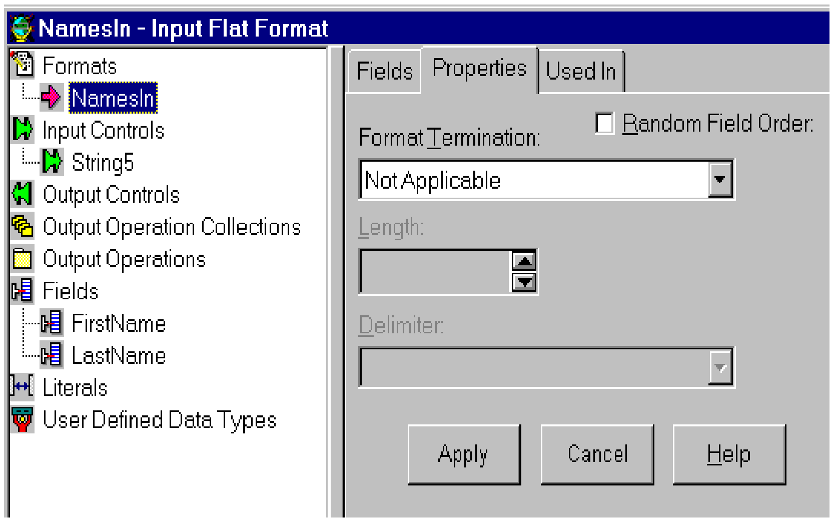


Figure 13. Tutorial: Creating a Flat Input Format

Change NewFormat_1 to NamesIn and press Enter. Next, you will see the window below. Leave the default for format termination and click on **Apply**.



Random Field Order: ☐

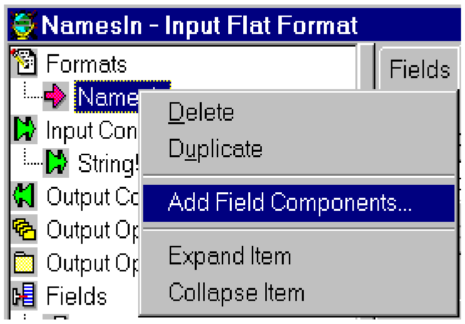
Format Termination: Not Applicable

Length:

Delimiter:

Figure 14. Tutorial: Input Flat Format

Next, we have to associate the input fields with the input format.

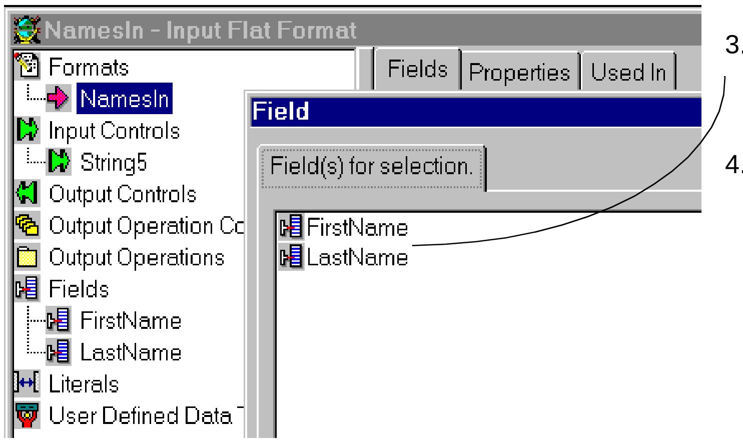


1. Right-click **NamesIn.**

2. Click **Add Field Components.**

   This brings up the Field window shown below.

Figure 15. Tutorial: Add Fields to a Format



3. Select the names in the window.

4. Click the **Accept Selection** button at the bottom of the window.

Figure 16. Tutorial: Select Fields for a Format

When you highlight NamesIn in the Formatter window and click the Fields tab you will see the list of fields in the format. Since in our message the first name is followed by the last name, we don't have to re-order the fields.

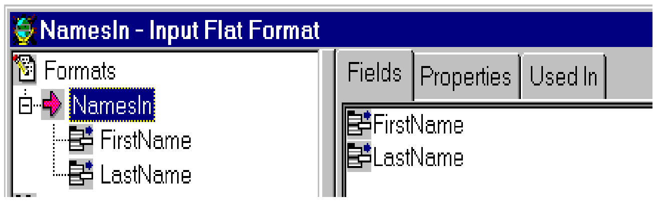


Figure 17. Tutorial: Input Format with Fields

To re-order, click an item in the fields list, drag it upward and drop it on another field. The field will be inserted before the field you dropped it on. You cannot move a field from the top of the list to the bottom.

So far we have defined the two fields and specified where they are located within the input format (message). For the Formatter to parse the message it has to know how the fields looks. In this case, both fields have the attributes defined in the input control String5.

To assign the input format String5 to an output field, highlight the field under the format. This brings up a window that shows the cutoff you see in Figure 18. Select **String5** from the Input Control Name list and click **Apply**. Repeat this for LastName, too.

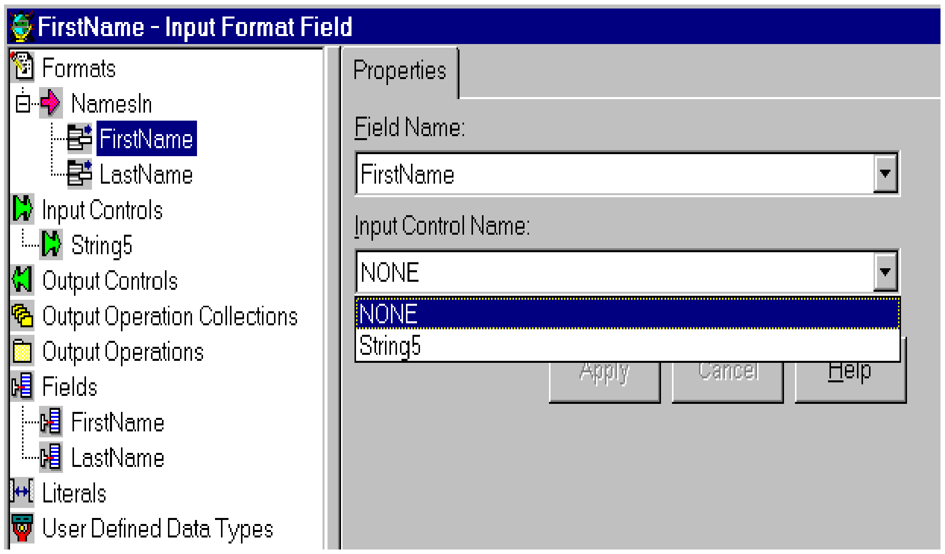


Figure 18. Tutorial: Assign an Input Control to a Field

Note: We found that you must select the name from the tree view on the left side of the window. Choosing it from Field Name on the right side and then assigning an input control to it did not work.

This completes the definition of the input format. At this point you could use the Visual Tester and check if your definitions are correct. How to do this is explained in 3.3.1, “How to Test If the Input Message Is Defined Correctly” on page 46.

3.1.4 How to Define an Output Control

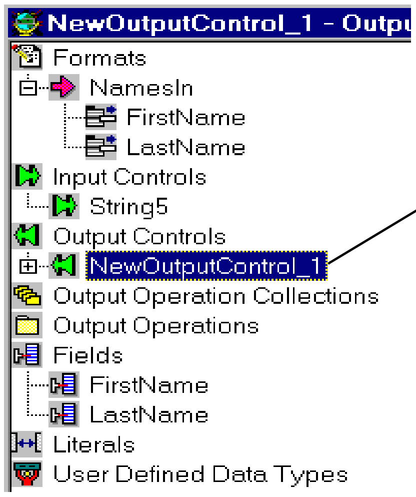
For the output format we use the same fields as for the input. We defined the field names in 3.1.1, “How to Define a Field” on page 30.

For the Formatter to re-format the message we also have to specify an output control. There are some predefined output operations. They are:

- CENTER_JUSTIFY
- LEFT_JUSTIFY
- RIGHT_JUSTYFY
- LOWER_CASE
- UPPER_CASE
- NONE

Since we use one of the predefined output operations we don’t need to define one.

Figure 19 shows part of the Formatter window that explains how to create a new output control.

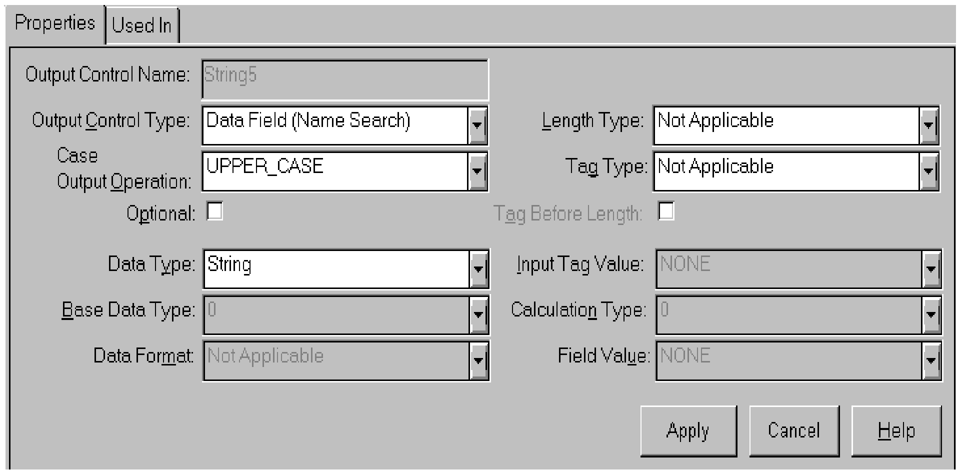


1. Right-click **Output Controls.**
2. Select **New** from the pop-up.
3. Change the name of the object to String5.
4. Press Enter.

Figure 19. Tutorial: Creating an Output Control (2)

Figure 20 on page 36 shows what you will see next on the right side of the Formatter window.

Change the output control type from NONE to UPPER_CASE and then click **Apply** at the bottom of the window.

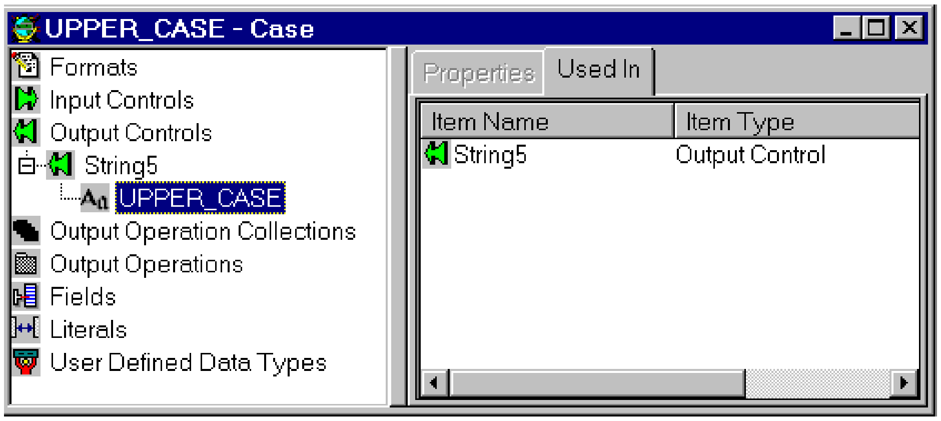


Output Control Name: String5
Output Control Type: Data Field (Name Search)
Length Type: Not Applicable
Case Output Operation: UPPER_CASE
Tag Type: Not Applicable
Optional: ☐
Tag Before Length: ☐
Data Type: String
Input Tag Value: NONE
Base Data Type: 0
Calculation Type: 0
Data Format: Not Applicable
Field Value: NONE

Figure 20. Tutorial: Creating an Output Control (2)

Figure 21 shows what you see when you expand the Output Controls tree. It clearly states that the output operation UPPER_CASE is used in the output control String5.

You can display the output operations when you click in the drop-down box in the window shown in Figure 20 or **Output Operations** on the left side on the Formatter window.



Item Name	Item Type
String5	Output Control

Figure 21. Tutorial: Output Control Example

3.1.5 How to Define an Output Format

The output format is a flat format that contains the same two fields as the input format, however in reversed order. Right-click on **Formats** and then select what's highlighted in Figure 21.

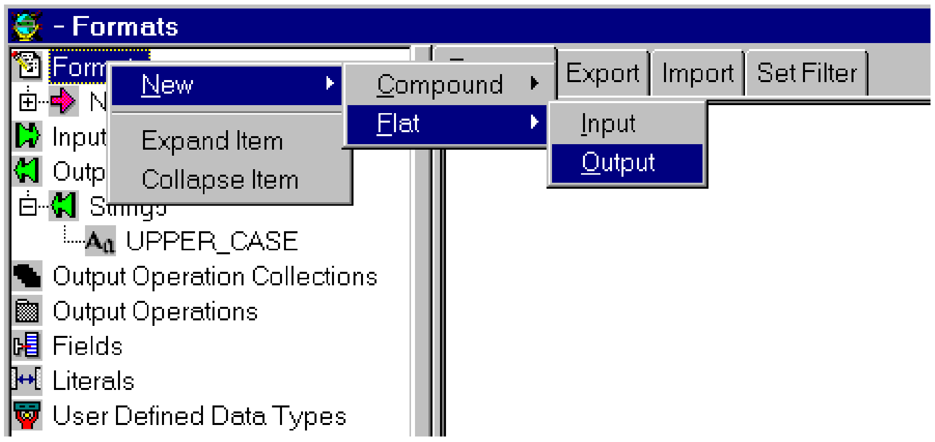


Figure 22. Tutorial: Creating a Flat Output Format

Then give the new object the name **NamesOut** and press Enter. This causes an entry to be made in the MQSI database.

Right-click **NamesOut** and select **Add Field Components** as shown in Figure 15 on page 33. This brings up the Field window shown in Figure 16 on page 33. Highlight FirstName and LastName and click **Accept Selection.** In this case, there should be only those two fields in the list.

Next, you have to order the fields as they have to appear in the output message, last name first and then the first name. Highlight FirstName, drag it upwards over LastName and then drop it.

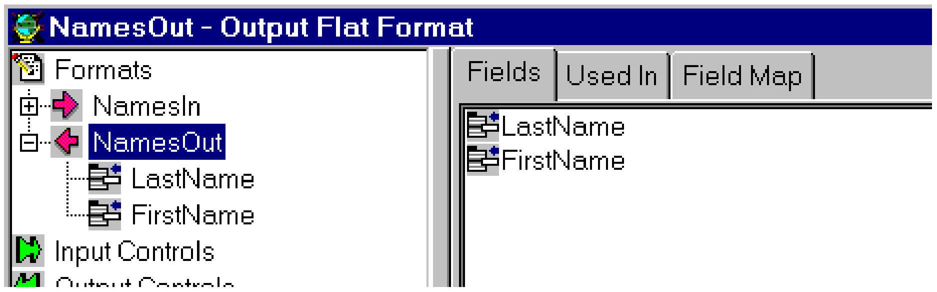


Figure 23. Tutorial: Output Format with Fields

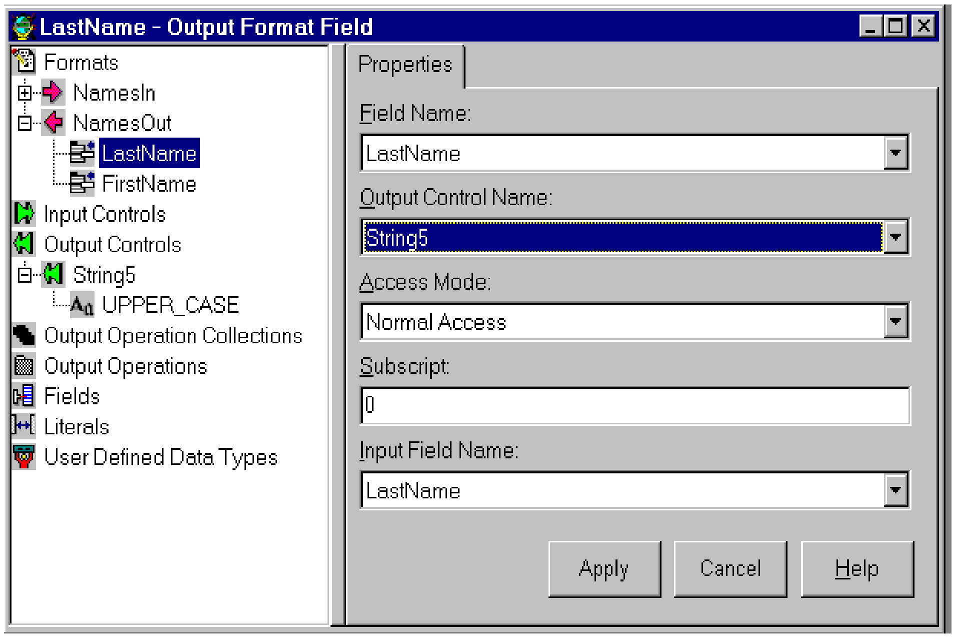
So far we have defined that the message consists of two fields and we also specified the order of the fields. What we have to do now is to tell the Formatter what to do with fields when it re-formats the message.

We have to assign the output control String5 to both fields. This control includes the output operation that converts all characters to upper-case. Click **LastName** and you see the properties shown in Figure 24. The names of the input and output fields are already set. When you click the Output Control Name list two choices are displayed:

- NONE
- String5

Select **String5** and click **Apply**.

Repeat this scenario for FirstName.



Field Name: LastName
Output Control Name: String5
Access Mode: Normal Access
Subscript: 0
Input Field Name: LastName

Figure 24. Tutorial: Assign an Output Control to a Field

This concludes the definition of the output format. At this point you could use the Visual Tester and check if your definitions are correct. How to do this is explained in 3.3.2, "How to Test If the Output Message Is Defined Correctly" on page 48.

3.2 Working with Rules

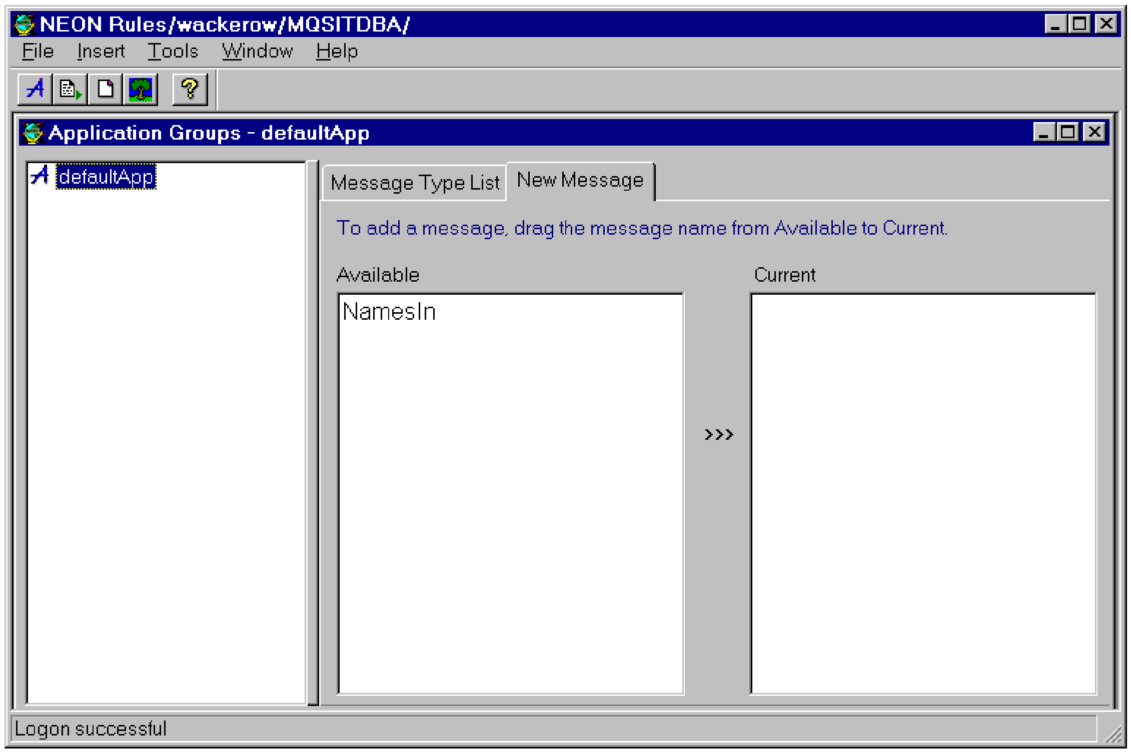
In this section, we will create the rule that tells the MQSI run-time program, the rules processor, to get a message, parse and reformat it, and then put it on an output queue. To bring up the Rules GUI click the **Start** button and select from the menu:

Programs
  MQSeries Integrator
    Rules

This will bring up the logon window where you have to enter the following:

User ID:	Your User ID
Password:	Your password
DBMS:	Select **ODBC - DB2(ODBC)** from the drop-down list
Driver:	Select the alias database name, here **MQSITDBA**
Qualifier:	For DB2, leave this field blank

Click on **OK** to log on. Next, you will see the Rules window:


NEON Rules/wackerow/MQSITDBA/
File Insert Tools Window Help
Application Groups - defaultApp
defaultApp
Message Type List
New Message
Available
Current
>>>

To add a message, drag the message name from Available to Current.

NamesIn

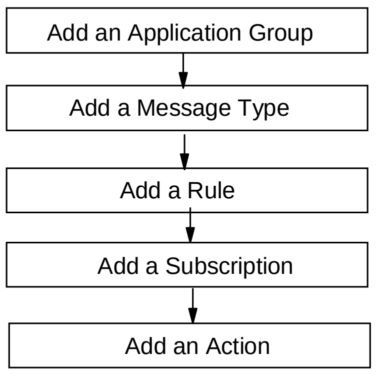
Logon successful

Figure 25. MQSI Rules Window

Note: The left pane in the Rules window shown in Figure 25 on page 39 contains only one object, the default application group "defaultApp". This group is always there.

When you click the New Message tab you will see a list of all input formats in the database. In our case, that is our only input message, NamesIn.

The steps to create a rule for an input message are:



- The MQSI database can contain definitions for several application groups. For this exercise, we use the default application group.
- An application group may have many messages. For example, the programs that comprise the payroll application may process messages that contain personal data changes, overtime payments and payroll deductions, to name some. Each of these messages is referred to as a message type.

  Note: Don’t confuse the MQSI message type with the MQSeries message type (request, reply, datagram).
- Each message type has at least one rule associated with it. Rules determine what happens to the message, for example, which output format is associated with which input format.
- Each rule is associated with a subscription. A subscription defines one or more actions, such as put a message on a queue.

To complete the definitions for this tutorial, we have to do the following:

1. Add the message (type) NamesIn to the default application.
2. Define the rule NamesRule.
3. Add and define the subscription NamesSub to the subscription list.
4. Associate the subscription with the rule.

3.2.1 How to Add a Message Type to an Application

You see in Figure 25 on page 39 that the database contains only one message, NamesIn. To add this message to the application group defaultApp, click **NamesIn** in the Available panel and drag it into the Current panel to its right.

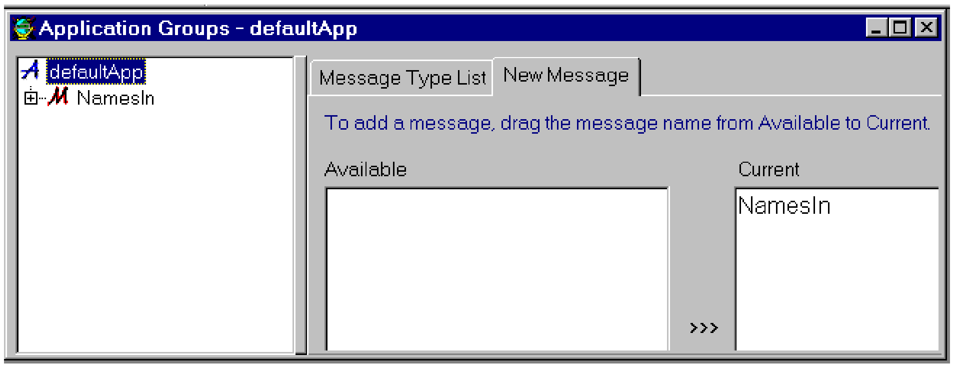

Application Groups - defaultApp
defaultApp
NamesIn
Message Type List
New Message
To add a message, drag the message name from Available to Current.
Available
Current
NamesIn
>>>

Figure 26. Tutorial: Creating a Message Group

Note: To remove a message from the application group (Current list), highlight it by clicking the name and pressing the Delete key.

3.2.2 How to Define a Rule

In the Rules window, right-click the message **NamesIn** and then click **New**.

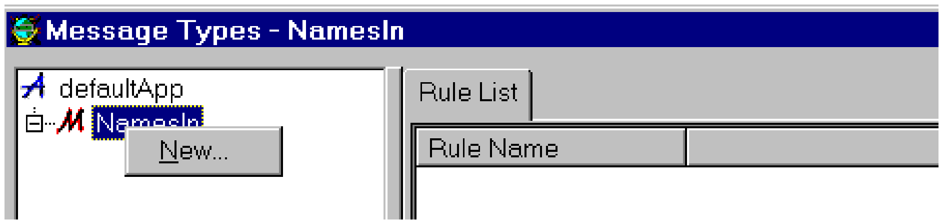
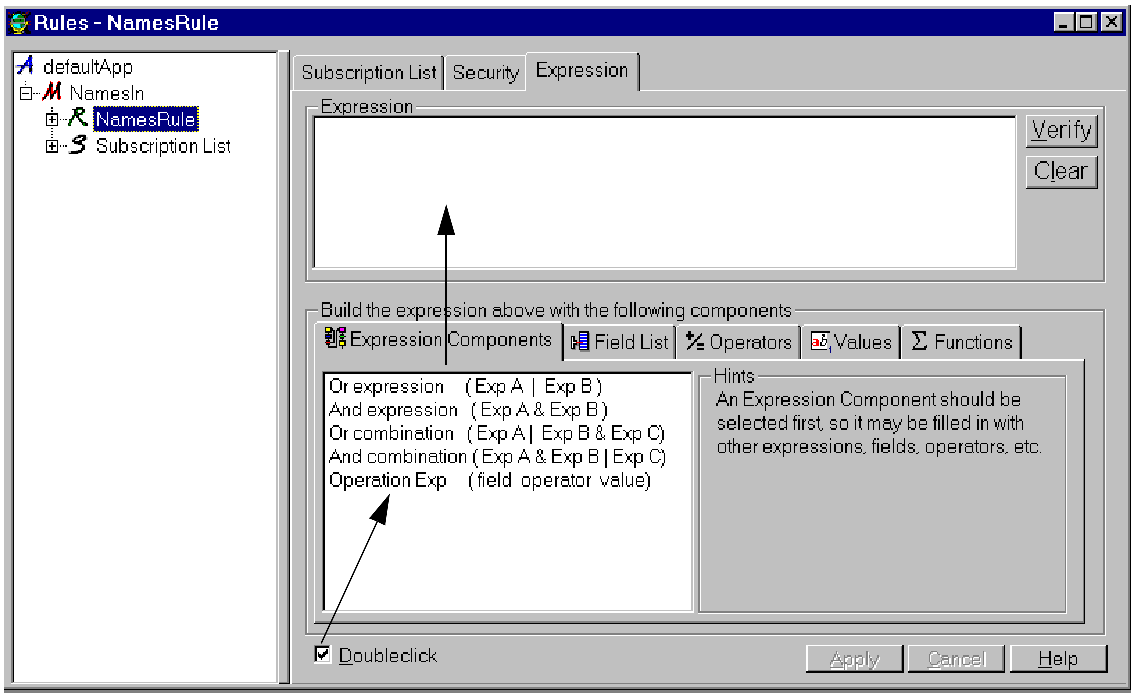


Figure 27. Tutorial: Adding a Rule

Then overwrite NewRule1 with the rule name and press Enter. We chose to name the rule NamesRule. Next, you will see the window in Figure 28 on page 42.

Note: Names of rules are only to help you organize and potentially for export and inport. They are nor used at run-time, except for diagnostic purposes. Users rarely need to care about rule names.



Or expression (Exp A | Exp B)
And expression (Exp A & Exp B)
Or combination (Exp A | Exp B & Exp C)
And combination (Exp A & Exp B | Exp C)
Operation Exp (field operator value)

Hints
An Expression Component should be selected first, so it may be filled in with other expressions, fields, operators, etc.

Figure 28. Tutorial: Defining a Rule

We want to define a rule that checks if both fields in the message, FirstName and LastName, are present, that is, if both fields have been properly parsed. You can type the rule in the Expression box or use the functions provided to build the expression. Let's do the latter:

1. Double-click **And expression** in the Expression Components list. In the Expression box you will now see the text:

   ```
   ( Exp A & Exp B )
   ```

2. Exp A is highlighted. Press the Delete key to erase this text.

3. Click the **Field List** tab. You will notice our two fields in the list. Double-click **FirstName**. The Expression box now shows:

   ```
   ( 'FirstName' & Exp B )
   ```

4. Click the **Operators** tab to display the operators MQSI provides. Double-click **EXISTS**. The Expression box now contains:

   ```
   ( 'FirstName' EXISTS & Exp B )
   ```

5. Delete Exp B, click the **Field List** tab and double-click **LastName**. In the Expression box you will now see:

```
( 'FirstName' EXISTS & 'LastName' )
```

6. Click the **Operators** tab again and double-click **EXISTS**. The expression is now complete:

```
( 'FirstName' EXISTS & 'LastName' EXISTS)
```

7. Click **Verify.** The MQSI will inform you that the syntax is OK.
8. Click **Apply**.

The rule is now defined.

3.2.3 How to Specify Actions for a Rule

In the Rules window, right-click **Subscription List** and select **New.** Then overwrite NewSubscription1 with a name of your choice, for example, NamesSub and press Enter.

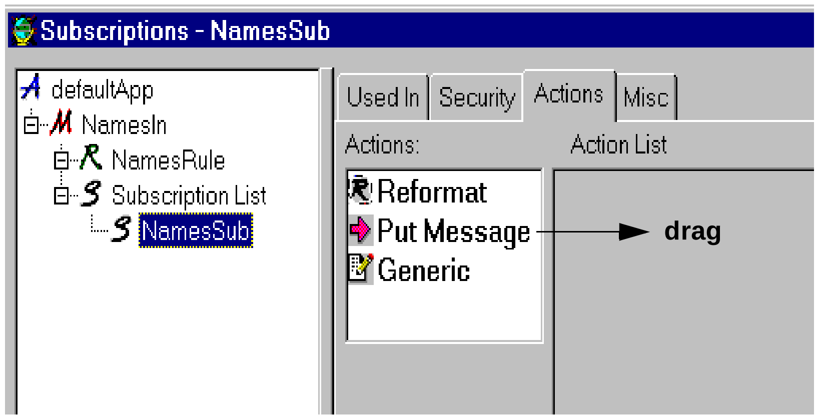
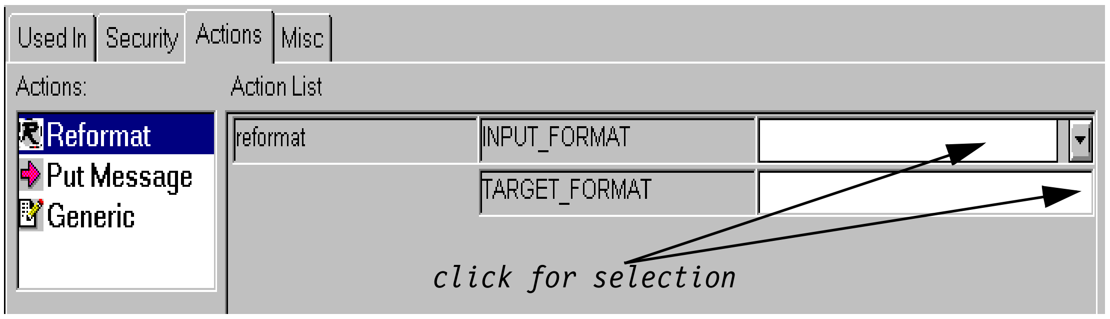


Figure 29. Tutorial: Creating a Subscription

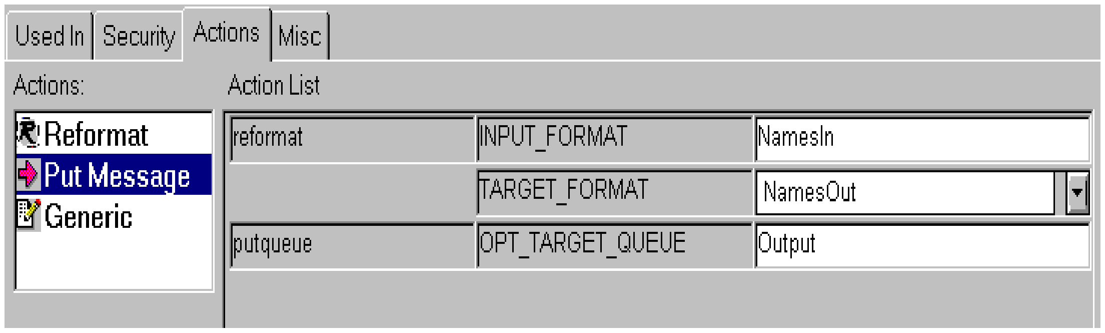
In this example, we want two actions applied to the message: reformat it and put it on a queue. In Figure 29 you see three actions. We are only concerned with the first two.These actions have to be placed in the action list:

1. Click **Reformat** and drag it into the Action List. The window changes now to what you see in Figure 30 on page 44.
2. Click the drop-down box to the right of INPUT_FORMAT. In our case, the list contains only one message, namely **NamesIn**. Select it.
3. Click somewhere in the field next to TARGET_FORMAT and then display the list of output formats available. Since we have only one, click **NamesOut**.
4. Click **Put Message** and drag it into the Action List field.



reformat	INPUT_FORMAT	
	TARGET_FORMAT	

Figure 30. Tutorial: Defining a Reformat Action



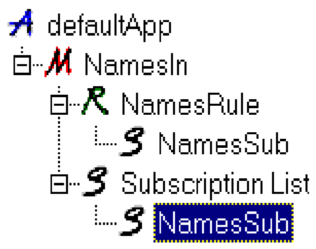
reformat	INPUT_FORMAT	NamesIn
	TARGET_FORMAT	NamesOut
putqueue	OPT_TARGET_QUEUE	Output

Figure 31. Tutorial: Defining a Put Message Action

5. You are required to enter the name of the queue where the MQSI shall put the output message. We use the queue Output. *You must specify a put message action or the message is lost.*

   Note: Queue names are case-sensitive.

6. To associate the subscription with the rule NamesRule click **NamesSub** under Subscription List and drag it on top of the rule. This completes the rules definition. The defaultApp tree shows the following objects:



```
defaultApp
⊟ NamesIn
   ⊟ NamesRule
      NamesSub
   ⊟ Subscription List
      NamesSub
```

3.3 Testing Formats and Rules with the Visual Tester

We found the Visual Tester, a Support Pack, a valuable tool to test input and output formats during definition time. In this section we explain how to use the product to check out the previously defined input and output formats, NamesIn and NamesOut.

It is assumed that this program is installed and usable. Detailed information on how to install and use this product can be found in 4.8, “The Visual Tester” on page 77. Before you start the Visual Tester create a *default queue manager* and start it, for example:

```
crtmqm /q QAQM
start qmgr
```

Note: You may have done this already for the IVP (see 2.1.8, “Verifying the Installation” on page 16).

Start the Visual Tester from the Start menu, usually by selecting:

Programs
  Visual Tester
    Visual Tester

When the logon window appears, type the following:

- Your user ID
- Your password
- The name of the database alias, here MQSITDBA, in the Driver field
- Leave the field Queue Session Name unchanged

Then click **OK**. This causes the Visual Tester window to appear. This window has four tabs:

1. Load Q
2. Test Format
3. Test Reformat
4. Test Rule

For this exercise, we will walk through the last three functions:

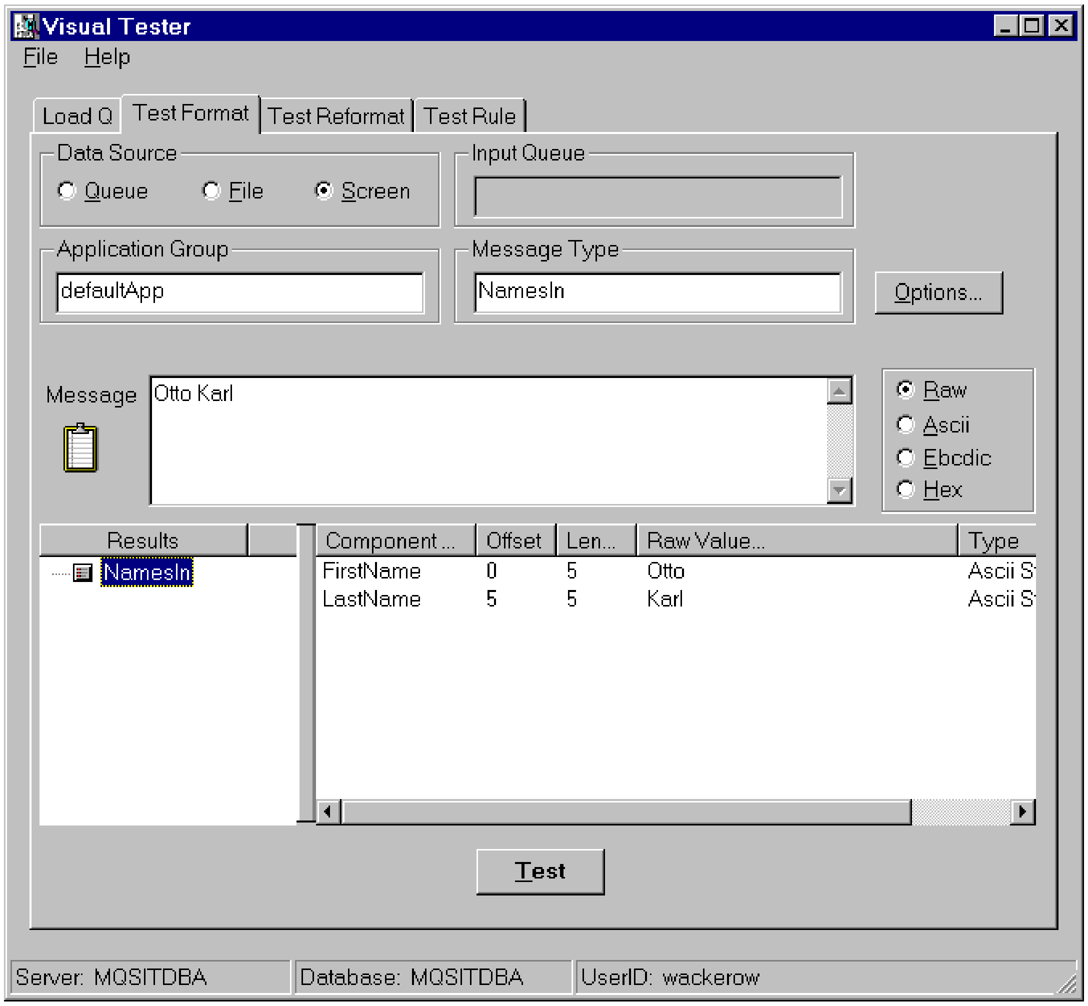
- Validate the definitions for the input format (test format).
- Use the reformatting function to see if the output message looks as expected.
- Find out how the rules processor processes our messages.

3.3.1 How to Test If the Input Message Is Defined Correctly

Figure 32 on page 46 shows the Visual Tester window with the tab Test Format selected. You see that the two input fields are parsed correctly.

The input message can be in a queue, a file, or you can type it in the Message field. Here, we selected **Screen** as the data source and typed the message.

Note: There is a blank at the end of the message to make it exactly 10 characters long.



Data Source: Queue File [✓ Screen]

Input Queue:

Application Group: defaultApp

Message Type: NamesIn

Message: Otto Karl

[✓ Raw] Ascii Ebcdic Hex

Results: NamesIn

Component ...	Offset	Len...	Raw Value...	Type
FirstName	0	5	Otto	Ascii S
LastName	5	5	Karl	Ascii S

Server: MQSITDBA | Database: MQSITDBA | UserID: wackerow

Figure 32. Tutorial: Test Input Format Using the Visual Tester

The fields for which you have to provide values are:

Application Group	Specify "defaultApp" as for the IVP. Actually, to test the input message you can enter any value or leave the field blank.
Message Type	This field is mandatory; specify NamesIn. The program displays an error message when a message with this name cannot be found in the database.
Screen	When this radio button is selected, you can type your input messages in the Message field.
Message	Type the input message as defined in the format. Be careful and type exactly 10 characters.

Note: Since we defined a fixed length message, you must type blanks at the end of the name if it is shorter than five characters.

If you make an input error, the program cannot parse the message and displays an error message, such as:

- Mandatory input field "FirstName" not found
- Mandatory input field "LastName" not found
- n trailing characters "abc..." after message parse

As the result of the test, the components or fields of the *input message* are displayed in the white area at the bottom of the window.

If You Make Corrections

If you correct your definitions you have to have to refresh the Formatter and rules data stored in the Visual Tester’s cache:

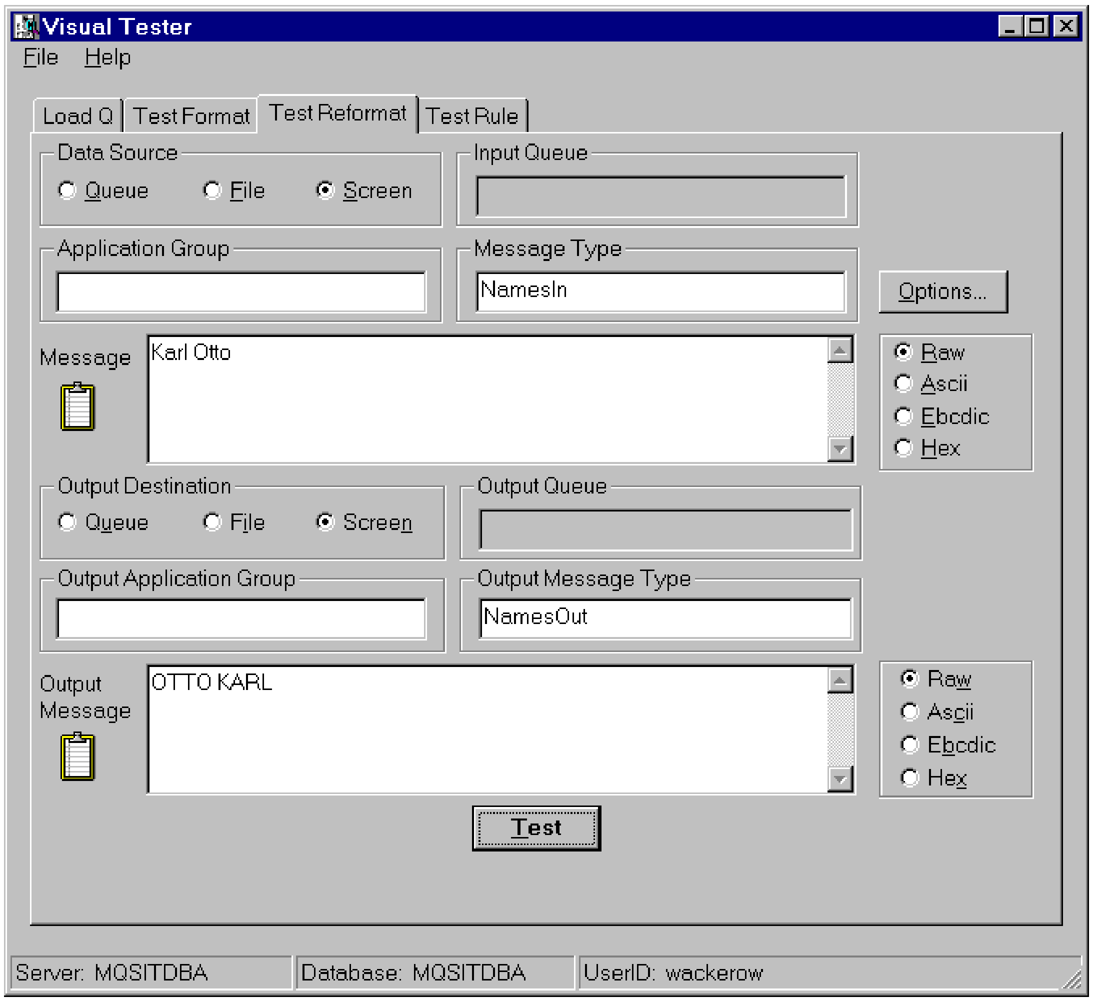
- Click **Options**
- Check Recache Rules/Formatter
- Click **OK**

3.3.2 How to Test If the Output Message Is Defined Correctly

Figure 33 on page 48 shows the Visual Tester window with the tab Test Reformat selected. You see that the two input fields are parsed correctly and then reversed and translated to upper-case.

As in the previous example, the input message can be in a queue, a file, or you can type it in the Message field.

Note: There is a blank at the end of the input message to make it exactly 10 characters long.



Data Source: Queue [✓ Screen] File
Input Queue:
Application Group:
Message Type: NamesIn
Message: Karl Otto
[✓ Raw] Ascii Ebcdic Hex
Output Destination: Queue File [✓ Screen]
Output Queue:
Output Application Group:
Output Message Type: NamesOut
Output Message: OTTO KARL
[✓ Raw] Ascii Ebcdic Hex
Server: MQSITDBA
Database: MQSITDBA
UserID: wackerow

Figure 33. Tutorial: Test Reformatting Using the Visual Tester

The fields for which you have to provide input are:

Application Group	Specify "defaultApp" as for the IVP. Actually, for this test you can enter any value or leave the field blank.
Message Type	This field is mandatory; specify the name of the input format, here NamesIn.
Output Message Type	This field is mandatory, too. Type the name of the output format, here NamesOut.
Data Source	Select the **Screen** radio button which allows you to type the input message into the Message field.
Output Destination	Select the **Screen** radio button so that the Formatter output appears in the Result field.

The error messages are the same as for testing the input field.

As the result of the test, the components or fields of the *output message* are displayed in the white area at the bottom of the window.

3.3.3 How to Test If the Rules Work Properly

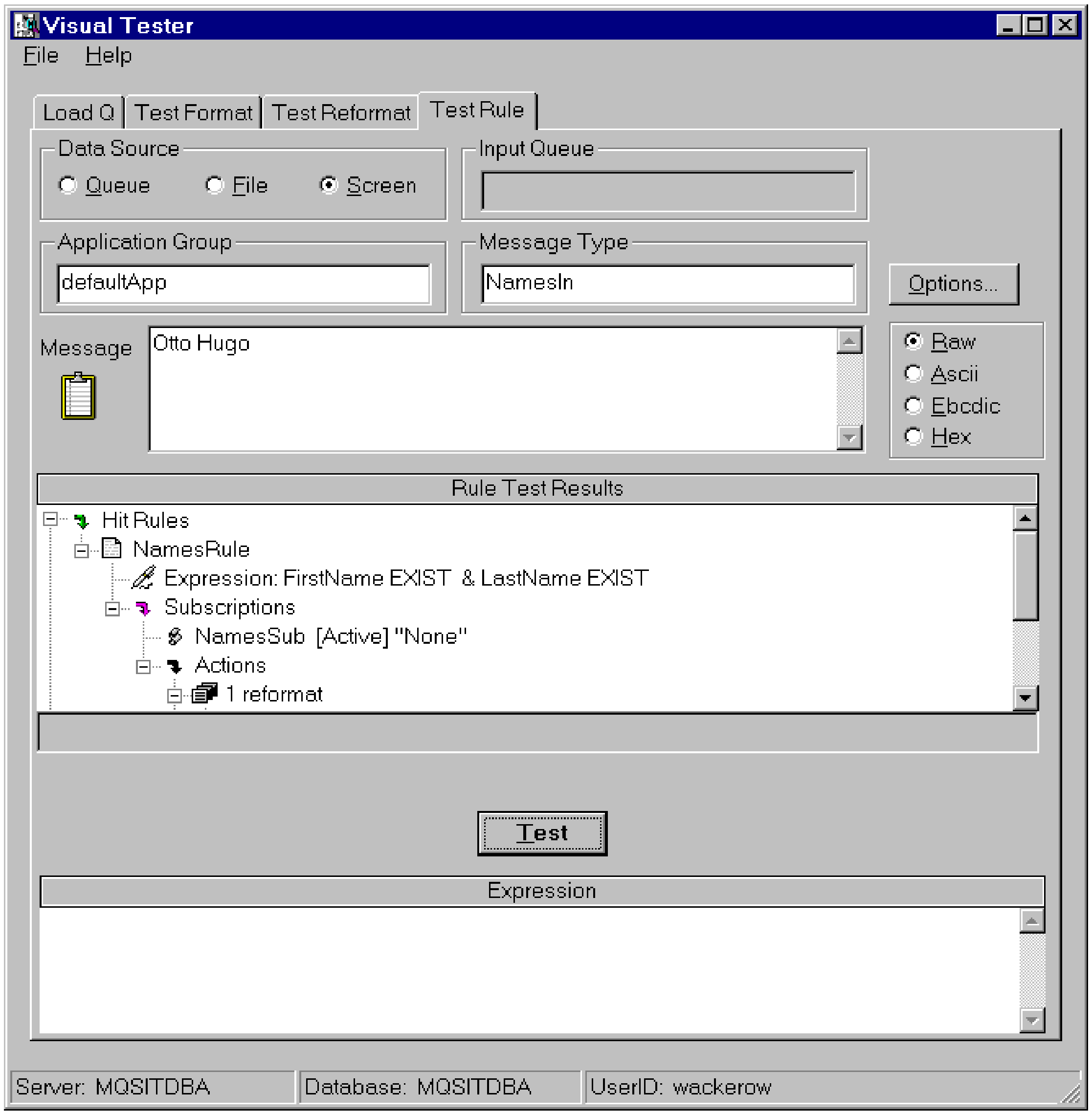
Figure 34 on page 50 shows the Visual Tester window with the tab Test Ruled selected. You see the fields of the input message and the rule test results.

First, the program parses the input message. If you don’t type the correct number of characters, here 10, it displays an error message. After the message is parsed correctly, the program checks if there is a rule that matches the input. In this case, we have only one rule defined. Therefore, we will always have a "hit".

Note: In the example, the rules are only tested when exactly 10 characters have been entered.

The fields you have to enter are:

Application Group	The name of the application group the input message belongs to, here defaultApp. We decided to use this name in 3.2, “Working with Rules” on page 39.
Message Type	This field is mandatory; specify NamesIn. We decided on this name in 3.2.1, “How to Add a Message Type to an Application” on page 41.
Data Source	Select the **Screen** radio button which allows you to type the input message into the Message field.



Data Source: Queue File [✓ Screen]

Input Queue:

Application Group: defaultApp

Message Type: NamesIn

Message: Otto Hugo

[✓ Raw] Ascii Ebcdic Hex

Rule Test Results

```
Hit Rules
  NamesRule
    Expression: FirstName EXIST  & LastName EXIST
    Subscriptions
      NamesSub  [Active] "None"
      Actions
        1 reformat
```

Expression

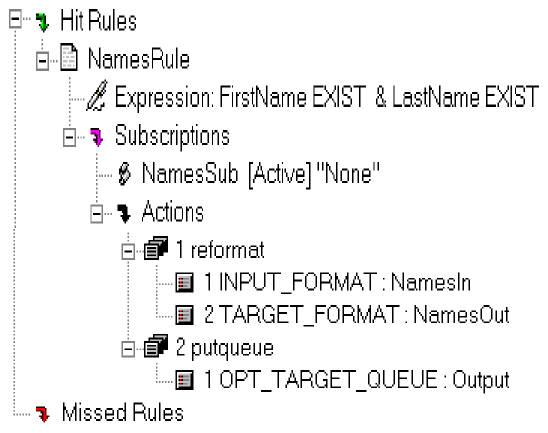
Server: MQSITDBA | Database: MQSITDBA | UserID: wackerow

Figure 34. Tutorial: Test Rules Using the Visual Tester

Message	Enter here the fields of the input message. Since it contains fixed length fields, type exactly 10 characters, otherwise you see a pop-up window telling you that the rules processor failed parsing the message. The error

message tells you the name of the format it used and the number of characters you typed.

As a result of the test, you see the following tree structure:



```
Hit Rules
  NamesRule
    Expression: FirstName EXIST & LastName EXIST
    Subscriptions
      NamesSub [Active] "None"
      Actions
        1 reformat
          1 INPUT_FORMAT : NamesIn
          2 TARGET_FORMAT : NamesOut
        2 putqueue
          1 OPT_TARGET_QUEUE : Output
Missed Rules
```

The tree structure shows:

- That we hit a rule; in this case we have only one
- What expression was used (first and last name must exist)
- The subscription name and the actions associated with it:
  1. The message has been reformatted
  2. The message is destined for the queue Output
- No rules were missed (since we have only one and that one matched)

Note: You can specify a queue or a file as a data source. However, the function to test the rules will not write an output message to a queue or file.

3.4 Using the Rules Processor

To test our application in a real life environment using the MQSI rules processor we have to do the following:

1. Prepare the environment.
2. Create the rules processor configuration file (mpf file).
3. Start the rules processor.

3.4.1 How to Set Up the Environment

Before we can start the rules processor we have to set up the MQSeries environment:

1. Create a queue manager. You create the default queue manager QAQM with this command:

   ```
   crtmqm /q QAQM
   ```

   The advantage of creating a default queue manager is that you don't have to type a queue manager name for any of the MQSeries commands or utilities.

1. Start the (default) queue manager with the command: `strmqm`

2. If not already done so for the IVP, start runmqsc and create the following queues:

   ```
   define qlocal('RulesIn') replace
   define qlocal('RulesNoHit') replace
   define qlocal('RulesFail') replace
   define qlocal('Output') replace
   ```

3. Use runmqsc to verify that the queues are empty:
   display ql(*) curdepth

   You should see the following output:

   ```
   QUEUE (RulesIn)      CURDEPTH (0)
   QUEUE (RulesNoHit)   CURDEPTH (0)
   QUEUE (RulesFail)    CURDEPTH (0)
   QUEUE (Output)       CURDEPTH (0)
   ```

4. Use amqsput to put some messages into the input queue:

   ```
   C:\>amqsput RulesIn
   Sample AMQSPUT0 start
   target queue is RulesIn
   abc
   Karl Otto
   JesseJames
   King Arthur
    <=== enter a blank line
   Sample AMQSPUT0 end
   ```

   The queue RulesIn now contains four messages, two have the correct length while the first one is too short and the last one too long.

3.4.2 What the Rules Processor Configuration File Is For

The rules processor needs a configuration file that describes what queues to use, to what queue manager to connect, and what default application to use, to name three.

The rules processor configuration file MQSIruleseng.mpf comes with the product. For this test, we make a copy of it, name it tutorial.mpf and modify it. The file is shown in Figure 35 on page 54.

Following are some notes to the fields in the file:

1. The name of the queue manager the rules processor connects to is QAQM. It is expected that the queue manager is running before the rules processor starts. This queue manager owns all queues the rules processor uses.
2. Here we define the three default queues required by the rules processor:
   - RulesIn contains the input messages.

     Note: In Version 1.0 of the MQSI, the rules processor accepts only one input queue.
   - If an input message does not match any of the rules it will be written to the RulesNoHit queue.
   - If a rule fails, the wrong message is written to the RulesFail queue. For example, this is the case when we provide a message with the wrong length.

   Note: The output queue is not specified in this file. It is defined in the rule. The rules processor is able to write messages to more than one queue.
3. The rules processor requires that we specify a default application group and a default message type. In this example, we use the same application group as the IVP. The default message type for the application group defaultApp is NamesIn. We decided this when we created the rule in 3.2.1, “How to Add a Message Type to an Application” on page 41.

   These defaults are used when the message itself does not contain an MQSI header that specifies application and (MQSI) message type. This subject is discussed later.
4. The rules processor has to know where to get the input, output and rules definitions. For DB2 we specify the database alias MQSITDBA.
5. Since we use DB2, the database type is set to 5.

```
[Queues]
# Parameters related to queues, MQSeries control, and rules engine control

# Alternate User Authority Flag
CredentialsEnabled = 0

# MQSeries queue manager name
QueueManagerName = QAQM                              1

# retry limit
MaxBackoutCount= 0

# these three queue names are mandatory!
InputQueueName   = RulesIn
NoHitQueueName   = RulesNoHit
FailureQueueName = RulesFail                         2

# rules default application group and message type values (mandatory)
DefaultAppGroup = defaultApp
DefaultMsgType  = NamesIn
                                                     3
[Logging]
# Log levels:
#3 - log only fatal errors
#2 - log errors, and fatal errors
#1 - log warnings, errors, and fatals
#0 - log informationals, warnings, errors, and fatals
LogFileName = mqsiruleng.log
LogLevel    = 0

[Rules Database ]
# all fields are mandatory)
ServerName       = MQSITDBA
UserId           = userabc
Password         = password                         4
DatabaseInstance = MQSITDBA
#
# DatabaseType is a numeric with these values:
#       SYBASE = 1      MQSQL = 2      ORACLE = 3
#       DB2    = 4      ODBC  = 5
#
DatabaseType = 5                                    5
```

Figure 35. Tutorial: Rules Daemon Configuration File tutorial.mpf

3.4.3 How to Start the Rules Processor

When the MQSeries environment is set up and a rules processor configuration file is present, we are ready to start the rules processor. In a command prompt window, issue the command:

```
mqsiruleng -p tutorial.mpf
```

The rules processor will never end unless you send a shutdown message or kill it yourself. It constantly checks the input queue for messages. You may minimize the window.

Now let us find out what happened to the four messages we put in the input queue RulesIn (see page 52).

In another command prompt window, start runmqsc and check how many messages are in the queue. The mqsc command you type is:

```
display ql(*) curdepth
```

The output should tell you the following:

- The queue RulesIn is empty, all messages have been processed.
- The queue Output contains two messages, namely the one with the correct length.
- The queue RulesFail contains two messages, the one that is too short and the one that is too long.

You can use amqsget to retrieve the messages from the queues:

```
C:\>amqsget Output QAQM
Sample AMQSGET0 start
message < Otto Karl >
message < JamesJesse>
no more messages
Sample AMQSGET0 end

C:\>amqsget RulesFail QAQM
Sample AMQSGET0 start
message < abc>
message < King Arthur>
no more messages
Sample AMQSGET0 end

C:\>
```

Chapter 4. Tools That Help with Development

The MQSeries Integrator consists of three main parts:

- The Formatter GUI, a program to develop format defintions
- The Rules GUI, a program to develop rules
- The Rules Daemon, the MQSI's run-time program

In addition, the product provides a number of tools to test if the definitions entered into the MQSI database work with the real data as intended. The tools allow you to quickly test your MQSeries Integrator formats and rules using input and output messages without the need to code programs.

You may also have the need to export and import definitions. This chapter describes a number of tools provided with the MQSl. We also include the Visual Tester which at the time of writing this book is available as a Support Pack. The tools we write about are listed in Table 3:

Table 3. MQSI Tools

Program Name	What It Does
msgtest	Tests message reformatting; input and output messages in files
ruletest	Tests rules; input message in file, output on screen
NNRTrace	Tests actions for a specific rule
MQSIputdata	Puts messages in a queue to be processed by the Rules daemon; can add MQSI header (RFH) and application type and message type; needs parameter file MQSIputdata.mpf
MQSIgetdata	Gets message from a queue put there by the Rules daemon; needs parameter file MQSIgetdata.mpf
NNFie	Imports and exports formats
NNRie	Imports and exports rules
Visual Tester	Tests formats, reformatting and rules; input can come from a queue, a file or it can be typed

Of course, you may also use some of the utilities supplied with MQSeries, such as runmqsc, amqsput. amqsget, and amqsgbr.

4.1 Using msgtest to Test If a Message Reformats Correctly

A simple tools to test formats is the msgtest program. It is in the Windows NT directory \mqi\bin. You can use it to test your input and output formats defined with the Formatter GUI.

To find out what parameters you can specify, open a command prompt window, change the directory to \mqi\bin and type the command `msgtest /?`. The output is shown below.

```
D:\MQI\bin>msgtest /?
 Formatter Reformatting Test Tool (msgtest)
 NeoNet Version 4.01
 Version 1.23.2.2.2.3, last changed on 1998/04/14 21:23:33
 Copyright (C) 1996-198, New Era Of Networks, Inc.
 All Rights Reserved.

usage: msgtest [-li] [-lo] [-lf] [-nv] [-d [<file name>] [-dcp] [-dcm] [-dco]]
        -li:    loud input
        -lo:    loud output
        -lf:    loud formatted value
        -nv:    no validation
       -d:     debug on (debug parse only if -dcp and -dcm and -dco not specified)
        -dcp:   debug parse on
        -dcm:   debug map on
        -dco:   debug output on

D:\MQI\bin>
```

Figure 36. Tools: Parameters for msgtest

Notes:

- The -li, -lo and -lf parameters are used to echo the input, output and format elements back to the screen for checking.
- The -d, -dcp, -dcm and -dco debug parameters give more details about the data type, length and content of the individual elements.
- In the case of -dco the output controls are used.

Example:

1. If you want to check the format defaultMsg provided with the IVP, create a small file called defaultMsg.txt in the \mqi\bin directory. Then type in the message text `hello;world;` and save.

Notes: Do not add a carriage return at the end of the text as this will count as another character in your message.
The file can contain only a single message.

2. To test the defaultMsg format, at the command prompt type the command `msgtest` with no parameters and press Enter. You will then be prompted for the input file name, output file name, input format and output format. For our example this would appear as shown in Figure 37. The bold text indicates your input.

```
D:\MQI\bin>msgtest
Enter the input file name:
defaultMsg.txt
Enter the output file name:
defaultOut.txt
Enter the input format name:
defaultMsg
Enter the output format name:
defaultOut
Message count: 1

Success.  Hit return.
```

Figure 37. Tools: Example of Using msgtest

Note: Format names are case-sensitive.

To end the program press Return and then Ctrl-C. You will find that you have an output file called defaultOut.txt in the \mqi\bin directory that contains your reformatted output text message WORLD HELLO.

4.2 Using ruletest to Evaluate Rules

This tool is used to test which rules and subscriptions *would* be executed and to evaluate which actions *would* take place for a specific input message. This program does not actually execute subscriptions using the formatter.

The ruletest utility gets the input message from a file. You have to provide the application group and the message type.

This program resides in the \mqi\bin directory. To find out what input parameters you can use, type on a command line the command `ruletest -?`.

```
D:\MQI\bin>ruletest /?

Rules evaluation test program (ruletest)
 NeoNet Version 4.01
Version 1.11.6.3, last changed on 1998/05/07 19:01:01
Copyright (C) 1996-1998, New Era Of Networks, Inc.
All Rights Reserved.

Usage:
        ruletest
                -i <Input-File-Name>
                -a <Application-Group>
                -m <Message-Type Format-Name>
                [-s <Session-Name>]     (default = "rules")
                [-v (Verbose)]


D:\MQI\bin>
```

Figure 38. Tools: Parameters for ruletest

You can run the program interactively. The example shown in Figure 39 on page 61 uses the formats and rules from the IVP. The input you have to provide is shown in bold.

Notes:

1. Use your favorite editor to create the input file. The file can contain only a single message.
2. Do not add a carriage return at the end of the text in the input file as this will count as another character in your message.
3. The names of the application group and message type are case-sensitive.
4. When prompted for the session name press Enter to use the default. This name is specified in the database connection file, sqlsvses.cfg. For more information refer to 2.1.7, “The Database Connection File” on page 15.
5. If you set the -v parameter on, the ruletest program logs to the screen, for example:

   ```
   Opening input file named defaultMsg.txt
   File opened successfully
   New message
   hello;world;
   End of message
   ```

6. You have to reload the rule set after you have made changes to the database that relate to the example your are testing.
7. Use Ctrl-C to end the program.

```
D:\MQI\bin>ruletest
Enter the session name (default = "rules"):

Enter the input file name:
defaultMsg.txt
Enter the application group name:
defaultApp
Enter the message type name:
defaultMsg
Verbose on [y/n]:
n
Do you want to reload the rule set? [y/n]:
n

New message
hello;world;
End of message

NO HIT RULES - Rule Name (Id)

HIT RULES - Rule Name (Id)
    defaultRule(4)

ACTIONS
    Action(Id): reformat(2)
        1 : INPUT_FORMAT - defaultMsg
        2 : TARGET_FORMAT - defaultOut
    Action(Id): putqueue(2)
        1 : OPT_TARGET_QUEUE - Output
```

Figure 39. Tools: Example of Using ruletest

4.3 Debugging Rules Using NNRTrace

Similarly to ruletest, we can use NNRTrace to test and debug rules. This program shows us whether a rule will hit or not, and what actions will be performed based on the subscriptions of the rule. The main difference between NNRTrace and ruletest is that for NNRTrace you specify the rule you want to test by name. Figure 40 on page 62 shows an example. The input is shown in bold.

NNTrace runs a message through a specified rule. The message comes from an input file. The program displays each argument with its appropriate field value derived from the message.

```
Enter the input file name:
defaultMsg.txt
Enter the application group name:
defaultApp
Enter the message type name:
defaultMsg
Enter the rule name:
defaultRule
Enter the session name (default 'rules'):

Enter the output file name (default none):

Verbose on [y/n]:
n
APPLICATION GROUP:  defaultApp
MESSAGE TYPE:  defaultMsg
RULE NAME:  defaultRule

EXPRESSION:  defaultfield1 EXIST  & defaultfield2 EXIST

INPUT MESSAGE:
hello;world;

FIELD NAME:  defaultfield1
VALUE 1:  hello
OPERATION:  EXIST

FIELD NAME:  defaultfield2
VALUE 1:  world
OPERATION:  EXIST

RULE WOULD HIT
SUBSCRIPTIONS:
        Subscription:  defaultSub

Action:  reformat
                        1 : INPUT_FORMAT - defaultMsg
                        2 : TARGET_FORMAT - defaultOut
                Action:  putqueue
                        1 : OPT_TARGET_QUEUE - Output
```

Figure 40. Tools: Example of Using NNRTrace

The mandatory parameters for NNRTrace are:

```
-i <Input-File-Name>
-a <Application-Group>
-m <Message-Type/Format-Name>
-r <Rule-Name>
```

Optional parameters are:

```
-s <Session-Name>       Default = rules
-o <Output-File-Name>   Default = Standard output
-v (Verbose)            Y = log to screen
```

4.4 Using MQSIputdata to Create Messages with RFH Header

At one time during the development process you have to test your formats and rules with real data from a queue. You can obtain messages from the production program that creates them or, if this program is not available yet, use the MQSIputdata tool. This program allows you to put one or more messages in a specific queue. You can also alter certain fields in the message descriptor (MQMQ) and add an MQSI Rules Format Header (MQHRF). This is useful if you don’t want to use the default application and default message type defined in the rules processor configuration file but the ones specified in the RFH header of the incoming message. Also, your target application may expect this header information.

The MQSIputdata tool is used in conjunction with:

- A parameter file, such as mqsiputdata.mpf, that is used to create the message header
- A flat file that contains the message data (one message only)

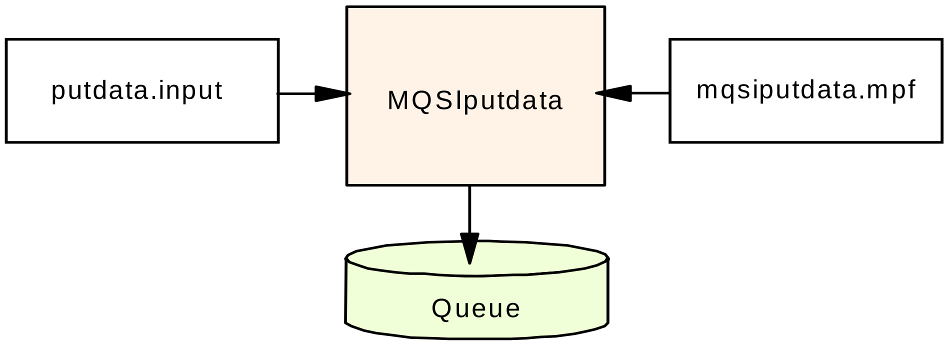


Figure 41. Tools: How MQSIputdata Works

The file putdata.input in Figure 41 on page 63 contains data of a single input message. The file mqsiputdata.mpf is used to set the following parameters:

Table 4. MQSIputdata Input Parameters

What you can specify	Values you can enter
inputFileNname	Name of file that contains data for a single message
queueName	Name of queue that will contains message(s)
queueManagerName	Name of queue manager MQSIputdata connects to
maxUserDataLength	Default =10,000 bytes of user data in a message
messageCount	Number of messages to put; the default is 1; you may replicate as many messages as you like
showStatistics	Whether to show statistics (1) or not (0)
MQMD fields that can be changed	
format	Value MQSIputdata puts in MQMD.Format, such as MQHRF (the default) or MQSTR
expiry	Value for MQMD.Expiry; the default is -1, MQEI_UNLIMITED
persistence	Value for MQMD.Persistence; the default is 0, MQPER_NOT_PERSISTENT
messageType	Value for MQMD.MsgType; the default is 8, MQMT_DATAGRAM (That's the only type supported)
MQSI Header	
includeHeader	Whether to include the RFH (1) or not (0)
dataFormat `Only when includeHeader=1`	The name to put in MQHRF.Format, for example, MQSTR (default); rules processor will put this in the MQMD.
putOptions `Only when includeHeader="1"`	Names of message group and message type: OPT_APP_GROUP = *name* OPT_MSG_TYPE = *name*

Figure 42 on page 65 shows the file mqsiputdata.mpf as it is supplied with the product. You start the program from the command line as shown (in bold) in Figure 43 on page 66. The statistics show that one message has been put.

Note: The queue manager must be running.

```
[Put Control]
     #Name of the file which contains the message data
     inputFileName                  =      putdata.input
     #Name of the queue where the message will be put
     queueName                      =      RulesIn
     #Name of the queue manager that owns the queue
     queueManagerName               =      QAQM

     #Maximum message size
     maxUserDataLength              =      10000
     #Number of messages to put
     messageCount                   =      1
     #Binary value indicating whether of not statistics information shoud
     #be output. 1 indicates yes, 0 indicates no.
     showStatistics                 =      1

[Put Message]
     # Populate the format field of the message descriptor with this value.
     format                         =     MQHRF  ◄──────────────── spelling!
     # Populate the expiry field of the message descriptor with this value.
     expiry                              =         -1
     # Populate the persistence field of the message descriptor with this value.
     #    Valid values for persistence
     #          MQPER_PERSISTENT               1
     #          MQPER_NOT_PERSISTENT           0
     #          MQPER_PERSISTENCE_AS_Q_DEF     2
     persistence                    =      0
# Populate the message type field of the message descriptor with this value.
#    Valid values for message type:
     #           MQMT_REQUEST                  1
     #           MQMT_REPLY                    2
     #           MQMT_REPORT                   4
     #           MQMT_DATAGRAM                 8
     messageType                    =      8

     # Specify whether or not to include the RF header
     # with the inbound message 1 = yes, 0 = no
     includeHeader                  =      1
     # Specify how to populate the MQRFH.Format field.
     # This parameter only takes effect if includeHeader == 1.
     dataFormat                     =     MQSTR

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

Figure 42. Tools: mqsiputdata.mpf

```
D:\MQI\bin>strmqm
MQSeries queue manager started.

D:\MQI\bin>mqsiputdata -p mqsiputdata.mpf
-Statistics-
        Messages Put    : 1

D:\MQI\bin>
```

Figure 43. Tools: Starting MQSIputdata

The following two examples demonstrate what messages MQSIputdata creates. If you want to do this on your own, make sure that the rules daemon is not running. It would process the messages you put on the queue immediately.

Example 1

Let's start with a simple example. In mqsiputdata.mpf, change the following three parameters:

```
messageCount   = 3
format         = MQSTR
includeHeader  = 0
```

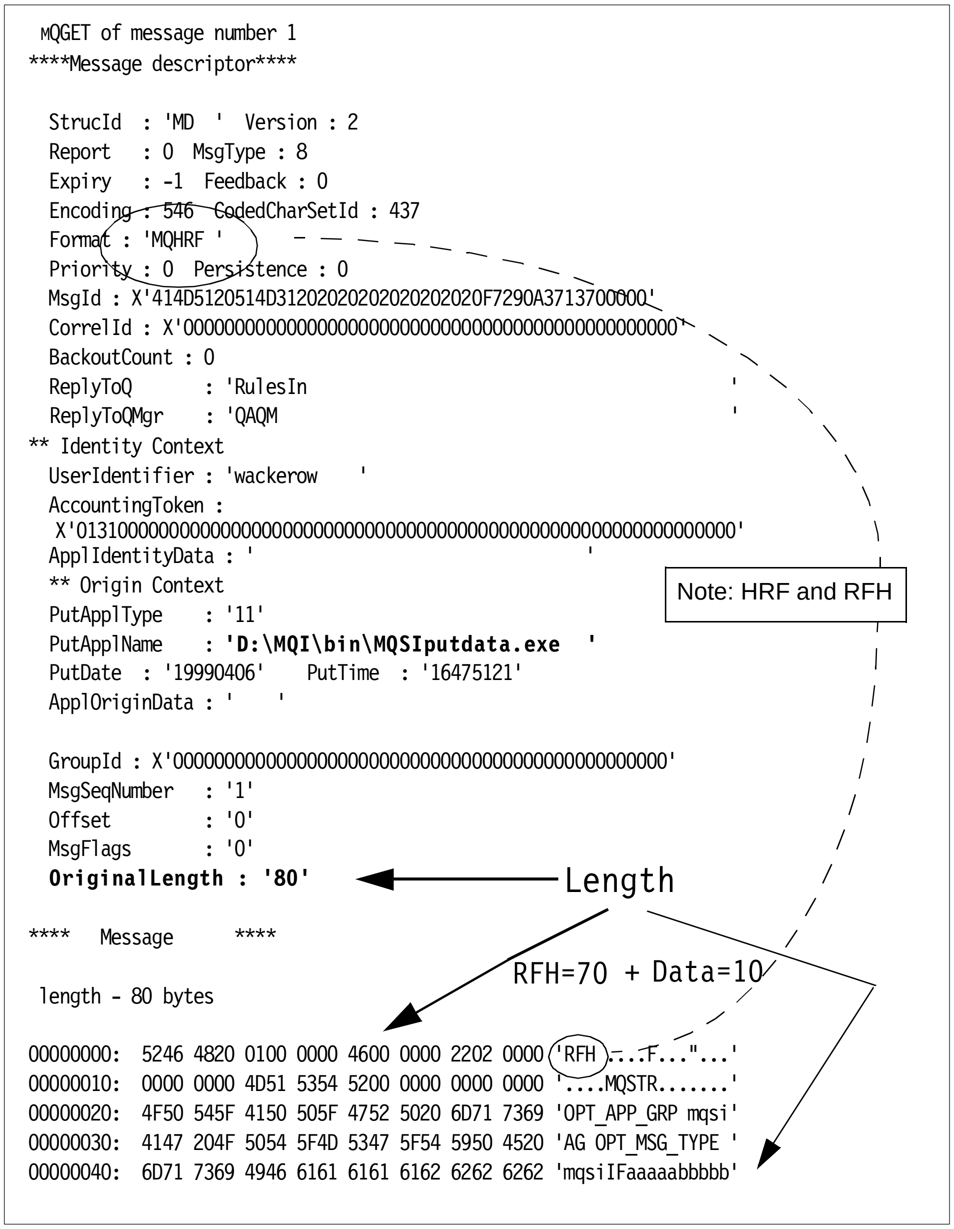
If you display the messages in the queue with amqsbcg, you will see that they look as if you had used amqsput to put them there.

Example 2

Now let us create a message that contains the RFH header and see how it looks. The parameters we have to specify in the .mpf file are as follows:

```
messageCount    = 1
format          = MQHRF     <------- note the spelling!
includeHeader   = 1
dataFormat      = MQSTR
OPT_APP_GRP     = mqsiAG
OPT_MSG_TYPE    = mqsiIF
```

The RFH becomes part of the message data. Since it contains binary data you cannot display the messages with amqsget or amqsgbr. Use amqsbcg instead; it displays the MQMD header, too. An example is in Figure 44 on page 67. You could use MQSIgetdata, but we will discuss this tool later.



```
 MQGET of message number 1
****Message descriptor****

  StrucId  : 'MD  '  Version : 2
  Report   : 0  MsgType : 8
  Expiry   : -1  Feedback : 0
  Encoding : 546  CodedCharSetId : 437
  Format : 'MQHRF '
  Priority : 0  Persistence : 0
  MsgId : X'414D5120514D31202020202020202020F7290A3713700000'
  CorrelId : X'000000000000000000000000000000000000000000000000'
  BackoutCount : 0
  ReplyToQ       : 'RulesIn                                          '
  ReplyToQMgr    : 'QAQM                                             '
** Identity Context
  UserIdentifier : 'wackerow    '
  AccountingToken :
   X'0131000000000000000000000000000000000000000000000000000000000000'
  ApplIdentityData : '                                '
  ** Origin Context
  PutApplType    : '11'
  PutApplName    : 'D:\MQI\bin\MQSIputdata.exe  '
  PutDate  : '19990406'    PutTime  : '16475121'
  ApplOriginData : '    '

  GroupId : X'000000000000000000000000000000000000000000000000'
  MsgSeqNumber   : '1'
  Offset         : '0'
  MsgFlags       : '0'
  OriginalLength : '80'

****   Message      ****

 length - 80 bytes

00000000:  5246 4820 0100 0000 4600 0000 2202 0000 'RFH ....F..."...'
00000010:  0000 0000 4D51 5354 5200 0000 0000 0000 '....MQSTR.......'
00000020:  4F50 545F 4150 505F 4752 5020 6D71 7369 'OPT_APP_GRP mqsi'
00000030:  4147 204F 5054 5F4D 5347 5F54 5950 4520 'AG OPT_MSG_TYPE '
00000040:  6D71 7369 4946 6161 6161 6162 6262 6262 'mqsiIFaaaaabbbbb'
```

Figure 44. Tools: Message With NEON Header (MQRFH)

In this example, data is 10 characters, following the RFH header. Table 5 on page 68 shows the structure of the header.

Table 5. MQSI Header (MQHRF)

Field Name	Field Length	Contents
StructId	MQCHAR4	Structure ID = RFH
Version	MQLONG	Version number = 1
StructLength	MQLONG	Length of the RFH header plus the length of the options string
Encoding	MQLONG	Default = MQENC_NATIVE
CodedCharSet	MQLONG	Default = zero
Format	MQCHAR8	Format of the data following the RFH The default = MQSTR
Flags	MQLONG	For future use
NEON Option Buffer		
Application Group	variable	OPT_APP_GRP followed by the application group name and a blank
Message Type	variable	OPT_MSG_TYPE followed by the message type

If the Rules daemon is not running, start it now with the following command:

```
mqsiruleng -p mqsiruleng.mpf
```

The rules processor will now process the messages in the queue RulesIn according to the formats and rules in the MQSeries Integrator database. The application ID and the message type must be in the database.

The MQSI header will be stripped off. The resulting messages are in the Output queue.

If you display the message in this queue with amqsbcg, you will see that the new MQMD.Format is now set to MQSTR. The rules processor got this information from the MQSI header. Figure 45 on page 69 shows some of the information displayed by amqsbcg for the message processed by the rules processor.

```
 MQOPEN - 'Output'


 MQGET of message number 1
****Message descriptor****

  StrucId  : 'MD  '  Version : 2
  Report   : 0  MsgType : 8
  Expiry   : -1  Feedback : 0
  Encoding : 546  CodedCharSetId : 437
  Format : 'MQSTR   '
  Priority : 0  Persistence : 0
- - - - - - - - - - - - - - - - - - - - - - - - - - - - -
OriginalLength : '10'

****   Message      ****

 length - 10 bytes

00000000:  414E 544F 4E48 494C 4C20                'ANTONHILL       '
```

Figure 45. Tools: Message Reformatted by the Rules Daemon

4.5 Displaying Messages with MQSIgetdata

MQSIgetdata displays messages that are in a queue. The messages may or may not carry the MQSI header. The program treats the header as data.

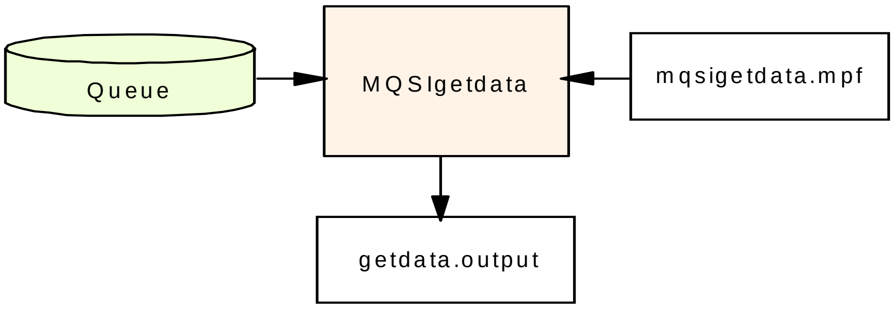


Figure 46. Tools: How MQSIgetdata Works

The functions of this tool are controlled through an mpf file, such as mqsigetdata.mpf supplied with the product. This file is shown in Figure 47 on page 70.

```
[Get Control]
     #Name of the file to put the message in.
     outputFileName                 =      getdata.output
     #Name of the queue to get the message from.
     queueName                      =      RulesIn
     #Name of the queue manager that owns the queue.
     queueManagerName               =      QAQM

     #Maximum message size that the application can get.
     maxUserDataLength              =      10000

     #ID of the message to get.  If this value is not defined
     #and correlID is not defined, the application gets the next
     #available message from the queue.  Notice that this field uses
     #an encoded hex representation for the messageId.
     #messageId                      =      414D51205141514D2020202020202020234EA17130000030D

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

     #Maximum number of messages to get.  The application will run until
     #messageCount messages have ben dequeued or until the queue is empty.
     messageCount                   =      1000

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

     #The following entries are binary attribute indicators
     # 1 indicates that the feature should be enabled.  0 indicates that
     # the feature should be disabled.

     #Show statistics about dequeued messages.
     showStatistics                 =      1

     #Should the output be sent to a file.  0 indicates that output
     #should be sent to stderr.
     outputToFile                   =      1
     #Should the message descriptor data be output.
      showDescriptor                 =      1
     #Should the message data be output.
     showData                       =      1

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

Figure 47. Tools: mqsigetdata.mpf

The output is similar to the output of amqsbcg; however, MQSIgetdata does not recognize the Version 2 header introduced with MQSeries Version 5. If you don’t have the need to get a message with a specific correlation ID or message ID you may just as well use amqsget. But note that amqsbcg limits the message size to 100 bytes and that is not very large. You start mqsigetdata as shown below in bold:

```
D:\MQI\bin>mqsigetdata -p mqsigetdata.mpf
Operation Complete.
-Statistics-
        Messages Got    : 1

D:\MQI\bin>
```

Figure 48. Tools: Starting MQSIgetdata

The output file is shown in Figure 49.

```
   StrucId             :'MD  '
   Version             :1
   Report              :0
   MsgType             :8
   Expiry              :-1
   Feedback            :0
   Encoding            :546
   CodedCharSetId      :437
   Format              :'MQSTR   '
   Priority            :0
   Persistence         :0
   MsgId               :'414D51205141514D2020202020202020898BC13613A00000'
   CorrelId            :'000000000000000000000000000000000000000000000000'
   BackoutCount        :0
   ReplyToQ            :'                                                '
   ReplyToQMgr         :'QAQM                                            '
   UserIdentifier      :'jorgen      '
   AccountingToken     :'16010515000000090722146F95F7B22C51DA912EF03000000000000000000000000B'
   ApplIdentityData    :'                                '
   PutApplType         :11
   PutApplName         :'D:\MQI\bin\MQSIputdata.exe  '
   PutDate             :'19990210'
   PutTime             :'20332797'
   ApplOriginData      :'    '
**********************
------------------------
 WORLD HELLO
------------------------
-Statistics-
Messages Got    : 1
```

Figure 49. Tools: Output of MQSIgetdata.

4.6 Importing and Exporting Formats

NNFie is provided for the purpose of importing and exporting formats to and from the MQSeries Integrator database. This is useful if you need to transfer or re-create format definitions in another MQSI database or on another machine.

You invoke this tool from the command prompt. If you type nnfie without parameters, NNFie displays all possible parameters as shown in Figure 50.

```
D:\MQI\bin>NNfie
 Formatter Import / Export utility (NNfie)
 NeoNet Version 4.01
 Version 1.6.2.9.2.2, last changed on 1998/04/01 21:53:45
 Copyright (C) 1996-198, New Era Of Networks, Inc.
 All Rights Reserved.

 usage:
 NNfie  ((-C <command file name>
         (-i <import file name> [-T]|-e <export file name>
          [-m <format name>+]
          [-s <session name>]))

 -C = Alternate command file: default file is NNFie.cmd
    if this option is given, NNFie will read command-line
    options from a file instead of the command line.
 -i = import: <import file name>: default file NNFie.exp
 -T = load import file as one transaction
 -e = export: <export file name>: default file NNFie.exp
 -s = session name: default is nnfie
 -m = message type/format: default results in export of all formats
```

Figure 50. Tools: Parameters for NNFie (Import and Export Formats)

If you worked through 2.1.8, “Verifying the Installation” on page 16, you have already used this command to import the defaultMsg and defaultOut formats used in the IVP. Remember, the command was:

```
nnfie -i d:\IVP\formats.fie -s new_format_demo
```

You specified the name of the file that contains the (previously exported) format and a session name specified in the database connection file, *sqlsvses.cfg.* Figure 51 on page 73 shows the entries relevant to DB2 in this file. The session names are highlighted.

```
# ---------------------------------------------------------------------
#
# Sqlsvses.cfg - Defines sessions to be used to access the Rules & Formatter
#                database.  The format of each line depends on the database
#                in which the NEON Rules and NEON Formats are to be stored.
# ---------------------------------------------------------------------------
#
# Format of lines for the various supported database types:
#
#      DB2:
# SessionName:dBaseName_or_Alias:dBaseUserid:dBasePassword:
#   (note that the fifth field is blank, so the last colon is required)
#
# ---------------------------------------------------------------------------
#
# SessionNames required to run specific utilities:
#
# The SessionName "new_format_demo" is required to run the test programs:
#     'apitest' and 'msgtest'.
# The SessionName "rules" is required to run the test programs:
#     'ruletest'
# Any SessionName may be used for the Import/Export utilities, NNFie and
# NNRie, and is specified by the '-s SessionName' option.  For example,
# a SessionName of "import" and/or "export" could be used.
#
# ---------------------------------------------------------------------------
#
# Example SessionNames (uncomment the SessionNames needed):
#
#       DB2 database (final colon required on each line):
 new_format_demo:MQSITDBA:wackerow:a1rich:
 rules:MQSITDBA:userid:password:
 import:MQSITDBA:userid:password:
#       (change Database, Userid, Password above)
#
# --------------- ( End of File ) -----------------------------------------
```

Figure 51. sqlsvses.cfg for Use with DB2

4.6.1 How to Export a Format

For example, let us export a format we have created called testMsg to a file with the name testMsg.fie. The command, executed from the \mqi\bin directory is as follows:

```
NNFie -e d:\temp\testMsg.fie -m testMsg -s new_format_demo
```

The parameters are as follows:

- new_format_demo is the session name for access to the MQSeries Integrator database as defined in the sqlsvses.cfg configuration file.
- NNFie will look in the database for a format with the name testMsg and export it.
- NNFie creates the file testMsg.fie in the directory d:\temp directory that will hold the exported format.

4.6.2 How to Import a Format

If you now delete the just exported format testMsg in the database (using the formatter GUI), you can re-create it with this command:

```
NNFie -i c:\temp\testMsg.fie -s new_format_demo
```

Alternatively, we could use this file on another machine running the MQSeries Integrator to import the identical Message Type/Format into its format database

In Case of an Error

During the export/import process you may see the following error message:

```
"ERROR NNFie.err file already exists"
```

Before you look for any errors, simply delete the error file in the \mqi\bin directory and try again.

4.7 Importing and Exporting Rules

With the NNRie tool you can import and export rules to and from the MQSeries Integrator database. It is useful if you need to re-create rules or transfer them between MQSI databases. You can replicate the entire rules database or part of it, such as the rules for a specific application group or just for a specific message type.

You invoke the program from the command prompt. If you type nnfie without any parameter, NNFie displays what you can specify as shown in Figure 52.

```
D:\MQI\bin>nnrie
Rules Import / Export utility (nnrie)
 NeoNet Version 4.01
Version 1.19.2.3.2.2, last changed on 1998/04/01 22:00:18
Copyright (C) 1996-1998, New Era Of Networks, Inc.
All Rights Reserved.

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

-C = Alternate command file: default file is NNRie.cmd
        if this option is given, NNRie will read command-line
        options from a file instead of the command line.
-V = return version information only (do no processing)
-i = import: <import file name>: default file NNRie.exp
-e = export: <export file name>: default file NNRie.exp
-o = overwrite file (export) or overwite data (import): default is off
-c = database configuration file: default is sqlsvses.cfg
-s = session name: default is nnrmie
-a = specific app name (export only): no default
-m = specific message name, requires -a (export only): no default
-r = specific rule name, requires -a and -m (export only): no default
No -a, -m or -r options means export entire database.

D:\MQI\bin>
```

Figure 52. Tools: Parameters for NNRie (Import and Export Rules)

4.7.1 How to Export Rules

Let us assume we have the rule testRule with the subscription testSub. We want to export this rule and the subscription with it. To export to the file testMsg.rie on the A-drive we issue this command:

```
NNRie -e a:\testMsg.rie -a defaultApp -m testMsg -r testRule -s new_format_demo
```

The parameters are as follows:

- new_format_demo is the session name for access to the MQSI database as defined in the database connection file sqlsvses.cfg. This file, shown in Figure 51, allows us to use one of three session names.
- testRule is the name of the rule we want to export.
- testMsg is the message the rule is for.
- defaultApp is the name of the application group to which testMsg belongs.
- testMsg.rie is the file that holds the exported rule.

The output from this command is as follows:

```
D:\MQI\bin>NNRie -e a:\testMsg.rie -a defaultApp -m testMsg -r testRule -s new_format_demo
Rules Import / Export utility (NNRie)
 NeoNet Version 4.01
Version 1.19.2.3.2.2, last changed on 1998/04/01 22:00:18
Copyright (C) 1996-1998, New Era Of Networks, Inc.
All Rights Reserved.

Import / export is set to "exporting"
Import / export file is "a:\testMsg.rie"
Configuration file is "sqlsvses.cfg"
Database session tag is "new_format_demo"
Export list contains 1 entries
Export node number 1
Application group name is "defaultApp"
Message type name is "testMsg"
Rule name is "testRule"
AMRnnnneSCPPCPnnnn
All done.
```

Figure 53. Tools: Export Example

To export all the rules associated with testMsg (or if this is the only rule) you can omit the -r parameter. Similarly, to export all the rules in the database omit the -a, -m and -r parameters. To export all rules that belong to a specific application group use just the -a parameter.

4.7.2 How to Import Rules

You may have already imported a set of rules when you verified the installation. Refer to 2.1.8, “Verifying the Installation” on page 16. Remember, the command was:

```
nnrie -i d:\IVP\rules.rie -s new_format_demo
```

With the above command you imported all rules in the file rule.rie into the MQSI database.

To import the rule testMsg into another database, issue the command as shown, in bold, in Figure 54.

```
D:\MQI\bin>NNRie -i a:\testMsg.rie -s new_format_demo
Rules Import / Export utility (NNRie)
 NeoNet Version 4.01
Version 1.19.2.3.2.2, last changed on 1998/04/01 22:00:18
Copyright (C) 1996-1998, New Era Of Networks, Inc.
All Rights Reserved.

Import / export is set to "importing"
Import / export file is "a:\testMsg.rie"
Configuration file is "sqlsvses.cfg"
Database session tag is "new_format_demo"
Importing from file to database
AMRnnescppcpSCPPCPnn
All done.
```

Figure 54. Tools: Import Example

In the above example, there is only one rule in the file testMsg.rie. If the input file contains more than the rules you want to import, use the -a, -m and -r parameters.

4.8 The Visual Tester

We found that the Visual Tester is the most useful tool available for testing both MQSI formats and rules. This tool is available as an MQSeries SupportPac from the World Wide Web:

How to Get It

MDI1: MQSeries Integrator - Visual Tester

This SupportPac has been updated to include fixes to the Visual Tester code and to include a User's Guide in Adobe Acrobat format.

IBM MQSeries SupportPacs may be accessed at the following URL:

`http://www.ibm.com/software/mqseries/txppacs`

The Visual Tester allows us to test both, formats and rules, in a GUI tool that is both visual and intuitive using queues, files or the screen as inputs and/or outputs.

Note: Some information on how to use this product is described in 3.3, “Testing Formats and Rules with the Visual Tester” on page 45.

4.8.1 Installing the Visual Tester

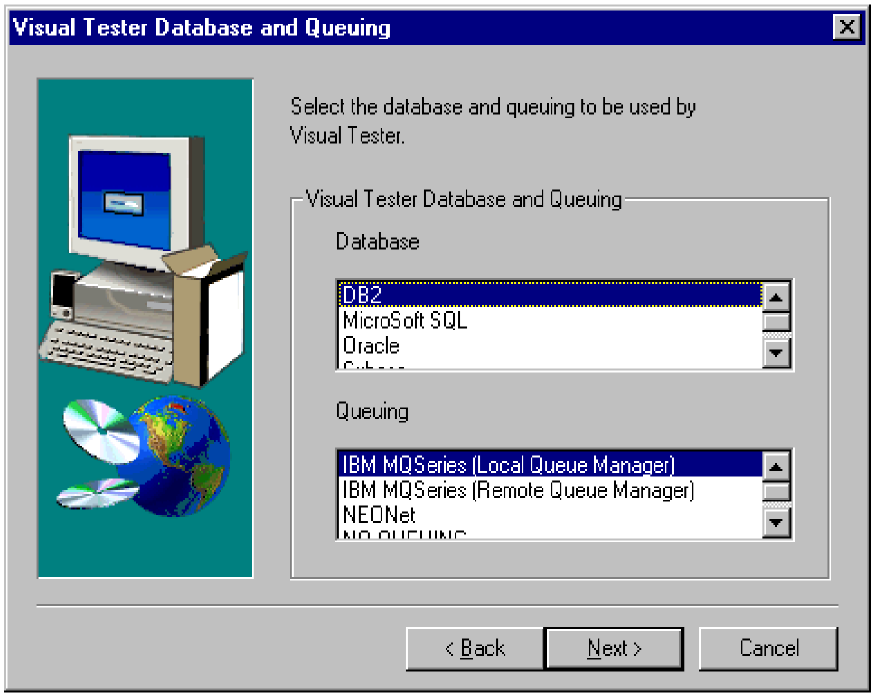
The code comes as a single self-extracting executable, visualtester.exe. To install it follow these steps:

1. Find the executable using the Windows NT Explorer and double-click on it. This starts the extraction process.
2. You will be asked, in a message box, if you wish to install the Visual Tester now. Click **Yes** to continue.
3. You are then asked where the installation files should be unpacked. Provided there is enough disk space on the C drive, accept the default of C:\TEMP\Visual Tester and click **Finish.**
4. Another prompt tells you: The specified folder does not exist. Create It ? Click **Yes** to continue.
5. When the Welcome window appears, click **Next** to carry out the installation.
6. You will then be presented with a dialogue asking you for the destination location. You can accept the default C:\NEON\Visual Tester or use the Browse button to change the location, for example, to D:\MQSI\VT. Then click **Next** to continue.
7. Next the window shown in Figure 55 on page 79 appears. Select the following:
   - The type of the *MQSI database* you are using, here DB2.
   - The *queuing method* being used, here MQSeries with a local queue manager installed (server).

   Then click **Next**.
8. If you are installing the Visual Tester with an MQSeries Integrator installation that uses MQSeries Version 5.1 you may see the message in Figure 56 on page 79.

   You have to manually register the OCX files. You can do this afterward. How to do this is described on page 80.

   Click **Yes** to continue.


Visual Tester Database and Queuing

Select the database and queuing to be used by Visual Tester.

Visual Tester Database and Queuing

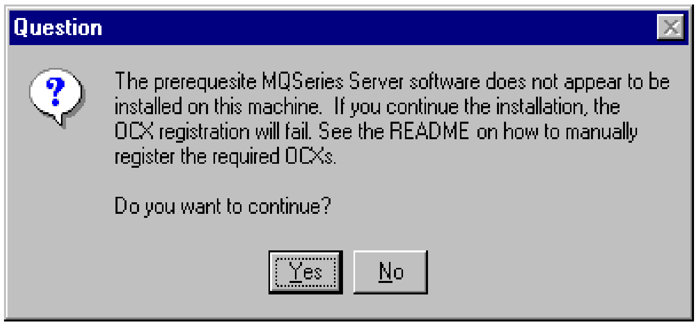
Database

DB2
MicroSoft SQL
Oracle

Queuing

IBM MQSeries (Local Queue Manager)
IBM MQSeries (Remote Queue Manager)
NEONet

Figure 55. Visual Tester Install: Select Database and Queuing


Question

The prerequesite MQSeries Server software does not appear to be installed on this machine. If you continue the installation, the OCX registration will fail. See the README on how to manually register the required OCXs.

Do you want to continue?

Figure 56. Visual Tester Install: OCX Registration Failure

9. Select a name for the program folder, for example, the default Visual Tester, and click **Next**.

10. The install program then copies the files and finishes with a dialog box, telling you that the installation has completed successfully. Click **Finish**.

Manually registering the OCX files:

If you are installing the Visual Tester with MQSeries 5.1 you need to manually register the OCX files used by Visual Tester, NNObjs.ocx and NNMgrs.ocx. To do this bring up a command prompt and type the following:

```
regsvr32 /s /c NNObjs.ocx
```

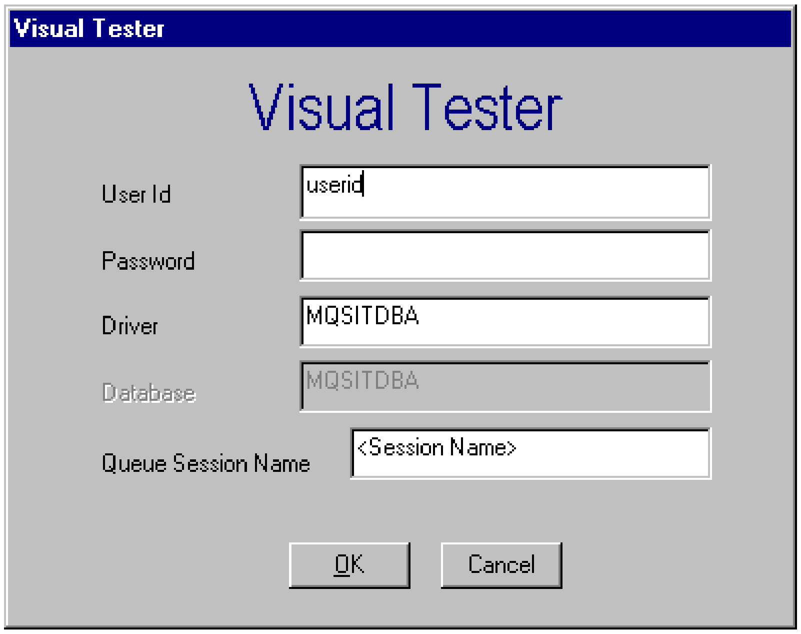
```
regsvr32 /s /c NNMgrs.ocx
```

Your Visual Tester installation is now complete.

4.8.2 Logging On to the Visual Tester

Selecting the Visual Tester from the programs folder brings up the logon window in Figure 57.

Note: Make sure that the queue manager is running.


Visual Tester

Visual Tester

User Id: userid

Password:

Driver: MQSITDBA

Database: MQSITDBA

Queue Session Name: <Session Name>

Figure 57. Visual Tester Logon Window

You have to enter your user ID and password as well as a Driver name. For DB2, this is the ODBC System Data Source name you created during the installation of the MQSeries Integrator, MQSITDBA.

Leave the Queue Session Name unchanged, provided the queue manager you use is the default queue manager. You have to click **OK** to start the Visual Tester.

As the four tabs in Figure 58 on page 82 indicate, the program lets you execute four functions:

- Load a message into a queue
- Test if an input format parses correctly
- Test if the reformatting (output format) works
- Test if the rules definitions are correct

4.8.3 Loading a Message into a Queue

The first tab in the Visual Tester is used to invoke a function that loads a message into a queue. The message data can come from another queue, a file or you may type it in the window.

If specified, the program puts the application group and message type in front of the data.

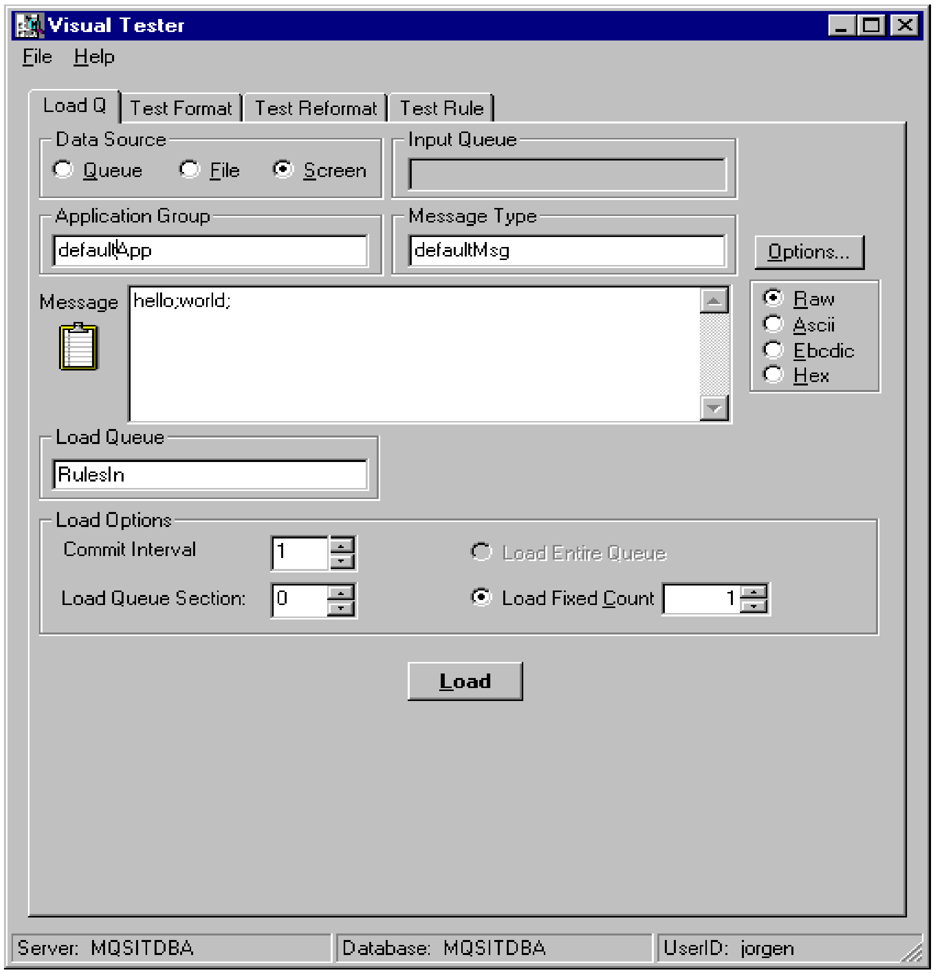
Figure 58 on page 82 demonstrates how to create a message for the IVP and put it into a queue. We specify the following values:

- The application group is defaultApp.
- The message type is defaultMsg.
- RulesIn is the name of the queue that will hold the message.
- The radio button Screen is selected to indicate the message data will be typed in the dialog box, here "hello;world;".

If you click **Load** the following message will be displayed:

```
Successfully put 1 message(s) to queue RulesIn.
```

If you do not enter an application group and/or a message type, a pop-up window will appear telling you that you didn’t. However, you may continue and create a message that contains only the message data.



Data Source: Queue, File, [✓ Screen]
Input Queue:
Application Group: defaultApp
Message Type: defaultMsg
Message: hello;world;
[✓ Raw] Ascii Ebcdic Hex
Load Queue: RulesIn
Load Options
Commit Interval: 1
Load Queue Section: 0
Load Entire Queue
[✓ Load Fixed Count] 1
Server: MQSITDBA
Database: MQSITDBA
UserID: jorgen

Figure 58. Visual Tester: Load Queue

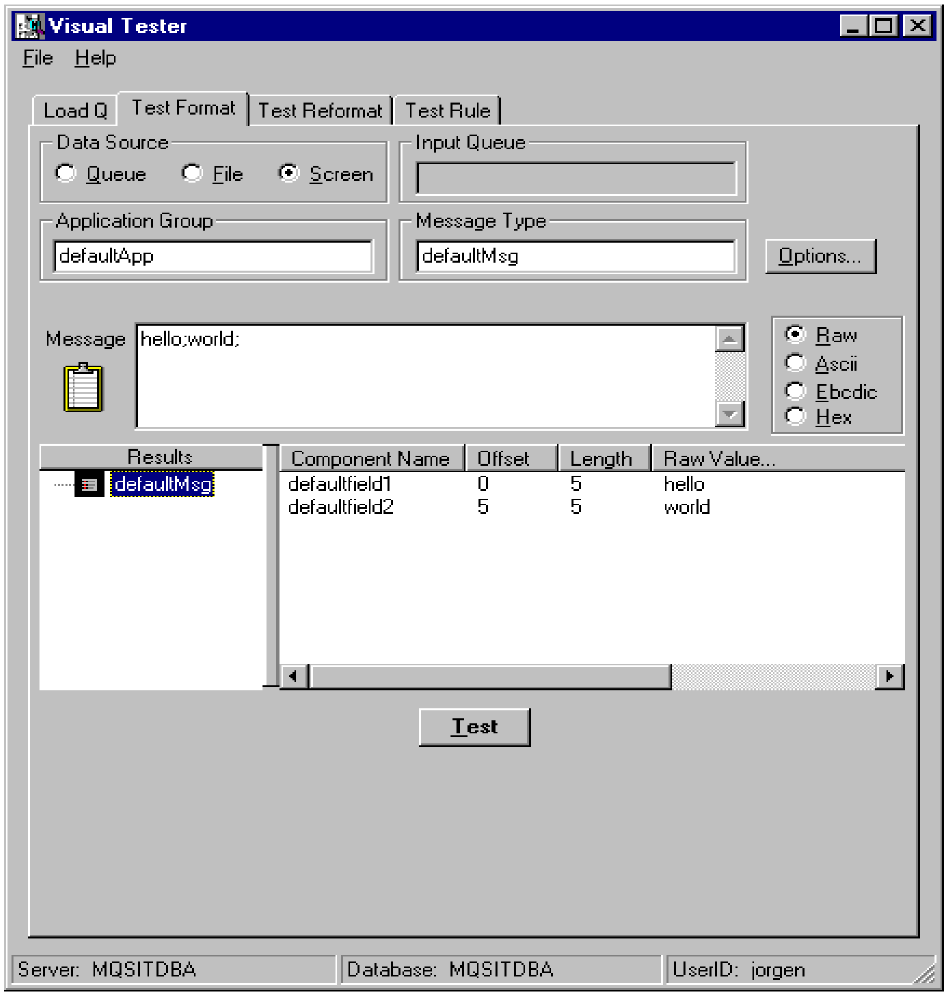
4.8.4 Testing an Input Format

The second tab in the window, Test Format, lets you check out the input format definitions you entered into the MQSI database with the Formatter GUI. Let us use the message we just created with the Load Q function and parse it:

- Click on **Test Format** tab.
- Select the **Queue** radio button for the data source.
- Enter the queue name, RulesIn

- Enter the application group defaultApp and the message type defaultMsg.
- Click **Test**.

Alternatively, you can select Screen as the data source rather than a queue and type in the message in the dialog box. Clicking **Test** shows the test results in Figure 59.



Data Source: Queue ( ) File ( ) [✓ Screen]

Input Queue:

Application Group: defaultApp

Message Type: defaultMsg

Message: hello;world;

[✓ Raw] Ascii Ebcdic Hex

Results: defaultMsg

Component Name	Offset	Length	Raw Value...
defaultfield1	0	5	hello
defaultfield2	5	5	world

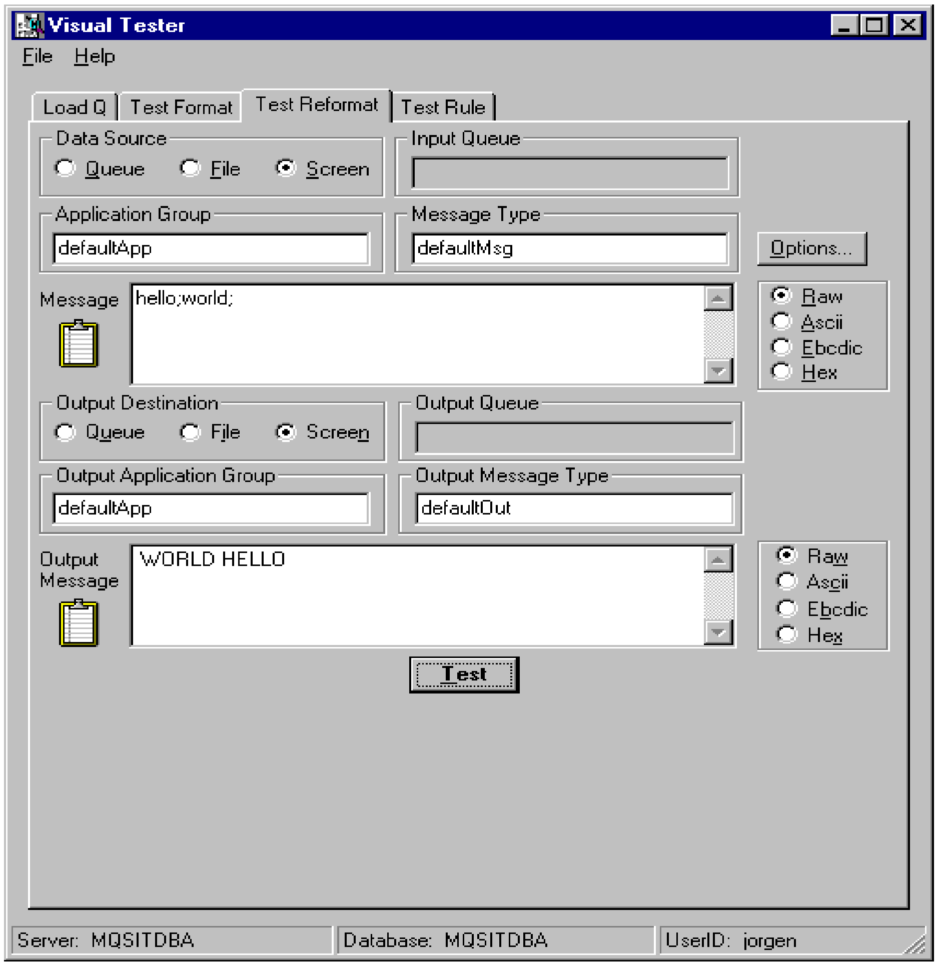
Server: MQSITDBA | Database: MQSITDBA | UserID: jorgen

Figure 59. Visual Tester: Test Format

Note: If you make any changes to the format you are testing while the Visual Tester is running, you will need to open the *Options* window and mark the check box for **Recache Rules/Formatter** under General Options for the changes to take effect. On occasion it has been found that you may need to do this several times when making several changes at a time.

4.8.5 Testing Reformat

The function behind the Test Reformat tab lets you check out your output format definitions.



Data Source: ( ) Queue ( ) File [✓ Screen]

Input Queue:

Application Group: defaultApp

Message Type: defaultMsg

Message: hello;world;

[✓ Raw] ( ) Ascii ( ) Ebcdic ( ) Hex

Output Destination: ( ) Queue ( ) File [✓ Screen]

Output Queue:

Output Application Group: defaultApp

Output Message Type: defaultOut

Output Message: WORLD HELLO

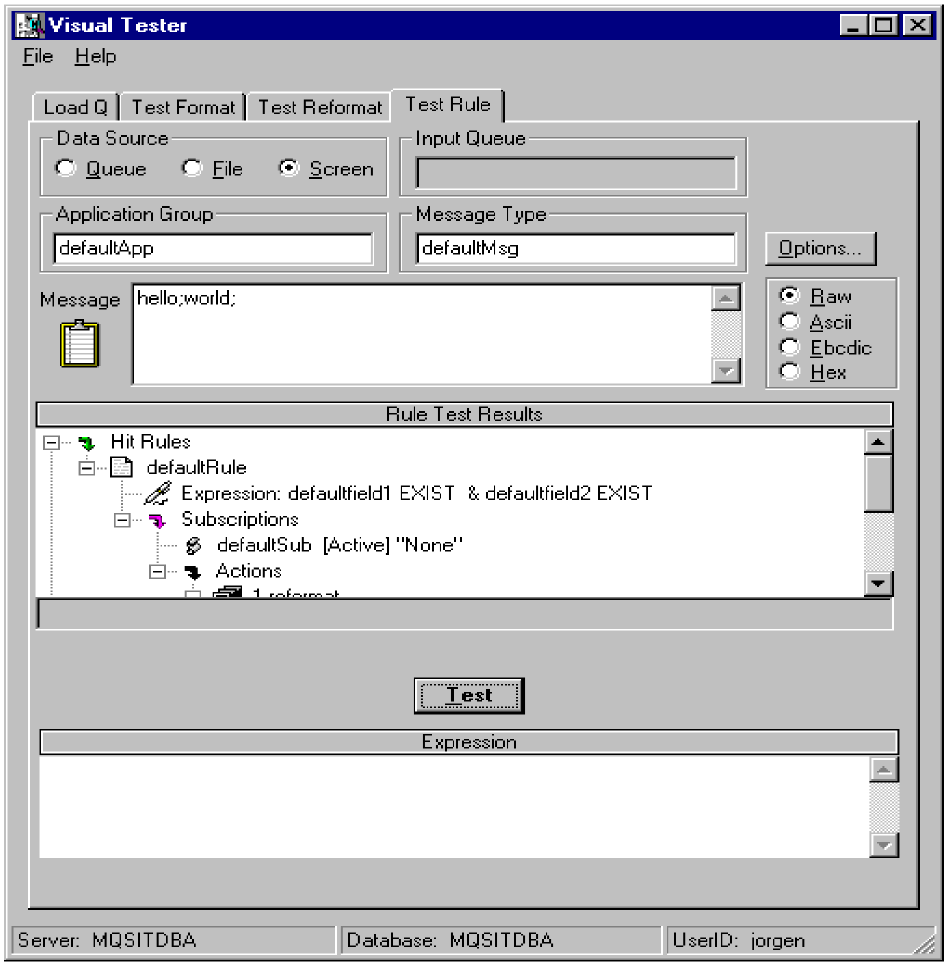
[✓ Raw] ( ) Ascii ( ) Ebcdic ( ) Hex

Server: MQSITDBA | Database: MQSITDBA | UserID: jorgen

Figure 60. Visual Tester: Test Reformat

If you want to test the IVP output format defaultOut, drag the test message in the Test Format window onto the Test Reformat tab and drop it. To do this, place the cursor on the Message clipboard icon, hold down the left mouse button and drag the contents onto the Test Message tab. When you then click the Test Reformat tab you will find that the message text box is already filled in, as is the application group and the message type. You can use this drag and drop technique between any of the tabs to reduce typing.

Continuing with your reformatting example, enter defaultApp as application group and defaultOut as output message type. The output application group defaults to the same as that for the input message. Click on Test and you see the result shown in Figure 60 on page 84.



Data Source: ( ) Queue ( ) File [✓ Screen]

Input Queue:

Application Group: defaultApp

Message Type: defaultMsg

Message: hello;world;

[✓ Raw] ( ) Ascii ( ) Ebcdic ( ) Hex

Rule Test Results

```
Hit Rules
  defaultRule
    Expression: defaultfield1 EXIST  & defaultfield2 EXIST
    Subscriptions
      defaultSub [Active] "None"
      Actions
```

Expression

Server: MQSITDBA   Database: MQSITDBA   UserID: jorgen

Figure 61. Visual Tester: Test Rules

4.8.6 Testing Rules

The fourth tab lets you test the rules you have entered into the MQSI database. To use the IVP message as an example, enter the application group and message type, and then type the message in the dialog box. Of course, the Screen radio button must be selected if you type the message.

In the Test Result window, you can see how the Rules Daemon evaluates the rules and expressions.

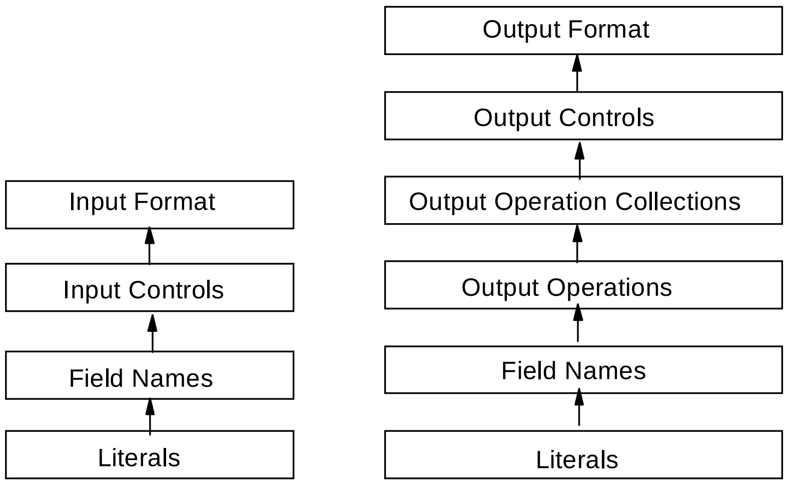
Refer also to 3.3.3, “How to Test If the Rules Work Properly” on page 49.

Chapter 5. Formatting Examples

The Formatter dynamically transforms and translates messages based on the requirements of the business and applications. The product has three basic functions:

- Parse a message
- Reformat the message
- Link the input to the output

In this chapter, we show some examples of those basic functions. We discuss the objects used to format and reformat messages. We also try to use a consistent technique when creating both input and output formats. Since this product was created with an object oriented design, it allowed us to use a technique that builds from the bottom up. Remember the order in which to create the objects for an input and output format:



How some of the objects are created is described in 3.1, “Working with the MQSI Format Administrator” on page 28. In the following sections, we discuss these objects and their properties in more detail.

The formatting examples will start with creating simple input and output formats. We expand this example to include new MQSI features and end with more involved examples, but not too complex so that the reader can easily understand what to do. We describe the different objects, then position them, and demonstrate their use with a reasonable example.

Now start the Formatter to bring up the GUI shown in Figure 62 on page 88. To show only the relevant information in the following screen captures we start with a clean database.

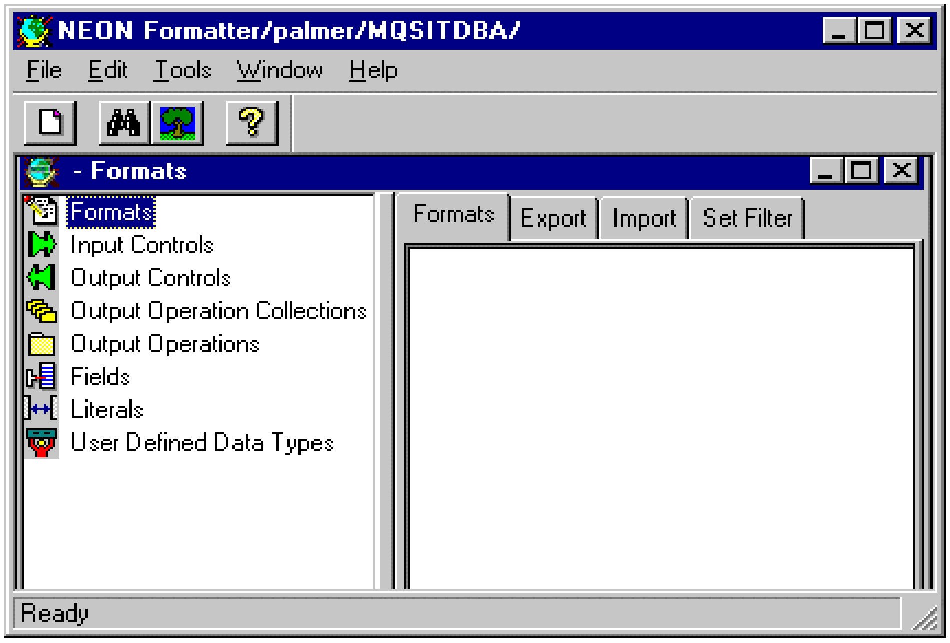


Figure 62. Formatter Window

In the following sections, we explain the following:

- How to process delimited fields and how to use literal values
- How to add prefixes and suffixes to fields for identification and better readability
- How to group output operations and form an output collection
- How to substitute values in the input message with values stored in the formatter
- How to process fields that contain field length and data
- How to process fields that are identified by tags (field IDs)
- How to manipulate date and time fields
- How to work with formats that contain other formats (compounds)

5.1 Using Delimited Fields

This section describes how the Formatter uses delimiters. We will also show an example of how delimiters can be used.

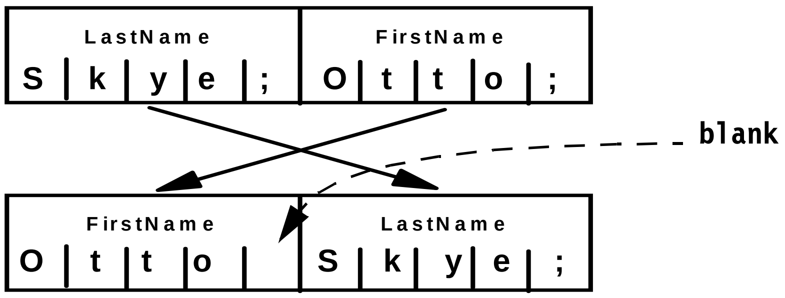
Basically, a delimiter is used to separate the data or fields in a message. There are several ways to locate fields in a message, by using spaces or symbols to separate them, or by using an exact length. The formatter can easily parse input when delimiters are used. Delimiters come in handy when the message contains variable length fields.

As an example on how to use delimiters let’s parse a message that contains two variable length fields, such as:

```
Skye;Otto;
```

The message contains a last name and a first name, separated by a semicolon and ending with a semicolon. In this case, both semicolons are delimiters. A delimiter can be any literal that is definable. A delimiter allows the formatter to look at the message and separate the fields because it knows where to stop looking. Whether the field is two or 32 characters long, the formatter can distinguish it because it ends with a semicolon.

In this example, we read an input message containing a last name and a first name. Then we swap the names and replace the semicolon between them with a blank, for example:



The objective of this exercise is to define an input format containing the fields LastName and FirstName, and an output format that swaps the fields and inserts separators.

5.1.1 Defining an Input Format with Delimited Fields

We are going to construct the input format Taxes_In with the following objects:

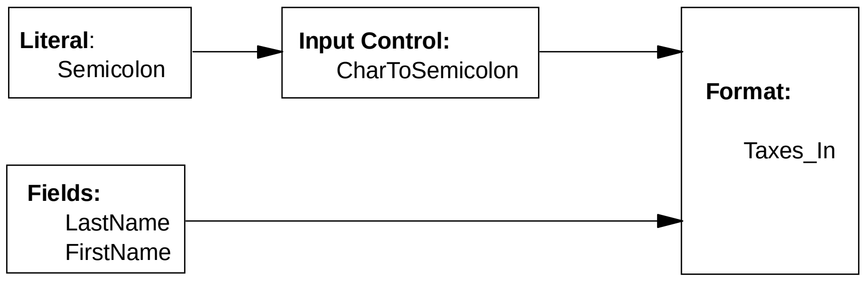


Figure 63. Delimited Fields Example: Input Objects

We create the literal first and then the two fields. After that we define the input control and assign it to the fields. At the end, we create the format definition and add the fields to it.

5.1.1.1 Defining a Semicolon Delimiter

The input message contains one delimiter, a semicolon. We define it in the following way:

1. Right-click **Literals** and then click **New**.
2. Type the literal name, Semicolon and press Enter. This will create the literal. Figure 64 on page 91 shows what you see next. Now you have to customize the properties.

   Note: If at any time a mistake is made, complete the process to define the literal. Then right-click the literal name and choose **Delete** from the subsequent menu.
3. After the literal has been created click on the **Properties** tab. You will see the name of the Literal in ASCII or EBCDIC and its hexadecimal representation.
4. Highlight the text **Semicolon,** delete it and replace it with the semicolon character ";". The hex value will change to "3b".
5. Click on **Apply**.

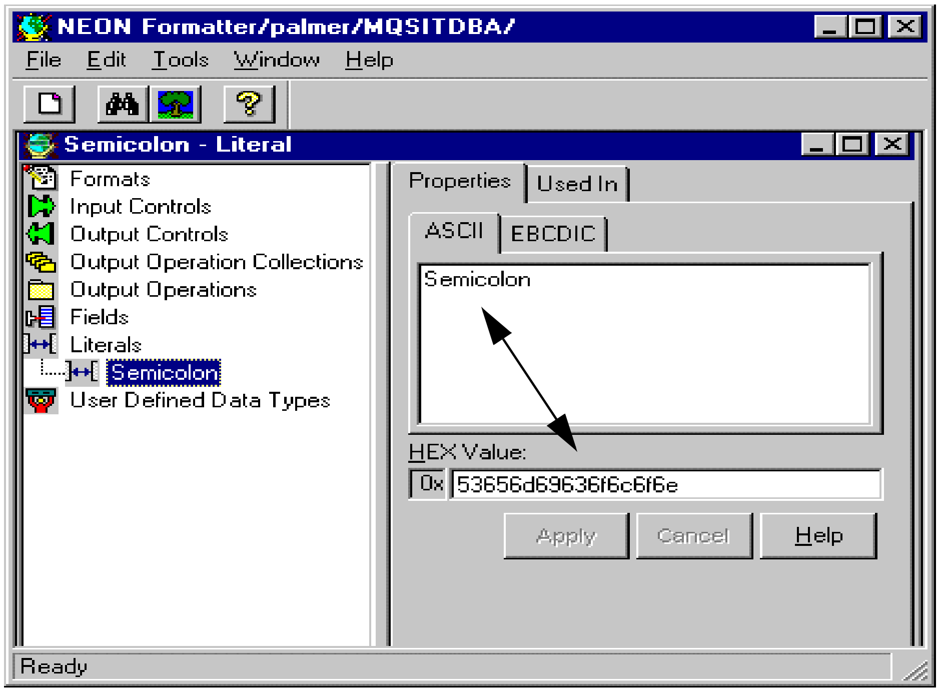
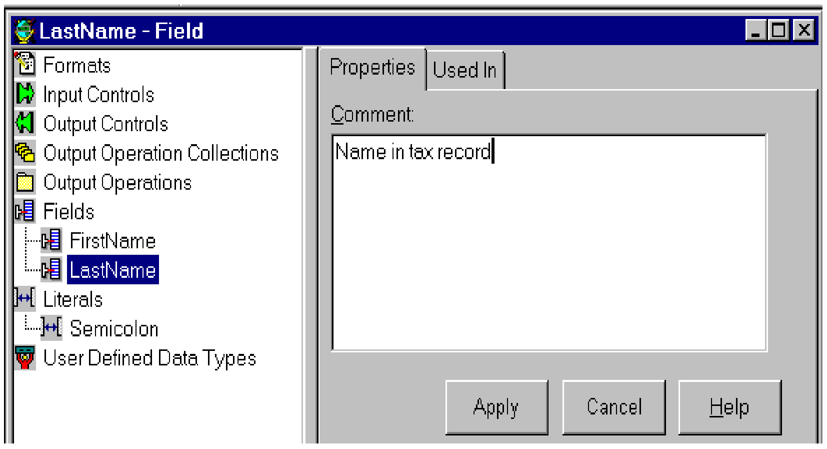


Figure 64. Defining the Literal Semicolon

5.1.1.2 Defining Fields

Figure 63 on page 90 shows that we have to create two fields, FirstName and LastName. We described how to create fields, in detail, in 3.1.1, “How to Define a Field” on page 30.

1. Right-click **Fields** and select **New** from the menu.
2. Type LastName in the field and press Enter.
3. You may add a comment in the Comment dialog box and then click **Apply**.
4. Perform the same actions for FirstName.



Comment:
Name in tax record

Figure 65. Example of Defining a Field

5.1.1.3 Defining Input Controls

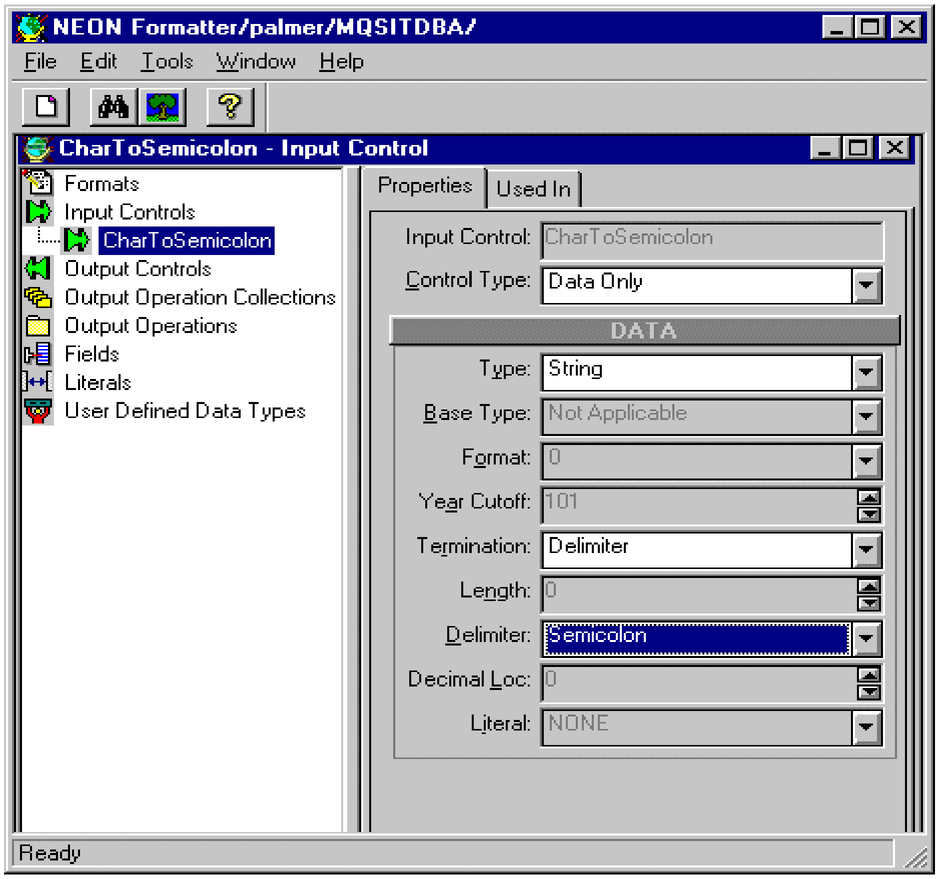
An input control tells the Formatter how to parse the input message. In this example, we want the Formatter to extract string data that ends with a semicolon.

1. Right-click **Input Controls** and select **New** from the menu.
2. Type the name CharToSemicolon and press Enter.
3. On the right side in the window you will see two panels. In the left panel there are three changeable fields:
   - Control Type
   - Data Type
   - Data Termination

   For this example, we will leave the defaults for Control Type (data only) and Data Type (string).
4. Expand the **Termination** drop-down list and select **Delimiter**.
5. Click **Apply**.
6. Now the Delimiter field becomes available to be changed. Expand this drop-down list and select **Semicolon**.

   Note: If Semicolon is not part of the selection then the literal semicolon has not been defined correctly.

Figure 66 on page 93 shows the significant parts of the Input Control window.



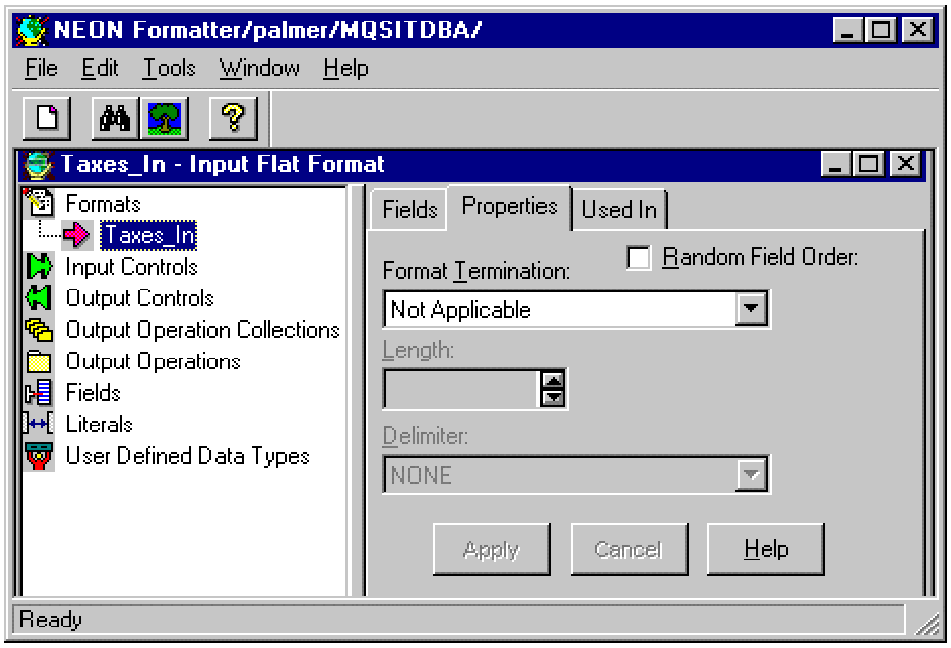
Input Control: CharToSemicolon
Control Type: Data Only
Type: String
Base Type: Not Applicable
Format: 0
Year Cutoff: 101
Termination: Delimiter
Length: 0
Delimiter: Semicolon
Decimal Loc: 0
Literal: NONE

Figure 66. Example of Defining an Input Control

5.1.1.4 Creating the Input Format

Now we have to create the input format, specify what fields belong in it (and in which order), and define how the fields have to be parsed. A detailed description on how to create an input format is in 3.1.3, “How to Define an Input Format” on page 32.

1. Right-click **Formats** and select from the menu **New**, then **Flat** and **Input**.
2. Type the format name, Taxes_In, and press Enter.
3. You will see now the Properties for Taxes_In as shown in Figure 67 on page 94. Make no changes here.

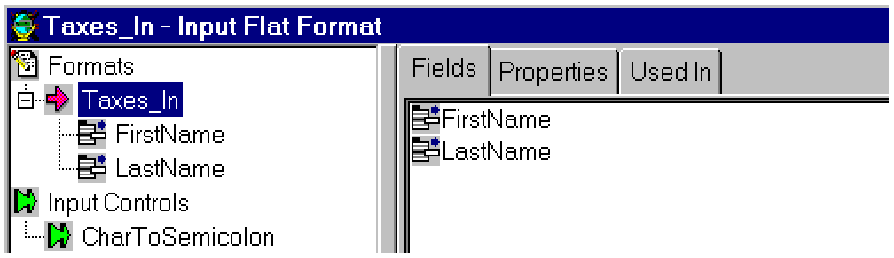


Format Termination: Not Applicable

Delimiter: NONE

Figure 67. Example of Creating a Format

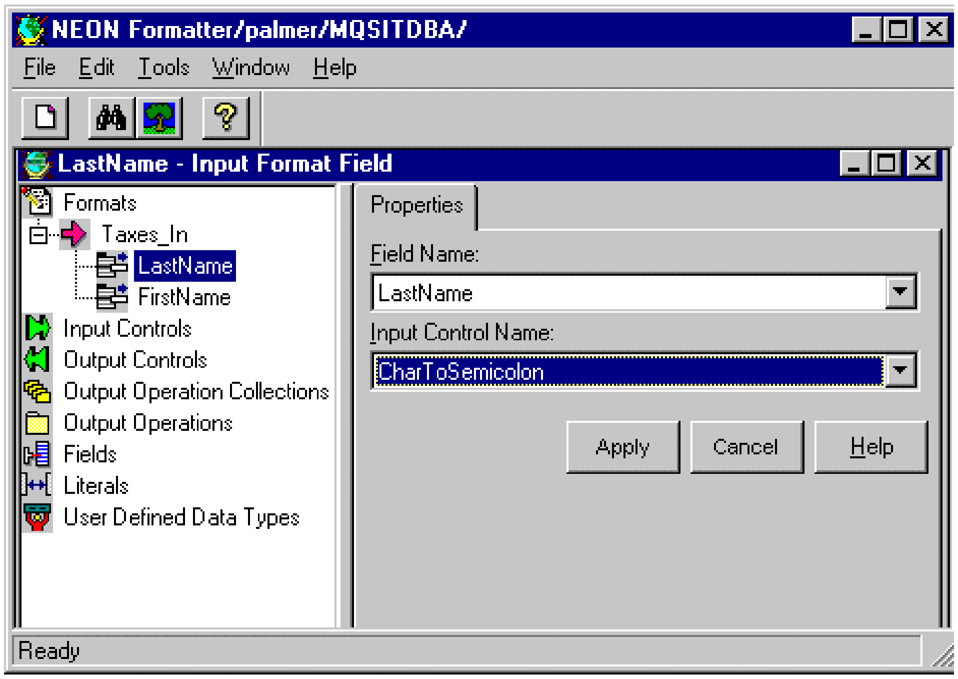
4. Right click Taxes_In and select the **Add Field Components** from the menu. You can now select the fields that belong in the message.
5. From the list of fields, select the appropriate fields, here LastName and FirstName. If you see only these two fields in the list, you can click **Select All**. Then click **Accept Selection**.



6. Depending on the way the fields were selected, they may not be in the correct order for our message. In our message, the last name is in front of the first name while the format shows these fields reversed.

   Left-click **Taxes_In** and make sure that the **Fields** tab is selected.

7. Click a field and drag it into its right place. You can only drag fields upwards! The order for our message is LastName followed by FirstName.
8. Next, click **LastName** in the panel on the left side of the window (under Taxes_In). Now the Properties tab as shown in Figure 68 on page 95 will appear on the right side of the window.
9. Expand the Input Control Name drop-down list and select the name **CharToSemicolon**.
10. Click **Apply**.
11. Repeat the above steps for FirstName.



Field Name: LastName

Input Control Name: CharToSemicolon

Figure 68. Example of Associating Fields with Input Control Names

At this point you should use the Visual Tester and check your definitions. Remember that the queue manager must be running when you start this program. Use any two names of any length, with a semicolon between them and one at the end. The names may contain blanks.

5.1.2 Defining an Output Format with Fields and Literals

Now let us create the objects for the output format. This is a very simple format. Since we don't do anything to the fields, we don't need output operations. The objects needed are shown below:

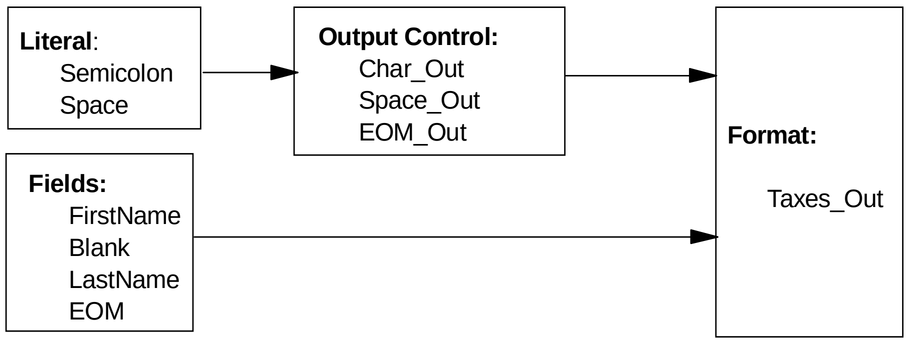


Figure 69. Delimited Fields Example: Output Objects

The two names fields and the semicolon have already been defined for the input format; we simply reuse them. To separate the first and last names in the output message we insert a space. This is a field which is not mapped from an input field. Its value comes from the literal Space. The end-of-message indicator (EOM) is another field with the literal value semicolon.

Here are the tasks to perform in order to create the output format:

5.1.2.1 Defining the Literal "Space"

We need a single space to insert between the first and last names. We create it in the same fashion as the semicolon in the previous section.

1. Right-click **Literals** and then click **New**.
2. Type the literal name, Space, and press Enter.
3. In the ASCII window, erase the word Space and replace it with a blank by pressing the Spacebar. The hex value will change to '20'.

5.1.2.2 Adding the Separator Fields

Let's create the fields Blank (containing one space) and the end-of-message indicator EOM (containing a semicolon).

1. Right-click **Fields** and select **New** from the menu.
2. Type Blank (or EOM) and press Enter.

5.1.2.3 Defining the Output Controls

Each field you output must be associated with an output control. In this example, we have two kinds of fields:

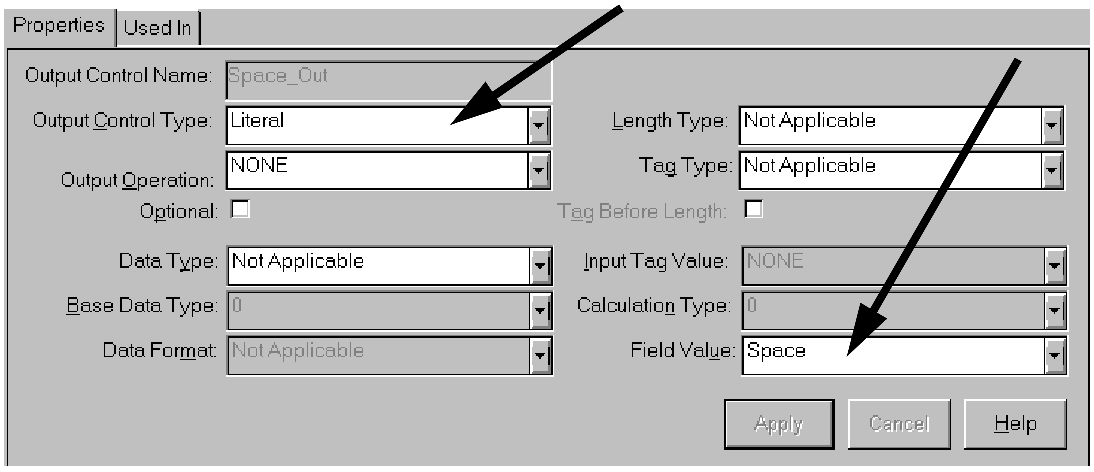
- Fields that are mapped from an input format, such as Char_Out
- Fields that get their values from one of the defined literals, such as Space_Out and EOM_Out

You define these objects as follows:

1. Right-click **Output Controls** and select **New** from the menu.
2. Enter the name Char_Out and press Enter. Leave the values in the Properties window as they are.

   The output control type is "Data Field (Name Search)" which means that the input field will be mapped to the output field by field name. There is no output operation associated with this control.

   This is a simple output control that puts the characters into the output field as they come from the input. Later we will examine ways to add spaces or other literals to the output controls to make the output more readable.
3. Again, right-click **Output Controls** and select **New** from the menu.
4. Type the name Space_Out and press Enter. Then change the values in the Properties panel as shown in Figure 70 on page 97 and click **Apply**.
5. Next, create EOM_Out with Semicolon as Field Value.



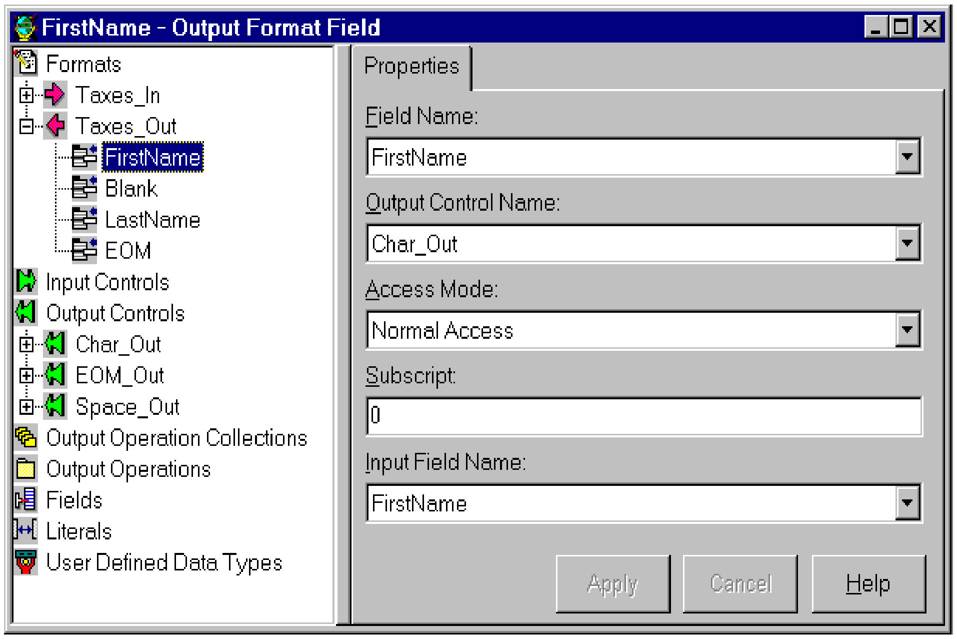
Output Control Name:	Space_Out		
Output Control Type:	Literal	Length Type:	Not Applicable
Output Operation:	NONE	Tag Type:	Not Applicable
Optional:	☐	Tag Before Length:	☐
Data Type:	Not Applicable	Input Tag Value:	NONE
Base Data Type:	0	Calculation Type:	0
Data Format:	Not Applicable	Field Value:	Space

Figure 70. Defining an Output Control for a Field with a Literal Value

5.1.2.4 Creating the Output Format

Now we have to define the output format and add the fields to it. Also we have to associate the fields with the appropriate output control. This process is described, in detail, in 3.1.5, “How to Define an Output Format” on page 37.

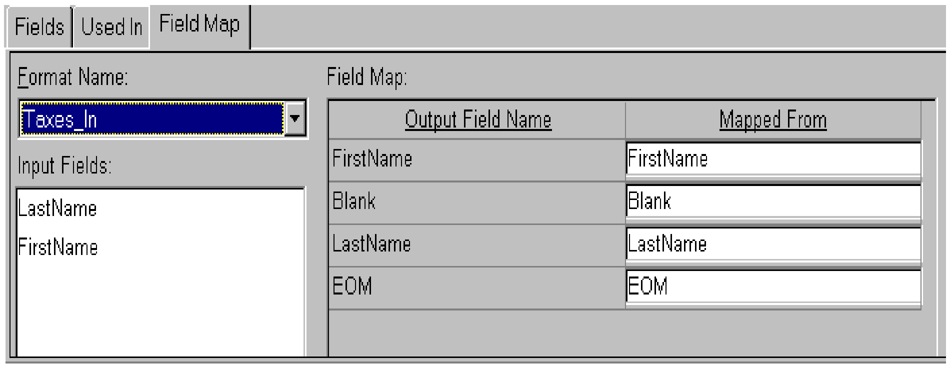
1. Right-click **Formats** and select **New**, **Flat** and then **Output** from the menu.
2. Type the format name Taxes_Out (or any other name you like).
3. Right-click **Taxes_Out** and select **Add Field Components**.
4. In the right side of the window, select the four fields and click **Accept Selection**.
5. You may have to reorder the fields by dragging them into their correct position as shown in Figure 69 on page 96. Remember: first name, space, last name, semicolon.
6. On the left side of the window, under Taxes_Out, select FirstName. Pull down **Output Control Name** and select Char_Out from the list. Also, make sure the input field name is correct. It has to be FirstName. Then **apply** the changes.
7. Assign the output control to LastName in the same way.



Field Name: FirstName
Output Control Name: Char_Out
Access Mode: Normal Access
Subscript: 0
Input Field Name: FirstName

Figure 71. Assigning an Output Control to a Field

8. Now we have to assign the output controls to the two literal fields. On the left side of the window, under Taxes_Out, select **Blank**. Pull down **Output Control Name** and select **Space_Out** from the list. The input field name is also Blank. Then click **Apply**.
9. Assign the output control EOM_Out to EOM in the same way.
10. Click **Taxes_Out** and then **Field Map** to see how the fields are mapped. You see that two fields come from the input format Taxes_In while two fields are created by the formatter using literal values.



Format Name: Taxes_In

Input Fields:
LastName
FirstName

Output Field Name	Mapped From
FirstName	FirstName
Blank	Blank
LastName	LastName
EOM	EOM

Figure 72. Mapping Input to Output Fields

This concludes the exercise about delimiters. Now use the Visual Tester to test the reformatting of some messages. For example:

"Skye;Otto;" becomes "Otto Skye;"

5.2 Using Fields with Prefixes and Suffixes

In this section we work with more literals and show examples of how to use them as a prefix and a suffix.

A literal (actually, literal value) is a single character or string. It is used:

- To mark the end of a piece of data
- As a prefix or suffix
- As a pad character
- As substitute value
- As tag or field identifier

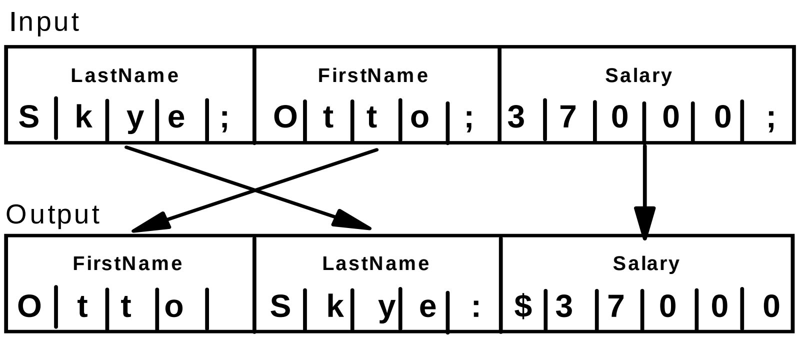
Two good examples of using literals are prefixes and suffixes. Prefixes are literals that are added to the beginning of an output field, while suffixes are added to the end of a field. For example:

- The prefix "$" can be associated with a dollar amount, as in $12.56.
- The suffix "%" can be associated with a percentage, as in 12%.

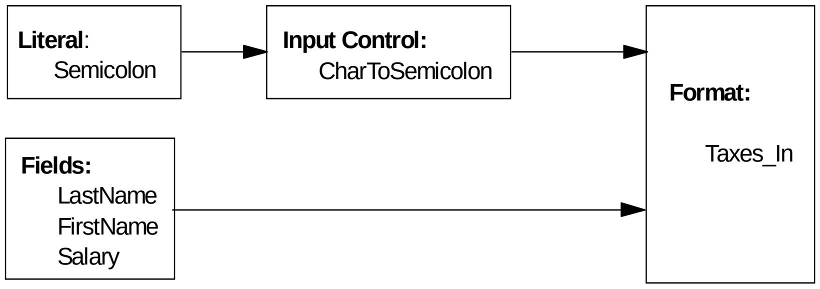
Note: In the previous example, we separated the first and last name with a blank and added a semicolon at the end of the message. We actually added two fields to the message. Now we replace them with suffixes added to both first and last name. The fields Blank and EOM as well as their output controls can be deleted.

The output format below shows that we have to create, both prefix and suffix literals. We will learn how the formatter treats them in the output control. For this example we add a salary field to the message.

.



For the input format we need to add only one object, the salary field.



Literal: Semicolon
Input Control: CharToSemicolon
Format: Taxes_In
Fields: LastName FirstName Salary

Figure 73. Prefix/Suffix Example: Input Objects

The output message contains the first and last name, separated by a space. The space is a suffix to FirstName. To LastName we add a colon suffix. The salary is prefixed by a dollar sign. Here we don't use an end-of-message indicator.

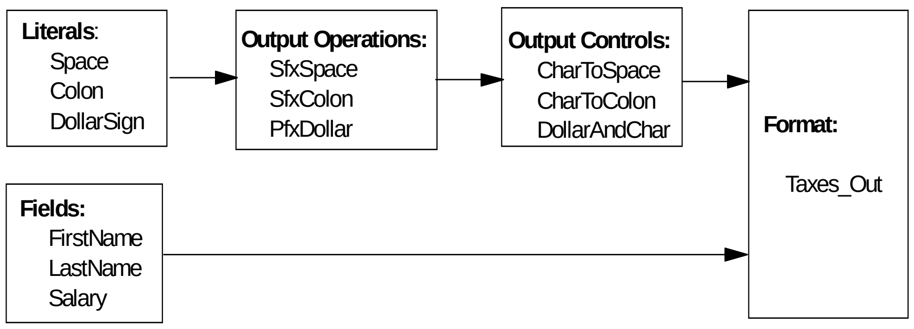


Figure 74. Prefix/Suffix Example: Output Objects

Some of the objects are already in the database. The good thing is that they appear in alphabetical order what makes them easy to find.

5.2.1 Adding Fields and Literals

1. Add the literals Colon and DollarSign as described in 5.1.2.1, “Defining the Literal "Space"” on page 96. Remember to change the ASCII value in the properties section to the correct hex value.
2. Add the field Salary. To parse the salary we use the existing input control CharToSemicolon.
3. Update the format Taxes_In by adding the field component Salary. Make sure the fields are in the correct sequence. Customize the properties of the field Salary with the proper input control name, CharToSemicolon.

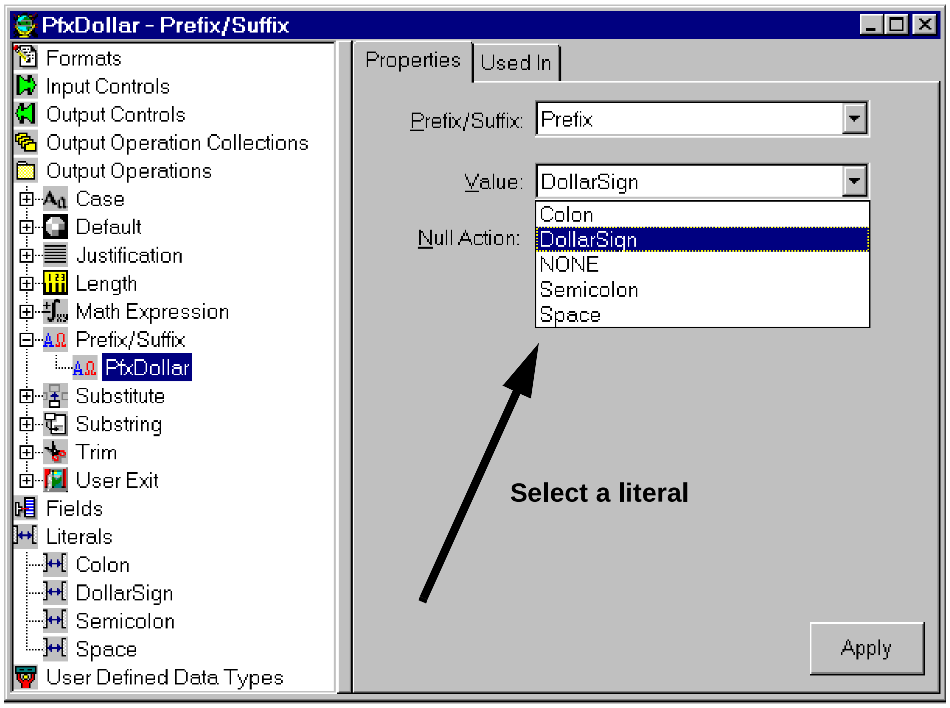
Note: Even though salary consists of numbers, we can still treat this field as character data.

At this point you can use the Visual Tester to test the input format. Input

5.2.2 Creating a Prefix and a Suffix

The literal values for the prefix and suffix are already defined. Now let us create the output operations PfxDollar and SfxSpace.

1. Double-click **Output Operations** and right-click **Prefix/Suffix**.
2. Select **New** and give the prefix the name PfxDollar.



Prefix/Suffix: Prefix

Value: DollarSign

Null Action:

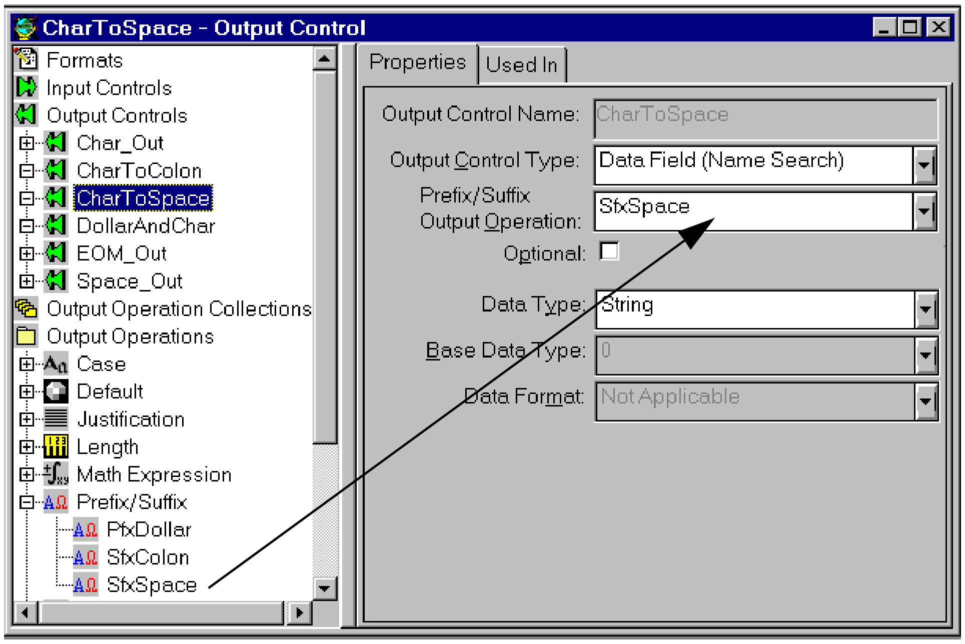
Figure 75. Defining a Prefix

3. In the Properties panel, select **Prefix**.
4. Expand the Value list and select **DollarSign**. As you can see in Figure 75, this list contains all defined literals.
5. Then click **Apply**.
6. Create the entries SfxSpace and SfxColon in the same fashion, but remember to create them as suffixes instead of prefixes.

5.2.3 Attaching Prefixes and Suffixes to Fields

We know that every field in an output format must be associated with and output control. So we use an output control to attach a prefix or a suffix to a field.

1. Right-click **Output Controls** and select **New**.
2. Type in the name CharToSpace. This is a simple output control that outputs the characters and then adds a space (suffix).
3. Don’t change the output control type and the data type.


CharToSpace - Output Control

Output Control Name: CharToSpace
Output Control Type: Data Field (Name Search)
Prefix/Suffix Output Operation: SfxSpace
Optional: ☐
Data Type: String
Base Data Type: 0
Data Format: Not Applicable

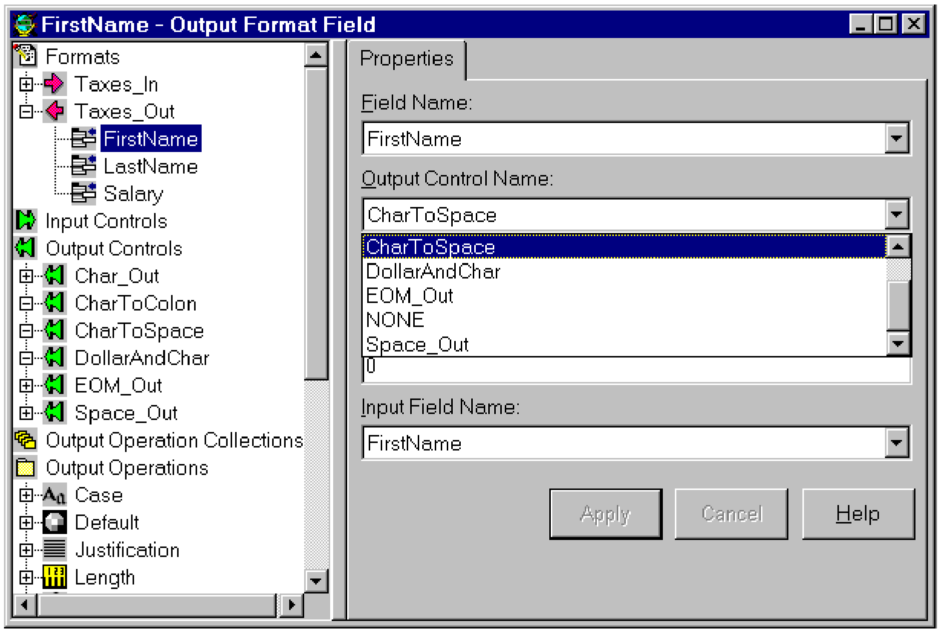
Figure 76. Output Control with Suffix

4. Expand the list of output operations and select **SfxSpace**.
5. Click **Apply**.
6. Define the output controls CharToColon and DollarAndChar using the appropriate output operations, SfxColon and PfxDollar.

Now it is time to assign the output control names to the fields of the output formats:

7. Double-click **Formats** to list the formats and then double-click **Taxes_Out** to expand the list of fields in this format.
8. Delete the fields Blank and EOM by right-clicking the field name and then selecting **Delete** from the menu.
9. Add the field Salary as you did for the input format above.
10. Click **FirstName** and change the Output Control Name from Char_Out to CharToSpace. This will produce a space after the name. Apply the change.
11. Next click **LastName**, change the Output Control Name to CharToColon and apply the change.

12. Click **Salary** and assign the output control name DollarAndChar to it.



Field Name: FirstName

Output Control Name: CharToSpace

CharToSpace
DollarAndChar
EOM_Out
NONE
Space_Out

0

Input Field Name: FirstName

Figure 77. Assigning an Output Control Name to a Field

Now we can use the Visual Tester to test if this example reformats correctly.

Note: There is no delimiter after the salary field.

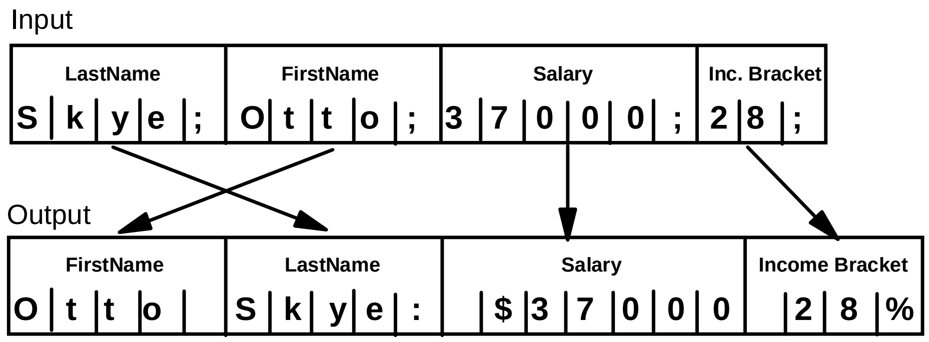
5.3 Output Operation Collections

Output Operation Collections allow you to group and sequence a series of output operations and/or other operation collections. In other words, we can perform more than one action on one field, such as change it to all capital letters, left-justify it, and then add a colon to the end of the field.

Let us add a tax bracket to the input of our example and then show it in the output precedes with a blank and with a percent sign at the end. To do this we need an output collection that consists of two output operations:

- Add a suffix percentage to the end of the tax bracket field.
- Add a prefix space to the beginning of the tax bracket field.

Also, add a second prefix, a space to the beginning of the salary field.



The objects we need to construct the output format are as follows:

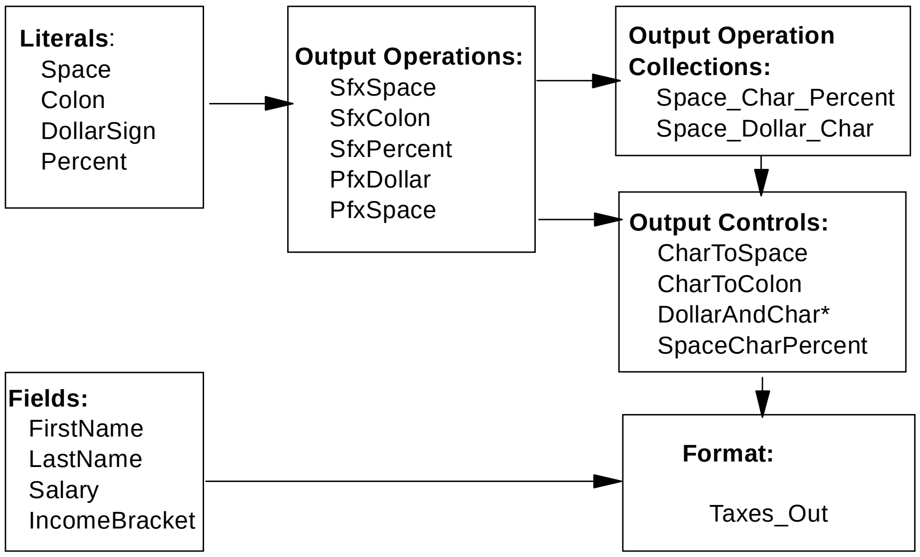


Figure 78. Output Operation Collections Example: Objects

Before we define the output operation collections we have to define some prerequisites:

1. To the input format we have to add the field IncomeBracket which uses the input control CharToSemiolon.
2. Add the literal Percent.

3. Add the suffix SfxPercent to the output operations. Make sure that suffix is selected; prefix is the default.
4. Add the prefix PfxSpace to the output operations.

5.3.1 How to Define an Output Operation Collection

We have to create two collections, one for the salary field and one for the income bracket.

1. Right-click **Output Operation Collections** and select **New.**
2. Type in the name Space_Char_Percent and press Enter.

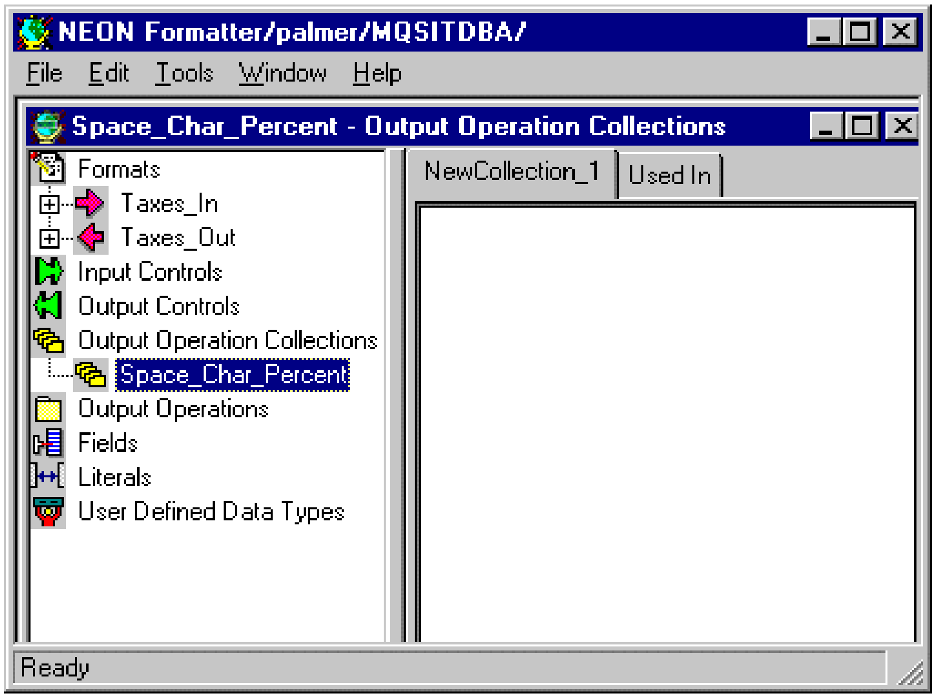
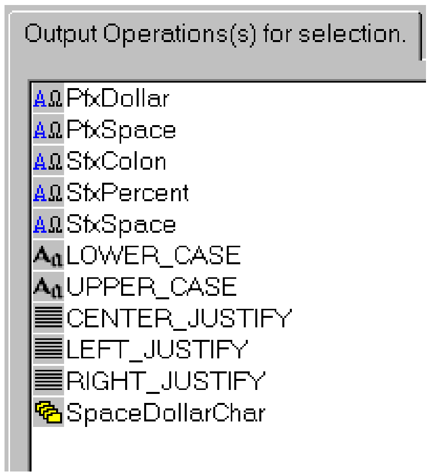


Figure 79. Defining an Output Operation Collection

3. Right-click **Space_Char_Percent** and select **Add Output Operations** from the menu.
4. This brings up the window shown in Figure 80 on page 107 from which you select two options, PfxSpace and SfxPercent. Hold the Control key down while selecting. Then click **Accept Selection**.


Output Operations(s) for selection.

PfxDollar
PfxSpace
SfxColon
SfxPercent
SfxSpace
LOWER_CASE
UPPER_CASE
CENTER_JUSTIFY
LEFT_JUSTIFY
RIGHT_JUSTIFY
SpaceDollarChar

Figure 80. Output Operations Available for Collection

5. Verify that the operations are in the right sequence. In this case, PfxSpace must be performed before SfxPercent.
6. Create another collection, Space_Dollar_Char. The order for the two output operations in this collection is PfxDollar followed by PfxSpace.

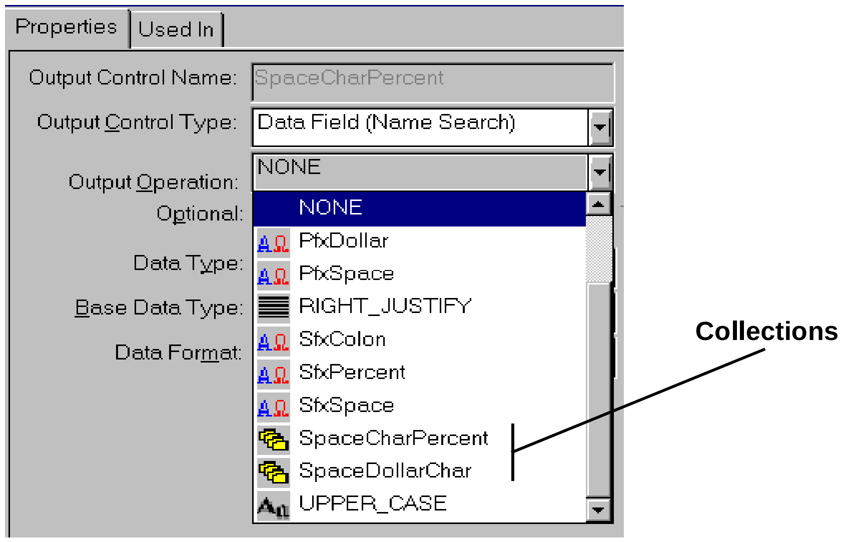
5.3.2 How to Assign an Output Operation Collection to a Field

First, we have to define an output operation that references the collection. Then we assign the operation to the field IncomeBracket.

1. Create a new output operation under the name SpaceCharPercent and select the output operation with the same name. Figure 81 on page 108 indicates that this operation is a collection.
2. Add the field IncomeBracket to the output format.
3. Space_Char_Percent as the Output Control Name in the properties of the field IncomeBracket.
4. For the salary field, change in the properties of the output control DollarAndChar the name of the output operation to SpaceDollarChar. This causes two prefixes to be added, a space and a Dollar sign.

We have just shown an example of output operation collections by creating a group of operations under one collection and assigning that collection to an output control.

You can use the Visual Tester and verify the exercise.


Properties | Used In

Output Control Name: SpaceCharPercent

Output Control Type: Data Field (Name Search)

Output Operation: NONE

Optional:

Data Type:

Base Data Type:

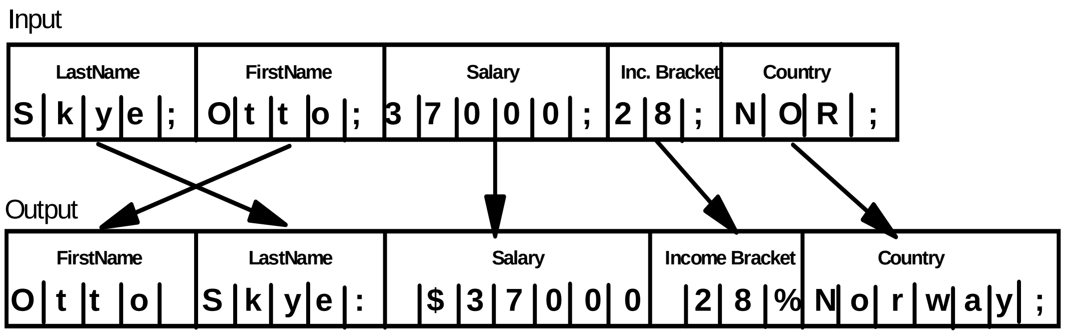
Data Format:

Figure 81. List of Output Operations

5.4 Substituting Field Values

Output Operations provide the different actions that can be performed on an output field. For example, you can change the case of output data, perform mathematical expressions based on input field contents, extract substrings, and much more. We have already used the output operation prefix and suffix. Now we will show an example of another useful operation, substitutions.

Let us work with the following input and output messages:



We will change a three-character abbreviation for countries to their full name, such as LUX becomes Luxembourg. Both values must be defined as literal strings. Therefore, we recommend that you do not use this functions when you substitute hundreds of values. Use a user exit instead. For our example we define the five countries listed in the table below.

Note: We also add a semicolon at the end of the country field.

Table 6. Substitution Example: Table for Countries

Three-character Country Code	Country Name
AUT	Austria
DEN	Denmark
GER	Germany
NOR	Norway
USA	United States

To the input format we have to add only the field Country. The objects for the output format are shown below:

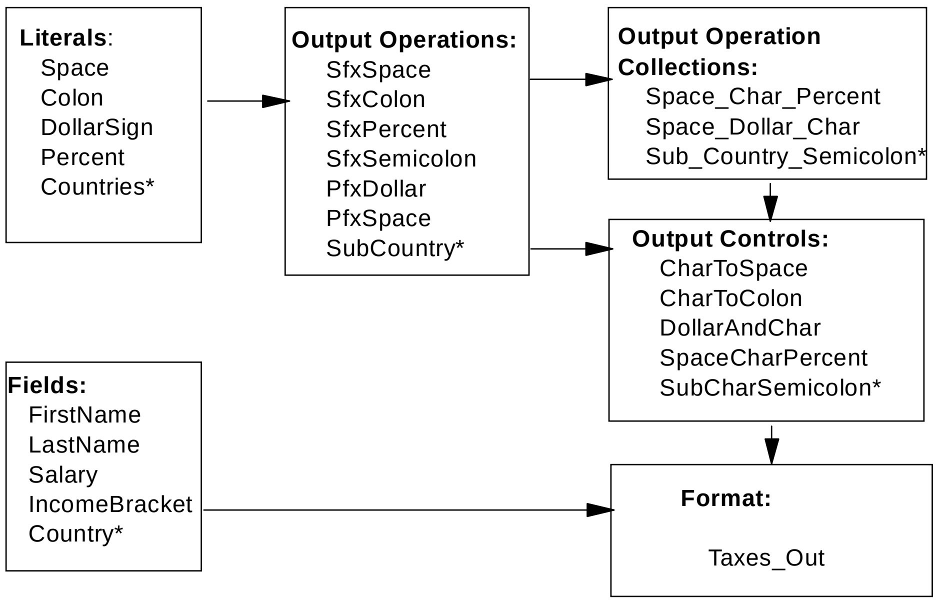


Figure 82. Substitution Example: Objects

Note: The new objects are marked with an asterisk. The country names and their abbreviations are listed in Table 6 on page 109.

Before we explain how to substitute values let us define the new field we use in the input and output formats:

1. Define the field Country.
2. Add it to the input format and specify CharToSemicolon as input control.
3. Add a new suffix, SfxSemicolon to the ouput operations. Don't forget to change prefix to suffix.

5.4.1 How to Define Substitution Values

1. Add the literals, both the three-character abbreviations of the countries and their full names. In this case you do not need to change the ASCII properties; they stay as they are.

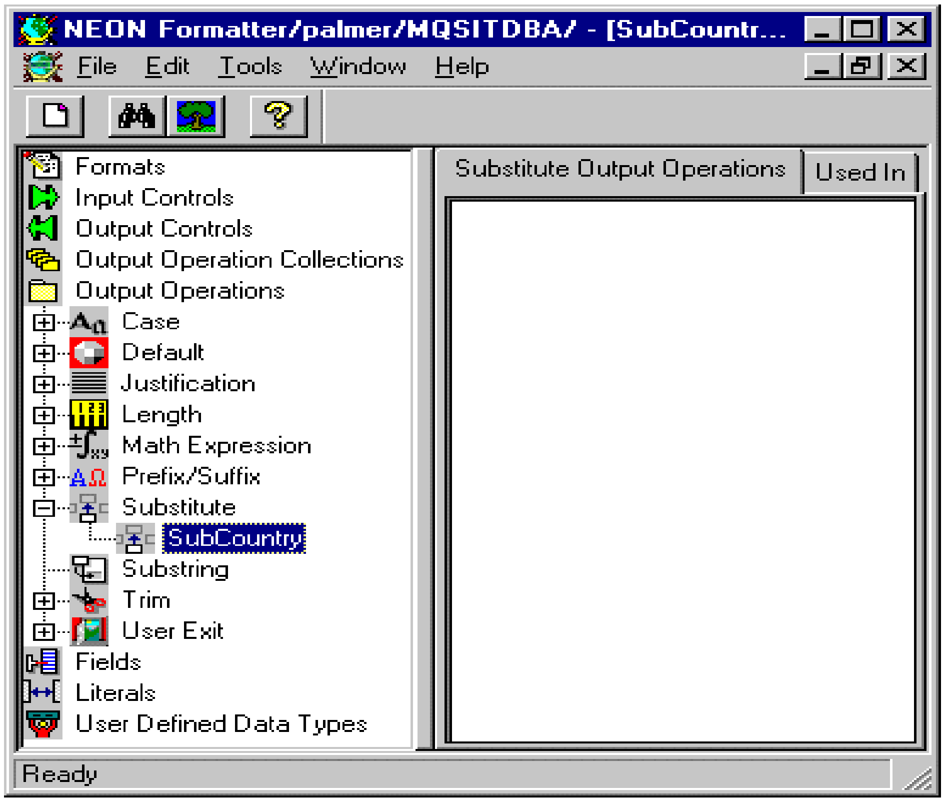


Figure 83. Defining a Substitution

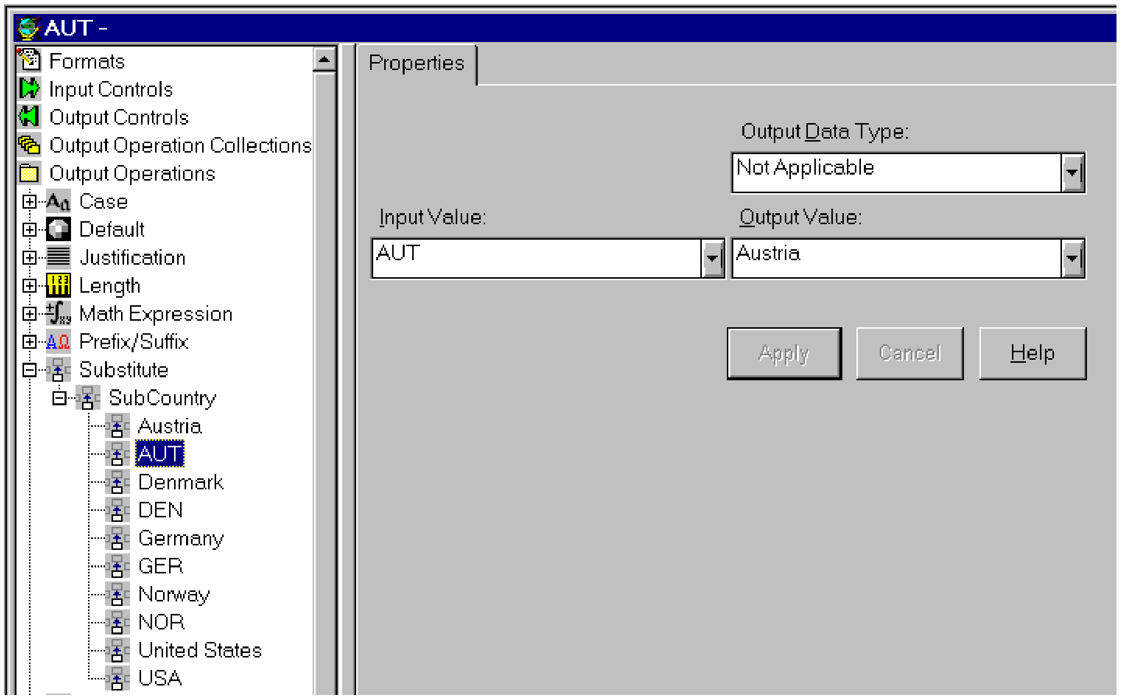
2. Add the appropriate Output Operations. Double-click **Output Operations** and right-click **Substitution**.

3. Select **New** from the menu and create SubCountry. Figure 83 on page 110 shows what you should see in your window.

4. Right-click the newly created object **SubCountry** and select **Add Substitute Items**. Add the three-character items along with their corresponding name. Under SubCountry you should now see ten items.

5. Then update each substitution literal (AUT, DEN, GER, NOR, USA) by associating each literal to what they are going to be substituted with. This is shown in Figure 84.

   Click **Substitute**, then **SubCountry**, and then one of the abbreviations, such as **AUT**. In the subsequent window, select the proper output value, here Austria, and click **Apply**.



Output Data Type: Not Applicable

Input Value: AUT

Output Value: Austria

Figure 84. Specifying Substitution Values

5.4.2 How to Define Substitutions for a Field

1. Create an Output Operation Collections, here Sub_Country_Semicolon. Right-click it and add to the collection the operations SubCountry and SfxSemicolon, in that order.
2. Create the Output Control Sub_Char_Semicolon and assign it to the collection Sub_Country_Semicolon you just created.
3. Update the Taxes_Out format by adding field Country.
4. Change the Output Control Name to Sub_Char_Semicolon in the properties of the field Country and apply the changes.

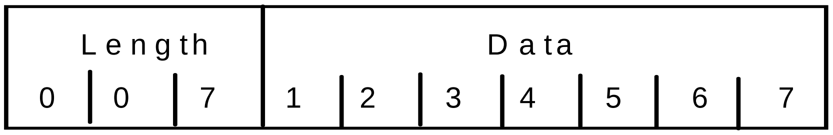
This concludes the substitution example.

5.5 Using Fields Containing Length and Data

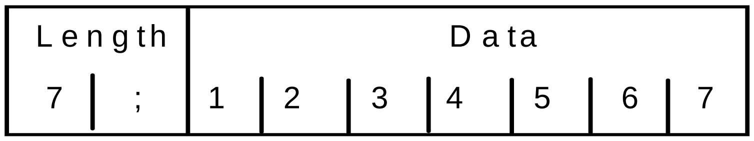
In the previous examples, we worked with fixed length fields and with variable length fields that are separated by a delimiter, such as a semicolon. In this section, we explain how to use fields that contain the field length, that is, one field consists of two sub-fields, length and data. The length field may be of fixed or variable length. Variable length fields need a delimiter to separate them from the data portion of the field. Such fields can play an important part with the message parsing.

In this context, we deal with the MQSI input control type "Length and Data" which allows these field types:

- Length field with exact length followed by data



- Variable length field (delimited) followed by data



You may also specify a minimum length in connection with a delimiter or use white space to separate the fields.

In this section, we demonstrate how to use fields that contain length and data in input and output formats. To do that we create two new formats, Address_In and Address_Out, with the following fields:

- First name
- Last name
- Street
- City

Below is an example of such a message. The field length portion of each field is exactly two bytes long, shown underlined.

```
04Otto06Kaiser11Ku-Damm 123111000 Berlin
```

Note: The data can contain blanks.

5.5.1 Parsing Input Fields with Length and Data

For the input format we need the following objects:

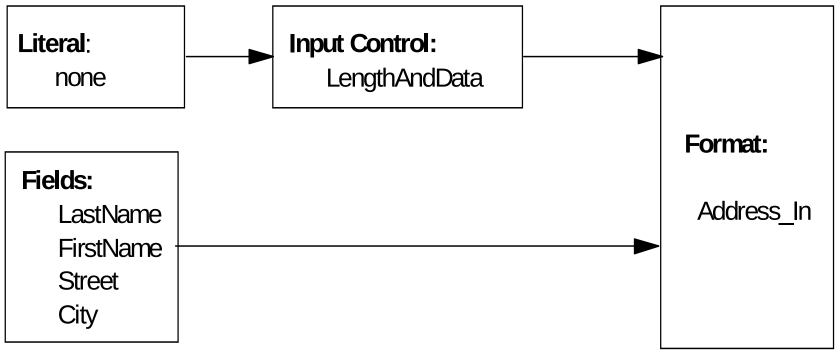
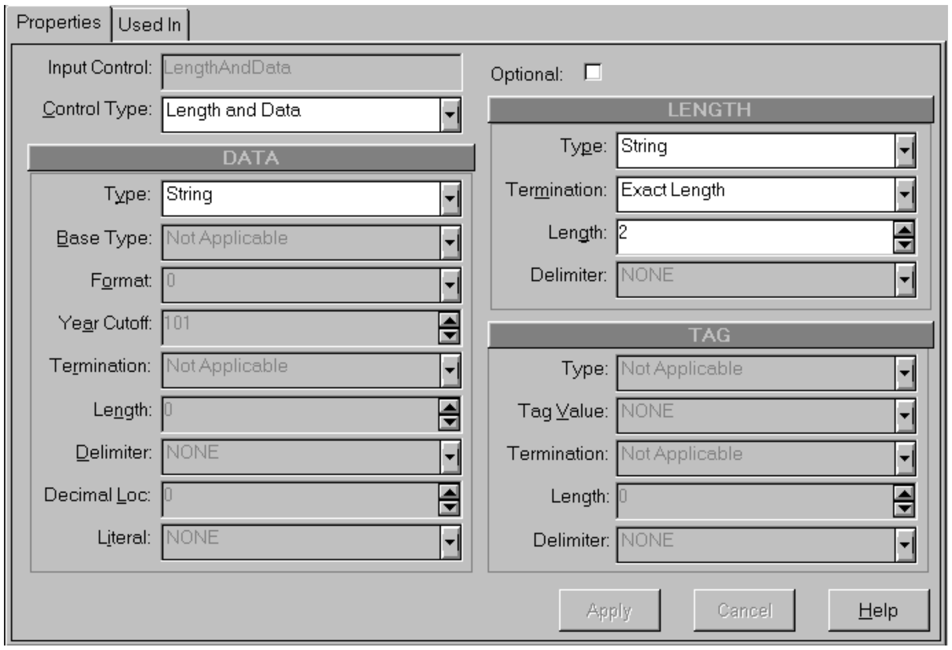


Figure 85. Length and Data Example: Input Objects

Two of the fields, LastName and FirstName, are already in the database. These are names only, without any attributes. You can reuse the fields in as many formats as you like and each time the attributes may differ. You assign attributes through an Input Control after the field has been added to the format.

The following steps describe what you have to do to define the format Address_In for the message shown above.

1. Add the fields Street and City to the database.
2. Create an input control that can be used for all fields:
   - Click **Input Controls** and select **New** from the menu.
   - Type the name LengthAndData and press Enter.
   - In the Properties window, shown in Figure 86, select:
     - Control type: **Length and Data**
     - Data type: **String**
     - Type of length field: **String**
     - Termination of length field: **Exact Length**
     - Length of the length field: **2** bytes
   - Click **Apply**.



Input Control: LengthAndData
Control Type: Length and Data
Optional: ☐

DATA
Type: String
Base Type: Not Applicable
Format: 0
Year Cutoff: 101
Termination: Not Applicable
Length: 0
Delimiter: NONE
Decimal Loc: 0
Literal: NONE

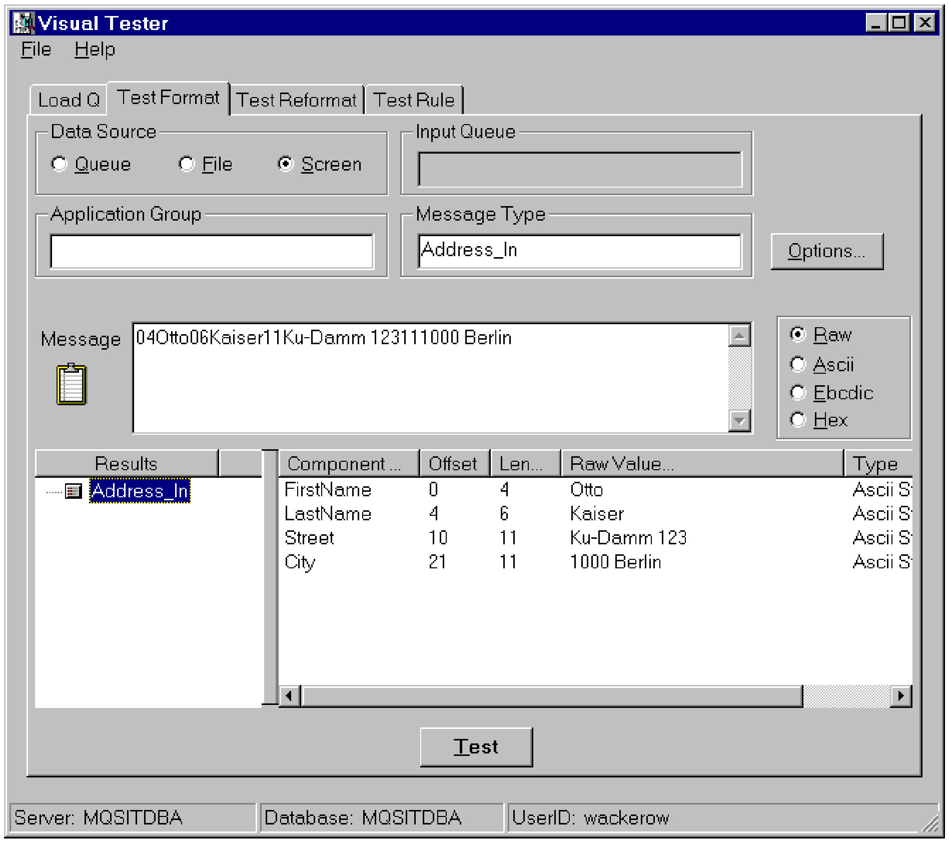
LENGTH
Type: String
Termination: Exact Length
Length: 2
Delimiter: NONE

TAG
Type: Not Applicable
Tag Value: NONE
Termination: Not Applicable
Length: 0
Delimiter: NONE

Figure 86. Input Control for Fields with Length and Data

3. Create a new input format with the name Address_In.
4. Add the four field components and put them in the right order.
5. Select and assign to each field the input control LengthAndData.

Now use the Visual Tester and check out your definitions. The result should be as shown in Figure 87.



Data Source: Queue, File, [✓ Screen]

Input Queue:

Application Group:

Message Type: Address_In

Message: 04Otto06Kaiser11Ku-Damm 123111000 Berlin

[✓ Raw] Ascii Ebcdic Hex

Results: Address_In

Component ...	Offset	Len...	Raw Value...	Type
FirstName	0	4	Otto	Ascii S
LastName	4	6	Kaiser	Ascii S
Street	10	11	Ku-Damm 123	Ascii S
City	21	11	1000 Berlin	Ascii S

Server: MQSITDBA Database: MQSITDBA UserID: wackerow

Figure 87. Test Input Format with Length and Data

5.5.2 Putting Fields with Length and Data

For the output format we use the same fields as we used for the input above. However, let us make the street 15 characters long, excluding the two-byte length portion. Based on the above input we want to see the following output:

```
04Otto06Kaiser15Ku-Damm 123    111000 Berlin
```

Also, let us express the field name as an integer. If you use a string the field is less than 10 bytes, the size of the length field will be one byte. The output control does not allow us to specify a fixed length.

For the output format we need the following objects:

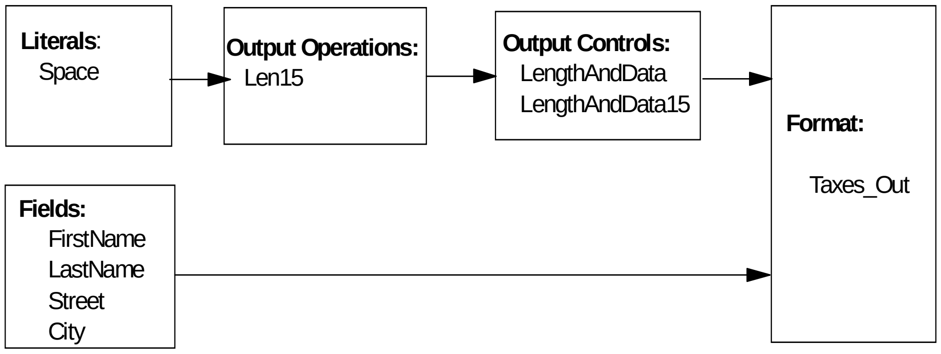
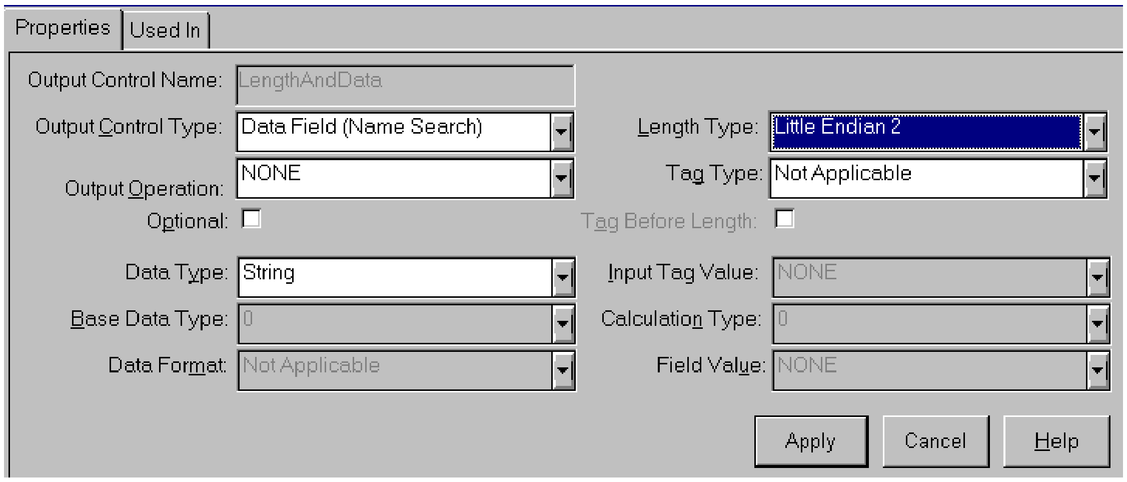


Figure 88. Length and Data Example: Output Objects

The following describes how to define the objects. The fields are already in the database.

1. Create an output operation of the type Length. This operation allows us to make the Street field exactly 15 bytes long.
   - Expand the **Output Operations** tree, right-click **Length** and then select **New**.
   - Type the name Len15.
   - In the Properties panel, select **Space** as pad character and **15** as length.
   - Click **Apply**.
2. Create the output control LengthAndData. This control is used for all but the street.
   - Right-click **Output Controls** and select **New** from the menu.
   - Type the name LengthAndData and press Enter.
   - In the Properties panel shown in Figure on page 117 specify the following and then click **Apply**:
     - Control type: Data Field (Name Search)
     - Output operation: None
     - Data type: String
     - Length type: Little Endian 2 for the integer that holds the field length.


Properties | Used In

Output Control Name:	LengthAndData		
Output Control Type:	Data Field (Name Search)	Length Type:	Little Endian 2
Output Operation:	NONE	Tag Type:	Not Applicable
Optional:	☐	Tag Before Length:	☐
Data Type:	String	Input Tag Value:	NONE
Base Data Type:	0	Calculation Type:	0
Data Format:	Not Applicable	Field Value:	NONE

Figure 89. Output Control for a Field with Length and Data

Note: Don't use String or Numeric as length type or your length field will be one or two bytes long as you see below:

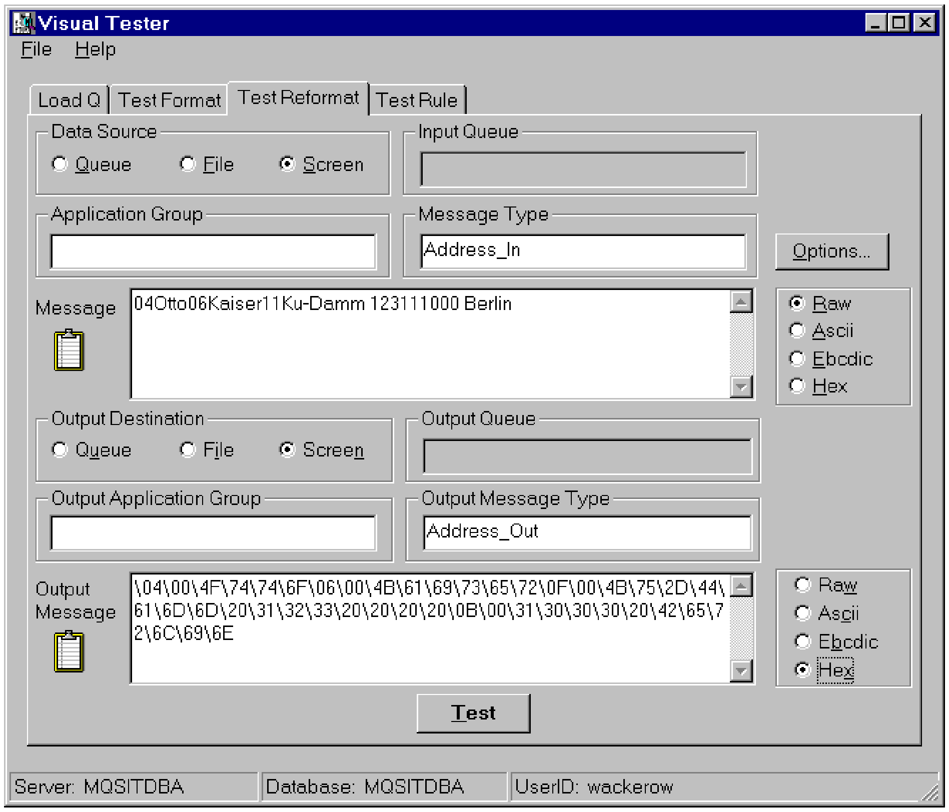
```
4Otto6Kaiser15Ku-Damm 123    111000 Berlin
```

You would have trouble processing such a message.

3. Create the output control LengthAndData15 to be used with the field Street.
   - Right-click **Output Controls** and select **New** from the menu.
   - Type the name LengthAndData15 and press Enter.
   - In the Properties panel shown in Figure on page 117 specify the following and then click **Apply**:
     - Control type: Data Field (Name Search)
     - Output operation: Len15
     - Data type: String
     - Length type: Little Endian 2
4. Create the output format Address_Out and add the four fields to it. Make sure they are in the correct order.
5. Click each field name and assign an output control to it:
   - LengthAndData15 to Street

- LengthAndData to all other fields

When you test the reformatting with the Visual Tester view the output data in hex representation as shown below.



Data Source: Queue File [✓ Screen]

Input Queue:

Application Group:

Message Type: Address_In

Message: 04Otto06Kaiser11Ku-Damm 123111000 Berlin

[✓ Raw] Ascii Ebcdic Hex

Output Destination: Queue File [✓ Screen]

Output Queue:

Output Application Group:

Output Message Type: Address_Out

Output Message:

```
\04\00\4F\74\74\6F\06\00\4B\61\69\73\65\72\0F\00\4B\75\2D\44\
61\6D\6D\20\31\32\33\20\20\20\20\0B\00\31\30\30\30\20\42\65\7
2\6C\69\6E
```

Raw Ascii Ebcdic [✓ Hex]

Server: MQSITDBA | Database: MQSITDBA | UserID: wackerow

Figure 90. Test Output Format with Length and Data

5.6 Using Fields Containing a Tag (Field ID)

Tags can be a useful item within your message. A tag or field ID describes the field. It allows fields to appear in any order within the message. Application programs and the Formatter can parse an input format by searching for the tag. Tags can appear together with a field length. They can be of fixed length or delimited. The input controls provide three choices:

- A tag followed by data
- A tag followed by a length field and data
- A length field followed by a tag and data

We will demonstrate the use of tags based on the address message defined in the previous section. We assign each field a tag or field ID:

11	First name
12	Last name
13	Street
15	City

5.6.1 Parsing Input Fields with Tag and Data

For this example, we chose that all tags are two bytes in length and numeric and that the fields end with a semicolon. This is an example of a message:

```
11Xaver;12Huber;13Rosenstr. 17;15Muenchen;
```

The tags are underlined. We will also allow that the fields appear in any order, for example:

```
15Muenchen;12Huber;11Xaver;13Rosenstr. 17;
```

Note: The Formatter parses the fields as they appear in the input message and not as they are defined in the input format.

We need the following objects:

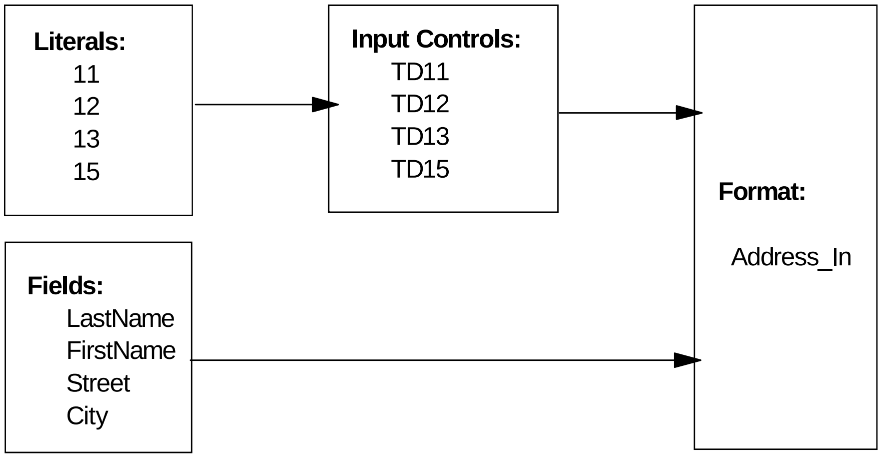
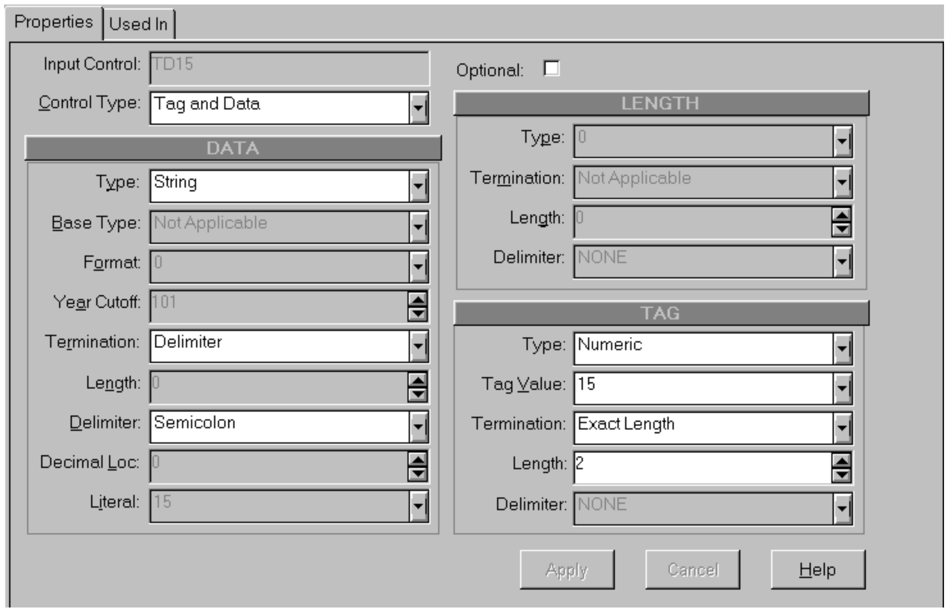


Figure 91. Tag and Data Example: Input Objects

Now let us create them and then test the format:

1. Create the four literals used as tags (11, 12, 13 and 15).
2. Create the input controls TD11, TD12, TD13 and TD15. TD stands for Tag and Data. In the Properties window select the values shown in Figure 92 and apply them:
    - The control type is of course Tag and Data.
    - The data is a string delimited with a semicolon.
    - The length parameters are not applicable because we do not have a length within the field.
    - The tag is numeric (you could also select String), its value is one of the literals defined, and the length of the tag is exactly two bytes.
3. Select each field in the input format Address_In and change the Input Control Name to TD11, TD12, TD13, or TD15.
4. Click the format name **Address_In** and mark the check box Random Field Order on the Properties panel.



Input Control: TD15
Control Type: Tag and Data
Optional: ☐

DATA

Type:	String
Base Type:	Not Applicable
Format:	0
Year Cutoff:	101
Termination:	Delimiter
Length:	0
Delimiter:	Semicolon
Decimal Loc:	0
Literal:	15

LENGTH

Type:	0
Termination:	Not Applicable
Length:	0
Delimiter:	NONE

TAG

Type:	Numeric
Tag Value:	15
Termination:	Exact Length
Length:	2
Delimiter:	NONE

Figure 92. Input Control for Fields with Tag and Data

5.6.2 Parsing Input Fields with Tag, Length and Data

To demonstrate this function we create another set of input controls with the names TLD11, TLD12, TLD13 and TLD15 (TLD stands for Tag, Length and Data).

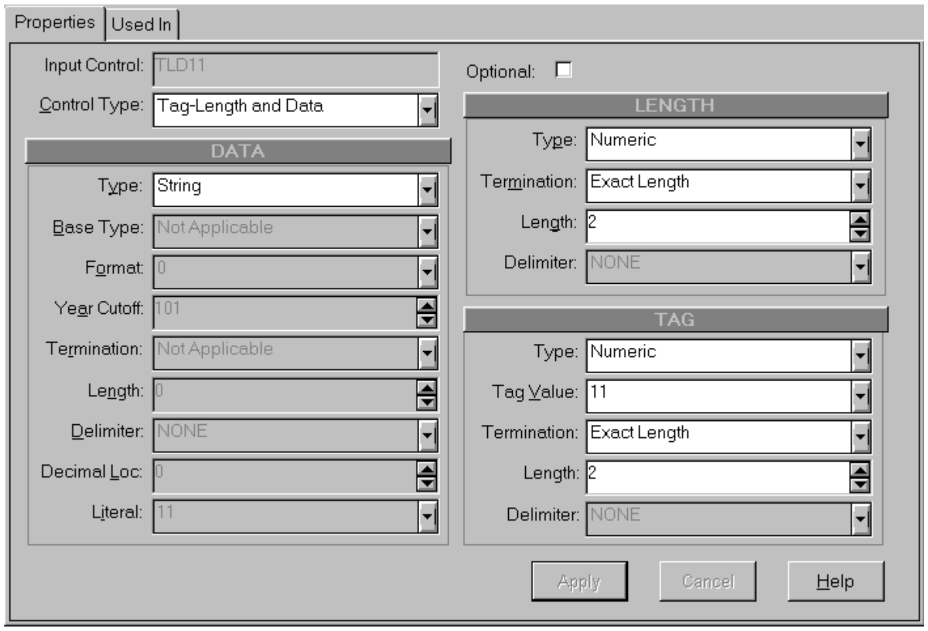
In the example message below the tag is underlined and the length is double-underlined.

```
1103Jim1206Miller1308Broadway1508New York
```

And this is what you have to do:

1. Create the new input controls TLD11, TDL12, TLD13 and TLD15 and specify the following properties:
   - The Control Type Tag-Length and Data
   - A numeric length field which is exactly two bytes long
2. All that's left to do is to change the input control names in the fields from TLxx to TLDxx.

Note: The specifications are the same for Length-Tag and Data.



Input Control: TLD11
Control Type: Tag-Length and Data
Optional: ☐

DATA
Type: String
Base Type: Not Applicable
Format: 0
Year Cutoff: 101
Termination: Not Applicable
Length: 0
Delimiter: NONE
Decimal Loc: 0
Literal: 11

LENGTH
Type: Numeric
Termination: Exact Length
Length: 2
Delimiter: NONE

TAG
Type: Numeric
Tag Value: 11
Termination: Exact Length
Length: 2
Delimiter: NONE

Figure 93. Input Control for Field with Tag, Length and Data

5.6.3 Putting Fields with Tag and Data

You can add tags to the fields in the output format. The tag can come from the input message (if it has a tag) or it can be added using a prefix output operation. In the following we show both possibilities. Again, we modify the objects belonging to the Address_In and Address_Out formats.

Let us assume that we receive a message that provides tags only for the first and last names but not for the street and the city. Since there is no length field, a semicolon is used to separate the fields. Below is an example of the input message. The tags are underlined.

```
11Anton;12Meier;Pilsener Str.;Jever;
```

The output message, shown below, contains tags for all fields. Two tags are taken from the input and two are added using field prefixes.

```
11Anton;12Meier;13Pilsener Str.;15Jever;
```

For this example we reuse the input format Address_In. We change only the output control name for the fields Street and City to CharToSemicolon.

For the output format we need the following objects:

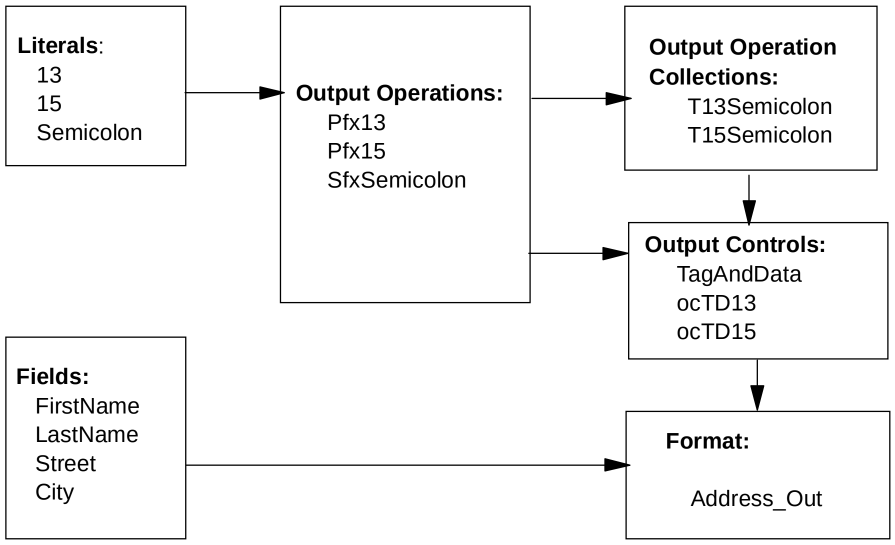
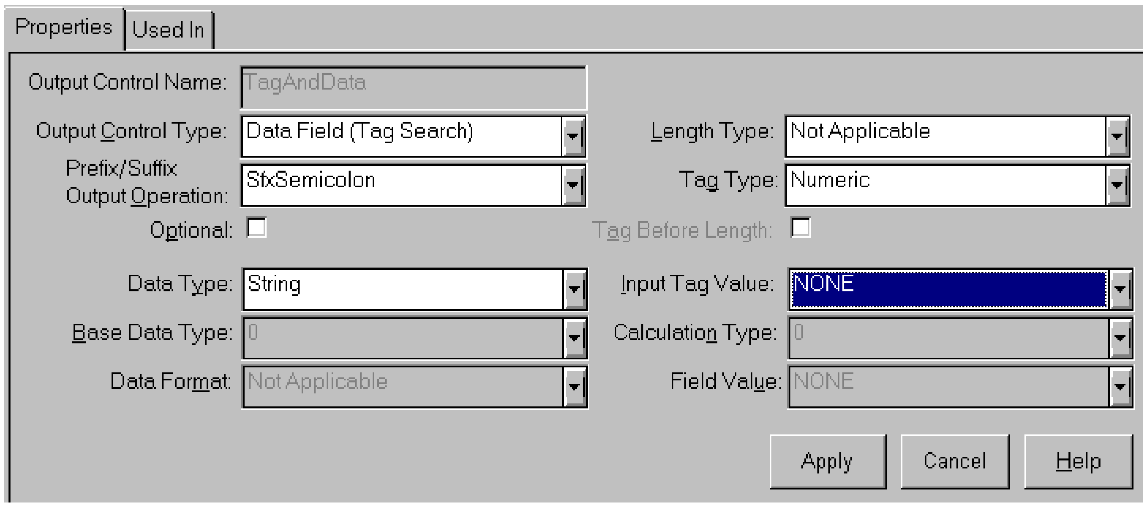


Figure 94. Tag and Data Example: Output Objects

To create the objects in the database follow these steps:

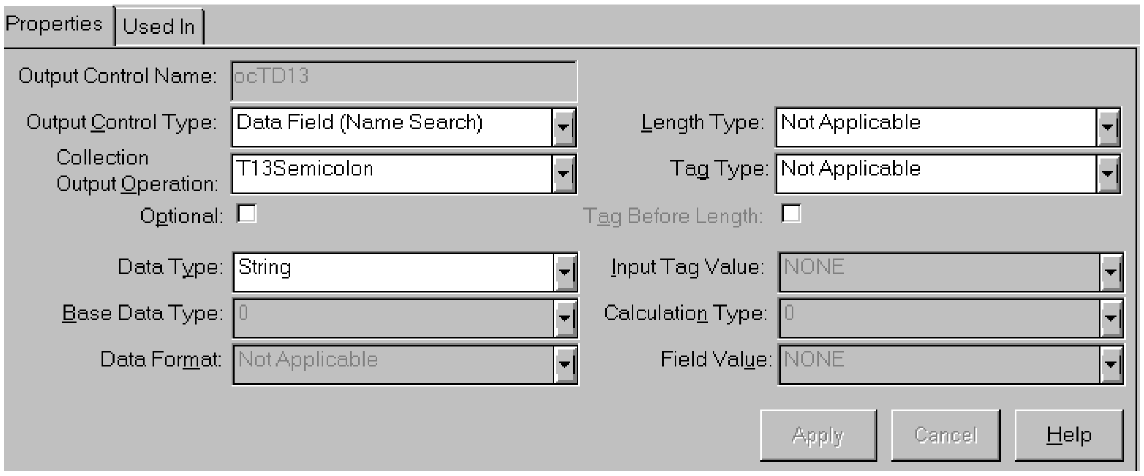
1. If not already done so, define the literals.
2. If not already done so, define the output operation SfxSemicolon.
3. Create the two new output operations, the prefixes containing the tag values 13 and 15 (for street and city). Name them Pfx13 and Pfx15.
4. Create two output operation collections. You need it for the fields Street and City. Both fields require a prefix, namely the tag (13 or 15) and the suffix semicolon to indicate the end of the field.
   - Right-click **Output Operation Collections** and select **New**.
   - Type the name, T13Semicolon or T15Semicolon (T stands for tag) and press Enter.
   - Right-click the name and select **Add Output Operations** from the menu.
   - From the list of output operations select Pfx13 or Pfx15 and SfxSemicolon (hold the Control key down) and click **Accept Selection**.
   - Make sure that the operations appear in the correct order, prefix before suffix.
5. Create the output control TagAndData for use by the two names fields. For these fields, we get the tag from the input and have only to add the semicolon suffix as field delimiter.
   - Right-click **Output Controls**, then click **New** and type the name TagAndData and press Enter.
   - In the Properties panel enter the values as shown in Figure 95 on page 124:
     - Since both fields are tagged, select **Data Field (Tag Search)** as output control type.
     - Select SfxSemicolon as output operation. This adds the delimiter at the end of the field.
     - As Tag Type select **Numeric** or **String**.
     - Keep **NONE** as Input Tag Value.

       If you choose 10 or 11 then you need two output controls or both output fields get the value you specify here.
     - Click **Apply**.



Output Control Name: TagAndData
Output Control Type: Data Field (Tag Search)
Prefix/Suffix Output Operation: SfxSemicolon
Optional: ☐
Data Type: String
Base Data Type: 0
Data Format: Not Applicable
Length Type: Not Applicable
Tag Type: Numeric
Tag Before Length: ☐
Input Tag Value: NONE
Calculation Type: 0
Field Value: NONE

Figure 95. Output Control for a Field with Tag (from Input) and Data



Output Control Name: ocTD13
Output Control Type: Data Field (Name Search)
Collection Output Operation: T13Semicolon
Optional: ☐
Data Type: String
Base Data Type: 0
Data Format: Not Applicable
Length Type: Not Applicable
Tag Type: Not Applicable
Tag Before Length: ☐
Input Tag Value: NONE
Calculation Type: 0
Field Value: NONE

Figure 96. Output Control for a Field with Tag and Data

6. Create the output controls for the two other fields. The input fields do not have a tag. We have to add it through a prefix. The fields also need a semicolon at the end.
   - Right-click **Output Controls**, then click **New** and type the name ocTD13 or ocTD15, respectively and press Enter.
   - In the Properties panel enter the values as shown in Figure 96:

- Since both fields don’t have tags, select **Data Field (Name Search)** as output control type.
- Select the collection **T13Semicolon** (or **T15Semicolon**) as output operation. This inserts the tag (13 or 15) at the beginning of the field and adds the delimiter at the end.
- Leave the Tag Type as is, this field is only for tags coming from an input field.
- Click **Apply**.

5.6.4 Putting Fields with Tag, Length and Data

You may want to include the length of the data in a field. The length can be before or after the tag. The two examples below show the tag underlined and the length double-underlined:

$\underline{\texttt{11}}\underline{\underline{\texttt{4}}}$`Fred;`$\underline{\texttt{12}}\underline{\underline{\texttt{10}}}$`Flintstone;`$\underline{\texttt{13}}\underline{\underline{\texttt{5}}}$`Hut 3;`$\underline{\texttt{15}}\underline{\underline{\texttt{7}}}$`Bedrock;`

$\underline{\underline{\texttt{6}}}\underline{\texttt{11}}$`Barney;`$\underline{\underline{\texttt{6}}}\underline{\texttt{12}}$`Rubble;`$\underline{\underline{\texttt{5}}}\underline{\texttt{13}}$`Hut 7;`$\underline{\underline{\texttt{7}}}\underline{\texttt{15}}$`Bedrock;`

To have the Formatter insert the length field, specify in the output control (shown in Figure 95 on page 124) these additional properties:

1. A Length Type, such as String, Numeric, Little Endian 2, Little Endian 4
2. The check box Tag Before Length

In the examples above we used the length type Numeric. You notice that the length portion is either one or two bytes, depending on the data length. That will cause problems for the application that will get the message. Specifying an integer would help.

This concludes the example about tags and data in output formats. As always, we recommend that you use the Visual Tester to verify your definitions.

5.7 Messages with Date and Time Fields

To parse and reformat date and/or time fields, special functions have been added to the input and output controls. There are default settings for date and time and for date and time in the same field. Furthermore, there are 30 custom date and time formats to choose from. This allows you to choose a different format for either date or time or both. Also, it allows you to convert time and/or date fields received from an input message to a different format for the output message.

The defaults are the following:

Date	YYYYMNDD
Time	HHMMSS
Date and time	YYYYMNDDHHMMSS
Custom	MN/DD/YYY

Note: MN stands for month and MM for minutes.

Where do I find the various formats?

- Input controls
  - Select the control type Data Only.
  - Select one of the data types Date, Time, Date and Time and you see the default in the grayed-out Format field.
  - Select the data type Custom Date and Time and you can choose from 30 different formats in the Format list.
- Output Controls
  - Select the output control type Data Field (Name Search) or Data Field (Tag Search).
  - Select one of the data types Date, Time or Date and Time and you see the default in the grayed-out Data Format field.
  - Select the data type Custom Date and Time and you can choose from 30 different formats in the Data Format List.

As an example, we show how to convert a date from one format to another. Both dates in Figure 97 on page 126 are custom dates.

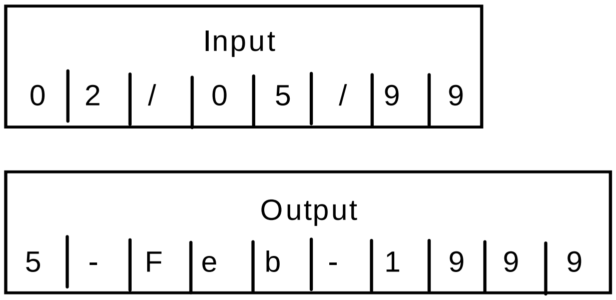


Figure 97. Conversion of a Date Field

5.7.1 Parsing a Date Field

To parse the input, let us define the input control icDate and assign it to the date field in the input format. For this example, we append the field Date to the format Address_In. The following steps explain how to do that:

1. Define a field with the name Data.
2. Right-click **Input Controls** and select **New** from the menu.
3. Type the name icDate and press Enter. This displays the window you see in Figure 98 on page 128.

   Note: The figure shows only the left side of the Properties panel.
4. Select the control type **Data Only**. This is the only one valid for date and time fields.
5. Select the data type **Custom Date and Time**.
6. Leave the base type, String, unchanged.
7. From the formats select **MM/DD/YY**. Note that there is also a format with a four-digit year.
8. When you specify a two-digit year (as in this case), you must change the 101 in Year Cutoff to a value in the range from 0 to 100. 101 is invalid and you get an error message if you don’t change it.

   What is the cutoff field for?

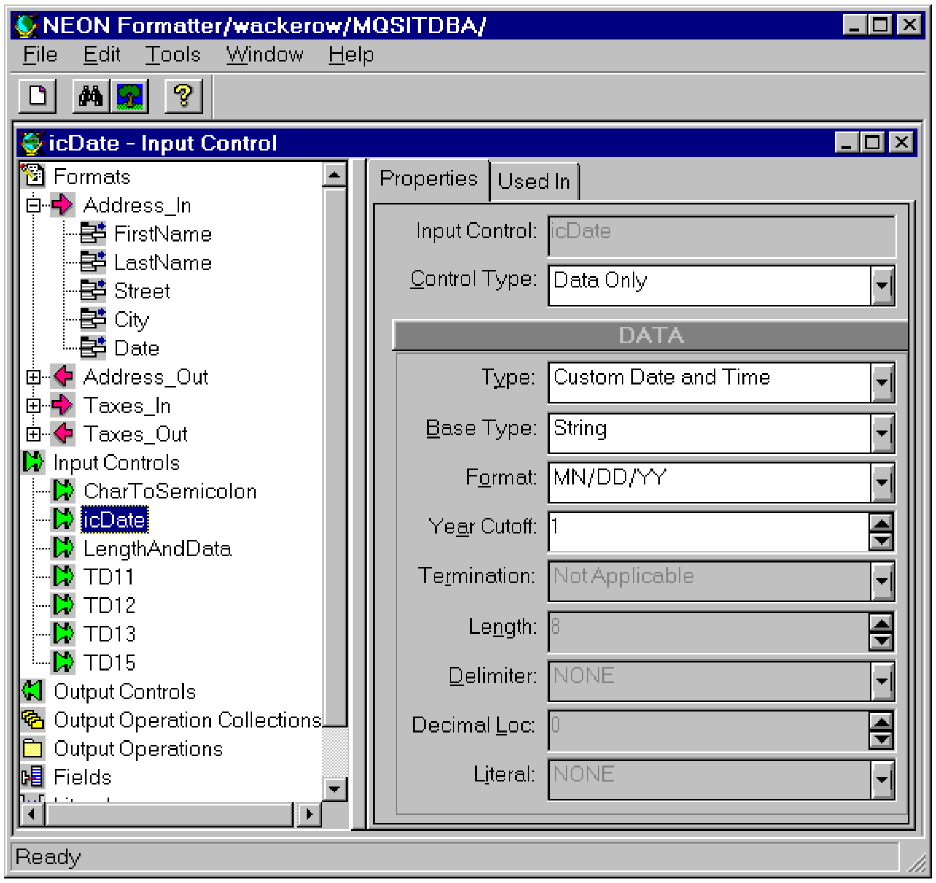
   This is the Formatter’s way to address Year 2000 compliance. It makes you decide what years belong in the 20th and 21st centuries. If you specify:

0	All years will get the prefix 19, as in 1900 or 1999.
100	All years will get the prefix 20, as in 2000 or 2099.
50	The years 50 to 99 get the prefix 19, as in 1950 or 1999; and the years 00 through 49 get the prefix 20, as in 2000 or 2049.
9. For this example, you may enter 1. Click **Apply**.
10. Add the field Date to the format Address_Out and assign it to the input control icDate.

Below is an example of an input message. There is no delimiter at the end of the date. The formatter treats it as a fixed length field.

```
11Peter;12Pan;Archway;Hursley;02/05/99
```

Use the Visual Tester to verify your definitions.



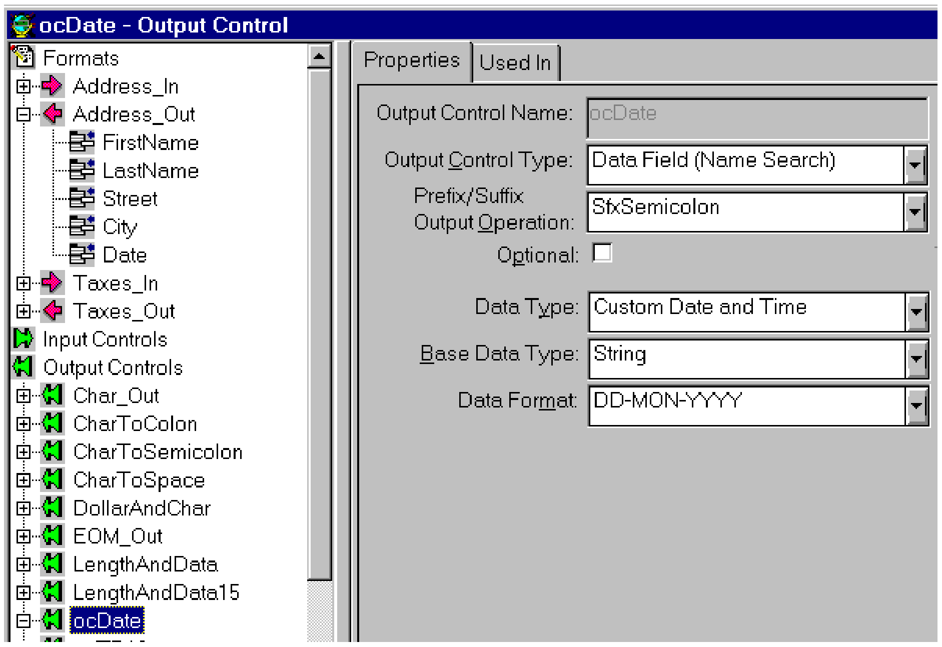
Input Control: icDate
Control Type: Data Only
Type: Custom Date and Time
Base Type: String
Format: MN/DD/YY
Year Cutoff: 1
Termination: Not Applicable
Length: 8
Delimiter: NONE
Decimal Loc: 0
Literal: NONE

Figure 98. Input Control for a Custom Date and Time Field

5.7.2 Putting a Date Field into a Message

To complete the example we have to place the date in the output message, however in a different format. We have to create an output control, ocDate, and assign it to the field Date which has to be added to Address_Out. The following steps explain how to do that:

1. Right-click on **Output Controls** and select **New** from the menu.
2. Type the name ocDate and press Enter.
3. Select the values shown in Figure 99 on page 129 and click **Apply**.
4. Add the field Date to the format Address_Out and assign to it the output control ocDate.


ocDate - Output Control

Formats
- Address_In
- Address_Out
  - FirstName
  - LastName
  - Street
  - City
  - Date
- Taxes_In
- Taxes_Out

Input Controls

Output Controls
- Char_Out
- CharToColon
- CharToSemicolon
- CharToSpace
- DollarAndChar
- EOM_Out
- LengthAndData
- LengthAndData15
- ocDate

Properties | Used In

Output Control Name:	ocDate
Output Control Type:	Data Field (Name Search)
Prefix/Suffix Output Operation:	SfxSemicolon
Optional:	☐
Data Type:	Custom Date and Time
Base Data Type:	String
Data Format:	DD-MON-YYYY

Figure 99. Output Control for a Custom Date and Time Field

Note: We also place a semicolon at the end of the date.

The reformatted message looks like this:

```
11Peter;12Pan;13Archway;15Hursley;05-feb-1999;
```

The tags are underlined.

The data conversion can be a very handy function because of the various international date standards or even to help solve the Y2K problem. You could, for example, change the date in all messages that flow between programs in your enterprise by routing them through an MQSI rules processor.

5.8 Adding Fields with Calculated Values

The output operations allow you to do some calculations using values from input fields and literals. In the following we use the output operation Math Expression to calculate taxes. The resulting value will be added to the output message in the field TaxPaid. We calculate the taxes with this formula:

TaxPaid = Salary / 100 * Income Bracket

To do this we have to create the following objects and add them to the output format Taxes_Out:

1. Fields: TaxPaid
2. Output Operations, Math Expression: CalcTax
    - Decimal Precision: 2
    - Round: Up
    - Mathematical Expression: Salary / 100 * IncomeBracket
3. Output Operation Collections: TaxPaid
    - Add Output Operations:
        - CalcTax
        - SfxSemicolon
4. Output Control: TaxOut
    - Output Control Type: Data Field (Name Search)
    - Output Operation: TaxPaid
    - Data Type: String
5. Output Format: Taxes_Out
    - Add Field Components: TaxPaid
    - Properties for field TaxPaid:
        - Output Control Name: TaxOut
        - Access Mode: Normal Access
        - Input Field Name: NONE

Here is an example:

Input message (Taxes_In):

```
Saetra;Morton;88000;20;NOR;
```

Output message (Taxes_Out):

```
Morton Saetra:$ 88000 20%Norway;22000.00;
```

Note: The tax value contains two decimal positions. You specify this in the Properties for the Math Expression.

5.9 Compound Formats

A compound format is a collection of flat and/or other compound formats. The flat or compound formats that reside within a compound format are called components. Compounds can help with the organization, grouping and parsing of messages. You can group sections of the total message.

Formats within a compound can be:

- Repeatable or single
- Mandatory or optional

Note: Optional component formats can have *missing mandatory fields* and the parse or reformat can still succeed.

In this section, we will develop two compound formats, an input and an output compound. The following example describes what we want to do:

- The input message contains the following fields:

  ```
  John;Bell;Main St.;Chicago;NOR;200;USA;600;GER;500;$
  ```

  The first four fields (underlined) compose the address. The fields are defined in the input format Address_In. All fields are delimited with a semicolon.

  The remainder of the message contains three sales to customers in three different countries. All fields end with a semicolon suffix. The number of sales varies, that is, the two fields Country and Sales (such as NOR;200;) can repeat several times.

  For this reason, we define another flat input format, SalesList, that contains those two fields. To tell the Formatter when the format stops repeating we add a delimiter at the end, in this case the dollar sign. Following that you could have another format.

  To summarize, the above compound message contains two components:

  - The flat input format Address_In with four mandatory fields
  - The repeatable flat input format SalesList with two mandatory fields

- The output message contains the following fields:

  ```
  John;Bell;Main St.;Chicago;$ 200;$ 600;$ 500
  ```

  This is also a compound, consisting of the format Address_Out and the repeatable format SaleItem. SaleItem contains only one field.

5.9.1 Objects for the Compound Example

We need the objects shown below. Most of them have already been defined for the previous examples.

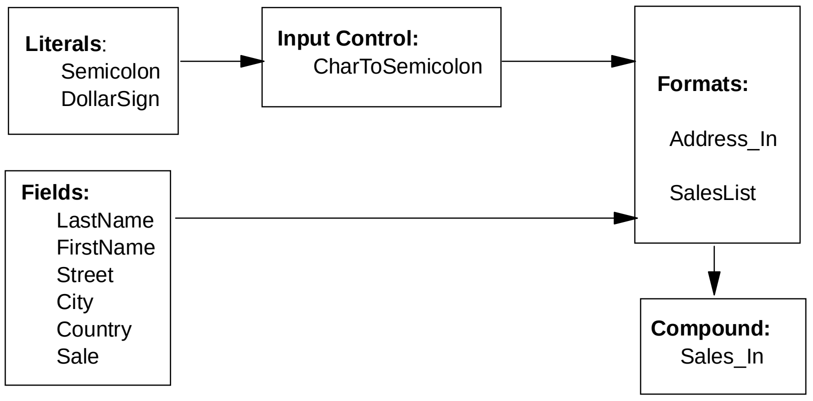


Figure 100. Objects for Input Compound

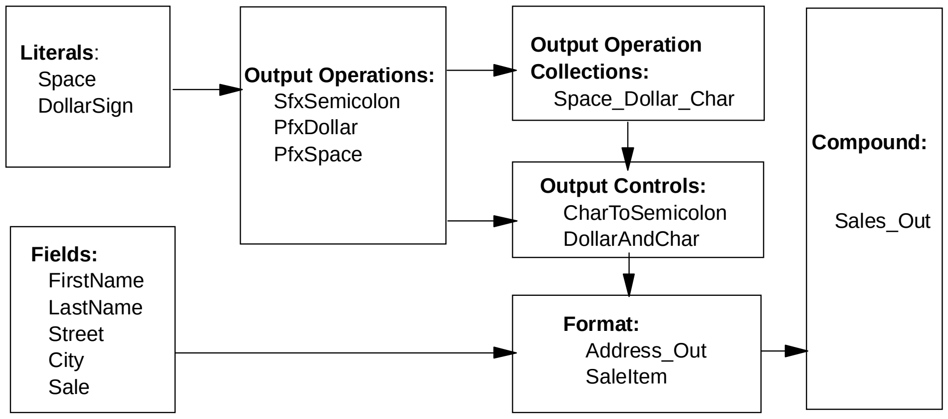


Figure 101. Objects for Output Compound

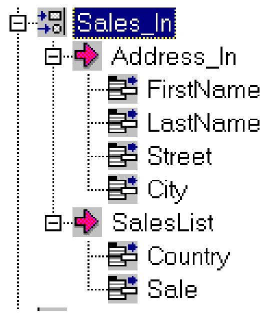
5.9.2 Defining an Input Compound Format

For this example, all literals, fields (with the exception of Sales) and input controls are already in the database. So is the format Address_In with all four fields using CharToSemicolon as input control.

We have to define the format SalesList with the two fields Country and Sale, both using CharToSemicolon as input control.

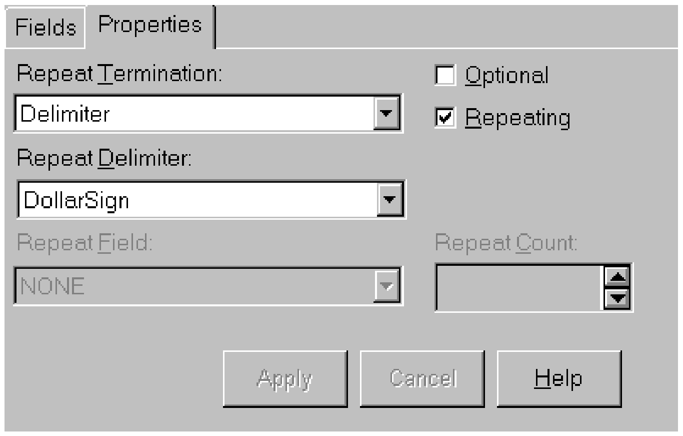
The compound format is created with the following steps:

1. Right-click **Formats** and select **New**, **Compound** and **Input** from the menu.
2. Type the format name, SalesIn, and press Enter.
3. Right-click the format name **SalesIn**, select **Add Format Components** from the menu.
4. From the list, select **Address_In** and **SalesList** and click on **Accept Selection**. This builds the compound as shown below:



```
Sales_In
  Address_In
    FirstName
    LastName
    Street
    City
  SalesList
    Country
    Sale
```

5. Make sure that all six fields use the input control CharToSemicolon.
6. Click **SalesList** and select the Properties window. Here, we define that the format is repeating and that the repeating fields end with a dollar sign. This is shown below.



Fields | Properties

Repeat Termination: Delimiter

Repeat Delimiter: DollarSign

Repeat Field: NONE

Repeat Count:

☐ Optional

☑ Repeating

Note: None of the properties is selected for Address_In.

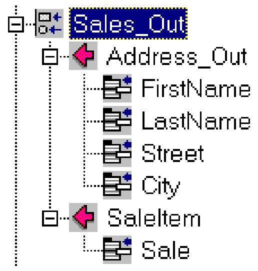
5.9.3 Defining an Output Compound Format

For this example, all fields, literals, output operations, the output operation collection SpaceDollarChar, and the output controls are already in the database.

We have to create the output format SaleItem which contains Sale as the only field. The field uses the output control DollarAndChar which puts the dollar sign and a space in front of the amount.

The compound format is created in the following way:

1. Right-click **Formats** and select **New**, **Compound** and **Output** from the menu.
2. Type the format name, Sales_Out, and press Enter.
3. Right-click the format name **Sales_Out**, select **Add Format Components** from the menu.
4. From the list, select **Address_Out** and **SaleItem** and click **Accept Selection**. This builds the compound as shown below:



5. Make sure that the fields in Address_Out use the output control CharToSemicolon.
6. Click **SaleItem** in the compound tree and make sure that the property **Repeating** is selected.
7. Select the field **Sale** in the format and specify the properties:
   - Output Control Name: DollarAndChar
   - Access Mode: Access using Relative Index
   - Input Field Name: Sale

The two compound formats are now defined and you can use the Visual Tester to verify your definitions.

Chapter 6. Writing Rules

NEON Rules is a component of the MQSI. It depends on the Formatter to parse messages for evaluations. NEON Rules enables you to evaluate a string of data (message) and react to the evaluation results. With the Rules GUI you can define the different components of rules. There are two ways to define rules, with the Rules GUI or with the Management API which you can use to write your own interface to Rules components.

A brief introduction to rules together with a small example is in 3.2, “Working with Rules” on page 39. There you find also information on how to log on to the Rules GUI. In the following sections we provide more detailed (and more challenging) examples. We explain how to use the Rules definitions with the rules processor to reformat and route messages.

6.1 About Rules

First, let us look at the objects we use in context with rules. The objects in the rules hierarchy are shown in Figure 102.

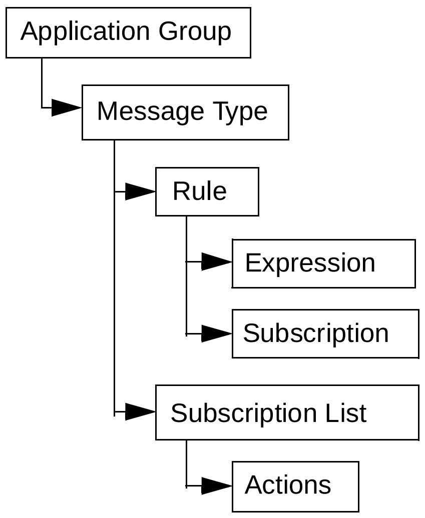


Figure 102. Rules Hierarchy

An application group is a rule set.

A rule set can be business or application oriented. We can define many application groups, however, one instance of the rules daemon (the run-time program) will only execute rules from one application group. We can have several rules daemons executing the same application group or different application groups.

An *application group* is a container which holds information about work to be done by the rules daemon, MQSI's run-time environment. For example, all messages and rules that have to do with payroll would be in one application group. Information regarding the sales department would belong to a different application group, thus requiring a second rules daemon.

An application group can contain one or more *message types*. A message type is an input format that you defined with the Formatter. Each message type may have one or more rules associated with it.

Rules determine what happens to a message, for example, which input format to map to which output format. In fact, from a single input format you can generate several messages, each with a different output format.

Each rule is associated with a *subscription list*, which is a set of actions. An *action* can be one of two, map an input message to an output message (reformat), or put a message to a queue.

Note: Without a put action messages will be lost!

The MQSI Rules Daemon does a few things differently based on subscription boundaries. For example, it starts the next reformat over from the incoming message. Also, you can associate multiple subscriptions with a rule and associate subscriptions to multiple rules.

You define the objects in the rules hierarchy from the top down. Remember, formats are defined from the bottom up. You start with the application group, the most inclusive definition, then you work your way down to the most specific part, the action.

6.2 Techniques for Creating Rules

The following example demonstrates how to use the Rules GUI in the application sample defined in Chapter 5, “Formatting Examples” on page 87. All message types defined in the Formatter are available in the Rules GUI. When using the Formatter, the message type is the same as the input format name.

When you create a new application group you will be presented with a choice of message types. These are input formats you defined with the Formatter.

Important

You have to restart the rules processor after you change a rule and want it to become effective. Rules are cached.

Alternatively, you can send a special control message, reload rule set, to the rules daemon and it will refresh the cache while up and running.

6.2.1 Creating a New Application Group

To create a new application group do the following:

1. Click the **A** in the Rules Toolbar.
2. Type a name for the application group, such as Taxes, and press Enter.

   This will present you with a window as shown in Figure 103. This window tells you what application groups exist. In our case, it is defaultApp which is always there and the one just created.

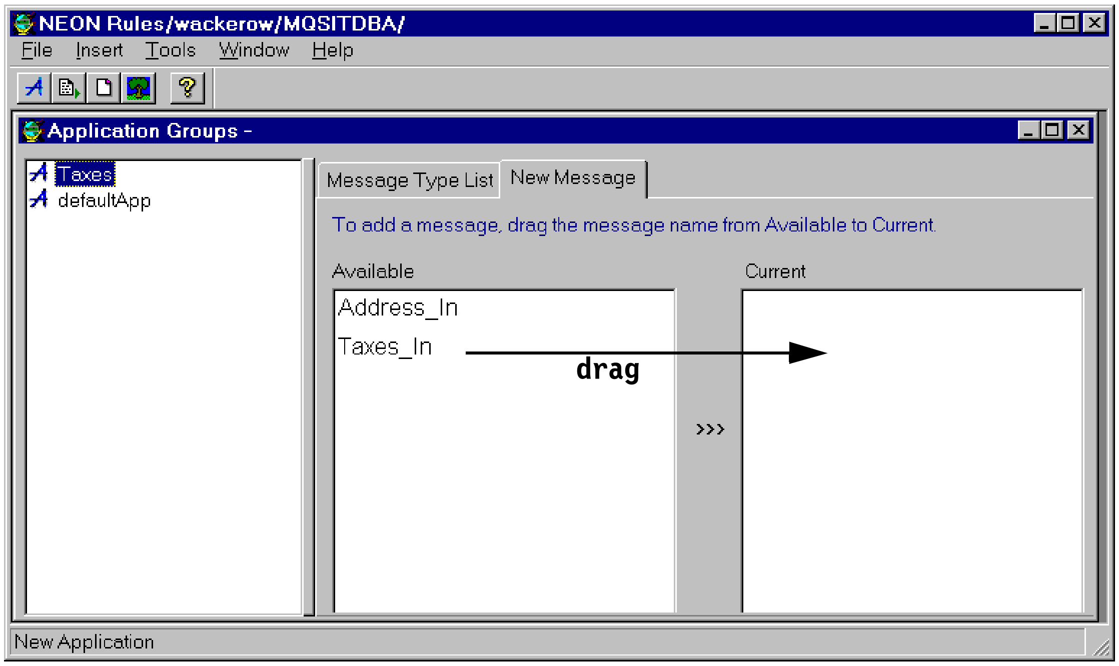

NEON Rules/wackerow/MQSITDBA/
File Insert Tools Window Help
Application Groups -
Taxes
defaultApp
Message Type List
New Message
To add a message, drag the message name from Available to Current.
Available
Current
Address_In
Taxes_In
drag
>>>
New Application

Figure 103. Adding a New Application Group

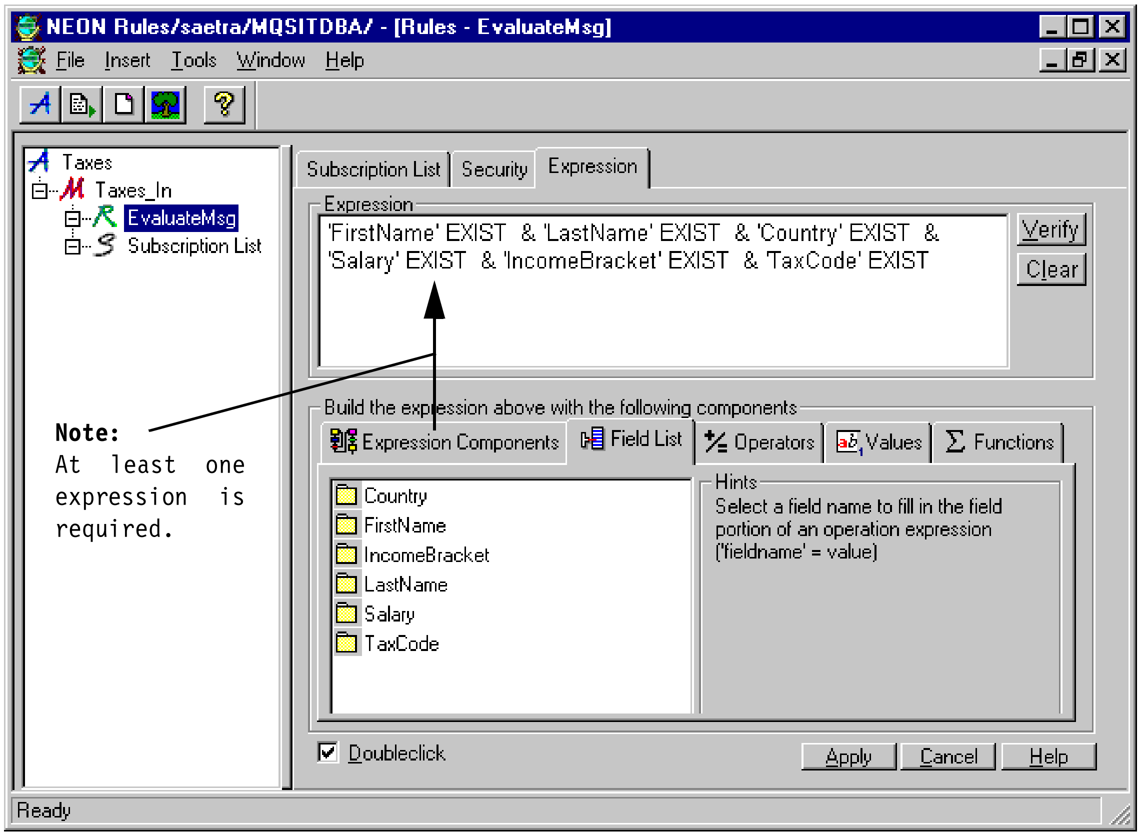
You see also what message types are available to be included in the application group. In this case there are two, Taxes_In and Address_In which were defined in Chapter 5 on page 87.

3. To add a message type to the application group, drag the name, such as Taxes_In from the Available list into the Current list.

Note: To *remove* a message type from the Current list, highlight it and press the Delete key. The name will then appear in the Available list again.

6.2.2 Creating a Rule

Let us create a simple rule that just checks that the input message contains all fields it is supposed to contain. We call it EvaluateMsg. To define a new rule for the message do the following:



'FirstName' EXIST & 'LastName' EXIST & 'Country' EXIST & 'Salary' EXIST & 'IncomeBracket' EXIST & 'TaxCode' EXIST

Select a field name to fill in the field portion of an operation expression ('fieldname' = value)

Note: At least one expression is required.

Figure 104. Rules Window with Simple Expression

1. In the left pane of the window, right-click the message name **Taxes_In**.
2. Select **New**.
3. Type the name of the rule, EvaluateMsg. Now you see a window as shown in Figure 28 on page 42, which is the same window as the one above, however, with the **Expression Components** tab selected.

   This window shows all expressions available to the rules. With the expressions, field list, operators, values, and functions you can build advanced rules to be performed on the message.

   The window shows all fields that have been defined in the Formatter GUI for this message type. You can pick from this list which fields you want to check and/or manipulate.
4. In 3.2.2, “How to Define a Rule” on page 41 is described, in detail, what to click and select to compose a rule as shown in Figure 104.

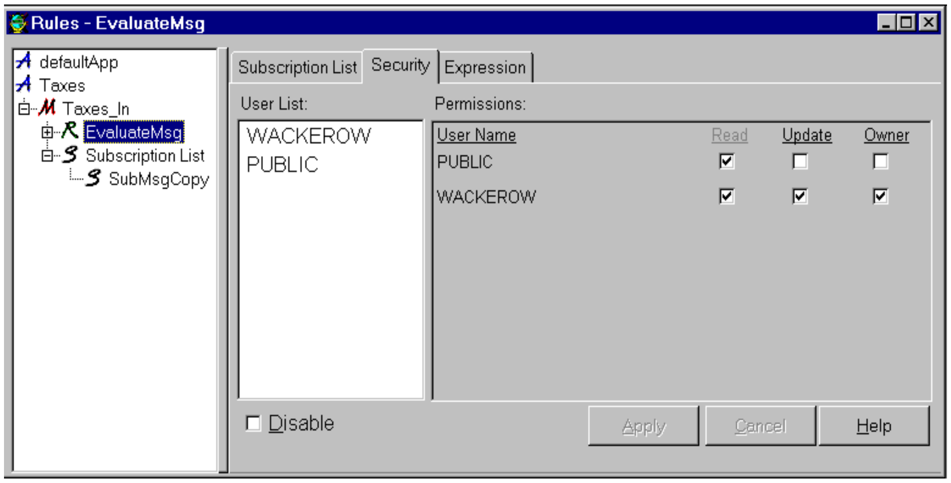
   As you can see, we have built a simple expression where all fields are checked for their existence. It does not matter in what order the fields appear in the list and in the expression window.
5. To activate the expression definitions click the **Apply** button at the bottom of the screen.

Be Aware

To see the subscription list in the window, you may have to collapse the tree displayed in the left pane so that only Application Groups is visible and then expand the tree again. It is important that the subscription list is visible because we have to click it when we define actions later.

The security tab in Figure 105 shows what level of security is applied to accessing rules definitions in the Rules GUI. This has nothing to do with message security. There can be only one owner of a rule, but several users can update it. All users of the GUI can read the rule definitions.

In MQSI Version 1.0, it is not possible to switch off the public read function. If a user who is not the owner of a rule deletes the rule, it will not be physically deleted but a disable flag will be set for it. For more information on security refer to Chapter 8., “Some Comments about Security” on page 163.


Rules - EvaluateMsg

defaultApp
Taxes
Taxes_In
EvaluateMsg
Subscription List
SubMsgCopy

Subscription List | Security | Expression

User List:
WACKEROW
PUBLIC

Permissions:

User Name	Read	Update	Owner
PUBLIC	✓	☐	☐
WACKEROW	✓	✓	✓

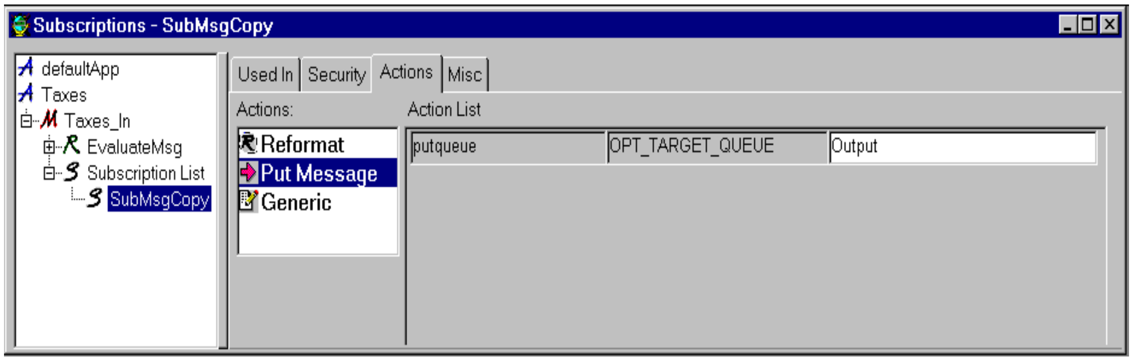
☐ Disable

Apply | Cancel | Help

Figure 105. Rule Security

6.2.3 Creating a Subscription and Action

We create a subscription with one action only, Put Message. That means no reformatting of the message will be done. This action will put the complete original message, not reformatted, to the target queue specified. If a subscription does not include the Put Message action, messages will not be put on any queue and will be lost.


Subscriptions - SubMsgCopy

defaultApp
Taxes
Taxes_In
EvaluateMsg
Subscription List
SubMsgCopy

Used In | Security | Actions | Misc

Actions:
Reformat
Put Message
Generic

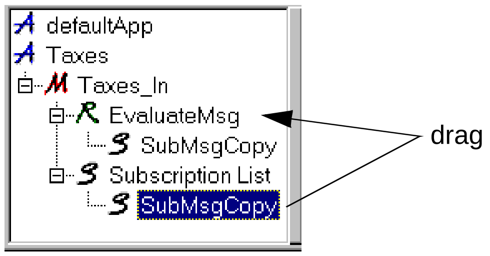
Action List

putqueue	OPT_TARGET_QUEUE	Output

Figure 106. Subscription - Simple Put Action

Figure 106 shows an action that simply puts the input message in the queue Output. You create the subscription SubMsgCopy the following way:

1. Right-click **Subscription List**, select **New** from the menu, enter the name SubMsgCopy and press Enter.
2. Select the **Actions** tab.
3. Highlight **Put Message** and drag it into the Action List.
4. Specify the name of the target queue, here Output.
5. On the left side of the window, select and drag it on top of EvaluateMsg.



Note: If you forget the last step you will get the following error message:

```
-1003 - Rules not configured and/or Operations missing for message
```

All changes to a suspicions, such as adding another action, must be done in the subscription list and not in the entry under the rule name. The rule is just a user, or subscriber, of the definition.

6.2.4 Testing a Rule

You can use one of the tools described in Chapter 4, “Tools That Help with Development” on page 57 to test the rule, or you may use the rules processor. In the latter case you have to change the mpf file you are using, for example, mqsiruleng.mpf, and reference the application group Test and the message group Test_In. Otherwise, you have to include an MQSI header in your message. The sample file mqsiruleng.mpf which comes with the product is in Figure 42 on page 65. The values you have to look at are:

- QueueManagerName = QM1
- InputQueueName   = RulesIn
- NoHitQueueName   = RulesNoHit
- FailureQueueName = RulesFail
- DefaultAppGroup = Taxes
- DefaultMsgType = Taxes_In

Now let us make three tests:

1. All fields are present in the message:

The message is put into the queue Output.

2. One or more fields are missing:

   The message is put into the queue RulesFail and mqsiruleng.log shows this error message:

```
Rules evaluation failed.
Application group   : 'Taxes'
Message type        : 'Taxes_In'
Rules Error #       : -2025
Rules Error Message : 'Formatter failed to parse input message -
check format -- Format -  'Taxes_In', msg length - 21'
```

3. We change the rule and require that only FirstName exists:

   The message is always put in the Output queue, regardless of the number of fields in it.

Note: No formatting takes place.

6.3 Adding Rules and Subscriptions

Let us add another rule to the message group Taxes_In. We want to achieve the following:

- Every message will be put "as is" on the queue Output.
- If the income bracket is over 50% the message will be formatted using TaxesOut and put on the queue Output1.Yes, these messages will end up in two queues. For this we define the rule HighIncBra.

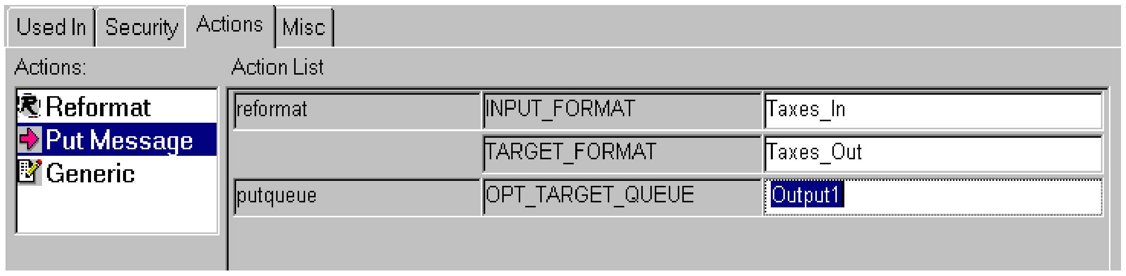
You can add a rule in two ways, create a new one or duplicate an existing one and modify it. The following steps explain how to duplicate and modify the rule EvaluateMsg:

1. Right-click **EvaluateMsg** and select **Duplicate** from the menu.
2. Type the name of the new message, **HighIncBra** and press Enter.
3. You will notice that the subscription is also duplicated. To delete it right-click **SubMsgCopy** under HighIncBra and select **Delete** from the menu. Then confirm the deletion.
4. You will see that the expression has been copied into the new rule. We add now the expression that checks the contents of the field IncomeBracket in the message:

   ```
   & IncomeBracket INT >= 51
   ```

5. Verify the new expression and click **Apply**.
6. Create a new subscription with the name SubIncBra as shown in Figure 107 and click **Apply**.
7. *Drag* the subscription over the rule HighIncBra and drop it.

Note: If one of the input fields is missing the message will be placed in the RulesFail queue.



reformat	INPUT_FORMAT	Taxes_In
	TARGET_FORMAT	Taxes_Out
putqueue	OPT_TARGET_QUEUE	Output1

Figure 107. Using Two Rules and Reformatting

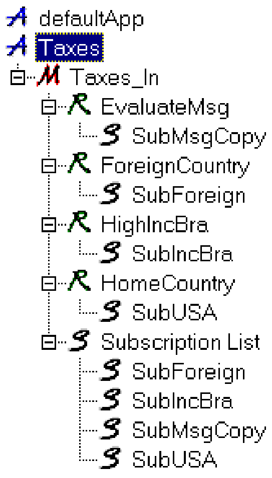
6.4 Multiple Rules

You can write as many rules as you like. Each must be associated with at least one subscription. Subscription actions are processed sequentially. All rules will be evaluated. If one of the required input fields is missing, the message will be put on the RulesFail queue. If the input is as expected but the message does not match any of the criteria defined in the rules, it is placed on the RulesNoHit queue. It is the user’s responsibility to take care of messages in those queues.

Now let us add two more rules to the two defined in the previous examples:

- With the rule ForeignCountry, we extract all messages that don’t have the country code USA and put them into the queue WorldQ.
- Since the MQSI does not know an "else", we need the rule HomeCountry to place all messages with the country code USA in the USAQ.

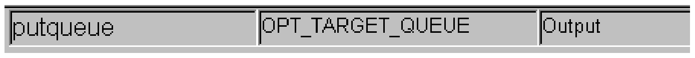
As you can see in the figure below, there are four rules for the message Taxes_In. They are evaluated in alphabetical order:



1. EvaluateMsg (as described in 6.2.2, “Creating a Rule” on page 138) checks if all input fields are present. If not, the message is put on the queue RulesFail. The expressions in the rule read as follows:

   ```
   FirstName EXIST & LastName EXIST & Salary EXIST &
   IncomeBracket EXIST & Country EXIST
   ```

   Subscription SubMsgCopy contains one action:

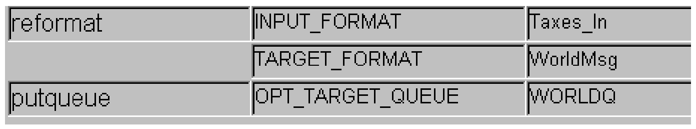


putqueue	OPT_TARGET_QUEUE	Output

2. ForeignCountry checks if the field Country in the input format does not contain the string USA. If this is the case, the message is formatted using the output format WorldMsg. The expression is:

   ```
   FirstName EXIST & LastName EXIST & Salary EXIST &
   IncomeBracket EXIST & Country EXIST & Country STRING <> USA
   ```

   Subscription SubForeign contains the following actions:

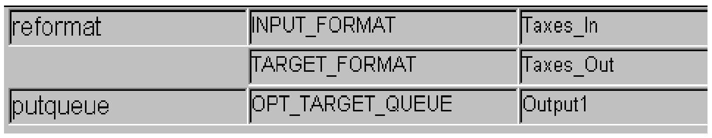


reformat	INPUT_FORMAT	Taxes_In
	TARGET_FORMAT	WorldMsg
putqueue	OPT_TARGET_QUEUE	WORLDQ

3. HighIncBra has been defined in 6.3, “Adding Rules and Subscriptions” on page 142. If the expression below is true the messages will be reformatted using TaxesOut and put on the queue Output.

```
FirstName EXIST & LastName EXIST & Salary EXIST &
IncomeBracket EXIST & Country EXIST & IcomeBracket INT >= 51
```

Subscription SubIncBra contains the following actions:

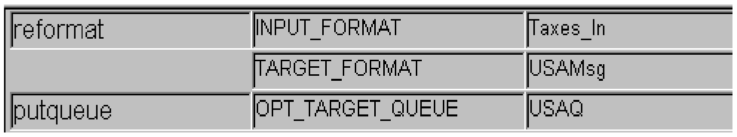


reformat	INPUT_FORMAT	Taxes_In
	TARGET_FORMAT	Taxes_Out
putqueue	OPT_TARGET_QUEUE	Output1

4. HomeCountry reformats the message using the output format USAMsg and writes it to the queue USAQ, provided the following expression is true:

```
FirstName EXIST & LastName EXIST & Salary EXIST &
IncomeBracket EXIST & Country EXIST & Country STRING => USA
```

Subscription SubUSA contains the following actions:



reformat	INPUT_FORMAT	Taxes_In
	TARGET_FORMAT	USAMsg
putqueue	OPT_TARGET_QUEUE	USAQ

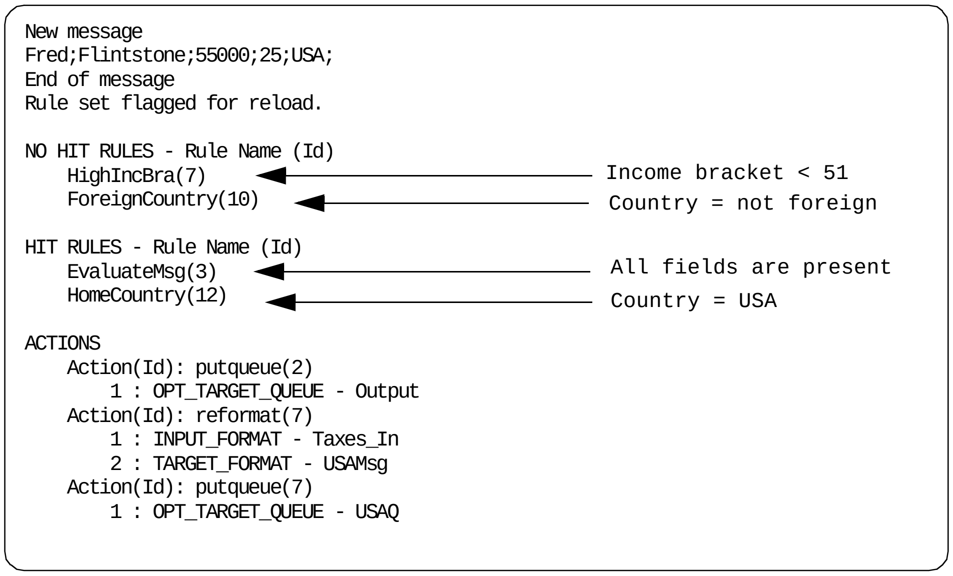
To get this application running you also have to define the following:

- Define the queues WORLDQ and USAQ. If you fail to do this, results may be unpredictable.
- Create the output formats WorldMsg and USAmsg, add some fields to them and don’t forget to assign each field an output control.

Figure 108 on page 146 shows the output of the ruletest tool. It is an example where two rules "hit" and two don’t.

Summary: The example above shows that you can:

- Make decisions based on the contents of a field
- Route messages to different queues (applications) based on that decision



```
New message
Fred;Flintstone;55000;25;USA;
End of message
Rule set flagged for reload.

NO HIT RULES - Rule Name (Id)
    HighIncBra(7)
    ForeignCountry(10)

HIT RULES - Rule Name (Id)
    EvaluateMsg(3)
    HomeCountry(12)

ACTIONS
    Action(Id): putqueue(2)
        1 : OPT_TARGET_QUEUE - Output
    Action(Id): reformat(7)
        1 : INPUT_FORMAT - Taxes_In
        2 : TARGET_FORMAT - USAMsg
    Action(Id): putqueue(7)
        1 : OPT_TARGET_QUEUE - USAQ
```

Figure 108. Test of Multiple Rules

6.5 Changing MQMD Fields

With the Rules definitions it is possible to change some of the fields in the message descriptor (MQMD header). The changes are done in Actions under the Subscription List, and they are part of the putqueue action.

However, there are a few things to consider before attempting to change fields in the MQMD header:

- In MQSI Version 1.0 all messages will be put to the target queue with the message type datagram.

  What does this mean?

  MQSeries has a field in the MQMD called MsgType. This field is used to convey information about what type of message this is:

Datagram	No response expected
Request	A request message that should invoke a reply.
Reply	A response to a request message
Report	A message usually generated in response to a report request

If the MQSI received a request message with the fields ReplyToQMgr and ReplyToQ set, then this information will not be given to the target application. It will be lost.

It also means that MQSI cannot participate in a conversational request/reply application architecture (unless the application programs are modified).

- It is planned that request/reply messages will be supported in Version 1.1.

6.5.1 The MQSI Header

MQSI uses a message header called MQSI Message Header. In the MQSI documentation for Version 1.0, this header seems to be unique to MQSI. However, this is the same header format as MQSeries Publish/Subscribe uses. The header is called MQRFH - Rules and Formatting Header. The MQPS/MQSI header has two parts, the MQRFH (in many places in the NEON documentation spelled MQHRF) and the Options (also called NEON Options). The options part is named *NameValueString* in Table 7.

The MQRFH header precedes the user data in an MQSeries message. To tell the MQSI Rules Daemon that the message is in MQSI format, the MQFMT field in the MQMD should be set to MQRFH or MQFMT_RF_HEADER. This could be done either in the originating application or in another MQSI Rules Daemon instance. The MQRFH structure is shown in Table 7.

Table 7. Fields in the MQRFH

Field	Description
StrucId	Structure identifier for the rules and formatting header
Version	Structure version number
StrucLength	Total length of MQRFH including string containing name/value pairs
Encoding	Numeric encoding
CodedCharSetId	Coded character set identifier. **Note:** When a message is put, this field must be set to the non-zero value that specifies the character set of the user data. If this is not done, it will not be possible to convert the message using the MQGMO_CONVERT option when the message is retrieved.
Format	Format name of the user data that follows the string containing the name/value pairs

Field	Description
Flags	Flags
NameValueString	String containing name/value pairs. This is a variable length character string.

The MQSI Options buffer (NameValueString) consists of a collection of space-delimited tag/value pairs. It contains information such as:

- Application Group
- Message Type

6.5.2 About Conversion

If the target application accepts the MQSI format (with MQSI header), then it cannot use the standard method on MQGET with convert as a way of doing codepage conversion. The codepage conversion has to be done on the sending side, that is, the MQSI Rules Daemon must convert to the codepage of the target application before the MQPUT is done. This may lead to unknown problems because of unsupported codepages on the MQSI platform, for example, European languages use many different codepages to support national characters on NT, UNIX and mainframe platforms.

Usually, headers that begin MQH are converted by the queue manager because they are defined in MQSeries.

The MQRFH is now converted on the distributed platforms. This has been added by CSD5 to support Publish/Subscribe which also uses the RFH header.

Unfortunately, RFH conversion is not supported on OS/390 yet. The work-around is to disable convert on the channel, or create a simple conversion exit that will convert the RFH header.

For MQSI on S/390 there is an exit installed to handle conversion of the RFH when the daemon reads a message off the input queue. With MQSI on distributed platforms the conversion of the RFH is done by the broker after the message has been read by the daemon, before any further processing is performed. The reason for these two approaches is that on the distributed platforms MQSeries will not call an exit for a structure that begins MQH, while on S/390 it will.

6.5.3 Get and Put Options

For MQGET, the MQSI Rules Daemon uses these options:

- Syncpoint
- Fail_if_quiescing
- Complete_msg
- Convert

The most important option here is the convert, it can use either yes or no, depending on whether the MQSI output format is defined to convert the data or not. Use convert *no* if the MQSI output format is defined to convert the data. Use convert *yes* if the output format is *not* defined to convert the data.

For MQPUT, it uses these options:

- Syncpoint
- Fail_if_quiescing
- Pass_all_context
- Alternate_user_authority

The alternate user authority flag is set with the variable CredentialsEnabled in the MQSIruleng.mpf file in the \mqi\bin\ directory.

What happens when there are two legacy applications that exchange messages?

Because none of them uses the MQSI format, the MQSI Rules Daemon will not find which application group and message type it should use to parse the message. The MQSIruleng (daemon) will therefore use the defaults that are set in the configuration file used by this instance of the rules processor, for instance the MQSIruleng.mpf file.

The putqueue actions that can be changed or overridden from the original MQSeries message are:

- MQS_FORMAT
  - Can change the MQMD FORMAT for the output message.
  - Can change a message from a standard to an MQSI message that will be processed by another MQSI Rules Daemon.
  - Can also make an MQSI message into a legacy message.
- MQS_PERSIST
  - If defined, can change the persistence of a message from persistent to non-persistent, or vice versa.

    - If undefined, the value in the input message’s MQMD will be the value in the output message’s MQMD header.
- MQS_EXPIRY
    - Can map the incoming value over to the output MQMD header as is.
    - Can clear this value and set it to no expiration.
    - Cannot change the expiration.
- MQS_PROPAGATE

  Reconstruct the MQSI header along with the MQMD header for the output message.
    - Should be set to propagate if there is another MQSI Rules Daemon downstream.
    - Should be set to no_propagate if the target is a legacy application.

The MQSI Rules Daemon strips off the MQMD and MQSI headers from the message before it parses the data. Then, based on the format and propagate definitions above, reconstructs either the MQMD header alone, or the MQMD and MQSI headers for the output message. See the *MQSeries Integrator Application Development Guide* for more information.

Note: A standard message can be converted into an MQSI-aware message. However, this is not automatic. You set the MQMD.Format field to "MQHRF   ', then build the RFH header and options segment in the output message.

6.6 Management API

The management APIs provide management definition ability via code rather than through GUIs. For the Formatter, the management API can manage elements such as:

- Read and create formats
- Change formats
- Return format information

The Rules management API can manage elements including:

- Rules
- Subscriptions
- Actions

The API commands are add, read, update, and delete.

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

For more information refer to the following manuals (available in PDF format):

- *MQSeries Integrator Programming Reference for NEON Rules*
- *MQSeries Integrator Programming Reference for NEON Formatter*

Chapter 7. MQSI As an Intelligent Router

In this chapter, we develop a more complex example of using the Formatter GUI and Rules GUI to do intelligent routing of messages using the content of the message. The application is a small "warehouse and manufacturing" process that produces fruit salad. The application architecture with data and message flow is shown in Figure 109.

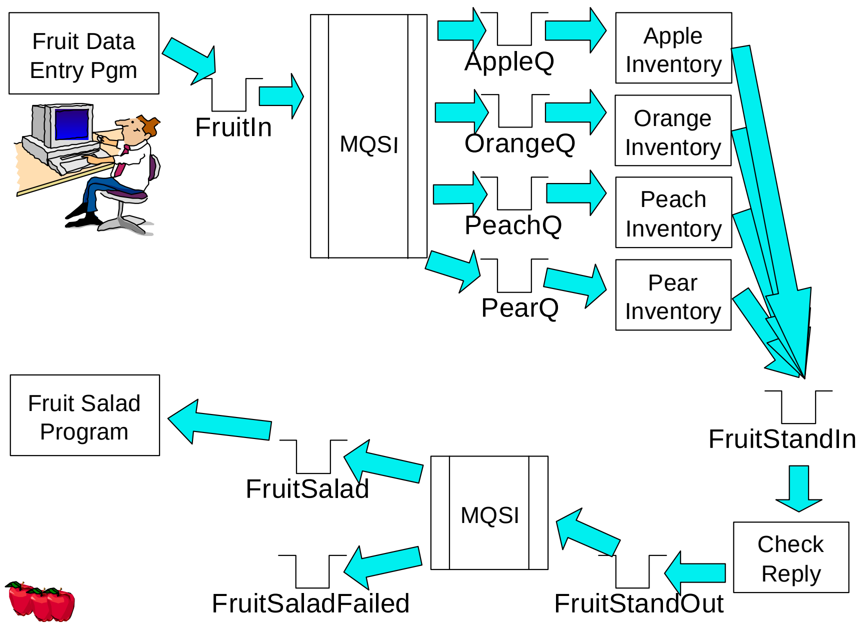


Figure 109. Fruit Salad Example - Overview

The purpose of this process is to produce fruit salad that can contain four different fruits, apples, oranges, peaches and pears. The customer phones the person at the data entry desk and places his order. The person interfacing the data entry program enters how many pieces of fruit from each kind are needed to make the fruit salad. The data entry program then builds an MQSeries message as shown in Table 8 on page 154 and puts it in the queue FruitIn. FruitIn is the input queue for an instance of the rules processor.

The MQSI gets the message and creates as many new messages as there are fruits in the order. Each message is sent to a warehouse application where the inventory is checked. If there are enough apples, for example, a field is appended to the message indicating whether the order can be filled. The messages are then sent to the Check Reply program, via the queue FruitStandIn.

The Check Reply program, written in C, performs a message broker function. It collects all messages that belong together. The MQSI cannot do message re-composition. The MQSI is an intelligent router with strong reformatting capabilities. The Check Reply program waits until all messages have arrived from the warehouse. Then it composes one message and puts it in the queue FruitStandOut.

Another instance of the rules processor is processing these messages and deciding whether the fruit salad can be made. The messages are placed either in the FruitSalad queue for production or in FruitSaladFailed for back order.

7.1 Data Entry and Message Routing

The instance of the rules processor that processes the messages from the data entry program expects the messages in this format:

Table 8. Fruit Salad Example - Message Format for FruitIn Queue

MQMD	Data
	2;apple:5;orange:0;peach:0;pear:7;

The message consists of the MQSeries message header and data, but does not contain an MQSI header with application group and (MQSI) message type. In the example above, five apples and seven pears are requested.All fields shown in Table 8 must be included in the message. The reason for this is that MQSI does not provide a standard way of handling repeating fields in a way suitable for this example. However, we could use a user exit to do this.

The first field in the message is a counter telling how many or different fruit types are in this order (message). The next (repeating) two fields describe the kind of fruit and the quantity needed.

Note: We use a colon as a separator between fruit name and the quantity, and a semicolon elsewhere.

The definitions for the various fields are shown in Figure 110. You recognize the same fields and their names in the Rules GUI in Figure 112 on page 157.

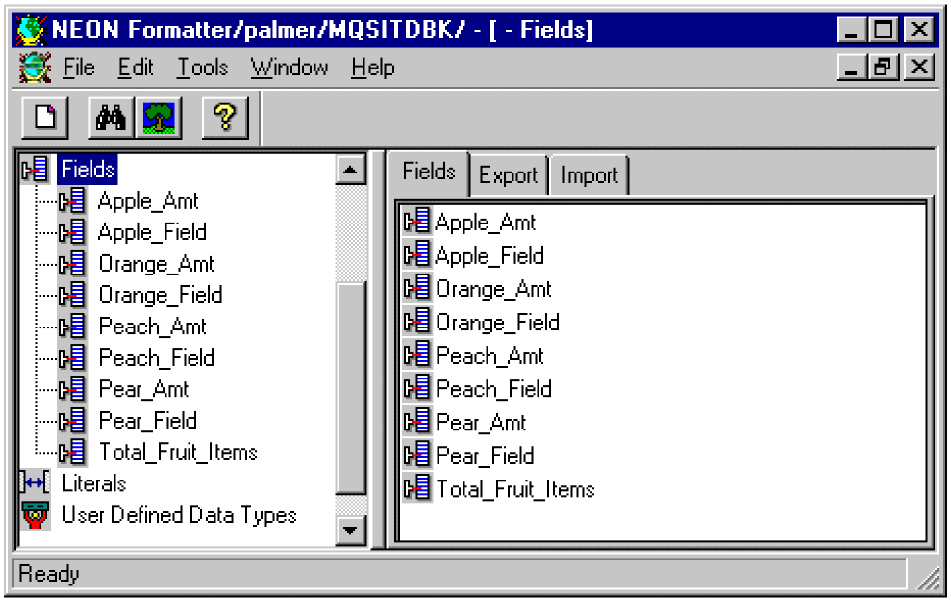


Figure 110. Fruit Salad Example - Fields

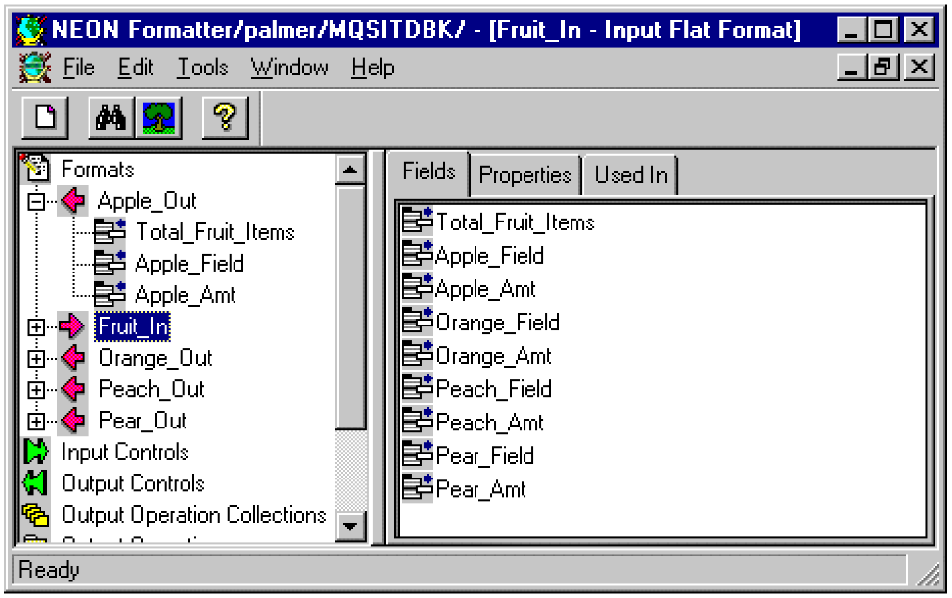


Figure 111. Fruit Salad Example - Formats

Figure 111 on page 155 shows the input format and the four output formats needed for this example. The input format Fruit_In applies to messages coming from the data entry program. The output formats are for the messages to be sent to the warehouse program.

The reason for having four output formats and an input format that must contain all four items is that the Rules Daemon lacks intelligence in handling compounds and repeatable fields in an input message. The rules processor will find the first occurrence of a repeatable field but not the following ones. It may be possible for a user exit program to handle this, but we consider that outside the scope of this book.

The MQSI Rules Daemon parses the message and by checking the contents of each field, determines what actions to perform. For instance, if the apple quantity field contains a number higher than zero, a message like the one in Table 9 will be put on the AppleQ and sent to the apple warehouse where the inventory application will check whether there are enough apples in stock to satisfy the request. If the apple amount field contains a zero, no message will be sent.

Table 9. Fruit Salad Example - Reformatted Message for AppleQ

MQMD	Data
	1;apple:5;

The Rules definitions required to do this are shown in Figure 112 and Figure 113 on page 157.

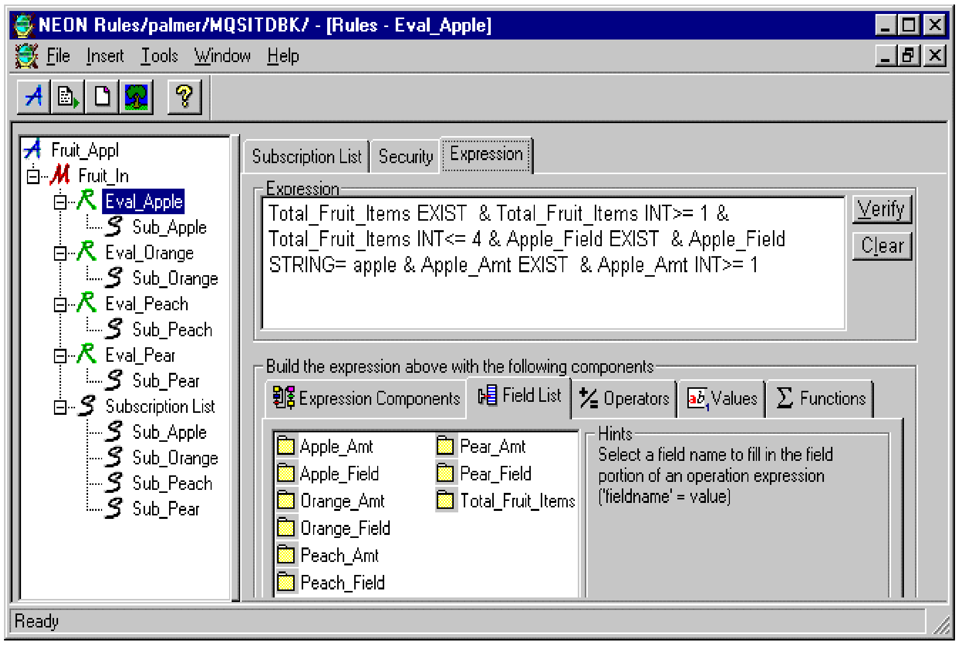
In Figure 112 on page 157, you see the expression the Rules Daemon uses to validate the input message. It may seem like overkill to do all this testing, but it is done to ensure that our data is correct.

7.2 Warehouse Application

The warehouse application decides whether a request for fruit can be satisfied or not. It builds a message as shown in Table 10. It contains a "y" if the request can be satisfied; otherwise an "n" will be appended. The message is then put on the FruitStandOut queue for processing by the Check Reply program.

Table 10. Fruit Salad Example - Message Format for FruitStandIn Queue

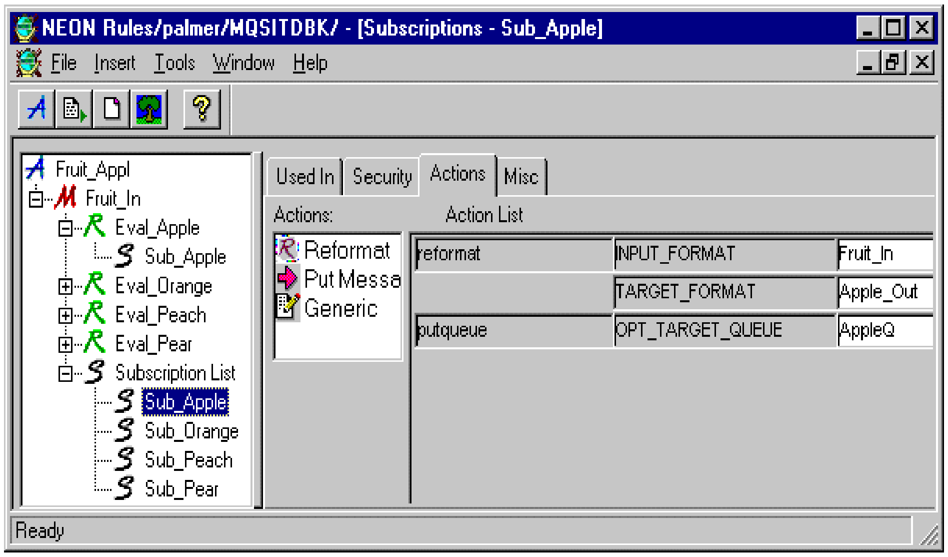
MQMD	Data
	1;apple:5;y;



Total_Fruit_Items EXIST & Total_Fruit_Items INT>= 1 &
Total_Fruit_Items INT<= 4 & Apple_Field EXIST & Apple_Field
STRING= apple & Apple_Amt EXIST & Apple_Amt INT>= 1

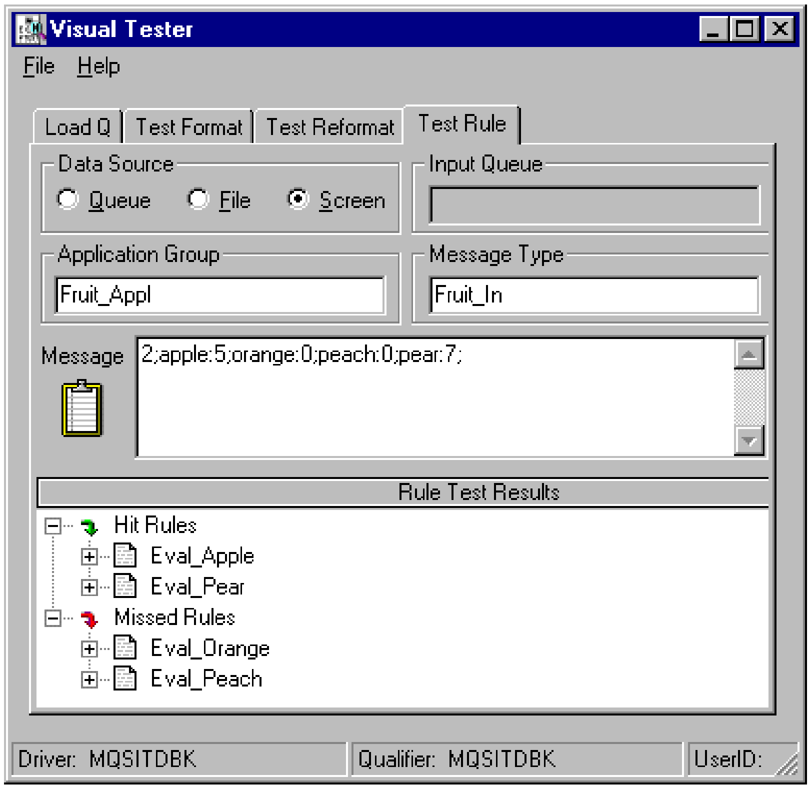
Select a field name to fill in the field portion of an operation expression ('fieldname' = value)

Figure 112. Fruit Salad Example - Rule Expression



reformat	INPUT_FORMAT	Fruit_In
	TARGET_FORMAT	Apple_Out
putqueue	OPT_TARGET_QUEUE	AppleQ

Figure 113. Fruit Salad Example - Subscription



Data Source: Queue File [✓ Screen]

Input Queue:

Application Group: Fruit_Appl

Message Type: Fruit_In

Message: 2;apple:5;orange:0;peach:0;pear:7;

Rule Test Results

- Hit Rules
  - Eval_Apple
  - Eval_Pear
- Missed Rules
  - Eval_Orange
  - Eval_Peach

Driver: MQSITDBK | Qualifier: MQSITDBK | UserID:

Figure 114. Fruit Salad Example - Rule Test

7.3 Re-Composition of the Message

The Check Reply program is a plain C program that re-assembles one single message. The code is in Appendix A., “Check Reply C Program” on page 167. The program first browses the message in the FruitStandIn queue to determine how many reply messages to expect. It will find that information as the first field in the message.

When the number of reply messages is known, it executes an MQINQ call to find the current depth of the queue. If the current depth equals the number of expected reply messages, it will MQGET all messages and build one message containing all replies from the warehouse.

The reasons we choose to do it this way are:

- MQSI Version 1.0 does not support request/reply type messages.

- MQSI cannot combine several messages or their contents into one message.
- To show the use of MQSI in more than one place in an application.

The format for the messages on the FruitStandOut queue is shown in Table 11. As we can see, this program just appends the reply messages received from the warehouse, one after another.

Table 11. Fruit Salad Example - Message Format for FruitStandOut Queue

MQMD	Data
	2;apple:5;y;2;pear:7;y;

This newly built message will then be put to queue FruitStandOut for processing by another occurrence of the MQSI.

7.4 Route Depending on Message Contents

The Check Reply program leaves it to the MQSI to decide whether the salad can be produced. Another instance of the rules processor reformats the message into a proper format for the final fruit salad program and puts it either in the FruitSalad queue for production or in the FruitSaladFail queue.

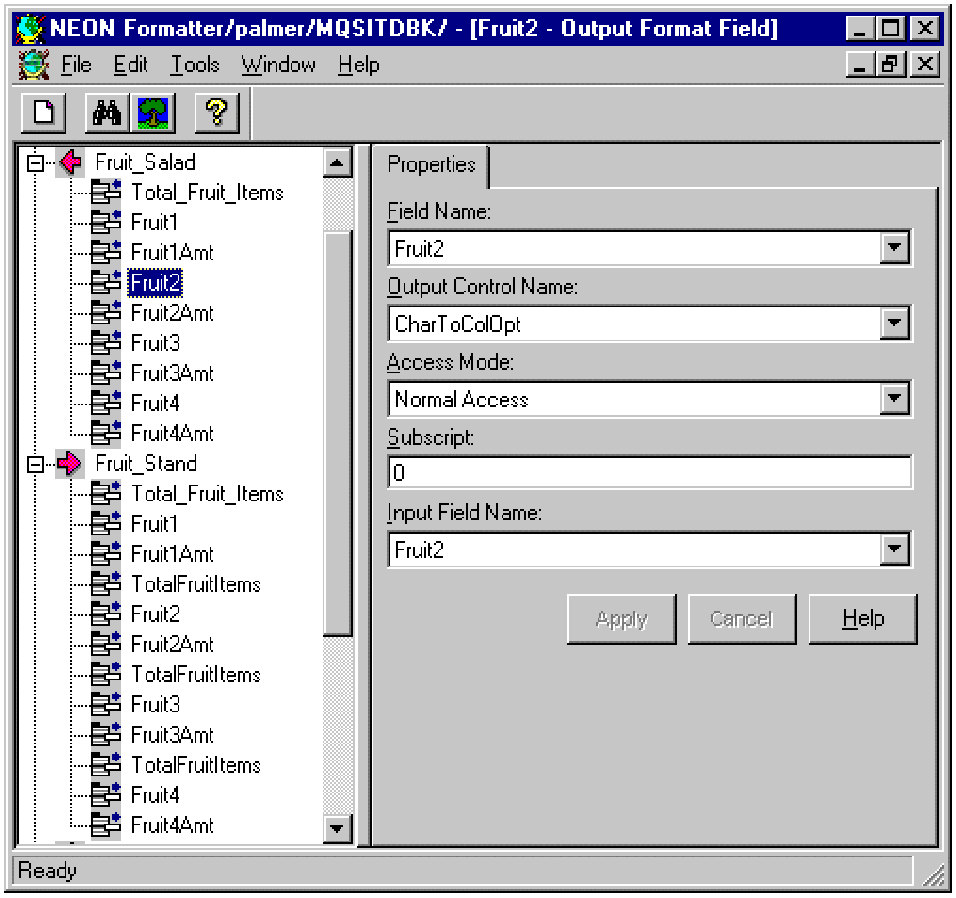
The format required by the final fruit salad program is shown in Table 12, and, as we can see, the MQSI strips off the "y" or "n" for each item. The first field is the number of fruit selections.

Table 12. Fruit Salad Example - Message Format for FruitSalad Queue

MQMD	Data
	2;apple:5;pear:7;

The new formats definitions used for this instance of MQSI are shown in Figure 115 on page 160. Because we do not know how many fruit fields there will be in the message and the sequence they will be in, we have defined more generic names for the fields. In addition, the first three fields, Total_Fruit_Items, Fruit1, and Fruit1Amt are defined as mandatory fields. The remaining fields may or may not be there so they are defined as optional using an Input or Output Control Name with optional fields. By doing it this way we can more easily handle a message containing from one to four fruit types with fewer definitions. We can also look at this as a different way of handling repeating fields.

Note also that in the input format Fruit_Stand, we have the field TotalFruitItems defined as optional for two reasons; the first reason is because we do not know if it will be there, the second reason is because we can use it in the rules to validate the message.



Field Name: Fruit2

Output Control Name: CharToColOpt

Access Mode: Normal Access

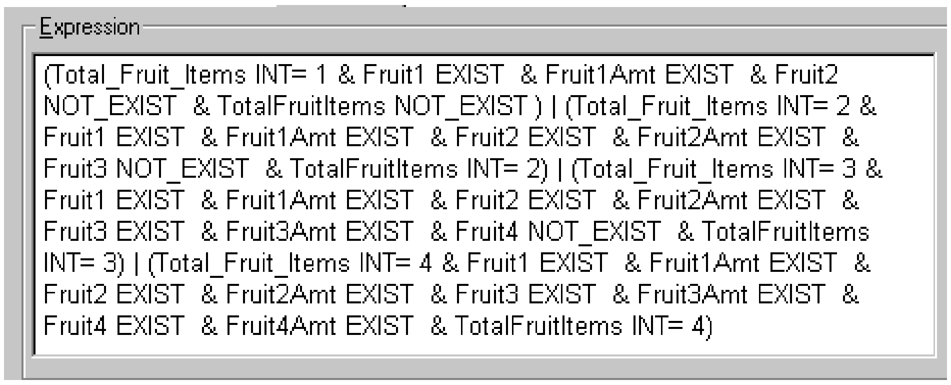
Subscript: 0

Input Field Name: Fruit2

Figure 115. Fruit Salad Example - Fields for Reply Messages

The new rules definitions used for the second instance of MQSI are shown in Figure 116 on page 161. Instead of making one rule for each of the possible formats of the message we have made one rule that covers the four scenarios, the FruitReply rule. This might seem to complicate the rule, but as a matter of fact, the rules definition and administration is simpler because of fewer rules.

The logic of the rule is that all arguments for one check are grouped together between a pair of parentheses, and this expression is ORed with another grouped argument. The subscription SubToSalad used by this rule is not shown here. It uses the input format Fruit_Stand, the target format Fruit_Salad, and the target queue FruitSalad.



```
(Total_Fruit_Items INT= 1 & Fruit1 EXIST  & Fruit1Amt EXIST  & Fruit2
NOT_EXIST  & TotalFruitItems NOT_EXIST ) | (Total_Fruit_Items INT= 2 &
Fruit1 EXIST  & Fruit1Amt EXIST  & Fruit2 EXIST  & Fruit2Amt EXIST  &
Fruit3 NOT_EXIST  & TotalFruitItems INT= 2) | (Total_Fruit_Items INT= 3 &
Fruit1 EXIST  & Fruit1Amt EXIST  & Fruit2 EXIST  & Fruit2Amt EXIST  &
Fruit3 EXIST  & Fruit3Amt EXIST  & Fruit4 NOT_EXIST  & TotalFruitItems
INT= 3) | (Total_Fruit_Items INT= 4 & Fruit1 EXIST  & Fruit1Amt EXIST  &
Fruit2 EXIST  & Fruit2Amt EXIST  & Fruit3 EXIST  & Fruit3Amt EXIST  &
Fruit4 EXIST  & Fruit4Amt EXIST  & TotalFruitItems INT= 4)
```

Figure 116. Fruit Salad Example - Rule Expression for Second MQSI Instance

We can also use the Rules Daemon to send messages that do not match a rule to a different queue from the default. To do this, we have to create a rule that will match all other formats of the message than our main rule. We have not done that in our example; however, that should be easy to accomplish.

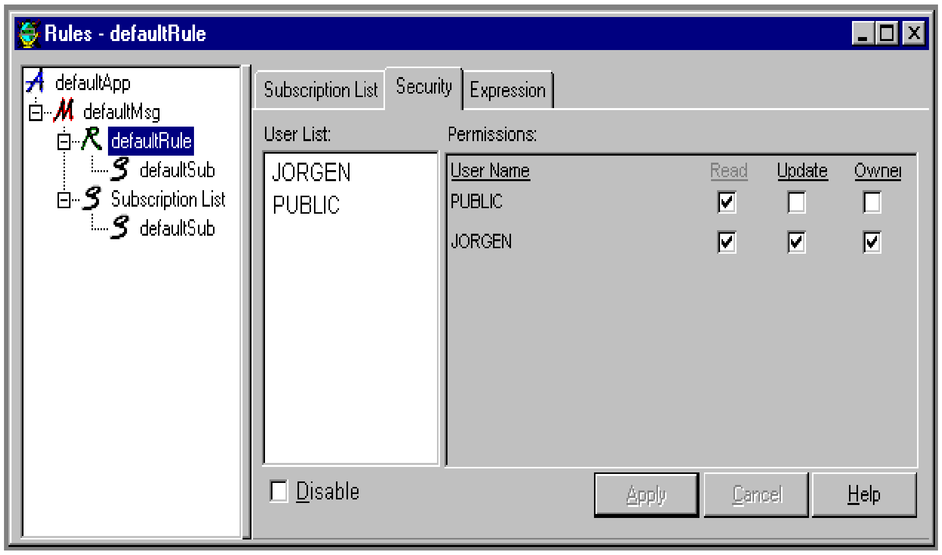
Chapter 8. Some Comments about Security

Security in MQSeries Integrator exists only within the Rules database. For security outside of this area it relies on the database security of the underlying database management system (DBMS) being used, such as DB2, SQL Server, SYBASE, Oracle.

All users automatically have read-only access to all rules within the MQSeries Integrator Rules database. Read access cannot be removed. This privilege is granted to the user group PUBLIC of which all users are a member.

Individual users who create their own rules and subscriptions immediately own and have read and update privileges to objects they create. In addition to this, the owner of a rule can pass ownership to another user as well as create and update the PUBLIC user group's permissions and their own.

Only one user can be the owner of a rule or subscription at any one time.



User List:

JORGEN
PUBLIC

Permissions:

User Name	Read	Update	Owner
PUBLIC	☑	☐	☐
JORGEN	☑	☑	☑

Figure 117. Security for a Rule

All security changes can be made from the Security tab in the Rules GUI. This is shown in Figure 117. JORGEN is the owner of the rule and he alone can update it while all members of the PUBLIC group have read access.

In addition to the security feature in the Rules GUI, the MQSI provides another tool, NNDBARuleOwnership. This utility is designed for system administration. It allows the Rules administrator to change the ownership of multiple rules or subscriptions at one time. For more details and examples of how this tool is used refer to the *MQSeries Integrator System Management Guide*, available in PDF format.

8.1 Adding a Database User

Using DB2 as the database you would need to carry out the following steps to add a user to the Rules database:

1. Connect to the database as the database owner:

   In the DB2 Command Window (DB2CMD) type:

   ```
   db2 connect to <databasename> user <userabc> using <passxyz>
   ```

2. Change the script 401_aliases.sql replacing every occurrence of <new user> with the name of the new user ID and every occurrence of <database owner> with the ID of the database owner.
3. Run this script which will create an alias for the owner of the tables using the syntax below:

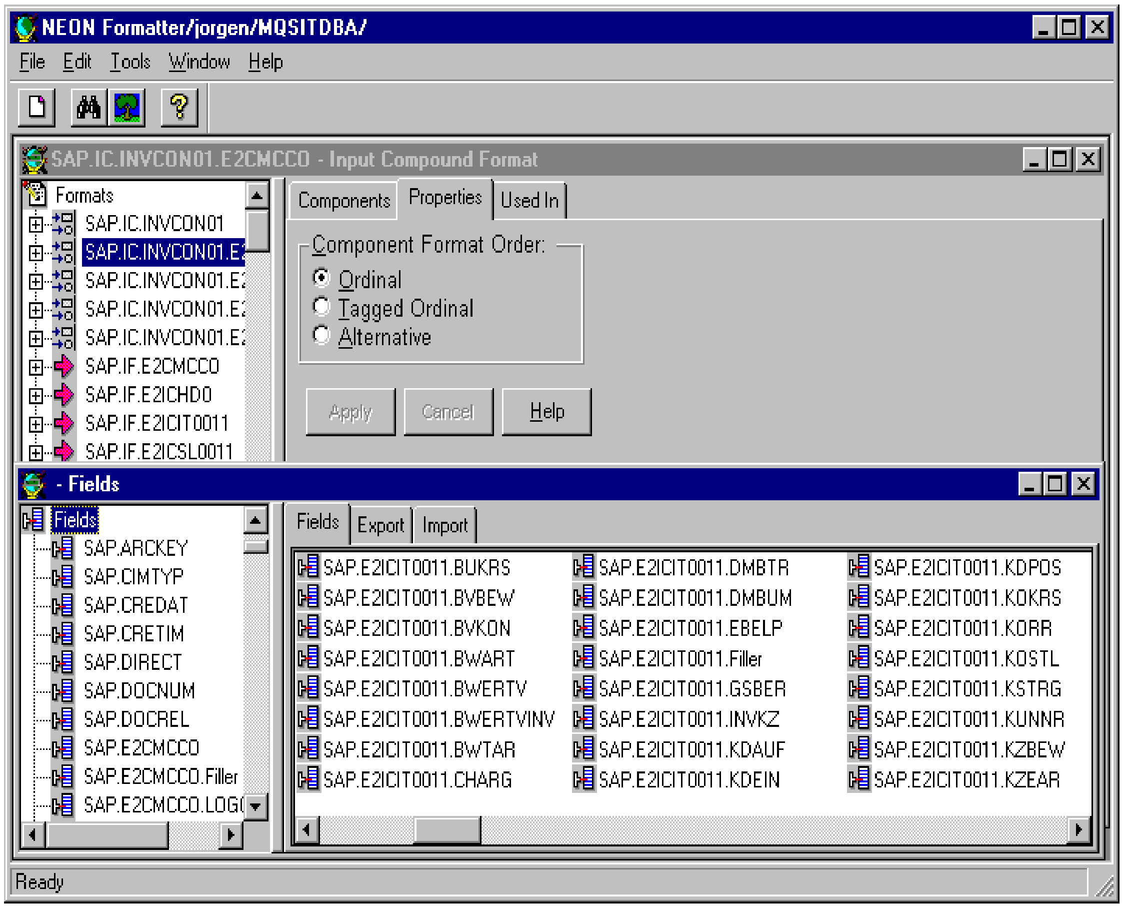
   ```
   db2 -f 401_aliases.sql -l 401_aliases.log -t
   ```

4. If it has not already been done then create the user ID using User Manager under Administrative Tools.
5. Add the user to the database using the DB2 Control Center. Expand the structure under the database name and right-click on **DB User** under User and Group Objects and select **Add**.
6. Finally, log on to the Rules GUI as the database owner and you will see the new user appear in the User List. To add the new user to the Permissions list drag it across under the User Name column.

Chapter 9. The MQSeries SAP Link

The IBM MQSeries Link for SAP R/3 allows us to both put and get data into and out of SAP R/3 from and to other data sources and legacy applications. It consists of three parts:

- MQSeries
- MQSeries Integrator
- The SAP Integration Libraries



Component Format Order: [✓ Ordinal] Tagged Ordinal Alternative

Figure 118. Sample SAP IDoc Loaded in Formatter GUI

The MQSeries SAP Link uses ALE to load SAP IDoc metadata into the MQSeries Integrator Formatter. This means that it can handle bidirectional IDoc formats.

A library of IDocs is provided and additional or customized IDocs can be loaded dynamically into the Formatter via the IDoc loader.

The IDoc loader is a tool with a GUI interface used to load IDocs from a specific SAP R/3 instance into the MQSeries Integrator Formatter. Figure 118 on page 165 shows an IDoc imported from SAP R/3 into the Formatter.

Formats may be taken either from the SAP Integration Libraries provided or re-created dynamically using the SAP Integration Libraries IDoc Loader and the Envelope Loader GUI tools. This eliminates the costly time-consuming process of creating or amending format information for all fields within an inbound or outbound IDoc.

The only remaining task is to map the incoming or outgoing application or legacy database data formats to the corresponding IDoc field formats.To aid in this process a GUI Formatter Mapping Tool is also provided to formats to SAP R/3 fields.

Appendix A. Check Reply C Program

```
/* Function:                                                       */
/*                                                                 */
/*   Program #3 of Fruit Salad (code name ASIL).                   */
/*                                                                 */
/*      -- This program is design to get a message from a queue.  */
/*                                                                 */
/*      -- Get the first char from the message. Convert it to an  */
/*         integer.                                                */
/*                                                                 */
/*      -- Get the depth of the Queue and compare the two numbers.*/
/*                                                                 */
/*      -- If the number from the message is > than the result    */
/*         of the depth then stop processing.                      */
/*                                                                 */
/*      -- If the number from the message is < than the result    */
/*         of the inquiry then get all of the messages on the     */
/*         queue.                                                  */
/*                                                                 */
/*      -- Concatenate all of the messages into one message.      */
/*                                                                 */
/*      -- Put the concatenated message onto a queue.             */
/*                                                                 */
/*   Program logic:                                                */
/*      - Connect to the QManager                                  */
/*      - MQOPEN queue for INPUT                                   */
/*      - Set up and perform the Inquiry                           */
/*      - Browse the first message                                 */
/*      - Compare the depth and the first field of the message    */
/*      - Stop processing (if depth is < first field) or get all  */
/*        of the messages                                          */
/*      - Concatenate all of the messages into one                 */
/*      - MQPUT1 the one message.                                  */
/*      - MQCLOSE and MQDISC                                       */
/*******************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
    /* includes for MQI  */
#include <cmqc.h>
```

```
int main(int argc, char **argv)
 {
   /*   Declare MQI structures needed                             */
   MQOD     od = {MQOD_DEFAULT};    /* Object Descriptor          */
   MQMD     md = {MQMD_DEFAULT};    /* Message Descriptor         */
   MQGMO   gmo = {MQGMO_DEFAULT};   /* get message options        */
   MQPMO   pmo = {MQPMO_DEFAULT};   /* put message options        */
   MQHCONN  Hcon;                   /* connection handle          */
   MQHOBJ   Hobj;                   /* object handle              */
   MQHOBJ   Hinq;                   /* handle for MQINQ           */
   MQLONG   O_options;              /* MQOPEN options             */
   MQLONG   C_options;              /* MQCLOSE options            */
   MQLONG   CompCode;               /* completion code            */
   MQLONG   OpenCode;               /* MQOPEN completion code     */
   MQLONG   Reason;                 /* reason code                */
   MQLONG   CReason;                /* reason code for MQCONN     */
   MQBYTE   buffer[101];            /* message buffer             */
   MQLONG   buflen;                 /* buffer length              */
   MQLONG   messlen;                /* message length received    */
   char     QMName[50];             /* queue manager name         */
   MQLONG   Select[3];              /* attribute selectors        */
   MQLONG   IAV[3];                 /* integer attribute values   */
   MQLONG   FruitAmtInt;            /* integer conversion value   */
   char     FruitAmt[1];            /* char                       */
   char     outputBuffer[100];      /* message buffer             */
   char     AdnerbBuffer[100];      /* transition buffer          */

   /****************************************************************/
   /* Create object descriptor for subject queue                   */
   /****************************************************************/
   strcpy(od.ObjectName, "FruitStandIn");    /***Input Queue   ***/
   QMName[0] = 0;   /* default */
   outputBuffer[0] = 0 ;

   /****************************************************************/
   /*   Connect to queue manager                                   */
   /****************************************************************/
   MQCONN(QMName,                /* queue manager                 */
          &Hcon,                 /* connection handle             */
          &CompCode,             /* completion code               */
          &CReason);             /* reason code                   */
   if (CompCode == MQCC_FAILED)
   {
     printf("MQCONN ended with reason code %ld\n", CReason);
     exit( (int)CReason );
   }
```

```
/**************************************************************/
/*   Open named queue for INQUIRE                              */
/**************************************************************/
O_options = MQOO_INQUIRE            /*open to inquire attributes*/
       + MQOO_BROWSE          /*open queue for browse  ***/
          + MQOO_INPUT_SHARED       /* open for shared         */
          + MQOO_FAIL_IF_QUIESCING;
 MQOPEN(Hcon,              /* connection handle            */
       &od,                /* object descriptor for queue   */
       O_options,          /* open options                  */
       &Hinq,              /* object handle for MQINQ       */
       &CompCode,          /* completion code               */
       &Reason);           /* reason code                   */
    if (CompCode != MQCC_OK)    /***** check the the open***/
    {
      printf("The open failed \n");
      exit( (int)CReason );
    }
    /**************************************************************/
    /*    Prepares for an inquire                                 */
    /**************************************************************/
    if (CompCode == MQCC_OK)
    { /
      Select[0] = MQIA_INHIBIT_GET;     /* attribute selectors  */
      Select[1] = MQIA_CURRENT_Q_DEPTH;
      Select[2] = MQIA_OPEN_INPUT_COUNT;
      MQINQ(Hcon,            /* connection handle              */
            Hinq,            /* object handle                  */
            3L,              /* Selector count                 */
            Select,          /* Selector array                 */
            3L,              /* integer attribute count        */
            IAV,             /* integer attribute array        */
            0L,              /* character attribute count      */
            NULL,            /* character attribute array      */
            &CompCode,       /* completion code                */
            &Reason);        /* reason code                    */
      /***********************************************************/
      /*   Did the Inquire work                                  */
      /***********************************************************/
      if (CompCode == MQCC_OK)
      {
         printf("IAV %ld \n",IAV[1]);
      }
      else      /* if MQINQ failed, flag that report is needed */
      {
           printf("The Inquiry failed \n");
      }
```

```
          /*********************************************************/
          /* Get messages from the message queue                   */
          /*********************************************************/
          buflen = sizeof(buffer) - 1; /* buffer size available for GET  */
          gmo.Options =  MQGMO_BROWSE_FIRST; /* get with browse   */

          MQGET(Hcon,                /* connection handle            */
              Hinq,                /* object handle                */
              &md,                 /* message descriptor           */
              &gmo,                /* get message options          */
              buflen,              /* buffer length                */
              buffer,              /* message buffer               */
              &messlen,            /* message length               */
              &CompCode,           /* completion code              */
              &Reason);            /* reason code                  */


        if (Reason != MQRC_NONE)       /* did the get work         */
        {
          printf("MQGET ended with reason code %ld\n", Reason);
          CompCode = MQCC_FAILED;
        }
        /*************************************************************/
        /* Display the message for testing                           */
        /* Get first number from buffer, convert to integer          */
        /*************************************************************/
      if (CompCode != MQCC_FAILED)
       {
         buffer[messlen] = '\0';           /*add terminator          */
         memcpy(FruitAmt,buffer,1);
         FruitAmtInt = atoi(FruitAmt) ;
       }

      }
    while (FruitAmtInt == IAV[1] && Reason == MQRC_NONE)
       {
          gmo.Options = MQGMO_ACCEPT_TRUNCATED_MSG; /* get messages  */
         /****************************************************************/
         /*                                                              */
         /*   In order to read the messages in sequence, MsgId and       */
         /*   CorrelID must have the default value.  MQGET sets them     */
         /*   to the values in for message it returns, so re-initialise  */
         /*   them before every call                                     */
         /*                                                              */
         /****************************************************************/
         memcpy(md.MsgId, MQMI_NONE, sizeof(md.MsgId));
         memcpy(md.CorrelId, MQCI_NONE, sizeof(md.CorrelId));
```

```
MQGET(Hcon,                /* connection handle            */
             Hinq,               /* object handle                */
             &md,                /* message descriptor           */
             &gmo,               /* get message options          */
             buflen,             /* buffer length                */
             buffer,             /* message buffer               */
             &messlen,           /* message length               */
             &CompCode,          /* completion code              */
             &Reason);           /* reason code                  */


       if (Reason != MQRC_NONE)
        {
          if (Reason == MQRC_NO_MSG_AVAILABLE || Reason ==
MQRC_NO_MSG_UNDER_CURSOR )
          {
           printf("no more messages\n");
          }
          else
          {
             printf("MQGET ended with reason code %ld\n", Reason);
            CompCode = MQCC_FAILED;
          }
        }
        else
         {
           /*****  Create a new message of all of the fruit responses  **/
           memcpy(AdnerbBuffer,buffer,buflen);
           strcat(outputBuffer,AdnerbBuffer);
         }
      }

      /**************************************************************/
      /*   Close the queue opened for INQUIRE                       */
      /**************************************************************/
         C_options = 0;          /* no close options             */
         MQCLOSE(Hcon,           /* connection handle            */
                 &Hinq,          /* object handle                */
                 C_options,
                 &CompCode,      /* completion code              */
                 &Reason);       /* reason code                  */
          if (Reason != MQRC_NONE)
          {
            printf("MQCLOSE ended with reason code %ld\n", Reason);
          }
```

```
if  (FruitAmtInt == IAV[1])
      {  /***********************************************************/
         /*  Put the combined message to a MQSI Queue to be process */
         /*   but first reset the MsgIds and CorrelIds.            */
         /************************************************************/

          strcpy(od.ObjectName, "FruitStandOut");
          memcpy(buffer,outputBuffer,sizeof(outputBuffer));
          messlen  = sizeof(outputBuffer);  /* length of reply */
         /***********************************************************/
         /*                                                          */
         /*   Put the message                                        */
         /*   Note - this sample stops if MQPUT1 fails; in some      */
         /*   applications it may be appropriate to continue         */
         /*   after selected Reason codes                            */
         /*                                                          */
         /************************************************************/
         MQPUT1(Hcon,            /* connection handle                */
                &od,             /* object descriptor                */
                &md,             /* message descriptor               */
                &pmo,            /* default options                  */
                messlen,         /* message length                   */
                buffer,          /* message buffer                   */
                &CompCode,       /* completion code                  */
                &Reason);        /* reason code                      */

         /* report reason if any  (loop ends if it failed)    */
         if (Reason != MQRC_NONE)
         {
           printf("MQPUT1 ended with reason code %ld\n", Reason);
         }
       }
   /****************************************************************/
   /*                                                              */
   /* Disconnect from MQM if not already connected                 */
   /****************************************************************/
   if (CReason != MQRC_ALREADY_CONNECTED )
   {
     MQDISC(&Hcon,                   /* connection handle          */
            &CompCode,               /* completion code            */
            &Reason);                /* reason code                */
     if (Reason != MQRC_NONE)
     {
       printf("MQDISC ended with reason code %ld\n", Reason);
     }
    }
```

```
  /*****************************************************************/
  /*                                                               */
  /* END OF Program #3 (ASIL)                                      */
  /*                                                               */
  /*****************************************************************/
    printf("Fruit Salad sample program #3 \n");
    return(0);
}
```

Appendix B. Special Notices

This publication is intended to help designers and developers to integrate applications that send messages in a different format from what the partner program understands. The information in this publication is not intended as the specification of any programming interfaces that are provided by the MQSeries Integrator Version 1.0. See the PUBLICATIONS section of the IBM Programming Announcement for MQSeries for more information about what publications are considered to be product documentation.

References in this publication to IBM products, programs or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM's product, program, or service may be used. Any functionally equivalent program that does not infringe any of IBM's intellectual property rights may be used instead of the IBM product, program or service.

Information in this book was developed in conjunction with use of the equipment specified, and is limited in application to those specific hardware and software products and levels.

IBM may have patents or pending patent applications covering subject matter 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 the IBM Director of Licensing, IBM Corporation, North Castle Drive, Armonk NY 10504-1785 USA.

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 Corporation, Dept. 600A, Mail Drop 1329, Somers, NY 10589 USA.

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

The information contained in this document has not been submitted to any formal IBM test and is distributed AS IS. The use of this information or the implementation of any of these techniques is a customer responsibility and depends on the customer's ability to evaluate and integrate them into the customer's operational environment. While each item may have been reviewed by IBM for accuracy in a specific situation, there is no guarantee that the same or similar results will be obtained elsewhere. Customers

Appendix C. Related Publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

C.1 International Technical Support Organization Publications

For information on ordering these ITSO publications see “How to Get ITSO Redbooks” on page 179.

- *MQSeries Version 5 Programming Examples,* SG24-5214
- *MQSeries Backup and Recovery*, SG24-5222
- *MQSeries Security: Example of Using a Channel Security Exit, Encryption and Decryption*, SG24-5306
- *MQSeries for Windows Version 2.1 in a Mobile Environment*, SG24-2103
- *Using MQSeries on the AS/400*, SG24-5236
- *Connecting the Enterprise to the Internet with MQSeries and VisualAge for Java*, SG24-2144
- *Internet Application Development with MQSeries and Java*, SG24-4896
- *Application Development with VisualAge for Smalltalk and MQSeries*, SG24-2117
- *Examples of Using MQSeries on WWW*, SG24-4882

C.2 Redbooks on CD-ROMs

Redbooks are also available on the following CD-ROMs. Click the CD-ROMs button at http://www.redbooks.ibm.com/ for information about all the CD-ROMs offered, updates and formats.

CD-ROM Title	**Collection Kit Number**
System/390 Redbooks Collection	SK2T-2177
Networking and Systems Management Redbooks Collection	SK2T-6022
Transaction Processing and Data Management Redbooks Collection	SK2T-8038
Lotus Redbooks Collection	SK2T-8039
Tivoli Redbooks Collection	SK2T-8044
AS/400 Redbooks Collection	SK2T-2849
Netfinity Hardware and Software Redbooks	SK2T-8046
RS/6000 Redbooks Collection (HTML, BkMgr)	SK2T-8040

CD-ROM Title	Collection Kit Number
RS/6000 Redbooks Collection (PDF Format)	SK2T-8043
Application Development Redbooks Collection	SK2T-8037

C.3 Other Publications

These publications are also relevant as further information sources:

- *MQSeries for AIX Quick Beginnings Version 5.0*, GC33-1867
- *MQSeries for Windows NT Quick Beginnings Version 5.0*, GC33-1871
- *MQSeries Command Reference*, SC33-1369
- *MQSeries Intercommunication*, SC33-1872
- *MQSeries: An Introduction to Messaging and Queuing*, GC33-0805
- *MQSeries System Administration*, SC33-1873
- *MQSeries Planning Guide*, GC33-1349
- *DB2 UDB Personal Edition Quick Beginnings V5*, S10J-8150
- *DB2 UDB for UNIX Quick Beginnings V5R2*, S10J-8148

These publications are available on the product CD in softcopy format:

- *MQSeries Integrator Application Development Guide*
- *MQSeries Integrator Installation and Configuration Guide*
- *MQSeries Integrator Programming Reference for NEON Formatter*
- *MQSeries Integrator Programming Reference for NEON Rules*
- *MQSeries Integrator System Management Guide*
- *MQSeries Integrator User's Guide*

How to Get ITSO Redbooks

This section explains how both customers and IBM employees can find out about ITSO redbooks, redpieces, and CD-ROMs. A form for ordering books and CD-ROMs by fax or e-mail is also provided.

- **Redbooks Web Site** `http://www.redbooks.ibm.com/`

  Search for, view, download or order hardcopy/CD-ROM redbooks from the redbooks web site. Also read redpieces and download additional materials (code samples or diskette/CD-ROM images) from this redbooks site.

  Redpieces are redbooks in progress; not all redbooks become redpieces and sometimes just a few chapters will be published this way. The intent is to get the information out much quicker than the formal publishing process allows.

- **E-mail Orders**

  Send orders via e-mail including information from the redbooks fax order form to:

	e-mail address
In United States	usib6fpl@ibmmail.com
Outside North America	Contact information is in the “How to Order” section at this site: `http://www.elink.ibmlink.ibm.com/pbl/pbl/`

- **Telephone Orders**

United States (toll free)	1-800-879-2755
Canada (toll free)	1-800-IBM-4YOU
Outside North America	Country coordinator phone number is in the “How to Order” section at this site: `http://www.elink.ibmlink.ibm.com/pbl/pbl/`

- **Fax Orders**

United States (toll free)	1-800-445-9269
Canada	1-403-267-4455
Outside North America	Fax phone number is in the “How to Order” section at this site: `http://www.elink.ibmlink.ibm.com/pbl/pbl/`

This information was current at the time of publication, but is continually subject to change. The latest information for customer may be found at `http://www.redbooks.ibm.com/` and for IBM employees at `http://w3.itso.ibm.com/`.

IBM Intranet for Employees

IBM employees may register for information on workshops, residencies, and redbooks by accessing the IBM Intranet Web site at `http://w3.itso.ibm.com/` and clicking the ITSO Mailing List button. Look in the Materials repository for workshops, presentations, papers, and Web pages developed and written by the ITSO technical professionals; click the Additional Materials button. Employees may also view redbook. residency, and workshop announcements at `http://inews.ibm.com/`.

IBM Redbook Fax Order Form

Please send me the following:

Title	**Order Number**	**Quantity**

First name	Last name	
Company		
Address		
City	Postal code	Country
Telephone number	Telefax number	VAT number

☐ Invoice to customer number

☐ Credit card number

Credit card expiration date	Card issued to	Signature

We accept American Express, Diners, Eurocard, Master Card, and Visa. Payment by credit card not available in all countries. Signature mandatory for credit card payment.

List of Abbreviations

AIX	Advanced Interactive Executive (IBM's flavor of UNIX)
CCA	Client Configuration Assistant
CLP	command line processor (DB2)
CSD	correctional service diskette
DB2	Database 2
DBMS	Database Management System
GUI	graphical user interface
HTML	Hypertext Markup Language
IBM	International Business Machines Corporation
ITSO	International Technical Support Organization
IVP	installation verification program
MQ	Message Queuing
MQHRF	MQSI Rules/Format Header
MQI	Message Queuing Interface
MQM	MQSeries Queue Manager
MQMD	MQ Message Descriptor
MQSI	MQSeries Integrator
MVS	Multiple Virtual Storage
NEON	New Era of Networks
OCX	OLE Control Module
ODBC	open database connectivity
PDF	Portable Document Format
PC	Personal Computer
PE	personal edition
PS	PostScript (file extension)
RFH	Rules/Format Header
SDK	Software Developers Kit
SMIT	System Management Interface Tool
UDB	Universal Database
URL	Uniform Resource Locator
WWW	World Wide Web

Index

ITSO Redbook Evaluation

Using the MQSeries Integrator Version 1.0
SG24-5386-00

Your feedback is very important to help us maintain the quality of ITSO redbooks. **Please complete this questionnaire and return it using one of the following methods:**

- Use the online evaluation form found at http://www.redbooks.ibm.com
- Fax this form to: USA International Access Code + 1 914 432 8264
- Send your comments in an Internet note to redbook@us.ibm.com

Which of the following best describes you?
_ Customer **_ Business Partner** **_ Solution Developer** **_ IBM employee**
_ None of the above

Please rate your overall satisfaction with this book using the scale:
(1 = very good, 2 = good, 3 = average, 4 = poor, 5 = very poor)

Overall Satisfaction __________

Please answer the following questions:

Was this redbook published in time for your needs? Yes___ No___

If no, please explain:

____

____

____

____

What other redbooks would you like to see published?

____

____

____

Comments/Suggestions: **(THANK YOU FOR YOUR FEEDBACK!)**

____

____

____

____

SG24-5386-00
Printed in the U.S.A.

Using the MQSeries Integrator Version 1.0

SG24-5386-00