



BEA eLink Information Integrator

User Guide

BEA eLink Information Integrator Version 1.1
Document Edition 1.1
July 2000

Copyright

Copyright © 2000, 1996-1999 BEA Systems Inc., or its suppliers, as applicable. All Rights Reserved.

Restricted Rights Legend

This software and documentation is subject to and made available only pursuant to the terms of the BEA Systems License Agreement and may be used or copied only in accordance with the terms of that agreement. It is against the law to copy the software except as specifically allowed in the agreement. This document may not, in whole or in part, be copied photocopied, reproduced, translated, or reduced to any electronic medium or machine readable form without prior consent, in writing, from BEA Systems, Inc.

Use, duplication or disclosure by the U.S. Government is subject to restrictions set forth in the BEA Systems License Agreement and in subparagraph (c)(1) of the Commercial Computer Software-Restricted Rights Clause at FAR 52.227-19; subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013, subparagraph (d) of the Commercial Computer Software--Licensing clause at NASA FAR supplement 16-52.227-86; or their equivalent.

Information in this document is subject to change without notice and does not represent a commitment on the part of BEA Systems. THE SOFTWARE AND DOCUMENTATION ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND INCLUDING WITHOUT LIMITATION, ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. FURTHER, BEA Systems DOES NOT WARRANT, GUARANTEE, OR MAKE ANY REPRESENTATIONS REGARDING THE USE, OR THE RESULTS OF THE USE, OF THE SOFTWARE OR WRITTEN MATERIAL IN TERMS OF CORRECTNESS, ACCURACY, RELIABILITY, OR OTHERWISE.

Trademarks or Service Marks

BEA, ObjectBroker, TOP END, and Tuxedo are registered trademarks of BEA Systems, Inc. BEA Builder, BEA Connect, BEA Manager, BEA MessageQ, BEA Jolt, M3, eSolutions, eLink, WebLogic, and WebLogic Enterprise are trademarks of BEA Systems, Inc.

All other company names may be trademarks of the respective companies with which they are associated.

BEA eLink Information Integrator User Guide

Document Edition	Part Number	Date	Software Version
1.1		July 2000	BEA eLink Information Integrator 1.1

About This Document

This document provides instructions for using BEA eLink Information Integrator to perform data translation.

This document covers the following topics:

- Understanding the BEA eLink Solution
- Using Information Integrator
- Using MsgDefAdmin
- Building Format Definitions
- Using the Tester
- Defining Rules
- Configuring Information Integrator and the IISERVER
- Additional Information Integrator Tools
- Message Explosion Example
- BEA eLink Platform Reference
- Data Types
- Operators
- eLink Information Integrator COBOL Copybook Importer

What You Need to Know

This document is intended for application programmers and users who will configure the Information Integrator and use it to execute information transfers.

e-docs Web Site

BEA product documentation is available on the BEA corporate Web site. From the BEA Home page, click on Product Documentation or go directly to the “e-docs” Product Documentation page at <http://e-docs.bea.com>.

How to Print the Document

You can print a copy of this document from a Web browser, one file at a time, by using the File—>Print option on your Web browser.

A PDF version of this document is available on the BEA eLink Information Integrator documentation Home page on the e-docs Web site (and also on the documentation CD). You can open the PDF in Adobe Acrobat Reader and print the entire document (or a portion of it) in book format. To access the PDFs, open the BEA eLink Information Integrator documentation Home page, click the PDF files button and select the document you want to print.

If you do not have the Adobe Acrobat Reader, you can get it for free from the Adobe Web site at <http://www.adobe.com/>.

Related Information

The following BEA publications are also available:

TUXEDO System 6 Reference Manual

TUXEDO System 6 Programmer's Guide, Volumes 1 and 2

Contact Us!

Your feedback on the BEA Information Integrator documentation is important to us. Send us e-mail at **docsupport@bea.com** if you have questions or comments. Your comments will be reviewed directly by the BEA professionals who create and update the BEA eLink Information Integrator documentation.

In your e-mail message, please indicate that you are using the documentation for the BEA BEA eLink Information Integrator 1.1 release.

If you have any questions about this version of the Information Integrator, or if you have problems installing and running the Information Integrator, contact BEA Customer Support through BEA WebSupport at **www.bea.com**. You can also contact Customer Support by using the contact information provided on the Customer Support Card, which is included in the product package.

When contacting Customer Support, be prepared to provide the following information:

- Your name, e-mail address, phone number, and fax number
- Your company name and company address
- Your machine type and authorization codes
- The name and version of the product you are using
- A description of the problem and the content of pertinent error messages

Document Conventions

The following documentation conventions are used throughout this document:

Item	Examples
Variable names	<p>Variable names represent information you must supply or output information that can change; they are intended to be replaced by actual names. Variable names are displayed in italics and can include hyphens or underscores. The following are examples of variable names in text:</p> <p><i>error_file_name</i></p> <p>The <i>when-return</i> value...</p>
User input and screen output	<p>For screen displays and other examples of input and output, user input appears as in the first of the following lines; system output appears as in the second through fourth lines:</p> <pre>dir c:\accounting\data Volume in drive C is WIN_NT_1 Volume Serial Number is 1234-5678 Directory of C:\BEADIR\DATA</pre>
Syntax	<p>Code samples can include the following elements:</p> <ul style="list-style-type: none">■ Variable names can include hyphens or underscores (e.g., <i>error_file_name</i>)■ Optional items are enclosed in square brackets: []. If you include an optional item, do not code the square brackets.■ A required element for which alternatives exist is enclosed in braces { }. The alternatives are separated by the pipe (vertical bar) character: . You must include only one of the alternatives for that element. Do not code the braces or pipe character.■ An ellipsis (...) indicates that the preceding element can be repeated as necessary.
Omitted code	<p>An ellipsis (...) is used in examples to indicate that code that is not pertinent to the discussion is omitted. The ellipsis can be horizontal or vertical.</p>

Item	Examples
Environment variables	Environment variables are formatted in an uppercase font. ENVFILE=\${APPDIR}
Key names	Key names are presented in boldface type. Press Enter to continue.
Literals	Literals are formatted in a monospace font. class extendSample
Window items	Window items are presented in boldface type. Window items can be window titles, button labels, text edit box names or other parts of the window. Type your password in the Logon window . Select Export to make the service available to the client.



Contents

About This Document

What You Need to Know	iv
e-docs Web Site	iv
How to Print the Document	iv
Related Information	v
Contact Us!	v
Document Conventions	vi

1. Understanding the BEA eLink Solution

BEA eLink Solution Overview	1-1
BEA eLink Information Integrator Overview	1-3
Formatter	1-4
MsgDefAdmin	1-5
Rules	1-5
Tester	1-6
IISERVER	1-6
Analyzing Your Data	1-7
Using Information Integrator	1-8
Defining Message Formats	1-9
Defining Field Mapping	1-10
Defining Rules	1-10
Editing the IISERVER Configuration File	1-10

2. Using Information Integrator

Sample Application Scenario	2-1
Sample Application Steps	2-2
Step 1. Define the Input Data	2-2

Step 2. Define the Database.....	2-4
Step 3. Create the Message Format Descriptions.....	2-4
Step 4. Create Output Operations.....	2-4
Step 5. Test the Message Format.....	2-5
Step 6. Define the Actions.....	2-5
Step 7. Configure the IISERVER.....	2-6
Step 8. Execute the data transformation.....	2-6

3. Using MsgDefAdmin

Creating an MFL Document.....	3-1
Message Format Elements.....	3-4
Running MsgDefAdmin.....	3-8
Example.....	3-10

4. Building Format Definitions

Understanding Information Integrator Formats.....	4-2
Using the Formatter Interface.....	4-3
Starting Formatter.....	4-3
Using the Formatter Window.....	4-5
Opening a New Window.....	4-5
Renaming Components.....	4-5
Duplicating Components.....	4-6
Deleting Components.....	4-6
Using the Used In Tab.....	4-7
Using Open Item.....	4-7
Using Drag and Drop.....	4-8
Using Formatter to Build Definitions.....	4-9
Defining Literals.....	4-9
Defining Fields.....	4-11
Defining Output Operations.....	4-13
Creating Output Operation Collections.....	4-20
Defining Input Controls.....	4-22
Defining Output Controls.....	4-39
Building Formats.....	4-54
Alternative Input and Output Formats.....	4-60

Alternative Input Formats	4-60
Alternative Output Formats	4-63
Importing and Exporting Format Components	4-65
Exporting Components	4-65
Importing Components	4-66
Field Mapping and Transformation.....	4-68
Example.....	4-71

5. Using the Tester

Starting the Tester.....	5-1
Parsing and Reformatting Messages Using the Tester	5-5
Example.....	5-6

6. Defining Rules

Understanding Information Integrator Rules.....	6-1
Using the Rules Interface	6-3
Starting Rules	6-3
Using the Rules Window.....	6-5
Opening a New Window	6-6
Renaming Components	6-6
Duplicating Components.....	6-7
Deleting Components.....	6-7
Using Drag and Drop	6-8
Building Rules	6-8
Application Groups	6-9
Message Types	6-9
Rules	6-11
Disabling and Enabling a Rule.....	6-11
Expressions.....	6-12
Creating or Modifying an Expression	6-13
Subscriptions	6-15
Associating Subscriptions with Rules.....	6-16
Actions	6-16
Adding a Reformat Action	6-18
Adding an Enqueue Action	6-19

Adding a Post Action	6-19
Adding a Call Action.....	6-19
Adding a Call Async Action	6-20
Adding an Explode Action.....	6-20
Adding a Cast Action	6-20
Adding a Return Action	6-21
Deleting an Action	6-21
Combining Actions to Create Subscriptions	6-21
Example	6-22

7. Configuring Information Integrator and the IISERVER

Configuring the Design Environment.....	7-2
Creating the Database Schema	7-2
Creating the Database Session Configuration File	7-3
Required Parameters	7-3
Optional Parameters	7-4
Sample Database Session Configuration File	7-4
Creating User Accounts.....	7-5
Creating Users on Oracle Systems	7-5
Creating Users on SQL Server Systems	7-7
Configuring the Execution Environment.....	7-9
Setting the Environment Variables.....	7-10
Adding the IISERVER to the UBBCONFIG File.....	7-12
Creating the Configuration File.....	7-14
Using the Rules Application	7-14
Using a Text Editor	7-21
Sample Information Integrator Configuration File.....	7-26
Creating the FML Field Table	7-27
Understanding the IISERVER.....	7-28
Understanding Message Processing	7-29
Request/Response Translation	7-29
Data Enrichment.....	7-31
Content-Based Routing	7-35
Message Explosion.....	7-37
Example — Step 2.....	7-41

Example — Step 7	7-42
Example — Step 8	7-44

A. Additional Information Integrator Tools

Defining the IIRESET Service	A-2
Converting Data Integration Option Formats to MFL	A-3
Using the sendBuf Utility	A-4
Using the ud32 Utility	A-6
Using the NNFie Utility	A-6
NNFie Import Syntax	A-9
NNFie Export Syntax	A-10
Using APITEST	A-11
Using MSGTEST	A-12
Configuration File	A-14
Using the NNRIE Utility	A-15
NNRIe Import Syntax.....	A-18
NNRIe Export Syntax.....	A-18
Command Line Functions	A-20

B. Message Explosion Example

Input Buffer Contents	B-2
MFL File Definitions.....	B-3
Rule Definitions.....	B-5

C. BEA eLink Platform Reference

BEA eLink Platform Architecture.....	C-1
ATMI Runtime Services.....	C-2
FML32	C-4
FML Buffers.....	C-5
Mapping Field Names to Field Identifiers	C-6
FML32 Primitives	C-7
eLink Commands.....	C-8
Commonly Used Tuxedo Commands	C-8
Commonly Used tadmin Commands	C-9

D. Data Types

E. Operators

F. eLink Information Integrator COBOL Copybook Importer

BEA eLink II CCBI Overview	F-1
Using eLink II CCBI	F-3
Importing the Converted File	F-3
Supported COBOL Data Types	F-5

Glossary

Index

1 Understanding the BEA eLink Solution

This section discusses the following topics:

- BEA eLink Solution Overview
- BEA eLink Information Integrator Overview

BEA eLink Solution Overview

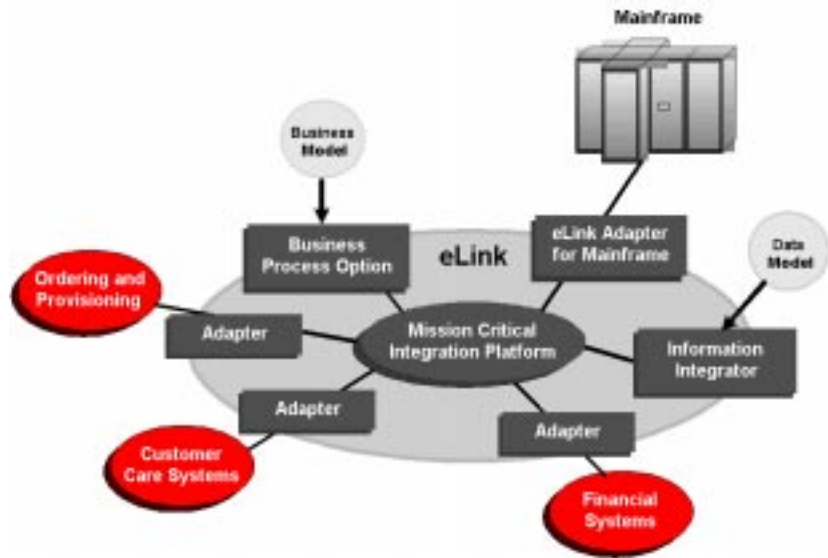
BEA eLink™ provides an open Enterprise Application Integration (EAI) solution that allows applications throughout organizations to communicate seamlessly. Using EAI, you gain the long-term flexibility and investment protection you need to keep up with today's ever-changing business environment.

Typically, companies use packaged applications to automate internal operations, such as financial, manufacturing, or human resources. While they successfully address the needs of these specific areas, these proprietary platforms often do not work together. To compete today, you need a much greater exchange of information. Systems need to communicate at a process level within your own organization, as well as with customer's and supplier's systems. BEA eLink Platform is the underlying basis of BEA eLink, a family of off-the-shelf enterprise application integration (EAI) products. BEA eLink leverages the BEA transaction platform to integrate existing legacy applications with customer-focused and business-to-business e-commerce initiatives.

1 Understanding the BEA eLink Solution

BEA eLink Platform provides a proven infrastructure for integrating applications within the enterprise and across the Web. BEA eLink Platform ensures high-performance, secure transactions and transparent access to mission-critical applications and information throughout the enterprise and across the Web. Figure 1-1 illustrates the eLink logical architecture and shows where the eLink Adapters fit into the process.

Figure 1-1 BEA eLink Solution Illustration



The entire BEA eLink family (including all options and adapters) is highly scalable. Multiple instances of BEA eLink components can collaborate so that work is divided between eLink domains. BEA eLink includes Simple Network Management Protocol (SNMP) integration for enterprise management.

The current BEA eLink Platform leverages the BEA Tuxedo infrastructure because it is based on a service-oriented architecture. Both BEA Tuxedo and BEA eLink communicate directly with each other and with other applications through the use of services. Multiple services are grouped into “application servers” or “servers”. The terms Tuxedo services/servers and eLink services/servers can be used interchangeably. Because this document is specifically addressing the eLink family, the terms “eLink service” and “eLink server” are used throughout.

The BEA eLink Platform complies with the Open Group's X/Open standards including support of the XA standard for two-phase commit processing, the X/Open **ATMI** API, and XPG standards for language internationalization. C, C++, COBOL, and Java are supported. The BEA eLink Platform connects to any RDBMS, OODBMS, file manager or queue manager, including a supplied XA-compliant queuing subsystem.

The following components operate with BEA eLink Platform:

- The **eLink Information Integrator** translates data models used by different applications. It provides a cost-effective alternative to writing or generating programs to perform this function. It also handles complex translation with great power and scalability.
- The **eLink Business Process Option** helps automate tasks in the distributed global business process and dynamically responds to business events and exceptions. The BPO is currently implemented by integrating eLink with technology based on InConcert workflow management software.
- An **eLink Adapter** provides the interface between the BEA eLink Platform and external applications with out-of-the-box functionality.

BEA eLink Information Integrator Overview

BEA eLink Information Integrator is an application integration component of eLink. It simplifies the daunting task of sharing data between applications and databases that run on different distributed computing environments within and beyond the enterprise. Information Integrator provides an easy to use set of tools that allow you to define how data will be interpreted and reformatted based on rules that you define. All format and rule configuration information is stored in a database allowing easy access to this information. When you actually deploy Information Integrator as part of an integration solution, a runtime server (IISERVER) uses the rules and format definitions stored in the database to process data transformation requests. The configuration information is cached by IISERVER insuring efficient processing.

BEA eLink Information Integrator consists of the following components:

- Formatter
- MsgDefAdmin
- Rules
- Tester
- IISERVER

These components are described in the following sections.

Formatter

BEA eLink Information Integrator Formatter allows you to store the necessary information that enables Information Integrator to parse an input message into individual components based on a layout (format) for the data. With Formatter, you can define input formats that match input data, define output formats that determine the appearance of output data, and view and edit formats created either by MsgDefAdmin or the Formatter graphical user interface.

Using Formatter, you define the following:

- Fields — Named divisions of message data tied to either a parse or output control. Each field can be associated with both an input and/or output control within a format.
- Literals — Single characters or strings used to mark the end of a piece of data, as a prefix or suffix, or as a pad character.
- Input controls — Controls used to parse input messages. Input control information describes the characteristics for the data in a field.
- Output Controls — Controls used to format output messages. Output control information describes how to format a component field for an output message.
- Output Control Collections — Formatter and operations you have defined used within output controls to further affect output data after data type transformations occur. Collections enable sets of operations to be grouped in the order they are to be executed.

- Flat and Compound Formats (input and output) — Combinations of fields or other formats used to describe formatting for an entire input or output message. Formats come in two main types – flat and compound. Flat formats contain a series of defined fields and the associated information needed to parse or format them (input controls or output controls, depending on whether the format is for input or output). Compound formats contain series of flat or other compound formats.
- Field Mapping — Mapping input fields to fields with different names in the output format. By default, input fields are mapped to output fields by matching names.

MsgDefAdmin

The MsgDefAdmin tool is a command line utility that allows the definition of a message format to be specified in an XML language called Message Format Language (MFL). Using your favorite text or XML editor, you can specify fields and the grouping of fields, according to the Message Format Language Document Type Definition (mfl.dtd). This XML document can then be imported using the MsgDefAdmin Tool to define the message format and all components (fields, controls, and formats).

Rules

Rules are used by the IISERVER to evaluate the contents of a message and perform actions based on the evaluation results.

Using Rules, you define the following:

- Application Groups — A logical grouping used to organize rules. For example, you may want to split your rules by project. Each application group would contain the messages and rules pertaining to that project.
- Messages — Identifies the input format used to parse an input buffer for the purposes of rules evaluation.
- Rules — Each rule has evaluation criteria (called an expression) that consists of fields from the message and associated Rules operators linked together with

Boolean operators. Fields, which are defined in the Formatter, can be compared against constant data or other fields within a message.

- Subscriptions — Groups of actions that are to be processed when a rule evaluates to true.

Tester

The Tester utility allows you to parse and reformat messages as a validation test. Using the Tester utility, you can make sure the message formats you built using the Formatter or MsgDefAdmin tool produce the expected results.

IISERVER

The IISERVER is a message format and routing server. It advertises available service names and their associated application names and input formats. The IISERVER translates messages from one format to another, simplifying message exchange between different systems and applications. The IISERVER breaks down complex messages into simple messages for processing. The IISERVER can translate formats containing fixed, tagged, or delimited fields, as well as multi-part nested formats.

The IISERVER uses control tables within the database to recognize and parse input and output messages. These tables describe message formats and format components. You create the data contained in these tables using the Formatter and MsgDefAdmin tools described earlier (see “Formatter” and “MsgDefAdmin” for more information). The IISERVER uses this information to interpret values from incoming messages and dynamically construct outgoing messages.

The following sections summarize the steps necessary to use Information Integrator to solve a business problem.

Analyzing Your Data

One of the first things you need to do prior to using Information Integrator is to analyze your data. This analysis consists of the following steps:

1. Determine the structure of the input data, including:

- fields and their names
- field data types
- delimiters and/or field length
- tags
- repeating fields

2. Determine the input message type. Supported message types are as follows:

STRING — A data structure that is an array of non-null characters terminated by the null character.

CARRAY — A data structure that is an array of characters any of which can be the null character.

FML — Attribute-value pairs called fields, in which the attribute is the field's identifier and the associated value represents the field's data content.

FML32 — Similar to FML, but FML32 uses 32-bit values for the field lengths and identifiers.

FMLVO — Similar to a STRING buffer, but put into FML format to accommodate data-dependent routing. FMLVO is an FML buffer where the message data is stored in the value of only one field. It may contain other fields that describe how to process the data.

3. Determine the structure of output data, including:

- fields and their names
- field data types
- delimiters and/or field length
- tags
- repeating fields

4. Determine the output message type. Supported output message types are the same as the input message types described in Step 2.
5. Determine the data mapping you want to use, including:
 - How the data structure maps from input to output
 - How you want to translate the field delimiters and field length
 - The data concatenation or partitioning you want to use during mapping
6. Determine the services you want to invoke and the message routing procedures you want to use. This analysis includes the following:
 - What services are to receive the transformed data?
 - What are the return values, if any?
 - What events are to be posted, if any?
 - What method of service invocation should be used?
 - Is any additional reformatting necessary?
 - What are the routing characteristics for data defined actions?

Once you have completed the data analysis, you are ready to configure and use BEA eLink Information Integrator to translate your data.

Using Information Integrator

Before data translation can occur, the following high level process must be followed to configure the Information Integrator Server (IISERVER) for the desired conversion process:

1. Define message formats in the format database.
2. Define mapping between message formats.
3. Define rule-based actions to take for defined message formats. Examples of actions include reformat, send message to another application, or enqueue a message.
4. Edit the IISERVER configuration file.

There are two phases involved in translating data using BEA eLink Information Integrator: the Design phase and the Execution phase.

- Design — During this phase, you create the formats and rules you want to use for processing your data.
- Execution — During this phase, your data is translated, using the formats and rules you created during the Design phase.

The following sections further explain the steps involved in the configuring data translation using Information Integrator.

Defining Message Formats

Message formats define the layout of data that is to be processed by IISERVER. This layout defines the actual “wire format” of the data that is communicated between collaborating applications. Each data item, such as a person's name, is defined as a field. For example, if a message consists of an employee's name and an employee's Social Security Number, the message contains two fields. For each defined field, an input control and an output control must be defined that indicate how to parse the field (input control), and how to output the field (output control). Groups of fields are called flat formats. Groups of flat formats are called compound formats. By defining fields, controls, and formats, the layout of the data is made available to the IISERVER component, so that it can perform the defined translation and actions during the execution phase.

Two utilities are provided with Information Integrator to define message formats:

- Formatter — GUI tool that allows the definition of all components of a message format. Using the Formatter component, you can visually define each field, control, and format.
- MsgDefAdmin — Command line utility that allows the definition of a message format to be specified in an XML language called Message Format Language (MFL). Using a text or XML editor, you specify fields and the grouping of fields, according to the Message Format Language Document Type Definition (mfl.dtd). This XML document can then be imported using the MsgDefAdmin utility to define the message format and all components (fields, controls, and formats).

Defining Field Mapping

After you define message formats, you can define field mapping to indicate how to map data from one message format to another. You do this using the Formatter component. You simply associate an output field of one message format with an input field of another message format, for each field that is to be mapped.

Defining Rules

After you define message formats and translations, you use the Rules component to specify the actions you want the IISERVER component to perform at run-time. Examples of these actions are: reformat a message, call an eLink Platform service with a message, explode a message into multiple messages, and enqueue a message to a queue. Using the Rules component, you can specify the actions to take if a specific message type is received, and additional rules that have to hold true before these actions are taken.

For example, suppose we have a message that contains payroll information for all employees in our company. We want IISERVER to explode this message into several messages, each containing payroll information for only one employee. Once we define the message formats, we use the Rules component to define a name for our set of rules and actions. In this example we define the set as “TRANSFORM_PAYROLL”. We then define a set of rules to evaluate when the payroll message is received. In our example we always want the IISERVER to perform our actions, so we define a single rule that is always true. We then define an explode action that explodes the larger payroll message into several messages for each employee.

Editing the IISERVER Configuration File

The last step is to edit the IISERVER configuration file. This can be done manually, using any text editor, or using the Rules application. In the configuration file, you specify the eLink Platform service name that is to be advertised for the previously defined set of rules. For example, we defined a set of actions called “TRANSFORM_PAYROLL”. In the IISERVER configuration file, we give this set of actions an eLink Platform Service Name. Then at run-time, when this service is invoked, the configured actions are performed.

2 Using Information Integrator

The BEA eLink Information Integrator installation CD-ROM includes a sample application to help you become familiar with the data translation process. This sample application is introduced in this chapter, and is carried out throughout the entire User Guide. Each chapter that follows includes an Example section which further explains the steps introduced here.

This chapter discusses the following topics:

- Sample Application Scenario
- Sample Application Steps

Sample Application Scenario

The XYZ Corporation has just hired a new employee as a buyer for their manufacturing plant. The new employee has been added to the payroll system at the plant. The payroll for all XYZ Corporation locations is generated at the home office on an SAP system.

We need to calculate the weekly pay for the new employee, reformat the data, and send it to the home office's SAP system.

Sample Application Steps

This section briefly describes the steps we need to take to calculate the weekly pay for the new employee and send the data to the home office. More detailed descriptions of these steps can be found in the “Example” sections of the chapters referenced in each step.

Step 1. Define the Input Data

Our first step is to define the input data. We have data records in the payroll system at the plant that contain the following fields:

- Employee Name
- Employee Number
- Three fields that repeat for each week the employee worked:
 - Pay Date
 - Employee Rate
 - Employee Hours

Our input data record appears in the payroll system database at the plant as follows:

Field Name	Data Type	Field Contents
NAME	String	Pete Walsh
EMPNO	Numeric	12345
PAYDATE	Date	01/15/00
RATE	Decimal	22.60
HOURS	Numeric	40
PAYDATE	Date	01/30/00
RATE	Decimal	22.60

Field Name	Data Type	Field Contents
HOURS	Numeric	32
PAYDATE	Date	02/15/00
RATE	Decimal	22.60
HOURS	Numeric	40

We want Information Integrator to calculate the pay for each of the weeks the employee worked and return the data in the following format:

12345 | 01/15/00 | 904.00 | 01/30/00 | 723.20 | 02/15/00 | 904.00

where:

12345

is the Employee Number

|

is used as a field separator for illustration purposes

01/15/00

is the first Pay Date

904.00

is the amount of pay for the first pay period, calculated as Hours (40) times Rate (22.60). We will create this calculation in “Step 4. Create Output Operations.”

01/30/00

is the second Pay Date

723.20

is the amount of pay for the second pay period (32 Hours * 22.60 Rate)

02/15/00

is the third Pay Date

904.00

is the amount of pay for the third pay period (40 Hours * 22.60 Rate)

Step 2. Define the Database

Before you can start using any of the Information Integrator tools, you must define your database, user ID, and password. To do this, you edit the `SQLSVSES.CFG` file provided on the installation CD-ROM. For specific instructions on editing the `SQLSVSES.CFG` file, refer to the “Example — Step 2” section in “Configuring Information Integrator and the IISERVER.”

Step 3. Create the Message Format Descriptions

The next step is to create the message format descriptions. Message formats define the layout of data that is to be processed. Information Integrator provides two ways to create message format descriptions: the Formatter tool and the `MsgDefAdmin` utility. The Formatter tool is a graphical user interface (GUI) tool that assists you in creating format descriptions. The `MsgDefAdmin` utility is a command line utility that allows the definition of a message format to be specified in an XML language called Message Format Language (MFL). In this example, we will use the `MsgDefAdmin` tool to create the format descriptions.

The MFL file needed to create the format descriptions is included in the `\sample` directory on the installation CD-ROM and is called `II_payroll_msg1.xml`. A second MFL file is needed to create the calculated pay field. This file is also included in the `\sample` directory and is called `II_payroll_msg2.xml`.

For details on creating an MFL file, refer to “Creating an MFL Document” in “Using `MsgDefAdmin`.”

For the specific steps to create this file for the sample application, refer to the “Example” section in “Using `MsgDefAdmin`.”

Step 4. Create Output Operations

The next step is to create an output operation to calculate the employee’s weekly pay. Output operations provide the different actions that can be performed on an output field. We will use the Formatter GUI to define the calculation (output operation) required to compute the amount of pay for each pay period and adjust the message definitions to use the new output.

For details on creating Output Operations, refer to “Defining Output Controls” in “Building Format Definitions.”

For the specific steps to the Output Operations for the sample application, refer to the “Example” section in “Building Format Definitions.”

Step 5. Test the Message Format

At this point, we are ready to test the message format using the Tester utility provided with Information Integrator. The Tester utility allows you to parse and reformat messages as a validation test. We recommend that you test your message formats before implementing them to make sure they are set up correctly, and that the output is formatted to fit your needs.

For details on testing message formats, refer to “Using the Tester.”

For the specific steps to test the message format for the sample application, refer to the “Example” section in “Using the Tester.”

Step 6. Define the Actions

The next step is to define the actions required to perform the data translation for the sample application. Actions hold subscription instructions. Subscriptions are lists of actions that are to be taken when a message evaluates true.

To define the actions, we use the Rules graphical user interface (GUI) to create the following:

- Application group
- Message type
- Rule
- Subscription lists

For details on defining actions in the Rules GUI, refer to “Defining Rules.”

For the specific steps to create the actions necessary for this sample application, refer to the “Example” section in “Defining Rules.”

Step 7. Configure the IISERVER

The next step is to configure the IISERVER to advertise the services required to perform the data transformation. The IISERVER is a message format and routing server that identifies available service names and their associated application names and input formats. To configure the IISERVER, we need to edit the Information Integrator .CFG configuration file and the FML field table.

The FML field table defines the FML fields, and is required if the messages you want to process with Information Integrator are contained in FML buffers. The file for this sample is included in the `\sample\payroll` directory on the installation CD-ROM and is called `fields.fml`. For details on creating an FML field table, refer to “Creating the FML Field Table” in “Configuring Information Integrator and the IISERVER.”

The .CFG configuration file, typically named `ii.cfg`, controls the operation of the Information Integrator server (IISERVER). For more information on the .CFG configuration file, refer to “Configuring Information Integrator and the IISERVER.” For the specific steps to create the .CFG file for the sample application, refer to the “Example — Step 2” section in “Configuring Information Integrator and the IISERVER.”

Step 8. Execute the data transformation

The final step is to execute the data transformation. This is done using the IISERVER and a command line utility called `sendBuf`. For details on the IISERVER, refer to “Configuring Information Integrator and the IISERVER.” For details on the `sendBuf` utility, refer to “Additional Information Integrator Tools.”

For the specific steps to execute the data transformation using the IISERVER and the `sendBuf` utility for the sample application, refer to the “Example — Step 2” section in “Configuring Information Integrator and the IISERVER.”

Note: To execute the data transformation for this sample, you must have eLink Platform loaded and running. The steps to do this are described in the “Example — Step 2” section in “Configuring Information Integrator and the IISERVER.”

3 Using MsgDefAdmin

The MsgDefAdmin Tool is a command line utility that allows the definition of a message format to be specified in an XML language called Message Format Language (MFL). Using your favorite text or XML editor, you can specify fields and the grouping of fields, according to the Message Format Language Document Type Definition (`mfl.dtd`). This XML document can then be imported using the MsgDefAdmin Tool to define the message format and all components (fields, controls, and formats).

The tasks involved in creating formats using MsgDefAdmin are as follows:

- Creating an MFL Document
- Running MsgDefAdmin

Creating an MFL Document

An MFL document is a description of a message that is sent to an application. To create an MFL document, you need to translate messages into a textual description that conforms to the MFL document type definition (`mfl.dtd`). The file `mfl.dtd` defines the elements and attributes necessary to create message formats.

To create an MFL document, follow the steps below. A sample MFL document is shown in Listing 3-2 following these steps.

1. Start a text editor.
2. Type the following as the first two lines in your MFL document.

```
<?xml version='1.0'?>
<!DOCTYPE MessageFormat SYSTEM 'mfl.dtd'>
```

Note: These two lines must always be the first two lines of your MFL document.

3. Specify the name of the message format in the `MessageFormat` element (the third line in the MFL document) as shown below:

```
<?xml version='1.0'?>
<!DOCTYPE MessageFormat SYSTEM `mfl.dtd`>
<MessageFormat name='message_name'>
```

4. Add a `FieldFormat` element for the first field in the message. The `FieldFormat` element specifies the name of the field, as well as the field's type attributes. An example is shown below:

```
<?xml version='1.0'?>
<!DOCTYPE MessageFormat SYSTEM `mfl.dtd`>
<MessageFormat name='message_name'>
  <FieldFormat name='field_name'
    type='type_attribute' />
```

5. If you need to group this field with others for purposes of defining a format for the group, add a `StructFormat` element. The `StructFormat` element defines the format for a group of fields. An example of the `StructFormat` element for a repeating field is shown below:

```
<?xml version='1.0'?>
<!DOCTYPE MessageFormat SYSTEM `mfl.dtd`>
<MessageFormat name='message_name'>
  <FieldFormat name='field_name'
    type='type_attribute' />
  <StructFormat name='structure_format_name'
    repeatField="repeating_field_name">
```

Note: Refer to “Message Format Elements” for a list of the elements you can include in an MFL document and their descriptions.

6. Add additional fields to the message format by defining `FieldFormat` elements for each of the additional fields. Set the type attribute according to each field's type.
7. Add the end tags for the `StructFormat` and `MessageFormat` elements. Listing 3-1 shows a sample MFL document. Refer to “Message Format Elements” for an explanation of the various elements in the MFL document.

Listing 3-1 Sample MFL Document

```
<?xml version='1.0'?>
<!DOCTYPE MessageFormat SYSTEM 'mfl.dtd'>
<MessageFormat name='PAYROLL_MSG'>
<FieldFormat name='NUM_EMPLOYEES' type='LittleEndian4' />
<StructFormat name='EMPLOYEE_REC'
  repeatField="NUM_EMPLOYEES">
<FieldFormat name='EMPLOYEE_NAME' type='String'
  delim='\xFF' />
<FieldFormat name='EMPLOYEE_SSN' type='String'
  length='11' />
<FieldFormat name='EMPLOYEE_PAY' type='Upacked'
  length='15' />
</StructFormat>
</MessageFormat>
```

8. Save your message definition and exit from the text editor. You can now use `MsgDefAdmin` to add this message definition to the Database. Refer to “Running `MsgDefAdmin`” for instructions.

Message Format Elements

This section lists the message format elements that make up the MFL document and provides a description of each element.

Note: In the examples below, the following characters have special meaning:

- Square brackets ([]) indicate optional parameters
- Curly brackets ({}) indicate choices
- Forward slash (/) indicates “OR”, as in “yes/no.”

MessageFormat

Acts as the top-level element for a message type's format information. For example:

```
<MessageFormat name="xxx">  
</MessageFormat>
```

where *xxx* is the name of the message.

StructFormat

Defines the formatting for a group of fields. For example:

```
<StructFormat name="xxx"  
  [ {repeat="99" repeatField="xxx"  
    repeatDelim="xxx" } ]  
  [alternative="yes/no"  
    [optional="yes/no" ] ]>  
</StructFormat>
```

Listed below are the parameters that may be included in the `StructFormat` element:

`name="xxx"`

The name of the message format. Required.

`repeat="99"`

Flags the message as repeating and indicates the number of times the message will repeat.

`repeatField="xxx"`

The name of the field that contains the repeat count for the message.

`repeatDelim="xxx"`

The delimiter that ends the repeating of this message.

`alternative="yes/no"`

Indicates whether the message is defined as an alternative format message.

`optional="yes/no"`

Indicates whether the message is optional.

FieldFormat

Defines the formatting for a single field. For example:

```
<FieldFormat name="xxx" type="xxx"
  [ basetype="xxx" cutoff="99" ]
  [ length="99" delim="xxx" ]
  [ optional="yes/no" ]
  [ accessMode="Normal/Current/Next/
    Controlling/Relative/Increment" ]
  [ value="xxx" ]>
</FieldFormat>
```

Listed below are the parameters that may be included in a `FieldFormat` element:

`name="xxx"`

The name of the field. Required.

`type="xxx"`

The field type. Required. Refer to Appendix D, "Data Types," for more information on the supported data types.

`basetype="xxx"`

Used only for Date and Time data types to indicate the base data type for the field. Valid values are: String, Numeric, and EBCDIC.

`cutoff="99"`

Used only for date formats containing a two digit year. Valid values are 00-99. This value is used to convert to a four digit year as follows:

```
yy >= cutoff --> 19yy
yy < cutoff  --> 20yy
```

`length="99"`

Defines an exact length for the field data. For data types that do not have an implicit length (String, Numeric, etc.), either an exact length or a field delimiter must be specified.

`delim="xxx"`

Defines a delimiter denoting the end of the field data. For data types that do not have an implicit length (String, Numeric, etc.), either an exact length or a field delimiter must be specified. Delimiter values may contain escaped decimal or hexadecimal values.

`optional="y/n"`

Indicates whether the field should be marked as optional on both the input and output message formats.

`accessMode="xxx"`

Must be one of the following values:

Normal - Access the instance in the same repeating component as the current controlling field instance. If there is no controlling field, access the first instance.

Current - Access the same field instance as on the previous access (the first access will get the first instance of the field).

Next - Access the next field instance relative to the previous access.

Controlling - This field is the controlling field for the repeating component. On each repetition, access the next field instance that is still a child of the current controlling field instance of the parent format. If there is no parent controlling field, the repetitions end with the last field instance from the input message.

Relative - The first field in a repeating component that Formatter encounters with this access mode is the controlling field for the repeating component (see Controlling field). Any other field in the repeating component with this access mode behaves as if it has access mode Access sibling instance or Normal access (access the sibling of the controlling field).

Increment - A field with this access mode is the controlling field for the repeating component (see Controlling field). This accesses the current value and increments it.

`value="xxx"`

The value of the literal. This attribute is only valid if the type of the FieldFormat is "Literal."

`TagField`

Defines a tag for a field format and the value to be matched. For example:

```
<TagField type="xxx"  
  [length="99" value="xxx"]>
```

`</TagField>`

Listed below are the parameters that may be included in a `TagField` element:

`type="xxx"`

Required. The type of the field. Refer to Appendix D, "Data Types," for more information on the supported data types.

`value="xxx"`

Required. Defines the value of the tag for this field.

`LenField`

Defines a length within the message data for a field. For example:

```
<LenField type="xxx"
  [length="99"]
</LenField>
```

`type="xxx"`

Required. The type of the field. Refer to Appendix D, "Data Types," for more information on the supported data types.

`length="99"`

Defines an exact length for the field data. For data types that do not have an implicit length (i.e. String, Numeric, etc.) either an exact length or a field delimiter must be specified.

Notes: The `TagField` and `LenField` elements must be child elements of a `FieldFormat` element. The order in which they appear in the `FieldFormat` element determines the order of the data in the message.

The same `StructFormat` element may not be defined in two different message definitions. If this happens, when these message definitions are imported, the `StructFormat` is redefined by the second message definition, leaving the first message definition in an invalid state. For example, suppose message definition A and message definition B both contain a structure called `PAY_TABLE`. When message definition A is imported, the import creates a format called `PAY_TABLE`. When message definition B is imported, the `PAY_TABLE` format is redefined. Message definition A then contains a reference to a format that no longer exists. When you try to access message definition A, you now receive an error.

Running MsgDefAdmin

To add message definitions created using an MFL file to the Information Integrator database, follow the steps below.

Note: Do not run the MsgDefAdmin utility and the Formatter GUI concurrently if they are connected to the same database.

1. Open a DOS command-prompt window.
2. Change to the directory where MsgDefAdmin is installed (for example, C:\BEAII)
3. Type **MsgDefAdmin** and press **Enter**. The MsgDefAdmin importer utility appears as shown in Listing 3-2.

Listing 3-2 MsgDefAdmin

```
- Copyright (c) 2000 BEA Systems, Inc.  
All Rights Reserved.  
Distributed under license by BEA Systems, Inc.  
BEA eLink Information Integrator is a registered trademark.  
  
This product includes software developed by the Apache Software  
Foundation (http://www.apache.org/).  
  
- - -      BEA eLink Information Integrator 1.1      - - -  
- - -      Message Definition Importer Tool        - - -  
  
Enter choice:  
1) Add Message Definition  
2) Get Message Definition  
3) Delete Message Definition  
4) List Message Definitions  
Q) Quit >
```

4. Type **1** to select the Add Message Definition choice and press **Enter**. An additional prompt displays asking you to specify the name of the file containing the XML message definition.

5. Type the path and name of the message file you created with your text editor and press **Enter**. MsgDefAdmin processes the file, stores the message definition in the Database, and returns a message that the definition was successfully created.
6. Type **Q** at the prompt and press **Enter** to quit MsgDefAdmin.

You can also run MsgDefAdmin in a non-interactive mode. There are two command-line options that allow you to import or export message definitions.

The command-line syntax is as follows:

```
MsgDefAdmin [-i | -e] <XML filename>
```

where

-i

Denotes an import operation. MsgDefAdmin will import the message definitions in `<XML filename>` into the Database. The following example shows how to use `-i`.

```
C:\BEAII>MsgDefAdmin -i payroll.xml employee_data.xml purchase_order.xml
Processing file: payroll.xml
Message Definition created!
Processing file: employee_data.xml
Message Definition created!
Processing file: purchase_order.xml
Message Definition created!
```

-e

Denotes an export operation. MsgDefAdmin will export the message definitions in `<XML filename>` from the Database. Each definition that is exported is saved to a file using the name of the message definition as the file name and `.xml` as the filename extension. The following example shows how to use `-e`.

```
C:\BEAII>MsgDefAdmin -e PAYROLL_MSG EMPLOYEE
XML Document retrieved:
Created file PAYROLL_MSG.xml
ML Document retrieved:
Created file EMPLOYEE.xml
```

`<xml filenames>`

Name of the XML file that contains or receives message definitions.

Example

This section outlines the procedure required to complete Step 3 in “Using Information Integrator.” In Step 3, you create the message format definitions. We need to define the layout of our data as a linear buffer. Since our input is actually an FML32 buffer, Information Integrator will map the data from FML to a linear buffer as defined by the input message format. This is done using MsgDefAdmin to create an XML file. We have provided two XML files for you to use in this example. You can use these files as a base for creating your own XML files to solve your own business needs.

In the steps that follow, we will view and discuss the XML files for this payroll example, and use MsgDefAdmin to create the message format definitions.

1. With your favorite text editor, open `\sample\payroll\II_payroll_msg1.xml`. This file is located in the `\sample\payroll` directory where you installed Information Integrator. The file is shown below:

```
<?xml version='1.0'?>
<!DOCTYPE MessageFormat SYSTEM 'mfl.dtd'>
MessageFormat name="II_payroll_msg1">
  <FieldFormat name="NAME" type="String"/>
  <FieldFormat name="EMPNO" type="BigEndian4"/>
  <FieldFormat name="$RPAYDATE" type="BigEndian2"/>
  <StructFormat name="II_payroll_msg1_rep" repeatField="$RPAYDATE">
    <FieldFormat name="PAYDATE" type="String" accessMode="Controlling"/>
    <FieldFormat name="RATE" type="String"/>
    <FieldFormat name="HOURS" type="String"/>
  </StructFormat>
</MessageFormat>
```

This file creates the input format for the data.

When you have finished viewing the first XML file, close it.

2. Open the file `\sample\payroll\II_payroll_msg2.xml`. This file is shown below:

```
<?xml version='1.0'?>
<!DOCTYPE MessageFormat SYSTEM 'mfl.dtd'>
<MessageFormat name="II_payroll_msg2">
  <FieldFormat name="EMPNO" type="String"/>
  <FieldFormat name="$RPAYDATE" type="String"/>
  <StructFormat name="II_payroll_msg2_rep" repeatField="$RPAYDATE">
    <FieldFormat name="PAYDATE" type="String" accessMode="Controlling"/>
    <FieldFormat name="PAY" type="String" controlName="CALC_PAY"/>
  </StructFormat>
</MessageFormat>
```

This file creates the output format for the data.

When you have finished viewing this file, close it.

3. Now you are ready to run `MsgDefAdmin` to create the format definitions.

From a command prompt, in the directory where you installed Information Integrator, type the following:

```
msgdefadmin -i \sample\payroll\II_payroll_msg1.xml
```

and press Enter. `MsgDefAdmin` processes the first XML file and displays a message when finished.

4. Type the following to process the second XML file:

```
msgdefadmin -i \sample\payroll\II_payroll_msg2.xml
```

and press Enter. `MsgDefAdmin` processes the XML file and displays a message when finished.

4 Building Format Definitions

This chapter describes Information Integrator Formats and gives instructions on using the Formatter application. The following topics are discussed:

- Understanding Information Integrator Formats
- Using the Formatter Interface
- Using Formatter to Build Definitions
- Field Mapping and Transformation
- Example

Understanding Information Integrator Formats

Message formats define the layout of data that is to be processed by the IISERVER. This layout defines the actual “wire format” of the data that is communicated between collaborating applications. Each data item, such as a person’s name, is defined as a field. For example, if a message consisted of an employee’s name and an employee’s Social Security Number, the message contains two fields. For each defined field, an input control and an output control must be defined. The input control indicates how to parse the field, and the output control indicates how to output the field. Groups of fields are called flat formats. Groups of flat formats are termed compound formats. By defining fields, controls, and formats, the layout of the data is made available to the IISERVER component, so that it can perform the defined translation and actions at run-time.

You can build and modify format definitions using one of the following methods:

- **MsgDefAdmin importer utility** — An easy-to-use command-line tool that allows you to store format definitions in the Information Integrator Database. For instructions on using the MsgDefAdmin utility, refer to “Using MsgDefAdmin.”
- **Formatter graphical user interface (GUI)** — Allows you to populate screens with format definition data and store the information in the Information Integrator Database.

Using the Formatter Interface

The Formatter graphical user interface (GUI) allows you to create formats from the “bottom up.” You create the formatter components first; then, you build the format. These components are generic and can be used for more than one format.

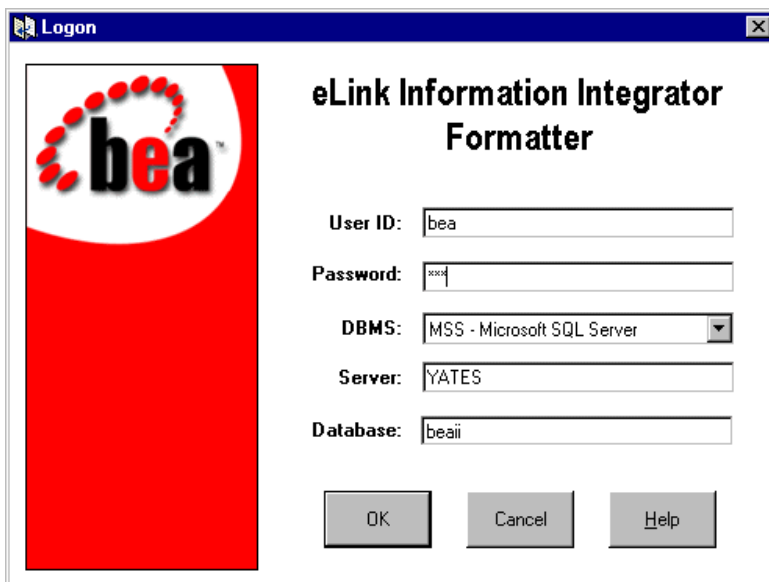
Starting Formatter

Start the Formatter GUI as follows:

1. Double-click the Formatter icon. The Formatter Logon dialog appears as in Figure 4-1.

Note: If you do not have an assigned user ID and password, ask your system administrator to create them for you.

Figure 4-1 Formatter Logon



The screenshot shows a Windows-style dialog box titled "Logon" for the "eLink Information Integrator Formatter". On the left is the BEA logo. On the right, there are five input fields: "User ID" (containing "bea"), "Password" (containing "xxxx"), "DBMS" (a dropdown menu set to "MSS - Microsoft SQL Server"), "Server" (containing "YATES"), and "Database" (containing "beaii"). At the bottom are three buttons: "OK", "Cancel", and "Help".

2. Enter your user ID in the User ID field and press **Tab**.

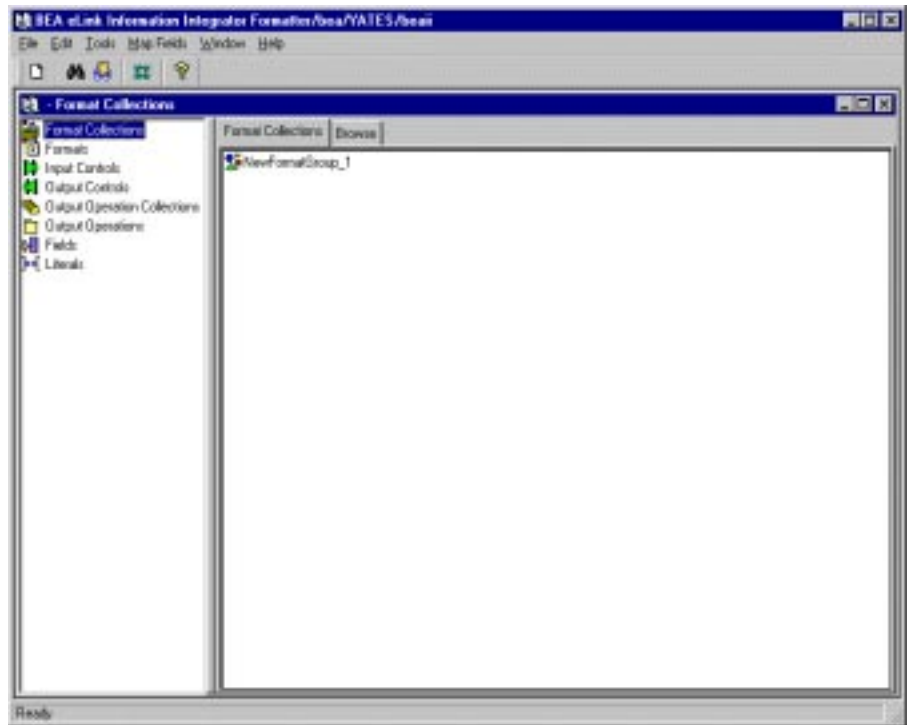
3. Enter your password in the Password field and press **Tab**.
4. Select a database type from the DBMS drop-down list box. The Database field appears or disappears based on your selection.

If you selected the Microsoft SQL server type, specify the server in the Server field and the database in the Database field.

If you selected an Oracle database type, specify the server in the Server field.

5. Click OK. The Formatter main window appears as in Figure 4-2.

Figure 4-2 Formatter Main Window



Formatter is divided into two panes. Format components are displayed in a hierarchical organization in the left pane. When you select a component in the left pane, the right pane of the window displays detailed information about the selected component. The detailed information is kept in property sheets labeled with tabs.

Using the Formatter Window

Each Formatter window is split into two panes. The left pane displays a list of all Format Collections, Formats, Input Controls, Output Controls, Output Operations, Output Operation Collections, Fields, and Literals. Single-click each category to display a list of its contents in the right pane. Double-click each category to expand or display its contents in the left pane. For example, double-click to expand the Formats category to show all defined input and output formats. You can then double-click a format name (or click the + sign to its left) to display its associated fields or formats.

The right pane, or Property Sheet, is where you'll be adding the majority of information. It uses tabbed sheets to display the details for whichever object is currently selected. For example, if you have a field in an input format selected, it displays the Field and Input Control Names for that field on a Properties tab.

Note: You can open more than one Formatter window at a time for access to multiple component types at the same time. This is necessary when assigning fields and input or output controls to formats.

Opening a New Window

To open a new window, select **New** from the File menu.

Tip When the Formatter window opens, it may default to a size too small to display all window components, including the Apply, Cancel, and Help buttons on some property sheets. Maximize the application or resize the window to see all window components.

Renaming Components

Formatter components can be renamed in the left pane of the Formatter window. To rename a component, follow these steps.

1. In the left pane of the Formatter window, select the component name.
2. Click again without moving the mouse a text box appears around the selected component name and is highlighted.
3. Edit the component name.

4. Press ENTER or click the mouse outside the component name text box. If there is no duplicate name, the component name is highlighted and moves to its alphabetical location in the list.

Note: Formatter component names must be 32 characters or less.

Duplicating Components

Formatter components can be duplicated in the left pane of the Formatter window. To duplicate a component, follow these steps.

1. In the left pane of the Formatter window, select any user-defined component and right-click. A popup menu appears.
2. Select Duplicate. A copy of the component is made and a text box appears around the duplicated component name.
3. Type the new component name.
4. Press ENTER or click the mouse outside the component name text box. If there is no duplicate name, the component name is highlighted and moves to its alphabetical location in the list.

Deleting Components

Formatter components can be deleted in the left pane of the Formatter window. To delete a component, follow these steps.

1. In the left pane of the Formatter window, select any user-defined component and right-click. A popup menu appears.
2. Select Delete. A dialog box appears asking if you are sure you want to delete the component.
3. Choose either Yes or No to confirm or cancel your action.

Tips To delete an input format with a field used in conditional branching, use one of the following methods:

- Export the format, delete the field, then import the format.
- Delete the rule from the field, then delete the field from the format.

Using the Used In Tab

The Used In tab displays a list of all places where the selected object is being used. For example, the Used In tab for an input control displays a list of all input formats containing the input control.

To view the Used In tab, follow these steps.

1. With the property sheet open for the object you are interested in, select the Used In tab. The Used In list appears.
2. To go to a specific component where the object is used, select the component and double-click. A new window opens with the component selected.

Using Open Item

The Open Item function opens a new window from within a properties tab or tree. The new window displays the information for the selected component in the tab. You can change the information in this window.

To find information about a component, follow these steps.

1. In a Formatter tab, right-click the item you want to see. A pop-up menu appears.
2. Choose Open Item. A new Formatter window appears displaying information for the selected item.
3. Either view or change the information in the window.

Using Drag and Drop

Use drag and drop to:

- Associate subordinate components from one window into the related Collection tab of another window. Example: associating fields or controls with a flat format.
- Associate input fields with output fields from the Field Map tab of a flat output format.
- Reorder fields (in flat input and output formats)
- Reorder formats (in compound and input and output formats)
- Map fields in the Field Mapping window
- Assign either a single or multiple output operations to an output control from the Output Operation Extended Properties tab
- Reorder the sequence of operations from the Output Operation Collections tab

Drag and drop works across both windows and panes. To copy an object using drag and drop, you must drag the object from the first window's tree to the second window's tab. You cannot drag and drop in the same pane unless you are reordering fields, formats, or output operations in a collection or using field mapping.

You can only drag and drop at the root level of the left pane tree view. When you drop an object, it's always added to the end of the list. Note that you can't drag and drop literals. You must select them from the appropriate drop-down lists.

Using Formatter to Build Definitions

The steps involved in creating a format definition using the Formatter GUI are as follows.

1. Defining Literals
2. Defining Fields
3. Defining Output Operations
4. Creating Output Operation Collections
5. Defining Input Controls
6. Defining Output Controls
7. Building Formats

These steps are described in the following sections.

Defining Literals

A literal is a string of characters that Formatter processes as:

- Delimiters, prefixes and suffixes
- Default values
- Trim and pad characters
- Tag values
- Data Validation values

Formatter uses literals in input controls, output controls, operations, format terminators, and repeating sequence terminators. Valid literals include the printable characters A-Z, a-z, 0-9, and any keyboard character available by pressing **Shift** with another printable character, as well as hex values that can include printable and non-printable characters.

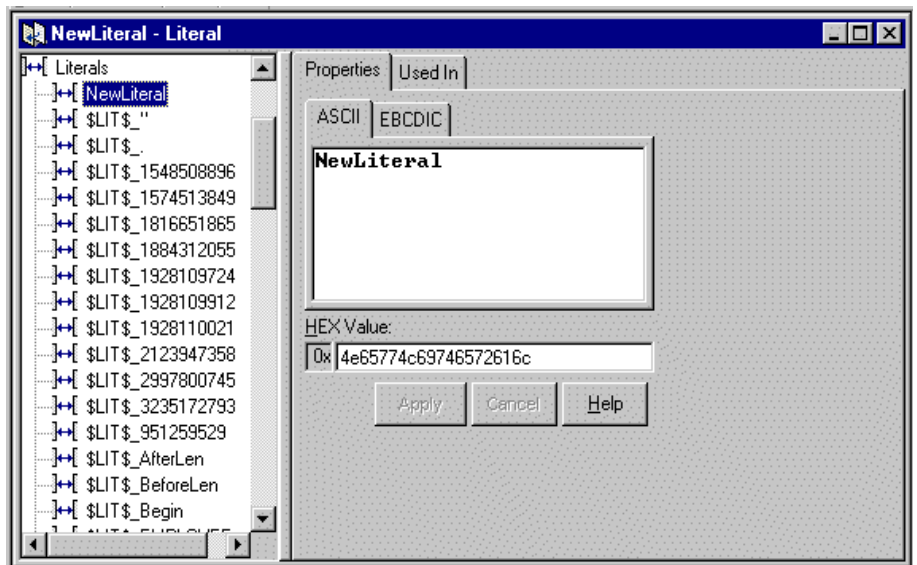
- Notes:** The maximum character length of a literal is 127.
Do not use single quotes in literal names.
When literals are used as pad characters, only the first character is used.

Creating a Literal

To create a Literal, follow these steps.

1. Right-click on Literals in the left pane of the Formatter window.
2. Select New from the pop up menu. A new literal component is added to the left pane, and the Literal property sheet is displayed in the right pane as shown in Figure 4-3.

Figure 4-3 Literal Property Sheet



3. Type the name you want to use to identify this Literal in the text box.
Note: The literal name must be 32 characters or less. Single quotes, double quotes, and spaces should not be used in names.
4. Press **Enter** to finish defining the literal. The name is highlighted and moves to its alphabetical location in the list. This name is the default value of the literal.

5. Enter the value for the literal in the appropriate Property tab. You can enter ASCII or EBCDIC values in their associated tabs or enter hex values in the HEX Value window. Note that the ASCII and EBCDIC fields have a 127-character maximum (which equals a 254-character hex value).
6. Click Apply to save your changes to the Database.

Tip To change the name of a literal in the list:

1. Select the literal name.
2. Click again without moving the mouse. The literal name is now highlighted and in a text box.
3. Type the new name.
4. Press ENTER or click the mouse outside the name text box. The literal name is highlighted and moves to its alphabetical location in the list.

Defining Fields

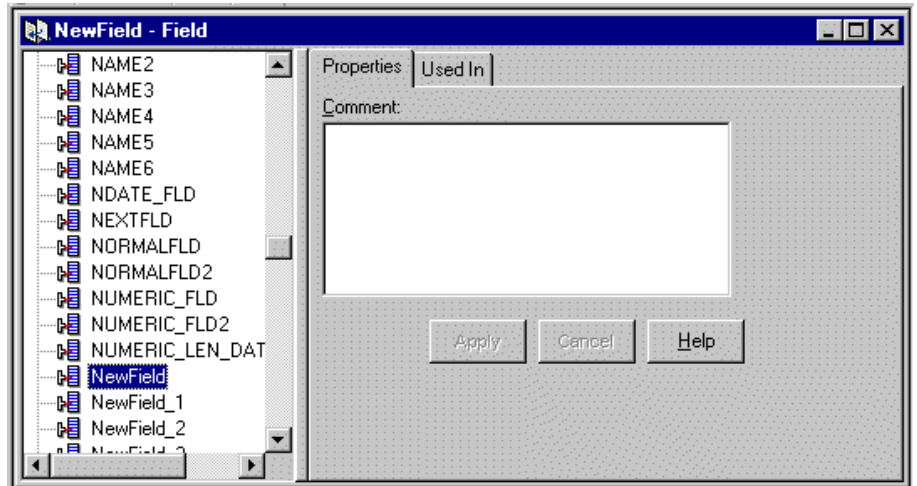
Fields are named items of message data and are the smallest possible container for information. In Formatter, the field name is the link between the input and output data. Each field can be used in multiple formats and associated with different input and output controls. To identify a field in a message, a combination of the field name and a control is used.

Creating a Field

To create a field, follow these steps.

1. Right-click on Fields in the left pane of the Formatter window.
2. Select New from the pop up menu. A new field component is added to the left pane and the fields property sheet is displayed in the right pane as shown in Figure 4-4.

Figure 4-4 Fields Property Sheet



3. Type a unique field name in the text box.
4. Press **Enter** to finish defining the field. The field name moves to its alphabetical location in the list and is highlighted.
5. Optionally click the Comment field on the property sheet to add a descriptive comment for the field and type the comment.

Notes: The Comment field does not retain formatted text when imported or exported.

Using carriage returns in the Comment field is not recommended.

6. Click Apply to save your changes to the Database.

Tip To change the name of a field:

1. Select the field name.
2. Click again without moving the mouse. The field name is now highlighted and in a text box.
3. Type the new name.
4. Press ENTER or click the mouse outside the name text box. The field name is highlighted and moves to its alphabetical location in the list.

Defining Output Operations

Output operations provide the different actions that can be performed on an output field. For example, using output operations, you can change the case of output data, perform mathematical expressions based on input field contents, and extract substrings.

Through the use of output operation collections, you can collect operations to perform them sequentially. For example, you can left justify and right trim a substring of the contents of an input field. The order in which these operations are defined in the collection is the order in which they are performed.

Available output operations are described in the following sections.

Case

Case operations affect the case of the field data. Information Integrator includes two pre-defined case operations, as described below:

- **LOWER_CASE** — Converts String data to all lowercase letters. For example, **ABC Corporation** becomes **abc corporation**.
- **UPPER_CASE** — Converts String data to all uppercase letters. For example, **ABC Corporation** becomes **ABC CORPORATION**.

Default

Default operations provide a default value for a field if an input field either does not exist in the input message or has a length of zero. To define a default operation, follow the steps below.

1. Select the Default operation category in the left pane and right-click. A popup menu appears.
2. Select New. A new operation is added to the left pane.

The cursor is positioned in the text box where you can type a unique operation name.

Note: Default operation names have a 32-character maximum length.

3. Press **Enter** to finish defining the Default operation. The new entry moves to its alphabetical location in the list and is highlighted.

4. Select the Properties tab in the right pane.
5. Select a literal from the Default Value drop-down list to use as the default value.
6. Click Apply to save your changes to the database.

Justification

Justification operations justify field data to the left, center, or right within the length of the field. Information Integrator includes three pre-defined Justification operations, as described below:

- `LEFT_JUSTIFY` — Left justifies the data in the field.
- `RIGHT_JUSTIFY` — Right justifies the data in the field.
- `CENTER` — Centers the data in the field.

Note: For pad characters, a Justification operation must precede a Length operation in a collection (see “Length” for more information). To specify a pad character other than the default (space), select the desired pad character in the associated Length operation. If `CENTER` is specified for Justification, the data can be padded on both the left and right.

Length

Length operations ensure that an output string is given a length. If the data length is longer than the specified length, the data is truncated. If the data length is shorter than the specified length, pad characters are used to fill in the remaining field length. Non-numeric data types are padded on the right; numeric data types are padded on the left.

Follow the steps below to define a Length operation.

1. Right-click the Length operation category in the left pane of the Formatter window. A popup menu appears.
2. Select New. A new operation is added to the left pane.
3. Type a new, unique Length operation name.

Note: Length Operation names have a 32-character maximum length.

4. Press **Enter** to finish defining the Length operation. The new entry moves to its alphabetical location in the list and is highlighted. The right pane displays the Length Properties tab.

5. Select a pad character from the drop-down list if the data length is less than the length specified for the output.

Pad characters are used to fill space to reach a specified field length, or when the Justify operations are used. For example, if the data length is 4, the field length is 7, and the padding character is an asterisk (*), the field would look as follows:

```
data***
```

6. Enter the exact length for the output field in the Length field.
7. Click Apply to save your changes to the Database.

Math Expression

Using Math Expression operations, you can output a value resulting from an arithmetic expression. The expression can be built using arithmetic operators, constants, and input field values.

The operators you can use in Math Expressions are: + - * / () and Unary -. These operators are processed in the following order: () * / + -. The operands you can use in Math Expressions are: Numeric, Constants, and Input Field Names.

Note: Field names with spaces or underscores must be surrounded by single or double quotes.

To define a Math Expression, follow the steps below.

1. Right-click the Math Expression output operation in the left pane of the Formatter window. A pop-up menu appears.
2. Select New. A new operation is added to the left pane.
3. Type a new, unique Math Expression name.
Note: Math Expression names have a 32-character maximum length.
4. Press **Enter**. The new entry is highlighted and moves to its alphabetical location in the list.
5. In the Decimal Precision text box, type the number of decimal points to which you want the expression result rounded.
6. Select whether the expression result should be rounded up or down.

7. Type the expression in the Math Expression field. Fields defined in mathematical expressions must be in quotes (for example, “field1” = “field2”).

Note: Field names cannot contain dashes.

8. Click Apply to save your changes to the Database.

Prefix/Suffix

Prefix/Suffix operations allow you to specify a literal for use as a prefix or suffix. Prefixes are added to the beginning of an output field. Suffixes are added to the end of an output field. For example, you may want to place a less-than sign (<) before and a greater-than sign (>) after the output data. This would require a Prefix of < and a Suffix of >. Any defined literal may be used as a prefix or suffix.

To define a prefix/suffix operation, follow the steps below.

1. Right-click the Prefix/Suffix operation category in the left pane of the Formatter window. A pop-up menu appears.
2. Select New. A new operation is added to the left pane.
3. Type a new, unique Prefix/Suffix operation name.
Note: Prefix/Suffix operation names have a 32-character maximum length.
4. Press **Enter** to finish defining the Prefix/Suffix operation. The new entry moves to its alphabetical location in the list and is highlighted. The Properties tab displays in the right pane.
5. Select either Prefix or Suffix from the drop-down list. Select Prefix if the literal is to come before the output data or Suffix if the literal is to come after the output data.
6. Select a literal from the Value drop-down list.
7. Select the Null Action box if the input field is not present in the input message.
8. Click Apply to save your changes to the Database.

Substitute

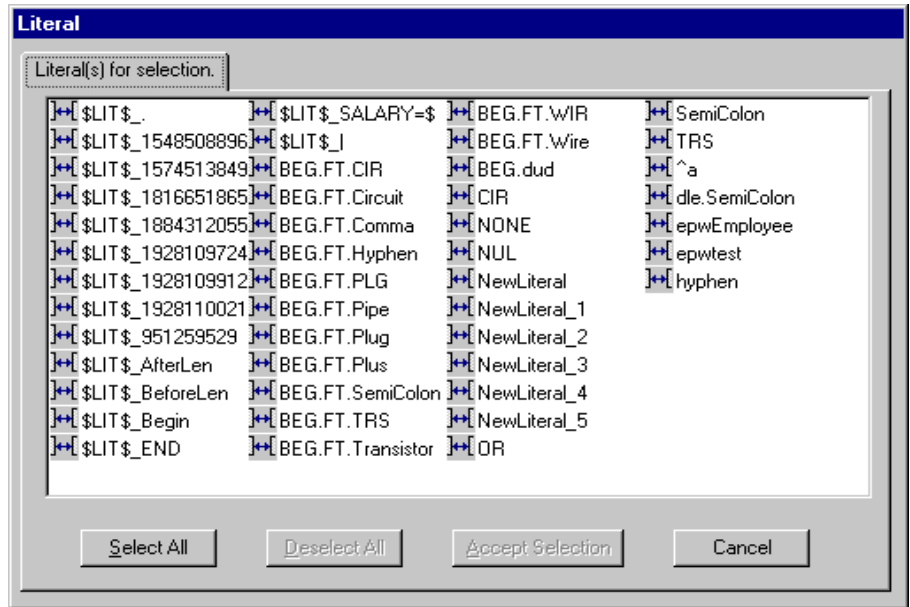
Substitute operations enable you to define a list of input strings to substitute and the output strings to replace them. For each substitute item within a substitute operation, you define a literal to look for as the input value, a literal to replace it with, and the data type in which to output the new data.

Note: You can add a literal with the value `NONE` and assign it an output value. This acts as a default substitution if there is no match for the input value.

To define a Substitute operation, follow the steps below.

1. Right-click the Substitute operation in the left pane of the Formatter window. A pop-up menu appears.
2. Select New. A new operation is added to the left pane.
3. Type a new, unique Substitute operation name.
Note: Substitute operation names have a 32-character maximum length.
4. Press **Enter** to finish defining the Substitute operation. The new entry is highlighted and moves to its alphabetical location in the list.
5. Right-click the new Substitute operation in the left pane of the Formatter window. A pop-up menu appears.
6. Select Add Substitute Items. The Literal dialog appears as in Figure 4-5.

Figure 4-5 Literal



7. Select the literals you want to use as input strings and click Accept Selection. The Substitute items are added under the Substitute operation in the left pane.
8. Click one of the new Substitute items in the left pane. The corresponding Properties tab appears in the right pane.
9. Select the following:
 - A data type from the Output Data Type drop-down list. For an explanation of the supported data types, please refer to “Data Types.”
 - A literal from the Output Value drop-down list. This value is substituted for the value specified in the Input Value drop-down list.
10. Click Apply to save your changes to the Database.
11. Repeat steps 8, 9, and 10 for each substitute item.

Substring

Substring operations extract part of an input field's contents (defined by byte position and length) and places it in the output field. Only the String, Numeric, EBCDIC, and Binary data types can be used for substrings.

To define a substring operation, follow the steps below:

1. Right-click the Substring operation category in the left pane of the Formatter window. A popup menu appears.
2. Select New. A new operation is added to the left pane.
3. Type a new, unique Substring operation name.
Note: Substring operation names have a 32-character maximum length.
4. Press **Enter** to finish defining the substring. The new entry moves to its alphabetical location in the list and is highlighted. The Properties tab displays in the right pane.
5. Specify the literal to use as a pad character in the Pad Character drop-down list. The pad character is used if the length of the substring is greater than the length of the data.
6. Specify the starting position for the substring in the Substring Start field.
7. Select End of Field if you want the substring to extend from the specified Start field to the end of the field. Or, specify the length of the substring in the Substring Length field.
8. Click Apply to save your changes to the Database.

Trim

Trim operations remove a defined trim character from the right and left of the output data.

To define a Trim operation, follow the steps below.

1. Right-click the Trim operation category in the left pane of the Formatter window. A popup menu appears.
2. Select New. A new operation is added to the left pane.
3. Type a new, unique Trim operation name.

Note: Trim operation names have a 32-character maximum length.

4. Press **Enter** to finish defining the Trim operation. The new entry moves to its alphabetical location in the list and is highlighted. The Properties tab displays in the right pane.
5. Select the literal you want to trim from the Trim Value drop-down list.
6. Select where you want to remove the Trim Value from the output field in the Trim Type drop-down list.
7. Click Apply to save your changes to the Database.

Creating Output Operation Collections

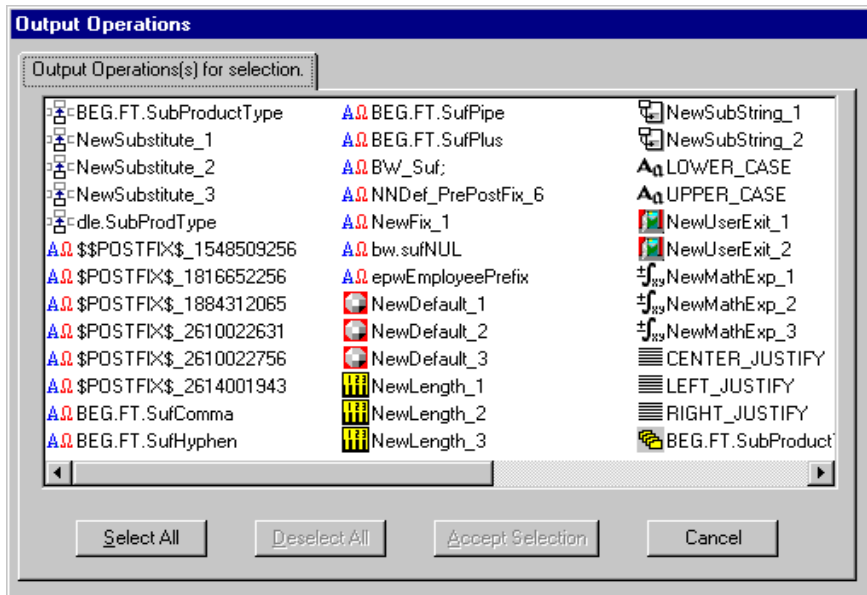
Use Output Operations Collections to group and sequence a series of output operations and other output operation collections. Operations are executed in the order in which they appear.

Follow the steps below to create an Output Operation Collection.

1. Right-click on Output Operation Collection in the left pane of the Formatter window. A pop-up menu appears.
2. Select New from the pop-up menu to create a new Output Operation Collection entry in the left pane.
3. Type a unique Output Operation Collection name in the text box.

Note: The name must be 32 characters or less. Single quotes, double quotes, and spaces should not be used in names.
4. Press **Enter** to finish defining the Output Operation Collection. The new entry moves to its alphabetical location in the list and is highlighted.
5. Right-click the new collection and choose Add Output Operations. The Output Operations window appears as in Figure 4-6.

Figure 4-6 Output Operations



6. Select the operations you want to add and click Accept Selection.

Notes: To reorder component output operations or output operation collections, click and drag the component to the highest level of the collection.

To move one operation to a partition preceding another operation, drag the operation that you want to move and place it on top of the operation that it should precede. You can rearrange the other components to fit the new order using this method.

Defining Input Controls

To build input formats, you must build input controls. Input controls are used to parse input data. The IISERVER uses input controls to determine how to find the beginning and end of the data in a field.

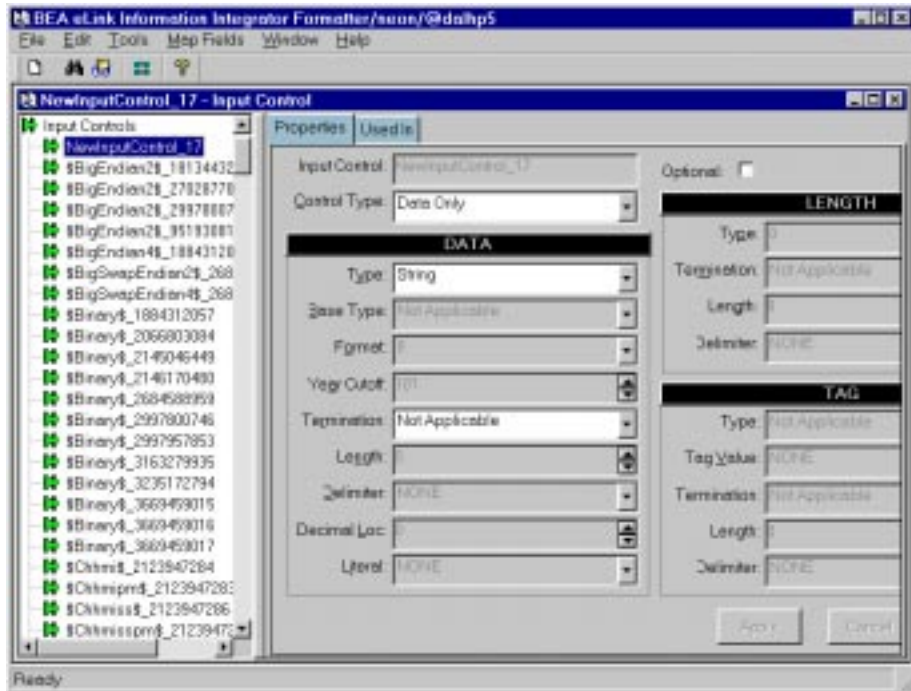
An input control also determines whether the data for a field is mandatory or optional. Mandatory means that Formatter considers the parse successful if the parse start and end strings are found, even if the associated data is empty. If the parse is successful, the field passes the mandatory test and the parse continues. If the parse parameters are not found, the field fails the mandatory test and the parse fails for the rest of the format. Optional means that the parse continues even if parse parameters for the field are not found.

Input controls can use tags to separate data. Tags are sets of bits or characters explicitly defining a string of data. For example, <NAME> could mark the beginning of a name field in a message.

You can create an input control using the steps described in the following procedure.

1. Right-click on Input Controls in the left pane of the Formatter window.
2. Select New. A new input control component is added to the left pane, and the input control property sheet is displayed in the right pane as shown in Input Control Property Sheet.

Figure 4-7 Input Control Property Sheet



3. Type a descriptive input control name in the text box.

Note: The input control name must be 32 characters or less. Single quotes, double quotes, and spaces should not be used in names.

Press **Enter** to finish defining the input control. The new input control moves to its alphabetical location in the list in the left pane and is highlighted.

The procedures for creating each input control type are described in the following sections.

4 Building Format Definitions

Examples of Input Controls

The following table contains examples of fields that might occur in message and the corresponding Input Control definition. In the table below, the following are previously defined Literals: BEA.Comma, <Name>, *Server, ^\\$\$

Example Field in Message	Description	Input Control Definition
John Doe,	String Delimited by a comma	Control Type = Data Only Type = String Termination = Delimiter Delimiter = BEA.Comma
76201	String having an exact length	Control Type = Data Only Type = String Termination = Exact Length Length = 5
<Name>John Doe,	String preceded by a tag with a delimiter	Control Type = Tag and Data Type = String Termination = Delimiter Delimiter = BEA.Comma Tag Type = String Tag Value = <Name> Tag Termination = Not Applicable
<Name>8,John Doe	String with a tag that has a length given in the order: tag, length, data	Control Type = Tag-Length and Data Type = String Length Type = Numeric Length Termination = Delimiter Length Delimiter = BEA.Comma Tag Type = String Tag Value = <Name> Tag Termination = Not Applicable
8,<Name>John Doe	String with a tag that has a length given in the order: length, tag, data	Control Type = Length-Tag and Data Type = String Length Type = Numeric Length Termination = Delimiter Length Delimiter = BEA.Comma Tag Type = String Tag Value = <Name> Tag Termination = Not Applicable

Example Field in Message	Description	Input Control Definition
8,John Doe	String that has a length given	Control Type = Length and Data Type = String Length Type = Numeric Length Termination = Delimiter Length Delimiter = BEA.Comma
3	Field that specifies the number of occurrences of another field, 1 digit only	Control Type = Repetition Count Type = Numeric Termination = Exact Length Length = 1
*Server	Literal that must be present in message	Control Type = Literal Termination = Not Applicable Literal = *Server
\$198	String that begins with a \$ followed by white space.	Control Type = Regular Expression Termination = White Space Delimited Reg Exp = ^\\$

Creating a Data Only Input Control

A Data Only input control specifies that, when parsing a message, any field associated with this input control has a data component only, not an associated length or tag data. For example, some message formats may have simple fields with Strings terminated by commas. The input control might look like:

```
Input Control Name: AsciiStringToComma
Control Type: Data Only
Optional/Mandatory: Mandatory
(Data) Type: String
(Data) Termination: Delimiter
(Data) Delimiter: Comma (Previously Defined Literal)
```

Note: The Input Control Type determines which fields require information. Fields containing “Not Applicable” do not require a value.

To create a Data Only input control, follow these steps.

1. With Data Only selected as the Input Control type, select a data type from the Type drop-down list.

2. If the selected data type is Date and Time, Date, Time, or Custom Date and Time, select the Base Type (String or Numeric) the data is output as.
3. If the selected data type is Custom Date and Time, select a date/time format from the Format drop-down list.
4. If your Custom Date and Time format string includes a 2-digit year, you also must choose a Year Cutoff value. The Year Cutoff value tells Formatter how to convert the 2-digit year to a 4-digit year. See Year Cutoff for details.
5. Select a termination type from the Termination drop-down list.
6. If you select Delimiter or Minimum Length + Delimiter as a termination type, select one from the Delimiter drop-down list.
7. If you select Exact Length, Minimum Length + Delimiter, or Minimum Length + White Space, enter the fixed length of the field or the minimum number of characters to parse before looking for a literal or white space in the Length field.
8. If the data type requires a decimal, the Decimal Loc field is active. Enter a decimal location or use the arrows to select the desired number. Only a positive value is allowed and it must not be more than the number of digits the field can hold. The value determines how many digits are to the right of the decimal location when Formatter interprets the message data. For example, an input value of 12345 with a decimal location of 2 is interpreted as 123.45.

Note: Note: Decimal location pertains only to IBM packed and zoned decimal data.

Creating a Tag and Data Input Control

The Tag & Data input control type indicates that there is a literal tag preceding the data. For example, some message formats use tagged fields to indicate the beginning of different sections of a message. If you're defining the start of the heading for a memo, you might define a Tag & Data input control specifying the To: section as: <MemoHead>FirstName, LastName. The input control for the first field might look like:

```
Input Control Name: AsciiStringMemoHeadTagToComma
Control Type: Tag & Data
Optional/Mandatory: Mandatory
Type: String
Termination: Delimiter
Delimiter: Comma (Previously Defined)
```

(Tag) Type: String
Tag Value: <MemoHead>
(Tag) Termination: Not Applicable

To create a Tag & Data input control, follow these steps.

1. With Tag & Data selected as the Input Control type, in the Tag section, select a data type for the tag data from the Type drop-down list.
2. From the Tag Value drop-down list, select a literal as the actual tag value to be searched for.

Note: The literal for the Tag Value must have been created previously.

3. In the Tag section, select the termination type for the tag from the Termination drop-down list.

The termination type tells Formatter when to stop parsing the tag value. When the tag delimiter is reached, the value that was parsed is compared to the value in the tag value field to determine if the data being searched for has been found.

4. If the tag termination is fixed-length, specify the exact tag length in the Length field.
5. If the tag termination type is Delimiter or Minimum Length + Delimiter, select a literal from the Delimiter drop-down list.
6. In the Data section, select the data type from the Type drop-down list.
7. If the selected data type is Date and Time, Date, Time, or Custom Date and Time, select the Base Type (String or Numeric) the data is output as.
8. If the selected data type is Custom Date and Time, select a date/time format from the Format drop-down list.
9. If your Custom Date and Time format string includes a 2-digit year, you also must choose a Year Cutoff value. The Year Cutoff value tells Formatter how to convert the 2-digit year to a 4-digit year. See Year Cutoff for details.
10. Select a termination type from the Termination drop-down list.
11. If the data termination type is Delimiter or Minimum Length + Delimiter, select a literal from the Delimiter drop-down list.

12. If you select Exact Length, Minimum Length + Delimiter, or Minimum Length + White Space, enter the fixed length of the field or the minimum number of characters to parse before looking for a literal or white space in the Length field.
13. If the data type for the data requires a decimal, the Decimal Loc field is active. Enter a decimal location or use the arrows to select the desired number. Only a positive value is allowed and it must not be more than the number of digits the field can hold. The value determines how many digits are to the right of the decimal location when Formatter interprets the message data. For example, an input value of 12345 with a decimal location of 2 is interpreted as 123.45.
Note: Decimal location pertains only to IBM packed and zoned decimal data.

Creating a Tag, Length, and Data Input Control

Tag, Length & Data input controls specify that field data starts with a tag, includes embedded length information, and contains data. For example, you might have a tagged field in a random ordered input format that includes an embedded length. If you're defining a tagged, variable-length price figure, you might define the field to look like: <Price>7,1035.25,.

The input control might look like:

```
Input Control Name: PriceTagAndLengthToComma
Control Type: Tag, Length & Data
Optional/Mandatory: Optional
(Data) Type: String
(Length) Type: Numeric
(Length) Termination: Delimiter
(Length) Delimiter: Comma
(Tag) Type: String
Tag Value: <Price>
(Tag) Termination: Exact Length
(Tag) Length: 7
```

To create a Tag, Length & Data input control, follow these steps.

1. With Tag, Length & Data selected as the Input Control type, in the Tag section, select a data type for the tag data from the Type drop-down list.
2. From the Tag Value drop-down list, select a literal as the actual tag value to be searched for.
3. In the Tag section, select the termination type for the tag from the Termination drop-down list.

The termination type tells Formatter when to stop parsing the tag value. When the tag delimiter is reached, the value that was parsed is compared to the value in the tag value field to determine if the data being searched for has been found.

4. If the tag termination is fixed-length, specify the exact tag length in the Length field.
5. If the tag termination type is Delimiter or Minimum Length + Delimiter, select a literal from the Delimiter drop-down list.
6. In the Length section, select a data type for the length from the Type drop-down list.
7. Select the termination type for the length portion of the data from the Termination type.
8. If you select Delimiter or Minimum Length + Delimiter as a termination type, select a literal from the Delimiter drop-down list.
9. If the Length termination type is Exact Length, Minimum Length + Delimiter, or Minimum Length + White Space, enter the fixed length or the minimum number of characters to parse before looking for a delimiter or white space.
10. In the Data section, select the data type from the Type drop-down list.
11. If the selected data type is Date and Time, Date, Time, or Custom Date and Time, select the Base Type (String or Numeric) the data is output as.
12. If the selected data type is Custom Date and Time, select a date/time format from the Format drop-down list.
13. If your Custom Date and Time format string includes a 2-digit year, you also must choose a Year Cutoff value. The Year Cutoff value tells Formatter how to convert the 2-digit year to a 4-digit year. See Year Cutoff for details.
14. Select a termination type from the Termination drop-down list.
15. If the data termination type is Delimiter or Minimum Length + Delimiter, select a literal from the Delimiter drop-down list.
16. If you select Exact Length, Minimum Length + Delimiter, or Minimum Length + White Space, enter the fixed length of the field or the minimum number of characters to parse before looking for a delimiter or white space in the Length field.

17. If the data type for the data requires a decimal, the Decimal Loc field is active. Enter a decimal location or use the arrows to select the desired number. Only a positive value is allowed and it must not be more than the number of digits the field can hold. The value determines how many digits are to the right of the decimal location when Formatter interprets the message data. For example, an input value of 12345 with a decimal location of 2 is interpreted as 123.45.

Note: Note: Decimal location pertains only to IBM packed and zoned decimal data.

Creating a Length and Data Input Control

Length & Data input controls can be used to parse fields with variable length data. The embedded length indicates the length for the actual field data. For example, messages containing the name and selling price for various pieces of equipment. The bigger the piece of the equipment, the bigger the price. Keeping the messages as compact as possible might be important if you were sending them over a phone line. We will send the length of the selling price, then the price itself:

```
Input Control Name: AsciiNumericLengthAndPrice
Control Type: Length & Data
Optional/Mandatory: Mandatory
(Data) Type: String
(Length) Type: Numeric
(Length) Termination: Delimited
(Length) Delimiter: Comma
```

To create a Length & Data input control, follow these steps.

1. With Length & Data Only selected as the Input Control type, in the Length section, select a data type for the length from the Type drop-down list.
2. Select the termination type for the length portion of the data from the Termination type.
3. If the Length termination type is Exact Length or Minimum Length + Delimiter, select a literal from the Delimiter drop-down list.
4. If you select Exact Length, Minimum Length + Delimiter, or Minimum Length + White Space, enter the fixed length of the field or the minimum number of characters to parse before looking for a delimiter or white space in the Length field.
5. In the Data section, select the data type from the Type drop-down list.

6. If the selected data type is Date and Time, Date, Time, or Custom Date and Time, select the Base Type (String or Numeric) the data is output as.
7. If the selected data type is Custom Date and Time, select a date/time format from the Format drop-down list.
8. If your Custom Date and Time format string includes a 2-digit year, you also must choose a Year Cutoff value. The Year Cutoff value tells Formatter how to convert the 2-digit year to a 4-digit year. See Year Cutoff for details.
9. Select a termination type from the Termination drop-down list.
10. If the data termination type is Delimiter or Minimum Length + Delimiter, select a literal from the Delimiter drop-down list.
11. If the data termination type is Exact Length, Minimum Length + Delimiter, or Minimum Length + White Space, enter the fixed length of the field or the minimum number of characters to parse before looking for a delimiter or white space.
12. If the data type for the data requires a decimal, the Decimal Loc field is active. Enter a decimal location or use the arrows to select the desired number. Only a positive value is allowed and it must not be more than the number of digits the field can hold. The value determines how many digits are to the right of the decimal location when Formatter interprets the message data. For example, an input value of 12345 with a decimal location of 2 is interpreted as 123.45.

Note: Note: Decimal location pertains only to IBM packed and zoned decimal data.

Creating a Repetition Count Input Control

Repetition Count input controls are used to indicate the number of times a component will repeat in the message. For example, an input format may include the daily sales figures for a traveling sales person. Some people in sales send daily updates as to the total amount sold per day. Others might send weekly updates, grouping 7 days together in a single message. The field containing the number of days being sent would need a repetition count input control like the following:

```
Input Control Name: AsciiNumericSalesRepCount
Control Type: Repetition Count
Optional/Mandatory: Mandatory
(Data) Type: Numeric
```

4 Building Format Definitions

```
(Data) Termination: Delimiter  
(Data) Delimiter: Comma
```

To create a Repetition Count input control, follow these steps.

1. With Repetition Count selected as the Input Control type, select a data type for the repetition count from the Type drop-down list.
2. Select a termination type from the Termination drop-down list.
3. If the termination type is Delimiter or Minimum Length + Delimiter, select a literal from the Delimiter drop-down list.
4. If you select Exact Length, Minimum Length + Delimiter, or Minimum Length + White Space, enter the fixed length of the field or the minimum number of characters to parse before looking for a delimiter or white space in the Length field.
5. If the data type requires a decimal, the Decimal Loc field is active. Enter a decimal location or use the arrows to select the desired number. Only a positive value is allowed and it must not be more than the number of digits the field can hold. The value determines how many digits are to the right of the decimal location when Formatter interprets the message data. For example, an input value of 12345 with a decimal location of 2 is interpreted as 123.45.

Note: Decimal location pertains only to IBM packed and zoned decimal data.

Creating a Literal Input Control

Formatter uses Literal input controls to make sure a specific literal value is in a message. If the message does not contain the literal and the Literal input control is mandatory, the parse fails. For example, a literal might mark the beginning or end of a component format within a compound input format. It may be as simple as "PART2:" as shown below:

```
Input Control Name: PART2Literal  
Optional/Mandatory: Mandatory  
Control Type: Literal  
(Data) Termination: Delimiter  
(Data) Delimiter: Colon  
(Data) Literal: PART2
```

To create a Literal input control, follow these steps.

1. With Literal selected as the Input Control type, select the data type from the Type drop-down list.
2. Select the literal in the drop down list.
3. Select a termination type from the Termination drop-down list.
4. If you select Delimiter or Minimum Length + Delimiter as a termination type, select one from the Delimiter drop-down list.
5. If you select Exact Length, Minimum Length + Delimiter, or Minimum Length + White Space, enter the fixed length of the field or the minimum number of characters to parse before looking for a delimiter or white space in the Length field.
6. If the data type requires a decimal, the Decimal Loc field is active. Enter a decimal location or use the arrows to select the desired number. Only a positive value is allowed and it must not be more than the number of digits the field can hold. The value determines how many digits are to the right of the decimal location when Formatter interprets the message data. For example, an input value of 12345 with a decimal location of 2 is interpreted as 123.45.

Note: Decimal location pertains only to IBM packed and zoned decimal data.

Creating a Length, Tag, and Data Input Control

Length, Tag & Data input controls specify that field data starts with embedded length information, has a tag, and contains data. For example, you might have a tagged field in a random ordered input format that includes an embedded length. If you're defining a tagged, variable-length price figure, you might define the field to look like:

```
7,<Price>1035.25,.
```

The input control might look like:

```
Input Control Name: PriceLengthAndTagToComma
Optional/Mandatory: Optional
Control Type: Length, Tag & Data
(Data) Type: String
(Data) Termination: Delimiter
(Data) Delimiter: Comma (Previously Defined)
(Length) Type: Numeric
(Length) Termination: Delimiter
(Length) Delimiter: Comma
(Tag) Type: String
```

Tag Value: <Price>
(Tag) Termination: Exact Length
(Tag) Length: 7

To create a Length, Tag & Data input control, follow these steps.

1. With Length, Tag & Data selected as the Input Control type, in the Length section, select the data type for the length from the Type drop-down list.
2. If the selected data type is Date and Time, Date, Time, or Custom Date and Time, select the Base Type (String or Numeric) the data is output as.
3. If the selected data type is Custom Date and Time, select a date/time format from the Format drop-down list.
4. If your Custom Date and Time format string includes a 2-digit year, you also must choose a Year Cutoff value. The Year Cutoff value tells Formatter how to convert the 2-digit year to a 4-digit year. See Year Cutoff for details.
5. Select the termination type for the length portion of the data from the Termination type.
6. If you select Delimiter or Minimum Length + Delimiter as a termination type, select a literal from the Delimiter drop-down list.
7. If you select Exact Length, Minimum Length + Delimiter, or Minimum Length + White Space, enter the fixed length of the field or the minimum number of characters to parse before looking for a delimiter or white space in the Length field.
8. In the Tag section, select a data type for the tag data from the Type drop-down list.
9. From the Tag Value drop-down list, select a literal as the actual tag value to be searched for.
10. In the Tag section, select the termination type for the tag from the Termination drop-down list.

The termination type tells Formatter when to stop parsing the tag value. When the tag delimiter is reached, the value that was parsed is compared to the value in the tag value field to determine if the data being searched for has been found.
11. If the tag termination is fixed-length, specify the exact tag length in the Length field.

12. If the tag termination type is Delimiter or Minimum Length + Delimiter, select a literal from the Delimiter drop-down list.
13. In the Data section, select the data type from the Type drop-down list.
14. Select a termination type from the Termination drop-down list.
15. If the data termination type is Delimiter or Minimum Length + Delimiter, select a literal from the Delimiter drop-down list.
16. If the data termination type is Exact Length, Minimum Length + Delimiter, or Minimum Length + White Space, enter the fixed length of the field or the minimum number of characters to parse before looking for a delimiter or white space.
17. If the data type for the data requires a decimal, the Decimal Loc field is active. Enter a decimal location or use the arrows to select the desired number. Only a positive value is allowed and it must not be more than the number of digits the field can hold. The value determines how many digits are to the right of the decimal location when Formatter interprets the message data. For example, an input value of 12345 with a decimal location of 2 is interpreted as 123.45.

Note: Decimal location pertains only to IBM packed and zoned decimal data.

Creating a Regular Expression Input Control

Regular Expression (RE) input controls express rules for string pattern matching. Instead of direct character-by-character matches, the input control value is interpreted as a regular expression to match the input. String-matching capabilities for this feature comply with the POSIX 1003.2 standard for regular expressions. You can only build REs for ASCII data types (String or Numeric). Within REs, only printable ASCII characters are valid.

The following rules apply for REs:

- Ordinary characters (not specially defined for REs) act as one-character REs to match themselves. For example, if your RE is “X” in a parse control for a repeating field and your message is “X,Y,Z”, you will get a match on the first repeating field.
- Backslashes (\) followed by special RE characters act as one-character REs that match the special character. Periods (.), asterisks (*), left square brackets ([), and backslashes (\) are special unless they are within square brackets (see below).

Note: Carets (^) and dollar signs (\$) are not supported. Do not use them in REs without preceding them with a backslash character (\).

- Periods (.) act as one-character REs that match any character except a new line character. For example, if your RE is “.” in a parse control for a field and the contents of the field is “This is a sample.”, you won’t get a match for the field (no space following the sentence).
- Non-empty strings of characters enclosed in square brackets ([]) act as one-character REs that match any one character in the enclosed string. “[a]” searches for the letter “a” in a fields contents.

The minus (-) character can be used to indicate a range of consecutive characters. For example, [0-9] is equivalent to [0123456789].

Note that the right square bracket (]) does not terminate such a string when it is the first character within it (after an initial caret, if any). For example “[a-f]” matches a right square bracket or one of the ASCII letters a through f inclusive. Also note that the four special characters (“.”, “*”, “\”, and “[”) represent themselves within the square brackets, so “[*]” searches for an asterisk within field contents.

- A one-character RE followed by an asterisk (*) matches zero (0) or more occurrences of the RE. A one-character RE followed by a plus sign (+) matches zero (0) or one occurrence of the RE. If there are multiple strings matching the RE, the longest leftmost string permitting a match is chosen.
- A one-character RE followed by {m}, {m,}, or {m,n} matches a range of occurrences of the one-character. m and n must be non-negative integers less than 256. {m} matches exactly m occurrences. {m,} matches at least m occurrences. And {m,n} matches any number of occurrences between m and n inclusive. If a choice exists, the RE matches as many occurrences as possible.

For example, “a{3,}” matches 3 or more concatenated “a” characters. This could also be done by REs of “aaa+” or “aaa*”. 7. If REs are concatenated, the merged RE matches the concatenation of the strings matched by each component of the RE. For example, “XY” matches strings containing those two letters side-by-side.

- A RE enclosed between parentheses (“(,)”) is a RE that matches whatever the non-parenthesized RE matches. There is no difference between parenthesized and non-parenthesized REs. This is useful when using a complex RE with the + or * operator as in (AB)+ which matches AB, or ABAB, or ABABAB...

- A pipe (|) character indicates an “or”. So “(RE1|RE2)” matches either RE1 or RE2.

Examples of regular expressions are:

Value	Expression
a	match a
\\	match \
^a	match a, only if it begins a string
a\$	match a, only if it ends a string
-	match any one character
[abc]	match one of the characters a, b, or c
[^abc]	match any one character that is not a, b, or c
a+	match a string of one or more of the character a
ab*	match the character a, optionally followed by any number of the character b
a{ 3 }	match 3 or more concatenated characters a. Equivalent to aaa+ or aaaa*
(a b \)	match either the character a or b or

To create a Regular Expression input control:

1. With Regular Expression selected as the Input Control type, select the data type from the Type drop-down list. Regular Expressions may only be defined using the String data type.
2. Define a Literal for the regular expression. See Defining Literals for details. Regular expressions are strings that express rules for string pattern matching.
3. Select the new regular expression literal from the Delimiter drop-down list.
4. Select a termination type from the Termination drop-down list.

5. If you select Delimiter or Minimum Length + Delimiter as a termination type, select one from the Delimiter drop-down list.
6. If you select Exact Length, Minimum Length + Delimiter, or Minimum Length + White Space, type the fixed length of the field or the minimum number of characters to parse before looking for a delimiter or white space in the Length field.

Saving an Input Control to an Output Control

You can save input controls as output controls. The output controls mirror (as closely as possible) the contents of the input control. You may have to modify the output control because certain properties of input controls do not have exact matches on the output control side.

To save an input control as an output control, follow these steps.

1. Right-click on the input control you want to save in the left pane of the Formatter window. A pop-up menu appears.
2. Select Save as Output. The input control is saved as an output control with the prefix OFC_#_ and appears in the Output Controls list.

For example, the first time the input control called Test is saved as an output control, it is saved as OFC_1_Test. The second time Test is saved, it is saved as OFC_2_Test, and so forth.

3. Select and modify the details on the Output Control property sheet to change or add information for the output control. See “Creating an Output Control” for more information about working with output property sheets.

Defining Output Controls

Output controls are used to format output data. IISERVER uses output controls to determine how to justify and trim data, add prefixes or suffixes, or perform an arithmetic expression. Some output control types also allow alternative output formatting. For more information on alternative output formatting, refer to “Alternative Input and Output Formats.”

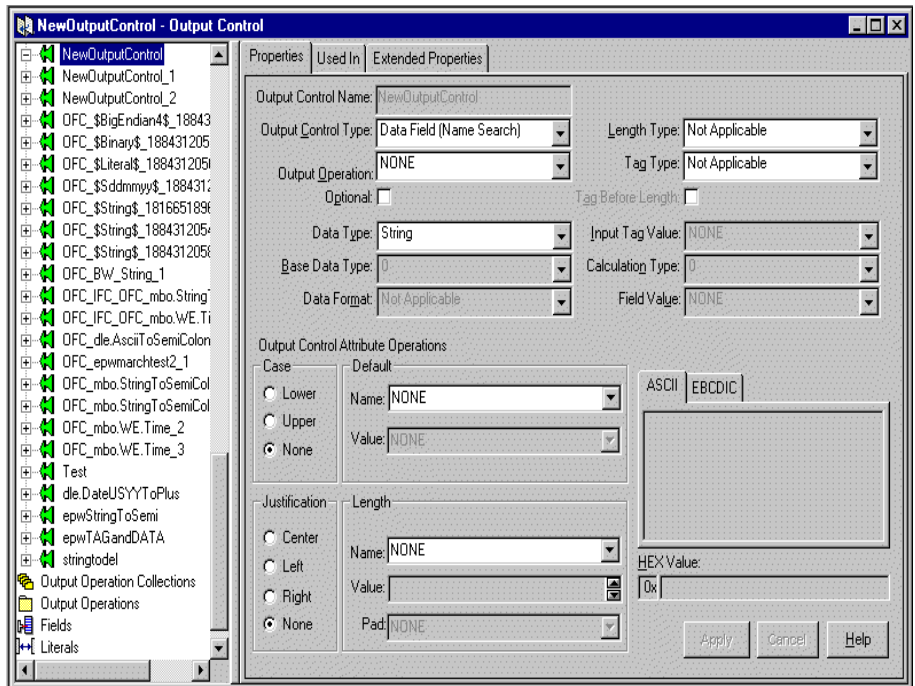
The Output Control Properties tab allows you to add a new output control, create and define default and length output operations, and assign case and justification operations to an output control.

Creating an Output Control

To create an output control, follow these steps.

1. Right-click on Output Controls from the left pane of the Formatter window. A pop-up menu appears, and the right pane displays the current list of output controls.
2. Select New from the pop-up menu to create a new output control entry in the left pane and open the Output Control property sheet in the right pane, as shown in Figure 4-8.

Figure 4-8 Output Control Property Tab



3. Type a unique output control name in the text box.

Note: The output control name must be 32 characters or less. Single quotes, double quotes, and spaces should not be used in names.

4. Press **Enter** to add the output control to the list of output controls. The output control is highlighted and its name appears in the Output Control name text box in the Output Control tab.

The procedures for creating each output control type are described in the following sections.

Creating a Data Field (Name Search) Output Control

Data Field (Name Search) output controls map the value of an input field directly to an output field. Formatter searches for the input field value by the field name specified. For example, if the contents of an output field are going to map directly to the contents of an input field, you could define a Data Field (Name Search) output control. The output control would look like:

```
Output Control Name: FieldNameWithComma
Output Control Type drop-down list: Data Field (Name Search)
Data Type drop-down list: String
Input Tag Value field: FirstName
Tag Before Length checkbox: (Not checked)
Optional checkbox: (Checked)
```

Notes: To add a comma suffix to the end of the field data, create a Suffix operation, then select it in the Output Operation drop-down list. See “Defining Output Controls” for more details.

The Output Control type determines which fields require information. Fields containing “Not Applicable” or “NONE” do not require a value.

To define a Data Field (Name Search) output control, follow these steps.

1. With Data Field (Name Search) selected in the Output Control Type drop-down list, select a data type for the output data from the Data Type drop-down list.
2. If the selected data type is Date and Time, Date, Time, or Custom Date and Time, select the Base Data Type (String or Numeric) the data will be output as.
3. If the selected data type is Custom Date and Time, select a date/time format from the Data Format drop-down list.
4. To perform an operation on the data, select one from the Output Operation drop-down list. See operations for details.
5. If the output data is to have an embedded length, specify the Length data type in the Length Type drop-down list.
6. If the output data is to have an associated tag, specify the data type for the tag in the Tag Type drop-down list.
7. If the tag should be output before the embedded length, check the Tag Before Length box.

8. If the output control parse is optional, check the Optional checkbox. Optional means that Formatter should continue with the output message if the field was NULL with no default value or was not referenced in the input layout.

Note: By default, output controls are mandatory. Mandatory means if the associated input field is either NULL or not referenced in the input format, and there is no default value, the mandatory test fails and Formatter will not output data.

Creating a Data Field (Tag Search) Output Control

Data Field (Tag Search) output controls map the value of an input field directly to an output field. Formatter searches for the input field value by tag, looking for the specified literal tag value in the parsed input message. For example, in a random-ordered flat format, you will need to search for a specific tagged field in the parsed input data. If you're looking for an input Tag & Data field with <FirstName> as the tag and want to output the data minus the tag, the output control might look like:

```
Output Control Name: InTagDataOutDataComma
Output Control Type drop-down list: Data Field (Tag Search)
Data Type drop-down list: Numeric
Input Tag Value field: FirstName
Tag Before Length checkbox: (Not checked)
Optional checkbox: (Not checked)
```

Notes: To add a comma suffix to the end of the field data, create a Suffix operation, then select it in the Output Operation drop-down list. See “Defining Output Controls” for more details.

The Output Control type determines which fields require information. Fields containing “Not Applicable” or “NONE” do not require a value.

To define a Data Field (Tag Search) output control, follow these steps.

1. With Data Field (Tag Search) selected in the Output Control Type drop-down list, select a data type for the output data from the Data Type drop-down list.
2. If the selected data type is Date and Time, Date, Time, or Custom Date and Time, select the Base Data Type (String or Numeric) the data will be output as.
3. If the selected data type is Custom Date and Time, select a date/time format from the Data Format drop-down list.
4. To perform an operation on the data, select one from the Output Operation drop-down list. See operations for details.

5. If the output data is to have an embedded length, specify the Length data type in the Length Type drop-down list.
6. If the output data is to have an associated tag, specify the data type for the tag in the Tag Type drop-down list.
7. After selecting a Tag Type, choose a Literal from the Input Tag Value drop-down list as the tag.
8. If the tag should be output before the embedded length, check the Tag Before Length box.
9. Select a literal from the Input Tag Value drop-down list as the tag value to search for in the input message. This name must match the tag value in the input message exactly. The tag value will be output with the field data.
10. If the output control parse is optional, check the Optional checkbox. Optional means that Formatter should continue with the output message if the field was NULL with no default value or was not referenced in the input layout.

Note: By default, output controls are mandatory. Mandatory means if the associated input field is either NULL or not referenced in the input format, and there is no default value, the mandatory test fails and Formatter will not output data.

Creating a Literal Output Control

Literal output controls insert a literal (static) value into the output message. For example, if there is a static value separating the header for a message from the body, you would add <BODY>. An output control might look like:

```
Output Control Name: Out<FirstName>Literal
Output Control Type drop-down list: Literal
Data Type (Data Types) drop-down list: String
Tag Before Length checkbox: (Not Checked)
Optional checkbox: (Not Checked)
```

Note: The Output Control type determines which fields require information. Fields containing “Not Applicable” or “NONE” do not require a value.

To define a Literal output control, follow these steps.

1. With Literal selected in the Output Control Type drop-down list, select a data type for the output data from the Data Type drop-down list.

2. If the selected data type is Date and Time, Date, Time, or Custom Date and Time, select the Base Data Type (String or Numeric) the data will be output as.
3. If the selected data type is Custom Date and Time, select a date/time format from the Data Format drop-down list.
4. To perform an operation on the data, select one from the Output Operation drop-down list. See operations for details.
5. If the output data is to have an embedded length, specify the Length data type in the Length Type drop-down list.
6. Select a literal from the Field Value drop-down list.
7. If the output data is to have an associated tag, specify the data type for the tag in the Tag Type drop-down list.
8. If the tag should be output before the embedded length, check the Tag Before Length box.
9. If the output control parse is optional, check the Optional checkbox. Optional means that Formatter should continue with the output message if the field was NULL with no default value or was not referenced in the input layout.
Note: By default, output controls are mandatory. Mandatory means if the associated input field is either NULL or not referenced in the input format, and there is no default value, the mandatory test fails and Formatter will not output data.

Creating a Conditional Field Output Control

Conditional Field output controls mark a field as output only if a specific field exists. One other field in the output format must be associated with an Existence Check Field output control. For example, you may want to output an account number only if a person's name is in the input message. The output control might look like:

```
Output Control Name: IfPersonNameOutAccountComma
Output Control Type drop-down list: Conditional Field
Data Type (Data Types) drop-down list: String
Tag Before Length checkbox: (Not Checked)
Optional checkbox: Checked
```

Notes: The Output Control type determines which fields require information. Fields containing “Not Applicable” or “NONE” do not require a value. Another field (in this example, the “PersonName” field) in the output format **MUST** be associated with an Existence Check Field output control for the Conditional Field output control to work correctly.

To define a Conditional Field output control, follow these steps.

1. With Conditional Field selected in the Output Control Type drop-down list, select a data type for the output data from the Data Type drop-down list.
2. If the selected data type is Date and Time, Date, Time, or Custom Date and Time, select the Base Data Type (String or Numeric) the data will be output as.
3. If the selected data type is Custom Date and Time, select a date/time format from the Data Format drop-down list.
4. To perform an operation on the data, select one from the Output Operation drop-down list. See operations for details.
5. If the output data is to have an embedded length, specify the Length data type in the Length Type drop-down list.
6. If the output data is to have an associated tag, specify the data type for the tag in the Tag Type drop-down list.
7. If the tag should be output before the embedded length, check the Tag Before Length box.
8. If the output control parse is optional, check the Optional checkbox. Optional means that Formatter should continue with the output message if the field was NULL with no default value or was not referenced in the input layout.

Note: By default, output controls are mandatory. Mandatory means if the associated input field is either NULL or not referenced in the input format, and there is no default value, the mandatory test fails and Formatter will not output data.

Creating an Existence Check Field Output Control

Existence Check output controls are used to define the field on which a Conditional Field output control depends. See “Creating a Conditional Field Output Control” for more information.

Note: The Output Control type determines which fields require information. Fields containing “Not Applicable” or “NONE” do not require a value.

To define an Existence Check Field output control, follow these steps.

1. Select Existence Check Field as the output control type.
2. If the output control parse is optional, check the Optional checkbox. Optional means that Formatter should continue with the output message if the field was NULL with no default value or was not referenced in the input layout.

Note: By default, output controls are mandatory. Mandatory means if the associated input field is either NULL or not referenced in the input format, and there is no default value, the mandatory test fails and Formatter will not output data.

Creating a Rules Field Output Control

Rules Field output control enables you to create several different output controls for a single output field by integrating the Rules application (see “Defining Rules” for more information) with the Formatter application. Formatter can then use the boolean logic capabilities of Rules to express and evaluate the conditions for formatting a field.

Based on the fields defined for the input format, you build different output controls for the same output field. This eliminates the need to create several output formats for a single input format. If no rule evaluates as “true,” output data is formatted to the other settings in the Rules Field output control. Otherwise, the data is formatted according to the output control specified by the rule.

Notes: When you define rules for the output control, you cannot delete the fields used in the rules without first deleting the rules that use the fields.
The Output Control type determines which fields require information. Fields containing Not Applicable or NONE do not require a value.

To define a Rules Field output control, follow these steps.

1. Select Rules Field in the Output Control Type drop-down list.

2. Select a data type for the output data from the Data Type drop-down list.
3. If the selected data type is Date and Time, Date, Time, or Custom Date and Time, select the Base Data Type (String or Numeric) the data will be output as.
4. If the selected data type is Custom Date and Time, select a date/time format from the Data Format drop-down list.
5. To perform an operation on the data, select one from the Output Operation drop-down list. See operations for details.
6. If the output data is to have an embedded length, specify the Length data type in the Length Type drop-down list.
7. If the output data is to have an associated tag, specify the data type for the tag in the Tag Type drop-down list.
8. If the tag should be output before the embedded length, check the Tag Before Length box.
9. If the output control parse is optional, check the Optional checkbox. Optional means that Formatter should continue with the output message if the field was NULL with no default value or was not referenced in the input layout.

Note: By default, output controls are mandatory. Mandatory means if the associated input field is either NULL or not referenced in the input format, and there is no default value, the mandatory test fails and Formatter will not output data.

To add the rules for the field, follow these steps.

1. Choose the Jump to Rules button. The Rules Login dialog box appears.
2. Login and type the rules for the field using the instructions found in “Defining Rules.”
3. Exit the Rules application. You are returned to the Formatter application.

Creating an Input Field Exists Output Control

Input Field Exists output controls indicate that the output format of which the control is part should be in the output message if it exists in the input message. For example, if a last name input field is in the input message, you may want to output the format that includes first, middle, and last names. You would have four fields in this format:

Field	Output Control Type
LastName	Input Field Exists
FirstName	Data Field (Name Search)
MiddleName	Data Field (Name Search)
LastName	Data Field (Name Search)

The output control for LastName might look like:

```
Output Control Name: IfMiddleNameOutData
Output Control Type drop-down list: Input Field Exists
Data Type (Data Types) drop-down list: String
Tag Before Length checkbox: (Not Checked)
Optional checkbox: Checked
```

Note: The Output Control type determines which fields require information. Fields containing “Not Applicable” or “NONE” do not require a value.

To define an Input Field Exists output control, follow these steps.

1. Select Input Field Exists as the output control type:
2. If the output control parse is optional, check the Optional checkbox. Optional means that Formatter should continue with the output message if the field was NULL with no default value or was not referenced in the input layout.

Note: By default, output controls are mandatory. Mandatory means if the associated input field is either NULL or not referenced in the input format, and there is no default value, the mandatory test fails and Formatter will not output data.

Creating an Input Field Value = Output Control

Input Field Value = output controls indicate that the output format of which the control is part should be output if the value of the field equals a specific value. For example, if an input field contains security information, you may want to limit the data in the output message based on that information. The output control might look like:

```
Output Control Name: IfSecurity=HighThenOut
Output Control Type drop-down list: Input Field Value =
Input Field Value drop-down list: High
Tag Before Length checkbox: (Not Checked)
Optional checkbox: Checked
```

Note: The Output Control type determines which fields require information. Fields containing “Not Applicable” or “NONE” do not require a value.

To define an Input Field Value = output control, follow these steps.

1. Select Input Field Value = as the output control type:
2. Select a Literal from the Field Value drop-down list to compare the field contents to.
3. If the output control parse is optional, check the Optional checkbox. Optional means that Formatter should continue with the output message if the field was NULL with no default value or was not referenced in the input layout.

Note: By default, output controls are mandatory. Mandatory means if the associated input field is either NULL or not referenced in the input format, and there is no default value, the mandatory test fails and Formatter will not output data.

Using the Extended Properties Tab

You can define and modify output operations and collections and assign them to an associated output control from the Extended Properties tab. The sections of the Extended Properties tab are labeled in Figure 4-9 and described in Table 4-1.

Figure 4-9 Extended Properties Tab

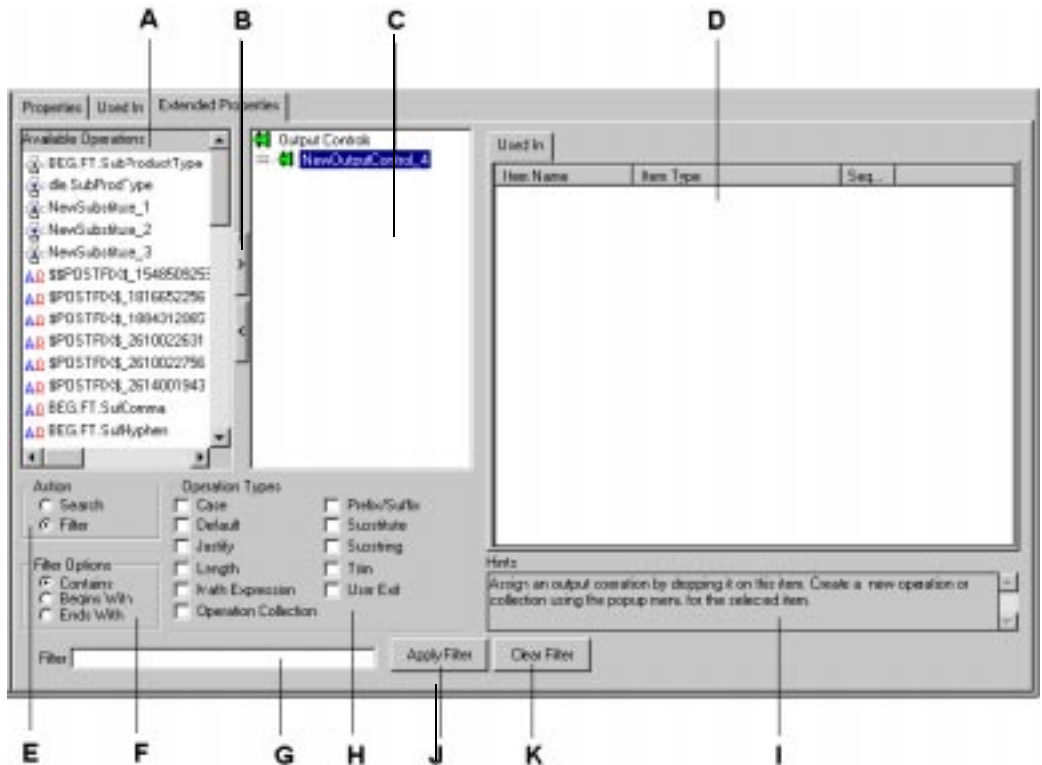


Table 4-1 Extended Properties

Label	Section	Description
A	Available Operations	Available output operations and output operation collections in read-only mode. This list can be filtered. You can select any or all of the items in this list and apply them to the output control by dragging and dropping the items onto the control.

Table 4-1 Extended Properties

Label	Section	Description
B	Add (>) and Remove (<) Buttons	<p>The Add (<) and Remove (>) buttons provide a fast way to perform specific tasks. The buttons behave differently, depending on what is selected.</p> <p>To assign operations to an output control, select the output control in the Formatter tree view, select the operations in the Available Operations list, and click >.</p> <p>To remove operations from an output control, select the output control in the Formatter tree view, select the operations in the Available Operations list, and click <.</p> <p>To remove an output operation or output operation collection, select it and click <. A confirmation box appears asking you to confirm the removal.</p>
C	Selected Operations	<p>The Selected operations list is a tree view containing a duplication of the output control from the main tree view; however, functionality not available in the main tree view is available in this list.</p> <p>With a collection selected, you can reorder the sequence of output operations in the output operations collections tab.</p> <p>Pop-up menus are available allowing you to create, remove, open, expand, and collapse items.</p> <p>Creating a new item with an output control selected in the tree view assigns the new item to the control.</p> <p>If an output operation or output operation collection is selected, you can access the pop-up menu and select Duplicate.</p> <p>Caution: A change made to any assigned component of an output control affects all controls that use that component.</p>

Table 4-1 Extended Properties

Label	Section	Description
D	Output Operation Collections	A list of output operation collections.
E	Action	Select Search to find items in the list of available output operations. Type the associated text in the Search text box. Select Filter to narrow the available output operations. Type the associated text in the Filter text box.
F	Filter Options	Use these options to either search or filter the available output operations. Only the Available Operations section is affected—the Formatter tree does not change.
G	Filter Text Box	Use the Filter text box to either search or filter criteria.
H	Operation Types	Select one or more items to filter. If you select Justify, Length is automatically selected also because these items are associated.
I	Hints	Hints guide you through the functionality of the tab. The hint changes to associate with the section in which you are positioned.
J	Apply Filter Button	Click this button to apply the filter. The list of operations is filtered to show only the selected items.
K	Clear Filter Button	Click this button to clear all parameters for searching and filtering. All available output operations are then displayed.

Notes: If you select more than one output operation, a new output operation collection is created that begins with the name of the output control.

If more than one output operation exists for a control, a number is appended to the name (for example, SS_1 for an output operation collection assigned to SS).

For more information on the output operations supported by Information Integrator, refer to “Defining Output Controls.”

Saving an Output Control as an Input Control

You can save output controls as input controls. The input controls mirror (as closely as possible) the contents of the output control. The input control may have to be modified, because certain properties of output controls do not have exact matches on the input control side.

You can save an output control as an input control using the steps described in the following procedure.

1. Select the output control you want to save as an input control in the left pane of the Formatter window and right-click. A pop-up menu appears.
2. Select Save as Input. The output control is saved as an input control with the prefix IFC_ExistingName_# and appears in the Input Controls list.

For example, the first time the output control Output_Control is saved as an input control, it is saved as IFC_1_Input_Control. The second time, it is saved as IFC_2_Input_Control.

To change or add information for the input control, select it and modify the details on the Input Control property sheet.

Building Formats

You use the Formats function in the left pane of the Formatter window to build both input and output formats. The formats can be either flat or compound.

Before you create flat or compound formats, you need to define their component parts. The component parts for each format are described below:

- Flat input format — Literals, fields, and input (parse) controls
- Flat output format — Literals, fields, and output controls, output operations, and output operation collections
- Compound input format — Flat input formats or other compound input formats
- Compound output format — Flat output formats or other compound output formats

Creating a Flat Input Format

You create flat input formats by adding fields and the desired associated input control, one by one, until you've fully defined the format for the message. You may want to define formats to uniquely identify only the data you're interested in, or you may want to explicitly define a format for each logical piece of the message data.

To create a new flat input format, follow the steps below.

Note: When building Flat Input Formats, have two windows available: one for the new Format and one for Input Controls to be associated with input fields.

1. Right-click Formats in the left pane of the Formatter window. A pop-up menu appears.
2. Select New. A submenu opens.
3. Select Flat; then select Input. A new format component is added to the left-hand pane.
4. Type a unique format name in the text box and press **Enter**. The name is alphabetically inserted into the list of formats, and the associated tabs are displayed in the right pane.
5. Select the Properties tab to define the properties for the flat input format.

6. If the fields for the format can appear in any order in the message, select the Random checkbox. If the fields for the format appear in a specific order in the message, the Random checkbox should be clear.
7. Select a termination type from the Format Termination drop-down list.
8. If the termination type is Delimiter or Minimum Length + Delimiter, select a literal from the Delimiter drop-down list.
9. If the termination type is Exact Length, Minimum Length + Delimiter, or Minimum Length + White Space, enter the fixed length of the field or the minimum number of characters to parse before looking for a delimiter or white space in the Length field.
10. To add field and input control components to the format, select the Fields tab. The Fields sheet appears.
11. Open another Formatter window and display either Fields or Input Controls, depending on what you want to add to the new format.
12. Drag and drop the Fields or Input Controls into the window containing the new flat input format.
13. When you have added all the desired fields and input controls to the format, click Apply to save your changes to the Database.

Creating a Flat Output Format

A flat output format contains a list of fields with their associated output controls. To create a new flat output format, follow the steps below.

Note: When building Flat Output Formats, have two windows available: one for the new Format and one for the Output Controls to be associated with output Fields.

1. Right-click Formats in the left pane of the Formatter window. A pop-up menu appears.
2. Select New. A submenu opens.
3. Select Flat; then select Output. A new format component is added to the left-hand pane.

4. Type a unique format name in the text box and press **Enter**. The name is alphabetically inserted into the list of formats.
5. Right-click on the output component and select Add Field Components from the pop-up menu.
6. Either select the field(s) that you want to add, or to filter the fields, click the Format Filter arrow to open the drop-down list displaying the available formats. You can filter the list to display only the fields contained in:
 - a specific flat format.
 - all of the flat formats contained in a compound format.After you select a filter, the Components window displays only the filtered fields. To display the entire list of available formats and fields, select Not Applicable from the Format Filter drop-down.
7. When you have selected all the fields you want to add to the format, click Accept Selection. The fields are added to the format tree.
8. Click Apply to save your changes to the Database.

Creating a Compound Input Format

You create compound input formats by adding flat or other compound formats to the format. To create a new compound input format, follow the steps below.

Note: When building Compound Input Formats, have two windows available: one for the new Compound Format and one for other Formats to be placed within it.

1. Right-click Formats in the left pane of the Formatter window. A pop-up menu appears.
2. Select New. A submenu opens.
3. Select Compound; then select Input. A new format component is added to the left-hand pane.
4. Type a unique format name in the text box and press **Enter**. The name is alphabetically inserted into the list of formats in the left pane, and the Components and Properties sheets are displayed in the right pane.

5. To specify the order of component formats in the compound, select one of the following:
 - To have component formats appear in the specified order in the input message, select **Ordinal**.
 - To have the first field of each component be defined as a literal, select **Tagged Ordinal**.
 - To have only one component format appear in a message, select **Alternative**. See “Alternative Input and Output Formats” for more information.
6. To begin adding flat and compound format components, select the **Components** tab. The **Components** sheet appears.
7. Open another **Formatter** window and display the format list.
8. Drag and drop the flat and compound formats into the window containing the new compound input format.
9. When you have added all the desired formats to the new compound format, click **Apply** to save your changes to the **Database**.

Creating a Compound Output Format

You create compound output formats by adding flat or other compound formats to the format. To create a new compound output format, follow the steps below.

Note: When building **Compound Output Formats**, have two windows available: one for the new **Compound Format** and one for other **Formats** to be placed within it.

1. Right-click **Formats** in the left pane of the **Formatter** window. A pop-up menu appears.
2. Select **New**. A submenu opens.
3. Select **Compound**; then select **Output**. A new format component is added to the left-hand pane.
4. Type a unique format name in the text box and press **Enter**. The name is alphabetically inserted into the list of formats in the left pane, and the **Components** and **Properties** sheets are displayed in the right pane.

5. To specify the order of component formats in the compound, select one of the following:
 - To have component formats appear in the specified order in the input message, select Ordinal.
 - To have only one component format appear in a message, select Alternative. See “Alternative Input and Output Formats” for more information.
6. To begin adding flat and compound format components, select the Components tab. The Components sheet appears.
7. Open another Formatter window and display the format list.
8. Drag and drop the flat and compound formats into the window containing the new compound input format.
9. When you have added all the desired formats to the new compound format, click Apply to save your changes to the Database.

Saving a Flat Input Format as a Flat Output Format

The flat output format mirrors, as much as possible, the contents of the flat input format; however, the termination type, length, and delimiter are defaulted for the output flat format.

You can save a flat input format as a flat output format using the steps described in the following procedure.

1. Select the flat input format you want to save as a flat output format in the left pane of the Formatter window and right-click. A pop-up menu appears.
2. Select Save as Output. A confirmation box appears asking if you are sure you want to save the flat input format as a flat output format. Click Yes. A new flat output format OFF_ExistingName_# appears in the Formats section of the tree.

For example, the first time the flat input format is saved as a flat output format, it is saved as OFF_NewInputFormat_1. The second time, it is saved as OFF_NewInputFormat_2.

The Objects Created Summary dialog appears containing a list with the new format name and any controls created.

3. Select a control and modify its details to change information for the output controls created.

Access Modes

Each output field has an associated Access Mode. Access Modes define how Formatter accesses fields in the input message to generate fields in the output message. You select output field access modes and associated input field names to tell Formatter how to map fields from the input message to fields in the output message.

Table 4-2 provides a description of each access mode supported in Formatter.

Table 4-2 Access Modes

Access Mode	Description
Not Applicable	Do not access any field instance. Use for the output of Literals.
Normal Access	Access the instance in the same repeating component as the current controlling field instance. If there is no controlling field, access the first instance. This behaves just like Access sibling instance.
Access with Increment	A field with this access mode is the controlling field for the repeating component. This accesses the current value and increments it.
Access Using Relative Index	The first field in a repeating component that Formatter encounters with this access mode is the controlling field for the repeating component. Any other field in the repeating component with this access mode behaves as if it is set to “Access sibling instance” or “Normal access” (access the sibling of the controlling field).
Access nth instance of field	Access the nth instance (n = 0 means get the first instance) of the field in the input message.
Controlling field	This field is the controlling field for the repeating component. On each repetition, access the next field instance that is still a child of the current controlling field instance of the parent format. If there is no parent controlling field, the repetitions end with the last field instance from the input message.
Access current instance	Access the same field instance as on the previous access (the first access will get the first instance of the field).
Access next instance	Access the next field instance relative to the previous access.

Access Mode	Description
Access parent instance	Access the instance that is the first ancestor of the current controlling field instance.
Access sibling instance	Access the instance in the same repeating component as the current controlling field instance. If there is no controlling field, access the first instance.

Alternative Input and Output Formats

Alternative formats are compound formats in which one format in a set of alternatives will apply to a message. For example, if an alternative format is named A, it may contain component formats B, C, and D. A message of format A may actually be of variation B, C, or D.

Note: At least one of the alternatives must apply to a message; otherwise, the entire alternative format does not apply.

An alternative format can be used anywhere a format can be used, and each component format can be any kind of its respective parent input or output format.

Compound formats may contain a mix of mandatory and optional component formats. Additionally, these component formats may contain a mix of mandatory and optional fields. If a mandatory component (of a format or field) is not present in a message, the compound or flat format does not apply. However, if an optional component is not present, the next component will be evaluated. We recommend that all component formats in an Alternative Format be mandatory.

Alternative Input Formats

If an alternative input format is applied to an input message, the first component of the alternative format is compared to the message. If it parses with the first component format, parsing is finished. If parsing fails, Formatter tries the second component format. If that fails, it tries the next component format. If all components fail to parse, the parse for the entire alternative input format fails.

If a format has optional fields, and the fields are delimited and not tagged, it may be impossible to determine which of the optional fields occur in an input message. For example, if a simple space-delimited format is:

```
[F1] F2 [F3] [F4]
```

The first, third, and fourth fields are optional. If an input message `value1 value2 value3` is received, there is no way, without some rules to remove ambiguity, to determine if the message is actually `F1 F2 F3`, `F2 F3 F4`, or `F1 F2 F4`.

Alternative input formats enable you to specify all possible configurations of mandatory and optional fields in a format. You must explicitly define all possible combinations to avoid possible parsing errors. Formatter does not recursively try all combinations.

As an example, if you have the following input format:

```
Field 1: optional, comma delimited  
Field 2: mandatory, colon delimited  
Field 3: optional, comma delimited  
Field 4: optional, forward-slash delimited
```

The input message `"field1,field2:field3,field4/"` parses into four fields correctly. However, `"field1 field2:field3,field4/"` fails. The first field of the format is comma delimited and parsed as `"field1 field2:field3"`. The second field is colon delimited, so Formatter looks for a colon in `"field4/"`, detects the end of the message, and fails to parse.

You have to explicitly define all possible combinations (`F1 F2 F3`, `F2 F3 F4`, `F1 F2 F4`). In this case, the second alternative format would have three fields, starting with the mandatory colon-delimited field.

For alternative formats, you determine the order in which alternative components are parsed. Remember that components are taken on a first-parsed, only-parsed basis. With that in mind, component order is critical.

For example, if you have a message `"abcde,"` you could specify two alternatives (or more). It could be parsed into a 5-byte field or two separate fields, one 2-bytes long and one 3-bytes long. If the 5-byte field format is the first alternative, you will never parse using the second format. If two parses are valid for the same input, only the first occurs.

Tagged Input Formats

A compound format can have a property of Tagged Ordinal. This means that the first field in each component format is a literal. The component format can be flat or compound.

Tagged input formats can be useful when used in conjunction with alternative formats. The following is an example of what might come in the data segment of a SWIFT message.

```
" :10:f1 f2 f3<CRLF>:20:C/1234/<CRLF>first description<CRLF>second
description<CRLF>:30:f4,f5<CRLF>"
```

The following is a loose definition of the format:

```
:10: field1 field2 field3 <CRLF>
:20: [C/acctnum/< CRLF>] (optional)
desc1 <CRLF>
[desc2 <CRLF>] (optional)
:30:first, second <CRLF>
```

Parse segment :20: using the following rules:

- If there are two slash-delimited fields before the <CRLF>, the credit code and account number exist.
- If not, continue with the mandatory desc1 field, followed by an optional desc2 field.
- Do not run into the :30: segment by parsing “:30:first, second” into the <CRLF>-delimited desc2 field when the desc2 field is not present (it is optional).

With tagged formats, you now have a way to ensure that you do not overrun the boundaries of the :20: segment. Any trailing optional fields in a tagged flat format can be parsed or determined to be absent by parsing only up to the component boundary, instead of looking for a field delimiter (or other termination) beyond the component boundary.

Define a compound tagged format with three components as follows:

```
Segment10 :
    First Field Literal ":10:"
    space-delim field
    space-delim field
    <CRLF> delim field
```



```
Segment20 : Alternative format with two components

Segment20_1 (this is the first alternative component)
  First Field Literal ":20:"
  Credit Code : slash delim, mandatory
  AcctNum : /<CRLF> delim, mandatory
  desc1 : <CRLF> delim, mandatory
  desc2 : <CRLF> delim, optional

  Segment20_2 (this is the second alternative component)
    First Field Literal ":20:"
    desc1 : <CRLF> delim, mandatory
    desc2 : <CRLF> delim, optional

Segment30 :
  First Field Literal ":30:"
  comma delim field
  <CRLF> delim field
```

When parsing a `Segment20`, the parser first attempts to parse `Segment20_1`. If it fails, it parses the second alternative. The credit code and account number are not part of this particular `:20:` segment.

You could have different first field literals for `20_1` and `20_2` (for example, `:20A:` and `:20B:`) like SWIFT sometimes does.

You can include tagged input format as a component of a compound, where the other components can be any other kind of input format. For SWIFT, you might define a SWIFT 570 message as a compound ordinal of three components:

```
Basic Header : Ordinal Flat Format
570 Data Segment : Tagged Compound Format
Trailer : Random Tagged Flat
```

Tagged formats do not apply to output formats. An output format can certainly have a first field literal, but calling it a tagged format does not gain anything. It is only when you need an additional way to determine message boundaries when parsing an input message where first field literals become necessary.

Alternative Output Formats

To format an alternative output format, Formatter attempts to create the first component of the alternative format. If creating the first component is successful, formatting is finished. If it fails, Formatter tries to create the second component, and so on. If all components fail to be created, formatting the alternative output format fails.

Alternative Output Format Example

To create an alternative output format, you define a set of alternatives, then define an alternative compound having all of the alternatives as components, sequenced in the order desired.

Using the previous description of a `:20:` segment, assume you have to create a `:20:` segment instead of parsing it. You do not know what kind of input message you had, whether it was a `Segment20_1`, `Segment20_2`, or some other input format that allowed a parse of some or all of the fields required for a `segment 20`.

You cannot just say you have some optional leading fields `CreditCode` and `AcctNum`, because you have to put in a `<CRLF>` if the fields exist, but not if they do not.

Create a set of two alternatives:

```
Seg20_1 1literal :20:
        2CreditCode Mandatory with / delim
        3AcctNum Mandatory with /<CRLF> delim
        4desc1 Mandatory with <CRLF> delim
        5desc2 Optional with <CRLF> delim

Seg20_2 1literal :20:
        2desc1 Mandatory with <CRLF> delim
        3desc2 Optional with <CRLF> delim
```

Notice the similarity between the two. It is easy to go from most specific to more general by using the `Formatter Save As` and `Field Delete` features. By having several mandatory fields in a format, you are ANDing their existence together. All must be present to create the given format. If you sequenced `Seg20_2` before `Seg20_1` in the alternative output (compound) format, `Seg20_1` would never be applied.

Importing and Exporting Format Components

The Formatter application enables you to import or export one type of Formatter component at a time. Using Formatter's import and export functions, you can export format data from one database and import it to another.

Note: The export file generated by the Formatter application has a different format than the one generated by the command-line utility, NNFie. The export file generated by NNFie cannot be imported by the Formatter application. See *BEA Information Integrator Installation and Administration Guide* for details on how to use NNFie.

Exporting Components

You can export Formatter components to a file that can be later imported into the same database or a different installation (of the same version). To ensure that all Formatter component information necessary is available, Table 4-3 lists the export requirements.

Table 4-3 Export Requirements

If you export	Also export
an Input Format	Literals, Fields, Input Controls, Flat Input Formats, and Compound Input Formats.
an Output Format	Literals, Fields, Output Operations, Output Operation Collections, Output Controls, Flat Output Formats, and Compound Output Formats.

Note: Files exported using the Formatter application may not be imported using the NNFie tool. The same tool must be used to import components as was used to export the components.

To export Formatter components, follow these steps.

1. Select the component category (Formats, Input Controls, Output Controls, Fields, Literals, or Operations) for the type of component you want to export. Be sure the property sheet for the component category is open in the right pane.

2. Select the Export tab. A list of defined components appears. For example, if you selected Formats, a list of input and output formats appears.
3. Select the components to export:
 - To select individual components, click on each one in the list.
 - To select all components listed, click Select All.
 - To deselect the components you have selected, click Deselect All.
4. Choose the Export button. The Export File dialog box appears.
5. Select the drive, directory, and filename to export the components to and click OK. The file is saved with a .fmt extension. The selected components are exported and the dialog box closes.

Importing Components

Import enables you to load components from a file you previously exported from another database. To import a complete format, you must import each of its components separately, starting from the lowest-level components. Subordinate components must be imported prior to parent components. For example, to import a compound input format, you must have already imported all subordinate flat input formats and their subordinate components.

To import a complete format, import the components in the following order:

1. Literals
2. Output Operations
3. Output Operation Collections
4. Fields
5. Input (Parse) Controls
6. Output Format Controls
7. Flat Input Formats
8. Flat Output Formats
9. Compound Input Formats

10. Compound Output Formats

To import Formatter components, follow these steps.

1. Select the component category (Formats, Input Controls, Output Controls, Fields, or Literals) for the type of component you want to import. Be sure the property sheet is open in the right pane.
2. Select the Import tab. The Import dialog box appears.
3. Enter the complete path to the import file; or, choose the Browse button and select the drive, directory, and filename. A list of the components contained in the file appears.
4. Select the components to import:
 - To select individual components, click on each one in the list.
 - To select all components listed, choose the Select All button.
 - To deselect the components you have selected, choose the Deselect All button.
5. Choose the Import button. The components are imported into the database.

Tip If a compound component contains other compound components, you must import the compound components it contains first. For example, the compound A12 within the compound B1 must be defined before compound B1 can be imported.

Field Mapping and Transformation

Information Integrator allows incoming data values to be mapped to different field names in the output. By default, input fields are mapped to output fields by matching names. However, when you define an output format, you can name your fields whatever you want, regardless of the field names in the input formats. Then, when you specify where to find the data from the input messages, you can map the input field name to your output field name.

Table 4-4 shows the input values and the output values to which they were mapped.

Table 4-4 Field Mapping Example

Input Values	Output Values
Acct X029	ACCT
Tx Dep	TX
Amt 4000	AMT
CC 044	CountryName

To map an input format field to an output format field, follow these steps.

1. Select the format containing the field or fields that you want to map. If you do not select a format now, you can select it after you open the Field Mapping window.

Note: Note that field mappings are attached to flat output formats.

2. To open the Field Mapping window (see Figure 4-10), choose one of the following:

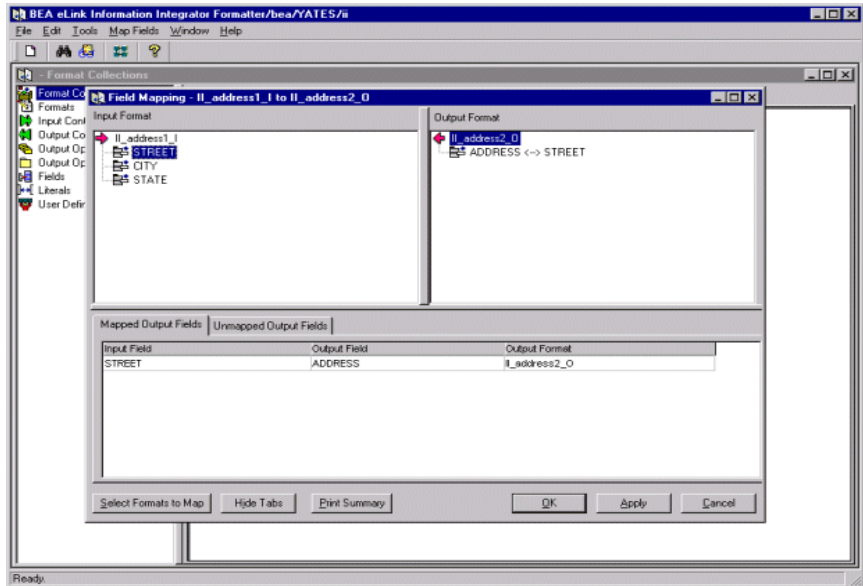
From the Map Fields menu, choose Open Mapped Fields Window.

From the toolbar, choose the Map Fields icon.

From the main Formatter tree, press the right mouse button to open the pop-up menu and select Map Fields. You must have a format selected to use this method.

The Field Mapping window opens displaying the selected format and its fields. If a single format was selected, some display data is available. If you did not select a format (or formats) in the Formatter tree, the Mapping Fields Reminder message box appears stating that you must open both an input and an output format. Click OK. The Field Mapping window opens with no data displayed.

Figure 4-10 Field Mapping Window



3. Click Select Formats to Map. The Select Formats for Mapping window appears displaying input formats in a tree in the left pane and output formats in a tree in the right pane. The format you previously chose is selected. To deselect a format, click it.

Note: You can double-click to open a format and view its details, but you cannot make changes to the format.

4. Select the input format and output format you want to map. You can only select one input and one output format.
5. Choose one of the following:

To map the formats and close the Field Mapping window, click OK.

To map the formats and keep the Field Mapping window open to map additional formats, click Apply.

To cancel and return to the previously selected formats, click Cancel. Depending on the size of the selected formats, processing may take several minutes. The status bar displays and updates the processing taking place. During this process, output fields are treated as follows:

If the output field was unmapped, Formatter searches for a matching field in the input format. If no match is found, it remains unmapped.

If the output field was previously mapped, Formatter searches for the mapped input field in the input format. If no match is found, Formatter searches for a matching field for the output field. If no match is then found, the output field becomes unmapped.

6. When processing completes, format information is displayed in the trees. If both an input and output format were selected, Mapped Fields is the default view. The Mapped Output Fields tab displays a list of fields from the output format and the input fields to which they are mapped. To have the two trees scroll and select the mapped fields, select a line in the list.
7. The Unmapped Output Fields tab displays a list of fields from the output format that could not be mapped to the selected input format. If there are unmapped fields, the tab is labeled Unmapped Output Fields. To have the output format tree scroll and select the unmapped field, select the line in the list.
8. To view only the input and output format trees, click Hide Tabs. The bottom pane of the window is hidden. To view this pane again, select Show Tabs.
9. The bottom pane of the window displays the mapped output fields. The information shown includes the input field and the output field it is mapped to, the output format the output field is associated with, and the field sequence (where the field is in the format). To view the list of unmapped output fields, choose the Unmapped Output Fields tab.
10. To view mapped fields for the output format, from the Map Fields menu, choose View Mapped Fields. To the right of each mapped output field is the name of the input field.
11. To view specific format and field information, from the Map Fields menu, choose one of the following:

Choose Format Types/Access Modes. The format type of Ordinal, Tagged Ordinal, or Alternative and access mode details are displayed.

Choose Optional/Repeating If a field is mandatory, nothing is displayed. If the field is optional or repeating, it is marked accordingly.

Choose Input/Output Control Types. The name and type of associated input and output controls for fields are displayed next to the field name.

The Field Mapping window displays the type of information you requested in the tree view list.

12. Choose one of the following:

To close the Field Mapping window and apply changes to the database, click OK.

To apply changes to the database and keep the Field Mapping window, click Apply.

Example

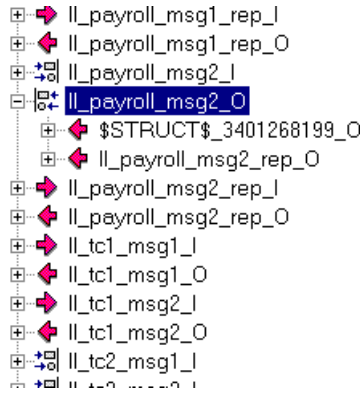
This section outlines the procedure required to complete Step 4 in “Using Information Integrator.” In Step 4, you create the output operation to calculate the employee’s weekly pay.

In the steps that follow, we will view formats created by MsgDefAdmin in Step 3 of the sample (see “Step 3. Create the Message Format Descriptions” in “Using Information Integrator”). We will also create the output operation to calculate the employee’s weekly pay, create a suffix to mark the end of the calculated field, and create an output operation collection containing the calculation and the suffix.

To view the formats, follow these steps.

1. Open the Formatter application.
2. Double-click Formats in the left pane of the Formatter window. The formats contained in your database display.
3. Find `II_payroll_msg2_O` in the left pane (see Figure 4-11).

Figure 4-11 II_payroll_msg_O in Formatter Window



This is the format that was created by running MsgDefAdmin on the file II_payroll_msg2.xml. Refer to the “Example” section in “Using MsgDefAdmin” for more information.

4. Expand II_payroll_msg2_O by clicking the plus sign (+) to the left of the format name.
5. Find II_payroll_msg2_rep_O. This is the repeating “pay” field that was created by II_payroll_msg2.xml.

We will now set up the calculation for this field. To set up the calculation, follow these steps.

1. Double-click Output Operations in the left pane of the Formatter window.
2. Right-click Math Expression and choose New from the popup menu.
3. Type **CALC_PAY** as the name of the new Output Operation in the left pane and press Enter.
4. Type **RATE * HOURS** in the Mathematical Expression text area and click Apply. This creates the expression to calculate the employee’s weekly pay. Figure 4-12 shows the math expression.
5. Select **2** in the Decimal Precision text area.

Figure 4-12 CALC_PAY Math Expression

Properties | Used In

Decimal Precision: 2

Round: Up Down

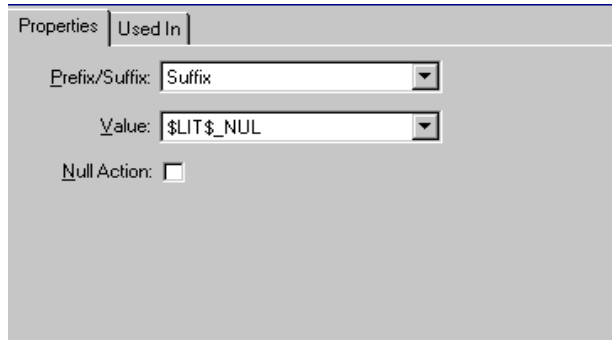
Mathematical Expression: RATE * HOURS

Apply Cancel Help

We will now create a NULL suffix to mark the end of the calculated field. To create the suffix, follow these steps.

1. Right-click Prefix/Suffix in the left pane and choose New from the popup menu.
2. Type **suffixNull** as the name of the new Suffix in the left pane and press Enter.
3. From the Prefix/Suffix drop down list, choose Suffix.
4. From the Value drop down list, choose \$LIT\$_NULL.
5. Click Apply. This creates the suffix to mark the end of the calculated field. Figure 4-13 shows the suffix.

Figure 4-13 NULL Suffix

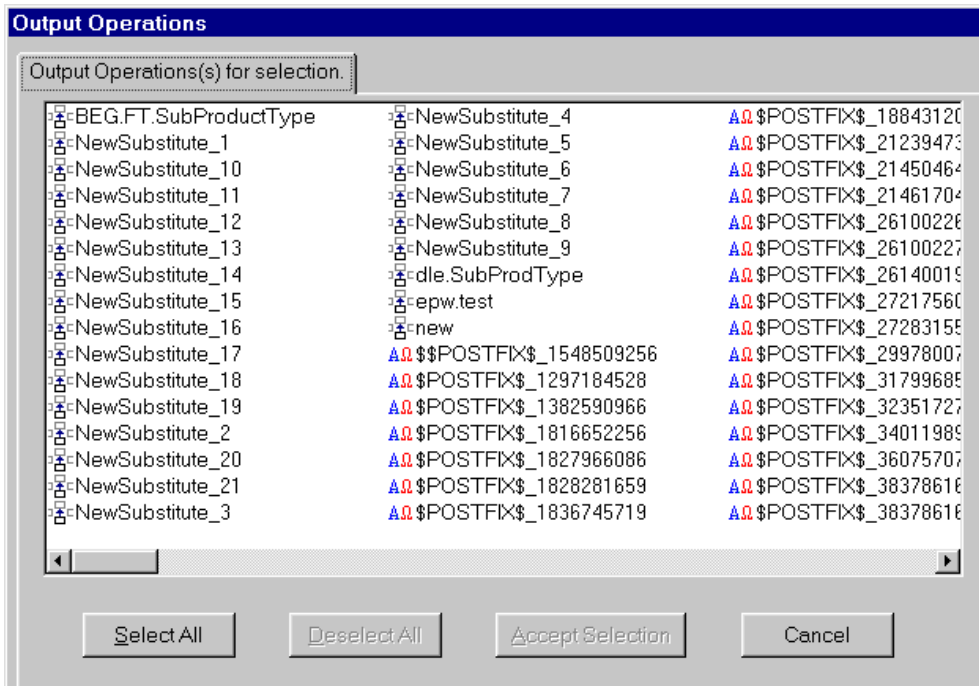


The screenshot shows a dialog box with two tabs: 'Properties' and 'Used In'. The 'Properties' tab is active. It contains three fields: 'Prefix/Suffix' with a dropdown menu showing 'Suffix', 'Value' with a dropdown menu showing '\$LIT\$_NUL', and 'Null Action' with an unchecked checkbox.

We will now create an output operation collection to hold the math expression and suffix. To create the collection, follow these steps.

1. Right-click Output Operation Collections in the left pane and choose New from the popup menu.
2. Type **CalcPay** as the name of the new collection in the left pane and press Enter.
3. Right-click CalcPay in the left pane and choose Add Output Operations from the popup menu. The Output Operations for Selection tab displays (see Figure 4-14).

Figure 4-14 Output Operations for Selection Tab



4. Select the **CALC_PAY** math expression and the **suffixNull** suffix from the list and click Accept Selection. This adds the two output operations to the collection.

Note: The **CALC_PAY** math expression should appear first in the output collection. If it does not, click on **CALC_PAY** and drag it to the top of the list.

We now need to assign the output operation collection to the **PAY** field so that the calculation will be performed on the field and the suffix will be added to the field. To do this, follow the steps below.

1. Double-click **Formats** in the left pane of the Formatter window. The formats contained in your database display.
2. Find **II_payroll_msg2_rep_O** in the left pane and expand it by clicking the plus sign (+) to the left of the format name.
3. Select the **PAY** field. The properties tab displays (see Figure 4-15).

Figure 4-15 Pay Field Properties

The screenshot shows a dialog box titled "Pay Field Properties". It has two tabs: "Properties" (which is selected) and "Browse". The dialog contains the following fields and controls:

- Field Name:** A dropdown menu with "PAY" selected.
- Output Control Name:** A dropdown menu with "CALC_PAY" selected.
- Access Mode:** A dropdown menu with "Normal Access" selected.
- Subscript:** A text box containing the value "0".
- Input Field Name:** A dropdown menu with "PAY" selected.

At the bottom right of the dialog, there are three buttons: "Apply", "Cancel", and "Help".

4. Right-click on the Output Control Name text box and select Open Item from the popup menu. The Output Control Properties tab displays (see Figure 4-16).

Figure 4-16 Output Control Properties Tab

Properties	Used In	Extended Properties
Output Control Name: <input type="text" value="CALC_PAY"/>		
Output Control Type:	<input type="text" value="Data Field (Name Search)"/>	Length Type: <input type="text" value="Not Applicable"/>
Collection	<input type="text" value="CalcPay"/>	Tag Type: <input type="text" value="Not Applicable"/>
Output Operation:	<input type="text" value="CalcPay"/>	Optional: <input type="checkbox"/>
		Tag Before Length: <input type="checkbox"/>
Data Type:	<input type="text" value="String"/>	Input Tag Value: <input type="text" value="NONE"/>
Base Data Type	<input type="text" value="0"/>	Calculation Type: <input type="text" value="0"/>
Data Format:	<input type="text" value="Not Applicable"/>	Field Value: <input type="text" value="NONE"/>
Output Control Attribute Operations		
Case	Default	<input type="text" value="ASCII"/> <input type="text" value="EBCDIC"/> <div style="border: 1px solid gray; height: 100px; width: 100%;"></div>
<input type="radio"/> Lower <input type="radio"/> Upper <input checked="" type="radio"/> None	Name: <input type="text" value="NONE"/> Value: <input type="text" value="NONE"/>	
Justification	Length	HEX Value:
<input type="radio"/> Center <input type="radio"/> Left <input type="radio"/> Right <input checked="" type="radio"/> None	Name: <input type="text" value="NONE"/> Value: <input type="text"/> Pad: <input type="text" value="NONE"/>	<input type="text" value="0x"/>
<input type="button" value="Apply"/> <input type="button" value="Cancel"/> <input type="button" value="Help"/>		

- From the Output Operation drop down list, choose the **CalcPay** collection and click Apply. This assigns the collection to the **Pay** field.

5 Using the Tester

The Information Integrator Tester utility allows you to parse and reformat messages as a validation test. Using the Tester utility, you can make sure the message formats you built using the Formatter or MsgDefAdmin tool produce the expected results. For more information on the Formatter tool, refer to “Building Format Definitions.” For more information on the MsgDefAdmin tool, refer to “Using MsgDefAdmin.”

This chapter discusses the following topics:

- Starting the Tester
- Parsing and Reformatting Messages Using the Tester

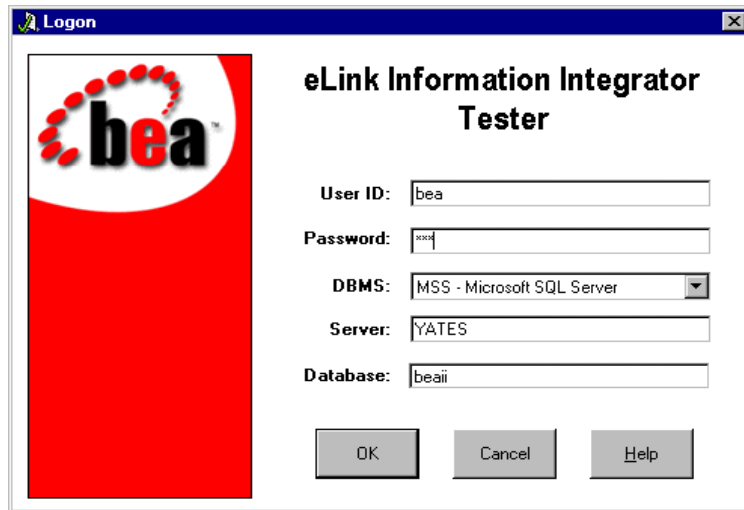
Starting the Tester

You can launch Tester using the steps described in the following procedure.

1. Double-click the Tester icon. The Tester Logon dialog box appears as in Figure 5-1.

Note: If you do not have an assigned user ID and password, ask your system administrator to create them for you.

Figure 5-1 Tester Logon



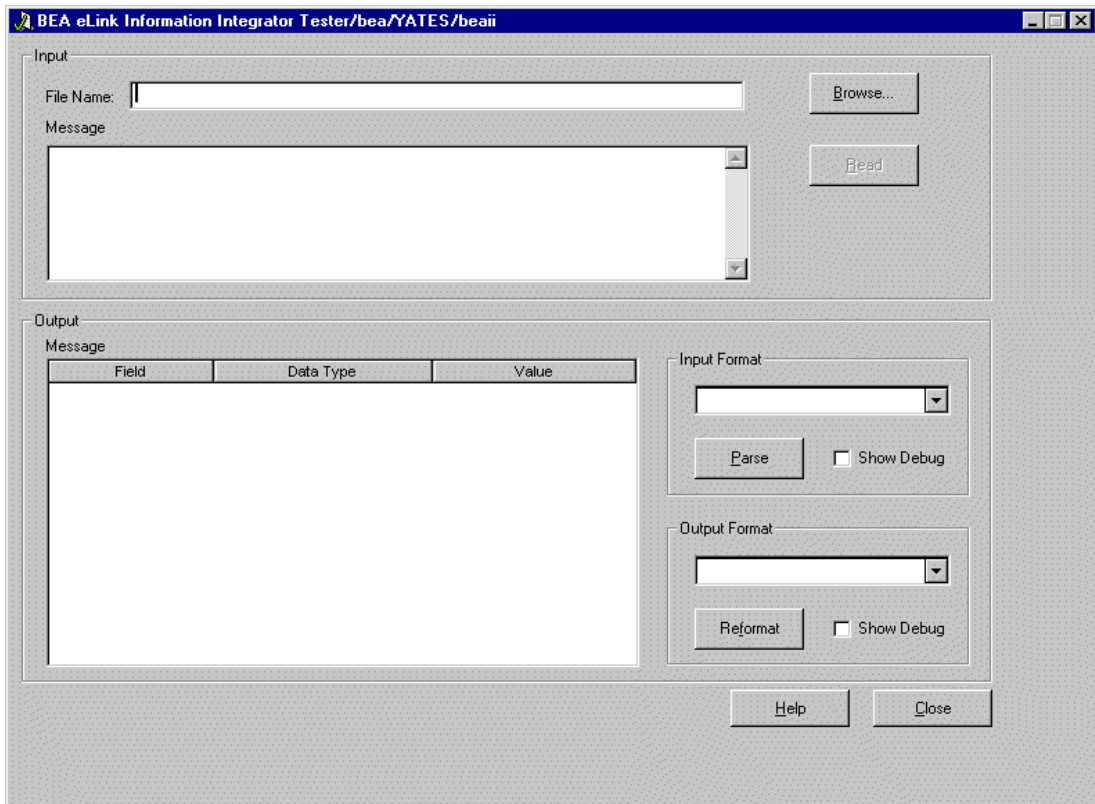
2. Enter your user ID in the User ID field and press **Tab**.
3. Enter your password in the Password field and press **Tab**.
4. Select a database type from the Database drop-down list box. The database field appears or disappears based on your selection.

If you selected the Microsoft SQL server type, specify the server in the Server field and the database in the Database field.

If you selected an Oracle database type, specify the server in the Server field.

5. Click OK. The Tester main window appears as in Figure 5-2.

Figure 5-2 Tester Main Window



This dialog contains the parameters for parsing Formatter messages. The parameters are described in Table 5-1 below:

Table 5-1 Tester Dialog Parameters

Parameter	Description
Filename	The name of the file that contains the messages to test. You can enter the file name directly into the field or click Browse to select from a list of files.
Browse	Displays the Open dialog, where you can select the file that contains the messages to test. The file name you select is copied to the File Name field when the Open dialog is closed.

Parameter	Description
Message	The message to test. You can enter the message directly into the Message text area or click Read to extract the message from the file specified in the File Name field.
Read	Extracts the message from the file name specified in the File Name field and displays the extracted message in the Message text area. This button is disabled until you enter a file name in the File Name field.
Input Format	Specifies the input format to use for parsing the message. The available values depend on the formats specified in the Formatter database.
Output Format	Specifies the output format to use for reformatting the message. The available values depend on the formats specified in the Formatter database.
Parse	Parses the message displayed in the input Message text area and displays the parsed message in the output Message text area. The output is divided into three columns: Field, Data Type, and Value.
Reformat	Parses and reformats the input message. The reformatted message is displayed in the output Message text area.
Show Debug	Displays details of the parsing or reformatting in the output Message text area.
Close	Closes this dialog.

Parsing and Reformatting Messages Using the Tester

You can parse and reformat messages using the Tester by following the steps below.

1. Choose one of the following options:

Specify the name of the file that contains the message you want to test in the File Name field, or click Browse to view a list of files.

If you click Browse, select the file that contains the message you want to parse in the Select File dialog. The file name is automatically copied to the File Name field when the Select File dialog is closed.

Once a file is specified in the File Name field, the Read button is enabled. Click Read to extract the message from the file specified in the File Name field. The message appears in the Message field.

Enter the message to test in the Message field. If you are parsing and reformatting binary data, you can use the following escape characters: `\a`, `\n`, `\r`, `\t`, and `\xFF`.

2. Select the input format for parsing from the drop-down list box in the Input Format group.
3. Click Parse to parse the message. The results are displayed in the Message field to the left of the Input Format group. If you want to view the details of the parse, select the Show Debug checkbox before clicking Parse.
4. Select the output format for reformatting from the drop-down list box in the Output Format group.
5. Click Reformat to reformat the message. The results are displayed in the Message field to the left of the Output Format group. If you want to view the details of the reformat, select the Show Debug checkbox before clicking Reformat.

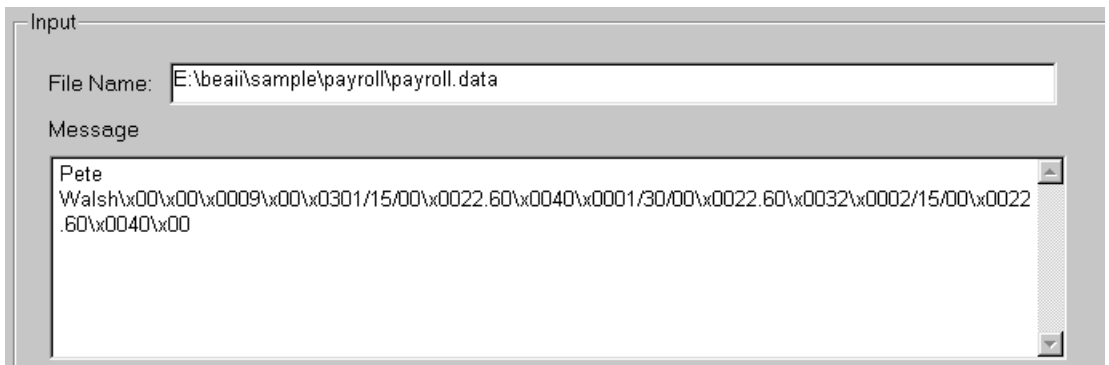
Example

This section outlines the procedure required to complete Step 5 in “Using Information Integrator.” In Step 5, you use the Tester utility to test the message formats you built in Step 3 (“Step 3. Create the Message Format Descriptions”) and Step 4 (“Step 4. Create Output Operations”). By testing message formats using the Tester utility, you can identify and correct errors in your formats before using them to parse your actual data.

In the steps that follow, we will load the input data in the Tester utility and test the formats.

1. Start the Tester utility.
2. Click Browse and open the file `\sample\payroll\payroll.data`. This file contains the input data for the sample application. It is provided on the installation CD-ROM, and is located in the `\sample\payroll` sub-directory under the directory where you installed Information Integrator.
3. Click Read. The file is read into the Tester and appears in the Message text area (see Figure 5-3).

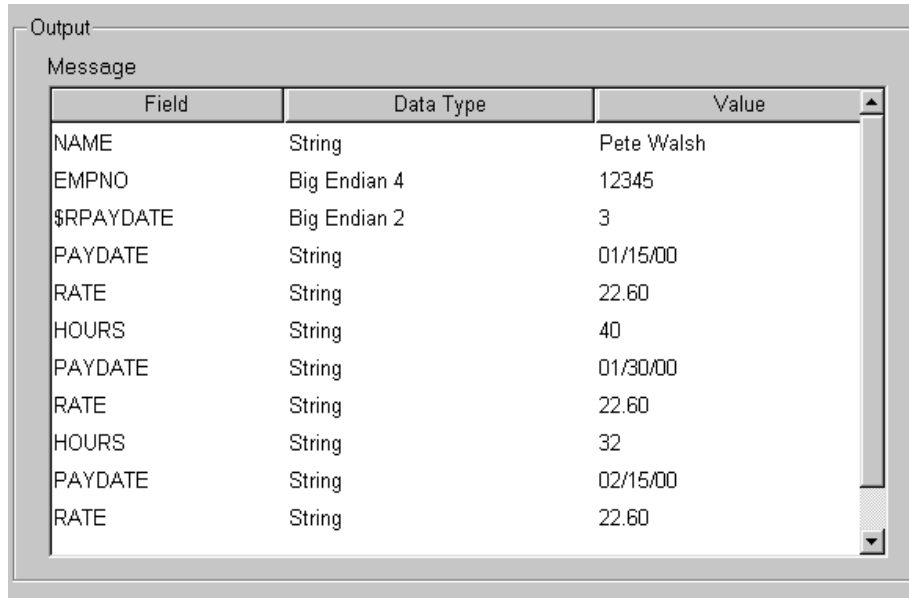
Figure 5-3 Tester window showing input data



4. From the Input Format drop down list, select `II_payroll_msg1_I`. This is the input format that was created when you processed the file `II_payroll_msg1.xml` using `MsgDefAdmin` (see “Step 3. Create the Message Format Descriptions” for more information).

5. Click Parse. The parsed message displays in the Message text area (see Figure 5-4).

Figure 5-4 Parsed message

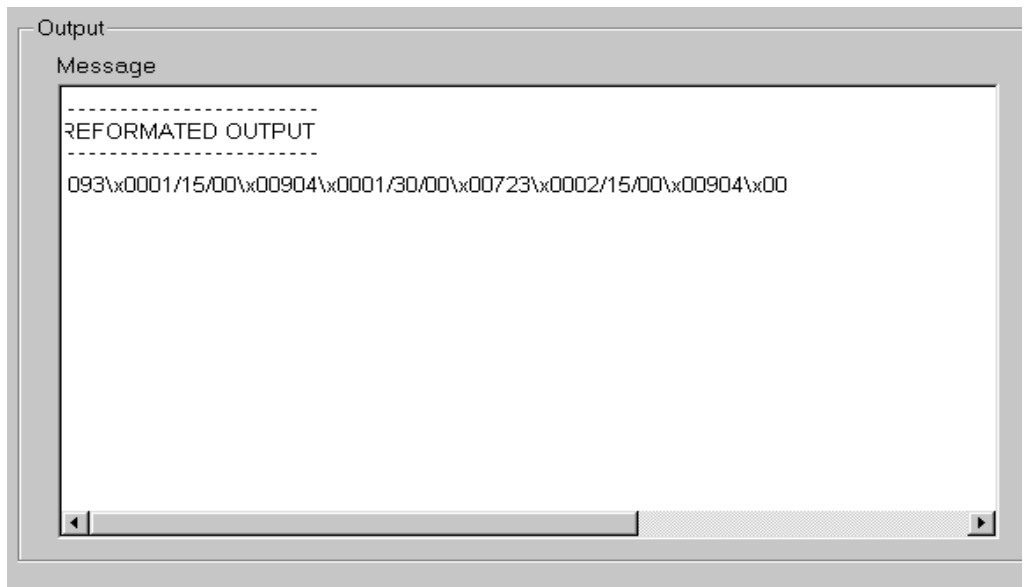


The screenshot shows a window titled "Output" with a sub-section "Message". Inside, there is a table with three columns: "Field", "Data Type", and "Value". The table contains the following data:

Field	Data Type	Value
NAME	String	Pete Walsh
EMPNO	Big Endian 4	12345
\$RPAYDATE	Big Endian 2	3
PAYDATE	String	01/15/00
RATE	String	22.60
HOURS	String	40
PAYDATE	String	01/30/00
RATE	String	22.60
HOURS	String	32
PAYDATE	String	02/15/00
RATE	String	22.60

6. From the Output Format drop down list, select `II_payroll_msg2_o`. This is the output format that was created when you processed the file `II_payroll_msg2.xml` using `MsgDefAdmin` (see “Step 3. Create the Message Format Descriptions” for more information).
7. Click Reformat. The reformatted message displays in the Message text area (see Figure 5-5).

Figure 5-5 Reformatted message



6 Defining Rules

This chapter describes Information Integrator Rules. The following topics are discussed:

- Understanding Information Integrator Rules
- Starting Rules
- Building Rules
- Example

Understanding Information Integrator Rules

Rules determine the runtime behavior of Information Integrator. Rules enable you to evaluate the data contained in a message and determine the Actions or tasks that will be performed based on the evaluation of that data.

A Rule consists of two parts: an Expression and a Subscription List. An Expression is the evaluation criteria for a Rule. It consists of a Boolean expression created from the fields in a message and comparison operators. A Subscription List contains multiple Subscriptions. Each Subscription consists of one or more Actions. An Action is an atomic task that is performed on the data in the message, such as, Reformatting the data or Enqueuing the data to a queue. When a Rule's Expression evaluates to TRUE, the Subscriptions in the Rule's Subscription List will be executed.

Application Groups and Message Types categorize Rules. The coarsest division of Rules is into Application Groups. Application Groups are logical divisions of sets of Rules for different business needs. For example, there might be an Application Group for each department in a company. The Message Type further categorizes Rules. It is the same as the Input Format that is used to parse the message (see “Building Format Definitions” for more information on Input Formats). Each Application Group can have multiple Message Types, and each Message type may contain multiple Rules.

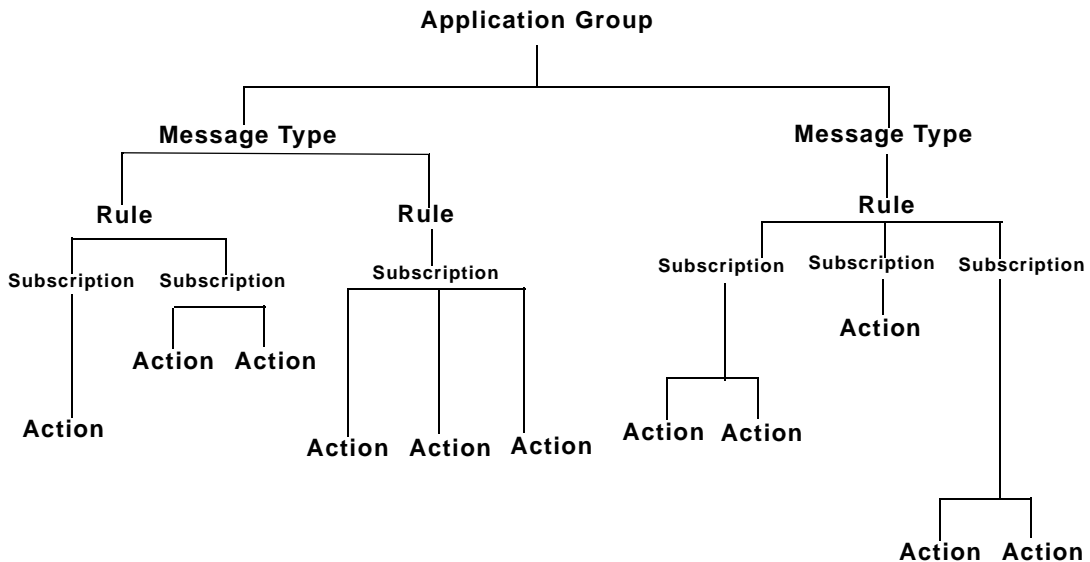
A service advertised by the Information Integrator Server is associated with a unique Application Group and Message type. When a client to the IISERVER makes a request for a particular service, the Expression for each Rule associated with that Message Type under the given Application Group is evaluated. The subscriptions associated with each TRUE Expression are then executed.

Using the Rules graphical user interface (GUI) you can define the various components of Rules in the following order:

- Application Groups
- Message Types
- Rules
- Expressions
- Subscriptions
- Actions

Rules are defined within an application group/message type pair, and are uniquely identified by the application group/message type/rule name. Figure 6-1 illustrates the Rules hierarchy.

Figure 6-1 Rules Hierarchy Diagram



Using the Rules Interface

You use the Rules graphical user interface (GUI) to define the rules components. The following sections describe the Rules GUI.

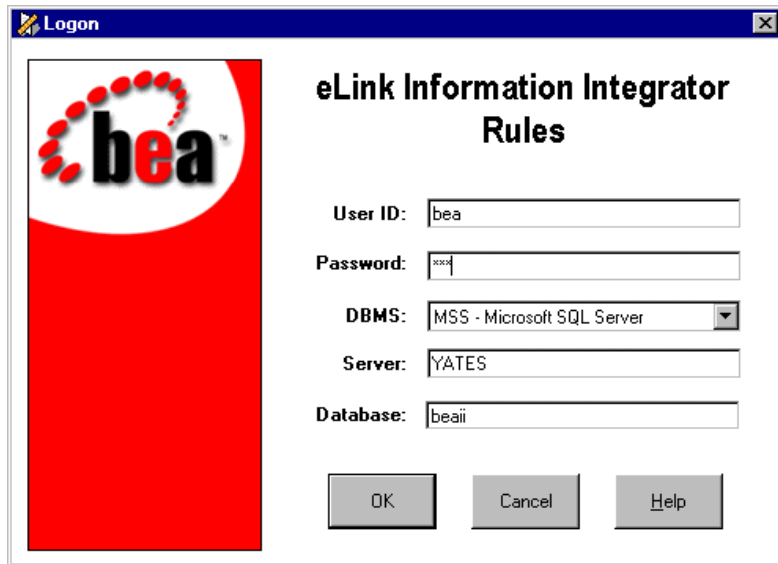
Starting Rules

Start the Rules GUI as follows.

1. Double-click the Rules icon. The Rules Logon dialog box appears as in Figure 6-2.

Note: If you do not have an assigned user ID and password, ask your system administrator to create them for you.

Figure 6-2 Rules Logon



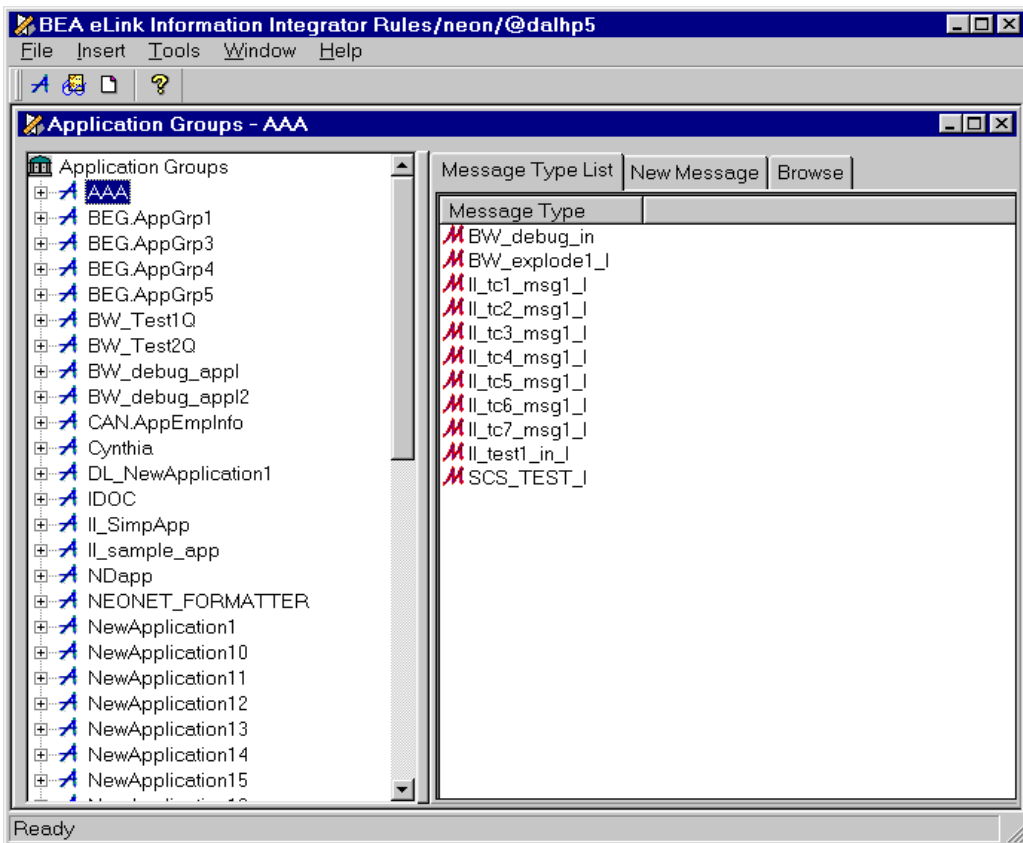
2. Enter your user ID in the User ID field and press **Tab**.
3. Enter your password in the Password field and press **Tab**.
4. Select a database type from the DBMS drop-down list box. The database field appears or disappears based on your selection.
 - If you selected the Microsoft SQL server type, specify the server in the Server field and the database in the Database field.
 - If you selected an Oracle database type, specify the server in the Server field.
5. Click OK. The Rules main window appears as in Figure 6-3.

Using the Rules Window

This section gives you instructions and tips for navigating through the Rules GUI. Refer to it for assistance in working in the Rules application as needed.

Each Rules window is split into two panes (see Figure 6-3). The left pane is a navigation tool and the right pane displays details about components selected in the left pane.

Figure 6-3 Rules Main Window



The left pane displays a list of all Application Groups and their associated Message Types, Rules, and Subscriptions. Single-click each category to just display a list of its contents in the right pane. Double-click each category to display its contents in the tree. For example, double-click to expand an Application Group. You can then double-click a Message Type name (or click on the + sign to its left) to display its associated Rules and Subscriptions.

The right pane, or Property Sheet, is where you will add the majority of information. It uses tabbed sheets to display the details for whichever object is currently selected. For example, if you have an Application Group selected, you can view the Message Type List and New Message tabs.

Note: You can open more than one Rules window at a time for access to multiple component types at the same time.

Opening a New Window

To open a new window, select **New** from the File menu.

Tip When the Rules window opens, it may default to a size too small to display all window components, including the Apply and Cancel on some property sheets. Maximize the application or resize the window to see all window components.

Renaming Components

Rules components can be renamed in the left pane of the Rules window. To rename a component, follow these steps.

1. In the left pane of the Rules window, select the component name.
2. Click again without moving the mouse a text box appears around the selected component name and is highlighted.
3. Edit the component name.
4. Press ENTER or click the mouse outside the component name text box. If there is no duplicate name, the component name is highlighted and moves to its alphabetical location in the list.

Note: Since Message Types are the same as the Input Formats created using Formatter or MsgDefAdmin, you cannot change Message Type names from the Rules window. Message Types must be renamed in Formatter. For more information, see “Using the Formatter Interface.”

Duplicating Components

Rules components can be duplicated in the left pane of the Rules window. To duplicate a component, follow these steps.

1. In the left pane of the Rules window, select any user-defined component and right-click. A popup menu appears.
2. Select Duplicate. A copy of the component is made and a text box appears around the duplicated component name.
3. Type the new component name.
4. Press ENTER or click the mouse outside the component name text box. If there is no duplicate name, the component name is highlighted and moves to its alphabetical location in the list.

Deleting Components

Rules components can be deleted in the left pane of the Rules window. To delete a component, follow these steps.

1. In the left pane of the Rules window, select a user-defined component and right-click. A popup menu appears.
2. Select Delete. A dialog box appears asking if you are sure you want to delete the component.
3. Choose either Yes or No to confirm or cancel your action.

Using Drag and Drop

You can use drag and drop to:

- Assign Actions to Subscriptions
- Assign Subscriptions to Rules

Drag and drop works across both windows and panes. To copy an object using drag and drop, you must drag the object from the first window's tree to the second window's property sheet.

Building Rules

This section describes how to build the following Rules components:

- Application Groups
- Message Types
- Rules
- Expressions
- Subscriptions
- Actions

Application Groups

An application group allows you to logically organize rules associated with a particular subject. For example, an application group could be the accounting department of a company.

To add an application group, follow these steps.

1. Select the New Application command under the Insert menu on the menu bar.

A text box appears at the top of the application group list. The cursor is positioned in the application group text box where you can type an application group name.

Note: The application group name must be 32 characters or less.

2. Press **Enter** to add the application group.

The application group is highlighted and alphabetically positioned in the list in the left pane, and the New Message property sheet appears in the right pane.

An application group should contain at least one message type, which corresponds to your input formats. For the procedure to add a message type, refer to “Message Types.”

Message Types

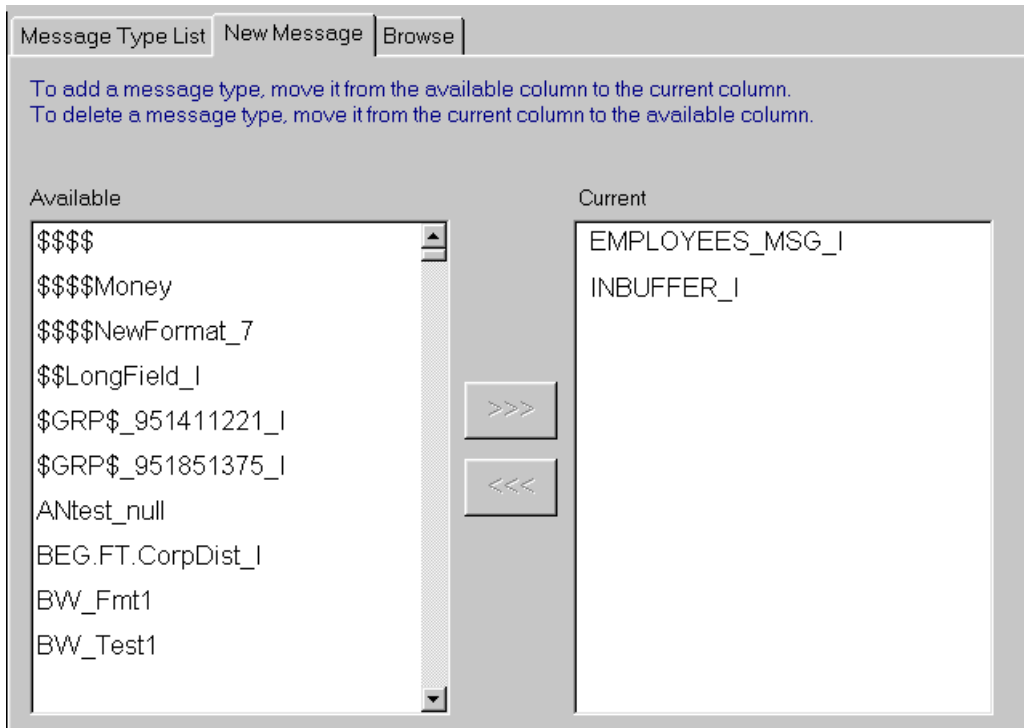
The message type is the input format name from Formatter. A rule is evaluated based on the message type.

An application group should contain at least one message type. For the procedure to add an Application Group, see “Application Groups.”

To add a message type, follow these steps.

1. Select the application group to which you want to add a message type. The list of current message types and the New Message tab is displayed in the right pane.
2. Select the New Message property sheet. The property sheet appears as in Figure 6-4.

Figure 6-4 New Message Property Sheet



The Available box on the left displays available message types. This list contains the input format names defined in Formatter.

The Current box on the right displays the message types currently associated with the application group.

3. Select the message type from the Available list box and click >>> or double-click to add a message type to the selected application group. The message type appears in the Current list box and in the left pane below the application group.
4. To delete a message type from the selected application group, select the message type from the Current list box and click <<< or double-click. The message type is removed from the Current list box.

Rules

A rule contains subscriptions that allow you to define message destination IDs, receiver locations, message formats, and any processes initiated upon message delivery.

To add a rule, follow these steps.

1. Right-click the message type for which you want to define a rule. A pop-up menu appears.
2. Select New Rule. The cursor is positioned in the rule text box where you can type a unique rule name.

Note: The rule name must be 32 characters or less.

3. Press **Enter** to add the rule. The rule is highlighted and alphabetically positioned in the list to create a new rule in the left pane. The Expressions property sheet appears in the right pane.

Note: When the first rule is created, a subscription list is automatically created for the application group.

For the Rules engine to correctly process a rule, you must define the rule's expression and subscription(s) or subscription list. The procedure to add an expression to a rule is described in "Expressions." The procedure to add a subscription to a rule is described in "Subscriptions."

Disabling and Enabling a Rule

You can temporarily "turn off" a rule by disabling it. This means that the message will not be evaluated against the disabled rule.

To disable a rule, follow these steps.

1. Right-click the rule you want to disable. A pop-up menu appears.
2. Select Disable. The rule is now disabled for the message.

When you want to "turn on" the rule again, follow these steps.

1. Right-click the rule you want to enable. A pop-up menu appears.
2. Select Enable. The rule is now enabled for the message.

Notes: Rules are automatically enabled when you create them, and remain enabled until you disable them. Rules remain disabled until you enable them again.

Expressions

The Expression tab is used to create, modify, or delete an expression for a selected rule. You can create an expression by entering it directly in the Expression text area, or by building it interactively using the sub-tabs at the bottom of the Expression tab (see Figure 6-5). The following sub-tabs appear within the Expression tab:

- Expression Components — Contains a list of expression templates, combining one or more sets of arguments and Boolean operators for a complete expression phrase.
- Field List — Contains a list of field names associated with the selected message type. You define field names when you create an input format.
- Operators — Comparison operator choices. For an explanation of all valid operators, refer to “Operators.”
- Values — Contains the matching data types of the operator in the operation expression field.
- Functions — Symbols used in the expression such as & (AND), | (OR), and parentheses

The evaluation criteria for a rule consists of a Boolean expression containing the Boolean operators & (AND) and | (OR), expressions, and parentheses to control the order of evaluation. You can use | to explicitly direct what Boolean operations do together.

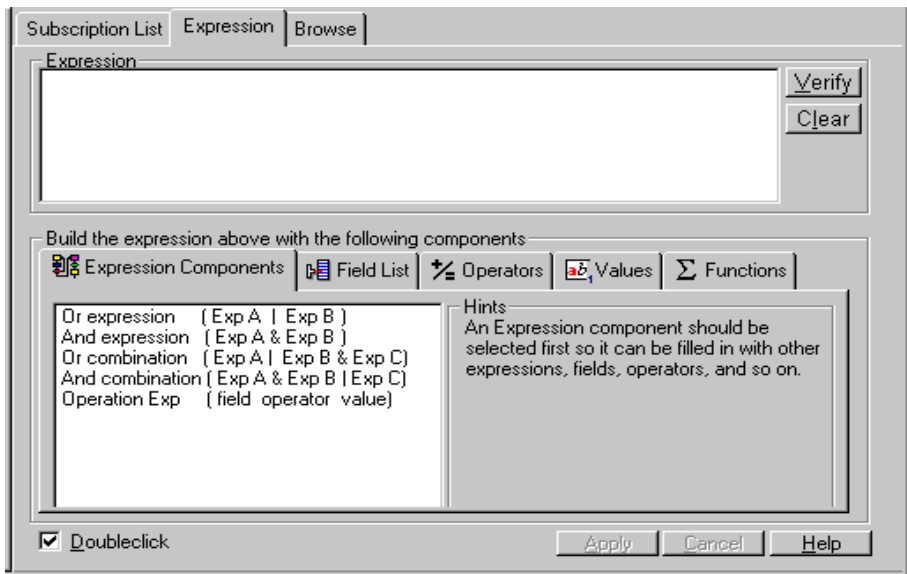
Notes: There must be at least one space between the field name and the Rules operator, as well as between the Rules operator and the comparison value. The `EXIST` and `NOT_EXIST` operators should be followed by at least one space before a parenthesis or a Boolean operator.

Creating or Modifying an Expression

To create or modify an expression, follow these steps.

1. Select the appropriate rule.
2. Select the Expression tab. The tab appears as in Figure 6-5.

Figure 6-5 Expression Tab



3. Select the Expression Components sub-tab.
4. If you want to double-click expression components to add them to the expression, check the Doubleclick checkbox. If you would rather single-click expression components to add them to the expression, clear the Doubleclick checkbox.
5. Build the rest of the expression by using the other subtabs (Field List, Operators, Values, Functions) and clicking or double-clicking their components as needed.
6. Click Verify if you want to check the final expression and make sure it is valid. If it is not, an error message displays, showing you the location of the error in the expression. Correct the error by doing one of the following:

- a. Click Input on the error message window and enter the correction in the User Input text area.
 - b. Click OK on the error message window and correct the error directly in the Expression text area.
- Note:** Click Clear to erase the contents of the Expression text area.
7. When you have finished creating or updating the expression, click Apply to save the expression to the Database.

Notes: The literals AND and OR do not evaluate correctly in a rules expression. If you enter these values, a message box appears stating the expression is valid; however, AND is changed to &, and OR is changed to |.

If you move out of the Expression tab by using the mouse or the keyboard, the expression is automatically saved.

Examples of expressions are shown in Table 6-1.

Table 6-1 Expression Examples

Description	Layout	Expression
Default to TRUE		TRUE
Single Argument	A	F1 STRING= Acct
Arguments use AND	A & B & C	F1 STRING= Acct & F2 INT= 100 & F3 INT= 150
Arguments use OR	A B C	F1 STRING=Acct F2 INT= 100 F3 INT= 150
Precedence	A B & C	F1 STRING= Acct F2 INT= 100 & F3 INT= 150
Nested Prens	(A ((B & (C)) D))	(F1 STRING= Acct ((F2 INT= 100 & (F3 INT= 150)) F4 INT= 200))

Subscriptions

Subscriptions are lists of actions to take when a message evaluates true. Each rule must have at least one associated subscription. Subscriptions enable you to define message destination IDs, subscriber locations, message formats, and any processes initiated on message delivery.

Subscriptions are created in Subscription Lists. There are two types of subscription lists. There is a subscription list for each application group/message type. This list is created when the first rule is added to the message type. There is also a subscription list for each rule. This list contains only those subscriptions that are associated with the rule (see “Associating Subscriptions with Rules” for more information).

To add a subscription, follow these steps.

1. Right-click the Subscription List for the Application Group/Message Type to which you want to add a subscription. A pop-up menu appears.
2. Select New Subscription. The cursor is positioned in the Subscription List text box where you can type a unique subscription name.
Note: The subscription name must be 64 characters or less.
3. Press **Enter** to add the subscription to the Subscription List. If there is no duplicate name for this subscription, the subscription is added and the Actions tab appears.
4. To add a comment to the Subscription, choose the Misc tab and type the comment in the Comments field.
5. Click Apply when finished. This creates the subscription list for the application group/message type.
6. Continue with the procedures described in “Associating Subscriptions with Rules.”

Notes: Each subscription must have at least one action. For the procedure to add an action, see “Actions.”

For hints on creating subscriptions by using a combination of actions, refer to “Combining Actions to Create Subscriptions.”

Associating Subscriptions with Rules

Creating a subscription does not automatically associate the subscription with a rule. In order for the actions in the subscription to be executed, you must associate the subscription with a rule by doing the following:

1. From the subscription list for the application group/message type, click and hold the left mouse button on the desired subscription.
2. While holding the left mouse button, drag the subscription from the right pane to the desired rule in the left pane.
3. When the cursor is over the desired rule, release the left mouse button. The subscription is now associated with the rule, and appears below the rule in the left pane.

Note: Subscriptions may only be assigned to a rule within the same message type and application group pair. A subscription can be assigned to several rules if the rules are in the same application group and message type.

Actions

Actions hold subscription instructions. The Actions affecting subscriptions are as follows:

- **Reformat** — Identifies the input and output message formats used when reformatting a message. In addition, the numbers of the input and output messages are assigned. These numbers may be used in other actions to identify the buffer to be used. For instructions on adding a Reformat action, refer to “Adding a Reformat Action.”
- **Enqueue** — Identifies all the data required to place a typed buffer on a Tuxedo queue. In addition, you can optionally specify the values to be used for reply queue, failure queue, and priority. For instructions on adding an Enqueue action, refer to “Adding an Enqueue Action.”
- **Post** — Permits entry of all the information required to post a typed buffer to the Tuxedo event broker. For instructions on adding a Post action, refer to “Adding a Post Action.”

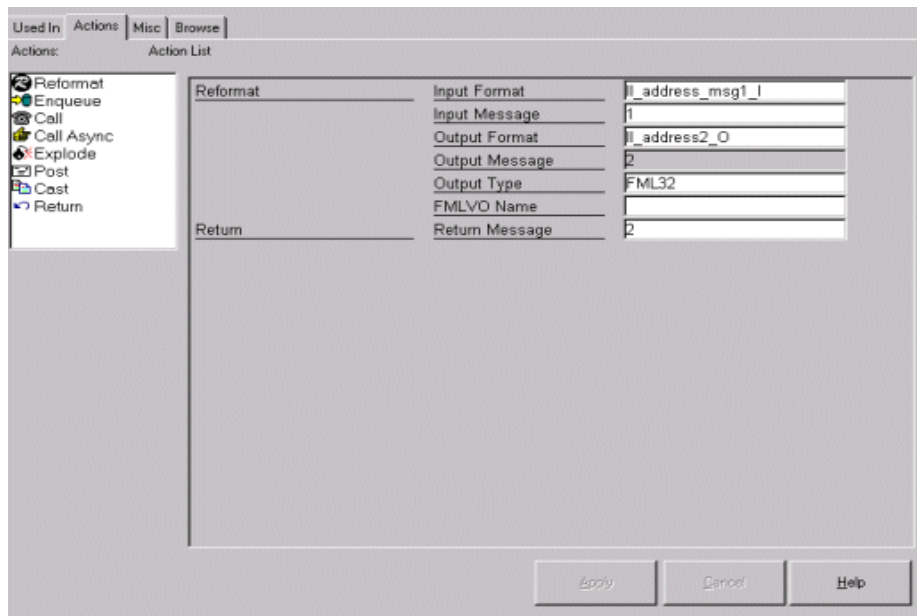
- **Call** — Obtains the necessary information to send a typed buffer to another Tuxedo service and accept a reply from that service. For instructions on adding a Call action, refer to “Adding a Call Action.”
- **Call Async** — Gathers information required to send a typed buffer to another Tuxedo service. For instructions on adding a Call Async action, refer to “Adding a Call Async Action.”
- **Explode** — Parses an input message into its component parts using a supplied input format. After parsing is complete, explode performs the next action in the subscription list for each member of the subcomponent. The explode action taken for each action type includes:
 - Enqueue—The enqueue action is performed for each matching subcomponent.
 - Post—The post action is performed for each matching subcomponent.
 - Call Async—The call async action is performed for each matching subcomponent.For instructions on adding an Explode action, refer to “Adding an Explode Action.”
- **Cast** — Reformats an entire message to a different type of buffer. For instructions on adding a Cast action, refer to “Adding a Cast Action.”
- **Return** — Identifies the buffer that is returned to the client. The return action must be the last action defined. For instructions on adding a Return action, refer to “Adding a Return Action.”

Note: The Reformat, Call, Cast, Explode, and Return actions cannot be used after an Explode action.

To add an action to a subscription, follow these steps.

1. Select the subscription to which you want to add an action. The Actions tab appears as in Figure 6-6.

Figure 6-6 Actions Tab



2. Select the action that you want to add, hold down the left mouse button, and drag the action to the Action List box. Then do one of the following:

To add the action to the bottom of the list, drop the action onto the gray space below the last action in the list.

To insert the action above an existing action, drop the action on top of the position you want the action to be above.

3. Click Apply to save the actions.

Adding a Reformat Action

1. Use the Input Format drop down list to select an Input Format.
2. Use the Input Message drop down list to select the message to be reformatted (if this is the first action, this will be message 1).
3. Use the Output Format drop down list to select an Output Format.

4. The Output Message is automatically generated.
5. Use the Output Type to select an Output type for this message.
6. If FMLVO is selected as the Output Type, enter the FML field name for output data.

Adding an Enqueue Action

1. Use the Input Message drop down list to select the message to be placed on a Queue (if this is the first action, this will be message 1).
2. Enter the name of the Queue Space where the message will be sent.
3. Enter the Queue Name where the message will be sent.
4. Enter the name of the Reply Queue (this field is optional).
5. Enter the name of the Failure Queue (this field is optional).
6. Select a Priority for this request (this field is optional).

Adding a Post Action

1. Use the Input Message drop down list to select the message to be Posted to the Tuxedo event broker (if this is the first action, this will be message 1).
2. Enter the Event Name to be Posted

Adding a Call Action

1. Use the Input Message drop down list to select the message to be input to the called service (if this is the first action, this will be message 1). The Output Message is automatically generated.
2. Enter the Service Name that will be Called.
3. Enter the FML field name for output data. This field is optional and is only used for FML value processing.
4. Enter the name of the Error Queue Space (this field is optional).
5. Enter the name of the Error Queue (this field is optional).

Adding a Call Async Action

1. Use the Input Message drop down list to select the message to be input to the called service (if this is the first action, this will be message 1).
2. Enter the Service Name to be Called.

Note: No reply is returned from the called service.

Adding an Explode Action

1. Use the Input Format drop down list to select an Input Format.
2. Use the Input Message drop down list to select the message to be reformatted (if this is the first action, this will be message 1).
3. Use the Output Format drop down list to select an Output Format.
4. Use the Output Type to select an Output type for this message.
5. If FMLVO is selected as the Output Type, enter the FML field name for output data.

Note: The Explode action cannot be followed by any action that creates new buffers, such as Reformat, Call, or Cast. Explode is used for messages in which there are some variable number of identical parts. The action that immediately follows the Explode action is applied to each of the items in the exploded message.

Adding a Cast Action

1. Use the Input Format drop down list to select an Input Format.
2. Use the Input Message drop down list to select the message to be reformatted (if this is the first action, this will be message 1). The Output Message is automatically generated.
3. Use the Output Type to select an Output type for this message.
4. If FMLVO is selected as the Output Type, enter the FML field name for output data.

Adding a Return Action

- Use the Input Message drop down list to select the message to be returned (if no buffer is to be returned, select 'NULL Message')

Deleting an Action

To delete an action, follow these steps.

1. Right-click on the action you want to delete in the Action List. A pop-up menu appears.
2. Choose Delete Action. The action is deleted from the Actions tab.
3. Click Apply to delete the action from the subscription.

Note: If you delete an action that generates a new buffer (Reformat, Call, or Cast), the message numbers for subsequent actions in the subscription may change. This is to account for the fact that the output message from the deleted action no longer exists. All input and output message numbers greater than the deleted output message numbers are decremented. Any input messages that used that output message number will be blank, and you must manually reset these to valid message numbers.

Combining Actions to Create Subscriptions

The Information Integrator Rules application allows you to accomplish multiple tasks on messages using subscriptions. Subscriptions are simply actions that you combine in the appropriate order to achieve the desired results. Any number of actions can be combined to create a subscription. Therefore, the incoming message must be able to be successfully parsed with the Input Format for the first action in a subscription. The Input Format for the first action in a subscription will generally be the same as the Application Group's Message Type. The Input Message for the first action in the subscription is automatically set to 1, signifying that the input buffer for this action is the original message buffer.

The Reformat, Call, and Cast actions generate new buffers. When any one of these actions is added to a subscription, the Output Message is automatically set to one number greater than the highest Message number in the subscription. The new message number is added to the drop down list of possible Input Message numbers for all actions added after that action.

The Explode action is used for messages with a variable number of identical parts. The action that immediately follows the Explode action is applied to each of the component pieces of the exploded message. The Input Message parameter for the action following Explode can be any of the possible values. Since it is the action following an Explode, the results of the Explode are automatically used as the Input Message for this action. Explode cannot be followed by any action that creates new buffers, such as Reformat, Call, or Cast. If the results of an Explode action need to be reformatted, this reformatting must be configured in a separate subscription.

If a Return action is included in a subscription, it must be the last action in the subscription. This is because rule evaluation terminates at the end of all rules or at the first Return action encountered.

Example

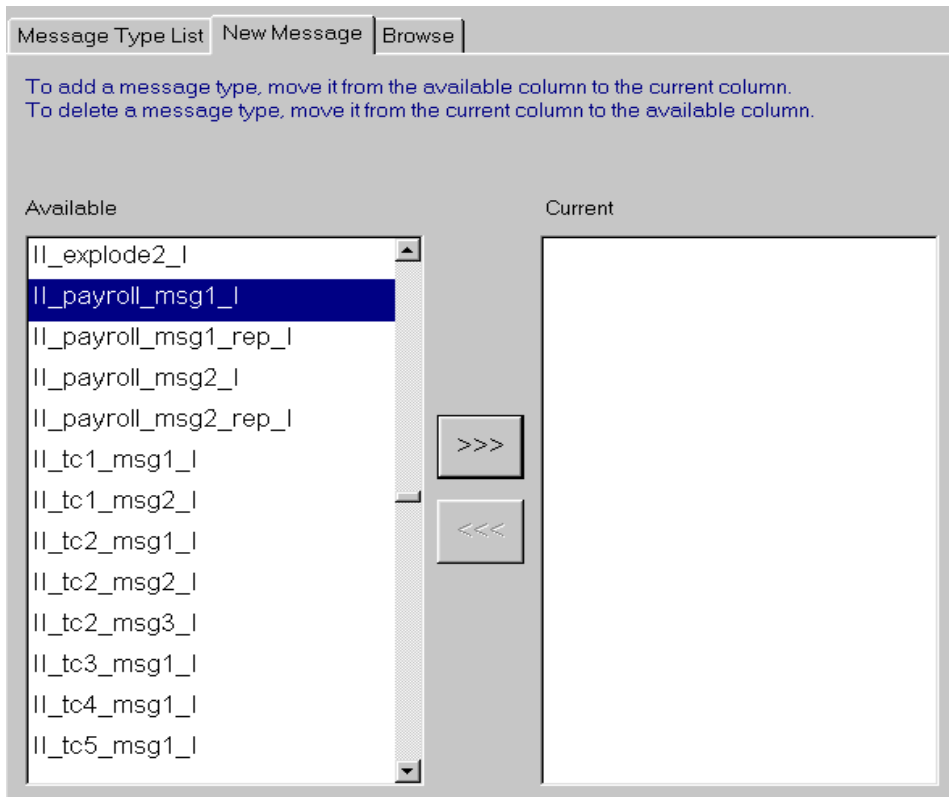
This section outlines the procedure required to complete Step 6 in “Using Information Integrator.” In Step 6, you define the actions required to perform the data translation for the sample application.

In the steps that follow, we will create a new application group, message type, rule, and subscription list, and assign the subscription list to the rule.

To create a new application group, follow these steps.

1. Open the Rules application.
2. Right-click Application Groups in the left pane of the Rules window and select New Application from the popup menu.
3. Type `II_sample` as the name of the new application group in the left pane and press Enter. The New Message tab displays in the right pane see (Figure 6-7).

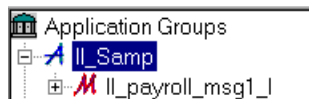
Figure 6-7 New Message Tab



We will now create a new message type in the new application group.

- From the New Message tab, select `II_payroll_msg1_I` from the Available list and click `>>>` to move it to the Current list. This adds the message to the `II_Sample` application group you created (see Figure 6-8).

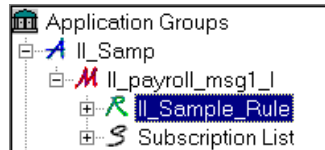
Figure 6-8 New Message Type



We will now create a new rule within in the new message type.

1. Right-click `II_payroll_msg1_I` in the left pane and select New Rule.
2. Type `II_sample_Rule` as the name of the new rule and press Enter. This adds the rule to the `II_payroll_msg1_I` message type you created (see Figure 6-9).

Figure 6-9 New Rule



We will now create an expression for the new rule.

1. On the Expression tab, type `TRUE` in the Expression text area. This tells Information Integrator that you want the actions in the rule to execute unconditionally.
2. Click Apply. This creates the expression for the `II_sample_Rule` (see Figure 6-10).

Figure 6-10 Expression for II_Sample_Rule

Subscription List | Expression | Browse

Expression

TRUE

Verify

Clear

Build the expression above with the following components

Expression Components | Field List | Operators | Values | Functions

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 can be filled in with other expressions, fields, operators, and so on.

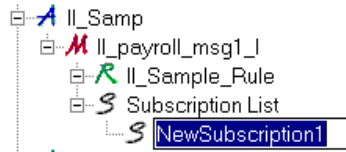
Doubleclick

Apply Cancel Help

We will now create a subscription list to hold the actions we want Information Integrator to perform.

1. Right-click **Subscription List** under **II_payroll_msg1_I** in the left pane and select **New Subscription**.
2. Type **II_sample_sub** as the name of the new subscription and press Enter. This adds the subscription to the subscription list for the **II_payroll_msg1_I** message type (see Figure 6-11).

Figure 6-11 New Subscription



We will now add actions to the new subscription. We want to reformat the input data and return the reformatted data.

1. On the Actions tab, click on the **Reformat** action and drag it into the Action List for this subscription.
2. From the Input Format drop down list, choose **II_payroll_msg1_I** to select it as the input format.
3. From the Input Message drop down list, choose **1** to assign the number “1” to the incoming message.
4. From the Output Format drop down list, choose **II_payroll_msg1_O** to select it as the output format.
Note: The Output Message defaults to **2**, assigning the number “2” to the output data.
5. From the Output Type drop down list, choose **FML32** to indicate that the output message is in FML32 format.
6. Click on the **Return** action and drag it into the Action List.
7. From the Return Message drop down list, choose **2**. This indicates that the data you want to return is the result of the Reformat action (Output Message 2).
8. Click Apply. This creates the subscription in the **II-sample_sub** subscription list (see Figure 6-12)

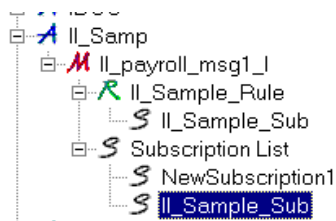
Figure 6-12 Subscription for II_Sample_Sub

Actions: Action List		
Reformat	Input Format	II_payroll_msg1_I
	Input Message	1
	Output Format	II_payroll_msg1_O
	Output Message	2
	Output Type	FML32
	FMLVO Name	
Return	Return Message	2

We will now add the subscription we just created to the `II_Sample_Rule`.

Note: This must be done in order for the actions in the subscription to be executed. Simply creating the subscription does not associate it with a rule.

1. Click `II_sample_sub` in the left pane of the Rules window and hold down the left mouse button.
2. Drag `II_sample_sub` to `II_Sample_Rule` and release the left mouse button. The subscription `II_sample_sub` appears under the `II_Sample_Rule`, indicating that it is associated with this rule, and its actions will be executed (see Figure 6-13).

Figure 6-13 `II_Sample_Sub` Associated with `II_Sample_Rule`

7 Configuring Information Integrator and the IISERVER

In order to run Information Integrator, you must configure the operating environment and define users. The configuration steps differ depending on whether you are running the Design environment or the Execution environment. In the Design environment, you create the formats and rules you want to use for processing your data. In the Execution environment, the IISERVER translates your data using the formats and rules you created during the Design environment.

This chapter discusses the following topics:

- Configuring the Design Environment
- Configuring the Execution Environment
- Understanding the IISERVER
- Creating the FML Field Table

Configuring the Design Environment

The following tasks are required to configure the Design environment of Information Integrator:

- Creating the Database Schema
- Creating the Database Session Configuration File
- Creating User Accounts

Creating the Database Schema

The first step in configuring the Design environment of Information Integrator is to set up the appropriate tables and stored procedures in the database. A script that does this for you is included on the installation CD-ROM. When you install Information Integrator, this script is placed in the `\InfoInt\install.sql` directory under your eLink Platform directory. Execute this script in a DOS window (for Windows NT systems) or at a UNIX command prompt by typing one of the following commands:

- On Oracle systems:

```
set SQLPLUS=<plus name>  
inst_db.sh SYS <SYS password> <Service Name>
```

For Oracle 7, *<plus name>* is **plus33**. For Oracle 8, *<plus name>* is **plus80**.

- On SQL Server systems:

```
inst_db.sh <username> <password> <servername> <dbname>
```

If you have not defined a password for your database username, use two single quotes to specify the password. For example:

```
inst_db.sh sa '' mycomputer
```

The required tables and stored procedures are automatically created in the Information Integrator database. A log of the schema creation procedure is written to `temp\inst_db.log`.

Creating the Database Session Configuration File

The next configuration step is to create the Database Session Configuration file (`sqlsvses.cfg`). A sample `sqlsvses.cfg` file that is commented out is provided on the installation CD-ROM. To edit the sample file with your site-specific information, uncomment the section that applies to your database type and edit the appropriate parameters.

Listing 7-1 shows the syntax for the Database Session configuration file.

Listing 7-1 Syntax for `sqlsvses.cfg`

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

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

Required Parameters

The following parameters must be defined in the `sqlsvses.cfg` file:

SESSIONNAME

Database session name to be used by Information Integrator executables. This can be any string as long as it is unique within the file.

SERVERNAME

Server where the Information Integrator database resides.

USERID

Database user ID.

PASSWORD

Database password.

DATABASENAME

Name of the database where the Information Integrator tables reside.

Note: The DATABASENAME parameter is not used for Oracle environments.

There must be a colon after the password in Oracle environments, even though the last parameter is not used.

Optional Parameters

There are no optional parameters for the `sqlsvses.cfg` file.

Sample Database Session Configuration File

Listing 7-2 shows the sample `sqlsvses.cfg` file included on the installation CD-ROM.

Listing 7-2 Sample Database Session configuration file

```
# -----  
#  
# Sqlsvses.cfg - Defines sessions to be used to access the Formatter  
# database. The format of each line depends on the database in which  
# the Formats are to be stored.  
  
#  
# Four fields are required in order to specify a connection to a  
# Oracle database. Five fields are required to specify a connection  
# to a SQLServer database.  
#  
# Entries can be commented by placing a "#" in column 1.  
# -----  
#  
# Format of lines for the various supported database types:  
#  
# MS SQLServer:  
# SessionName:dBaseServer:dBaseUserid:dBasePassword:dBaseInstance  
#  
# Oracle:  
# SessionName:dBaseService:dBaseUserid:dBasePassword:  
# (note that the fifth field is blank, so the last colon is required)  
#  
# -----  
#  
# The SessionName "import" is required to run the Message Definition  
# Importer Tool (MsgDefAdmin).  
#  
#  
# -----
```



```
#
# Example SessionNames (uncomment the SessionNames needed):
#
# MS SQL Server database: import:Server:Userid:Password:Instance
# (change Server, Userid, Password, and Instance above)
#
# Oracle database (final colon required on each line):
# import:Service:Userid:Password:
# (change Service, Userid, Password above)
# ----- ( End of File ) -----
```

Creating User Accounts

In order to use BEA eLink Information Integrator, each user must have an account created for them.

Creating Users on Oracle Systems

Users must have a user name and password for each database they want to access. Creating these user names and passwords can be done by the system administrator, security officer, or database owner. A single person can occupy one or more of these roles. Check with your database administrator for information about your specific environment.

During installation of Information Integrator on Oracle systems, the II_USER role is created for each user. This role gives users access to the Information Integrator database. To access databases, users must be created and associated with the II_USER role using the procedures described in the following sections.

Creating User Names

After you install Information Integrator, you must create user names in your database. User names identify individual users to the database. Listing 7-3 shows the syntax for creating users.

Note: This action must be performed while connected to the database as an administrator.

Listing 7-3 Syntax for creating users

```
create user <username> identified by <password>;
```

USERNAME and PASSWORD are required parameters.

For example, the following command creates a user with the login name “JOHNDOE” and the password “JDOE1”:

```
create user JOHNDOE identified by JDOE1;
```

Granting Roles to Users

Users must be given permissions to access the database data. You can either grant permissions to an individual user or create roles with certain permissions and associate users with specific roles. II_USER is a role created by the Information Integrator installation process. Listing 7-4 shows the syntax for granting user roles.

Note: This action must be performed while connected to the database as an administrator.

Listing 7-4 Syntax for granting user roles

```
grant II_USER to "username";
```

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

Defining User Synonyms

Once a user has been created, you must define the synonyms for that user. Connect to the database using the newly created USERNAME and PASSWORD; then, run the SQL_Plus executable `nn_synonyms.sql`, found in the `install.sql` directory. Listing 7-5 shows the syntax for defining synonyms.

Note: This action must be performed while connected to the database as the user for whom the synonym is being created.

Listing 7-5 Syntax for synonyms.sql

```
@nn.synonyms.sql
```

Creating Users on SQL Server Systems

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

Creating Login Accounts

Login accounts give users access to the SQL Server. They are created by the system administrator or security officer using the `sp_addlogin` system procedure.

Listing 7-6 shows the syntax for `sp_addlogin`.

Listing 7-6 Syntax for sp_addlogin

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

`loginName`

The login name being added.

`password`

The password associated with the login name being added.

`defdb`

Specifies a default database for the user.

`deflang`

Specifies the default language to use.

`full-name`

The full name of the user who owns this account.

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

Assigning Users to a Database

To use a database, the server login must be associated with a user name in the database. Users can be added to a database by the database owner (DBO) using the `sp_adduser` system procedure. This procedure must be run from the database in which the user is to be added. Listing 7-7 shows the syntax for `sp_adduser`.

Listing 7-7 Syntax for `sp_adduser`

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

loginName

The user's server login account.

nameInDB

The name for the user in the database. Defaults to the loginName if not specified.

group

Enables the DBO to add the user to an existing group in the database. If not specified, the user is placed in the default group, PUBLIC.

Defining User Groups

Each user added to the database must be granted permissions to access objects within that database, unless they are the database owner. During installation, the group `II_USER` is created for Information Integrator users. To access Information Integrator databases, users must be linked to the `II_User` group.

Users can be added using either the `sp_adduser` or `sp_changegroup` system procedures. Listing 7-7 describes the syntax for `sp_adduser`.

Listing 7-8 Syntax for `sp_changegroup`

```
sp_changegroup groupName, userName
```

`groupName`

Name of the group to which the user is added.

`userName`

The database user name.

Configuring the Execution Environment

The following tasks are required to configure the Execution environment of Information Integrator:

- Setting the Environment Variables
- Adding the `IISERVER` to the `UBBCONFIG` File
- Creating the Configuration File

Setting the Environment Variables

You need to set several environment variables before using Information Integrator. Table 7-1 lists the environment variables you need to set.

Table 7-1 Environment Variables Used By Information Integrator

Variable Name	Description
TUXDIR	Base directory of the eLink Platform software.
APPDIR	Base directory for applications.
PATH	Must include \$TUXDIR/bin.
TUXCONFIG	Full pathname of binary <code>tuxconfig</code> file.
LD_LIBRARY_PATH	Path where the libraries for the product and database reside. Must include \$TUXDIR/lib on systems that use shared libraries (except HP-UX and AIX).
SHLIB_PATH	Path where the libraries for the product and database reside. <i>HP-UX only</i> - Must include \$TUXDIR/lib.
LIBPATH	Path where the libraries for the product and database reside. <i>AIX only</i> - Must include \$TUXDIR/lib.
FLDTBLDIR	Path where FML field table resides.
FIELDTBLS	Name of the FML field table file.
FLDTBLDIR32	Path where FML32 field table resides. <i>FML32 only</i> .
FIELDTBLS32	Name of the FML32 field table file. <i>FML32 only</i> .

Scripts are often used to save time in setting environment variables in a development directory. The following scripts are provided to set these variables for you, but the scripts must be edited for your environment.

- \$APPDIR\setenv.bat for Windows NT systems
- \$APPDIR/setenv.sh for UNIX systems

A sample `setenv.bat` file is shown in Listing 7-9.

Listing 7-9 setenv.bat

```
set TUXDIR=<eLink Platform directory>
set APPDIR=<Application directory>
set TUXCONFIG=<Full pathname of the binary Tuxedo config file>
set FLDTBLDIR=%APPDIR%;%TUXDIR%\uataobj
set FIELDTBLS=Fieldtable.txt,Usysflds
set FLDTBLDIR32=%FLDTBLDIR%
set FIELDTBLS32=%FIELDTBLS%
set PATH=%PATH%;%TUXDIR%\bin
```

Notes:

On NT systems:

- eLink Platform sets the TUXDIR value and adds TUXDIR\bin to the PATH environment variable.
- You need to set the APPDIR, TUXCONFIG, FIELDTBLS32, and FLDTBLDIR32 variables.

On UNIX systems:

- eLink Platform generates a TUX.ENV file that sets the TUXDIR value and adds TUXDIR\bin to the PATH environment variables.
- You need to set the APPDIR, TUXCONFIG, FIELDTBLS32, FLDTBLDIR32, and the SHLIB_PATH (or LD_LIBRARY_PATH or LIBPATH) variables.
- LD_LIBRARY_PATH must include \$TUXDIR/lib on systems that use shared libraries (except HP-UX and AIX).
- SHLIB_PATH (HP-UX only) must include \$TUXDIR/lib.
- LIBPATH (AIX only) must include \$TUXDIR/lib.

Adding the IISERVER to the UBBCONFIG File

A message format and routing server (IISERVER) is part of Information Integrator. This server identifies available service names and their associated application names and input formats.

Before running Information Integrator, you must add the IISERVER to the eLink Platform configuration. This is done by identifying the IISERVER in the eLink Platform UBBCONFIG file. The UBBCONFIG file is a text file that defines an eLink application.

Table 7-2 describes the variables you need to set in the UBBCONFIG file for Information Integrator.

Table 7-2 UBBCONFIG File Variables used by Information Integrator

Variable Name	Description
APPDIR	Base directory for Information Integrator software. Make sure this value is the same as the APPDIR value in the environment file.
TUXCONFIG	Full pathname of binary <code>tuxconfig</code> file. Make sure this value is the same as the TUXCONFIG value in the environment file.
TUXDIR	Base directory of the eLink Platform software. Make sure this value is the same as the TUXDIR value in the environment file.
MACHINE_NAME	Name of the machine where Information Integrator is installed. On Windows NT systems, this value must be in uppercase.
*SERVERS	Defines the name of the Information Integrator server (IISERVER) and the server configuration file (<code>-- CLOPT -C ii.cfg</code>). For more information on the server configuration file, see “Creating the Configuration File.”
*SERVICES	Defines the name of the default service provided by the IISERVER (IISERVICE).

A sample UBBCONFIG file (called `SAMPLE.UBB`) is provided on the installation CD-ROM. You can use this sample file as a base for creating your own UBBCONFIG file.

Listing 7-10 shows the `SAMPLE.UBB` file. The variables you need to change are shown in boldface type.

Listing 7-10 Sample UBBCONFIG File

```
*RESOURCES

#Example:
#IPCKEY          123456

IPCKEY          123456

DOMAINID        IIAPP
MASTER          II
MAXACCESSERS    10
MAXSERVERS      10
MAXSERVICES     100
MODEL           SHM
LDBAL           N

*MACHINES
DEFAULT:
APPDIR="application directory"
TUXCONFIG="TUXEDO config file directory"
TUXDIR="TUXEDO directory"

<MACHINE NAME>  LMID=II

*GROUPS
GROUP1
                LMID=II GRPNO=1 OPENINFO=NONE

*SERVERS
DEFAULT:
                CLOPT="-A"

IISERVER        SRVGRP=GROUP1 SRVID=1 CLOPT= " -- -C ii.cfg"

*SERVICES
II_SERVICE
```

Creating the Configuration File

A configuration file, typically named `ii.cfg`, controls the operation of the Information Integrator server (IISERVER). Following are the sections of the configuration file and the parameters you can define for each section. A sample configuration file is provided in Listing 7-14.

Note: `ii.cfg` is a generic filename. You can name this file anything you choose, but the filename must match the name you specify in the UBBCONFIG file. (See “Adding the IISERVER to the UBBCONFIG File” for instructions on defining the IISERVER server in the UBBCONFIG file.) If you do not specify a path for the location of the configuration file, the system will search the current directory, then the path specified in the `APPDIR` variable in the UBBCONFIG file.

The Information Integrator configuration file is divided into the following sections:

SERVER

Contains the general parameters required during server startup. The configuration file can contain only one SERVER section.

SERVICE

Contains the name of a service to be advertised and the parameters required for the service. The configuration file can contain multiple SERVICE sections. One SERVICE section is required for each service advertised.

There are two ways to create the Information Integrator configuration file:

- Using the Rules Application
- Using a Text Editor

Using the Rules Application

The Rules application supports the creation and editing of Information Integrator configuration files. This is accomplished by the use of two windows: Configure Server (Figure 7-1) and Configure Service (Figure 7-2). To create an II Server configuration file you need to complete the following tasks:

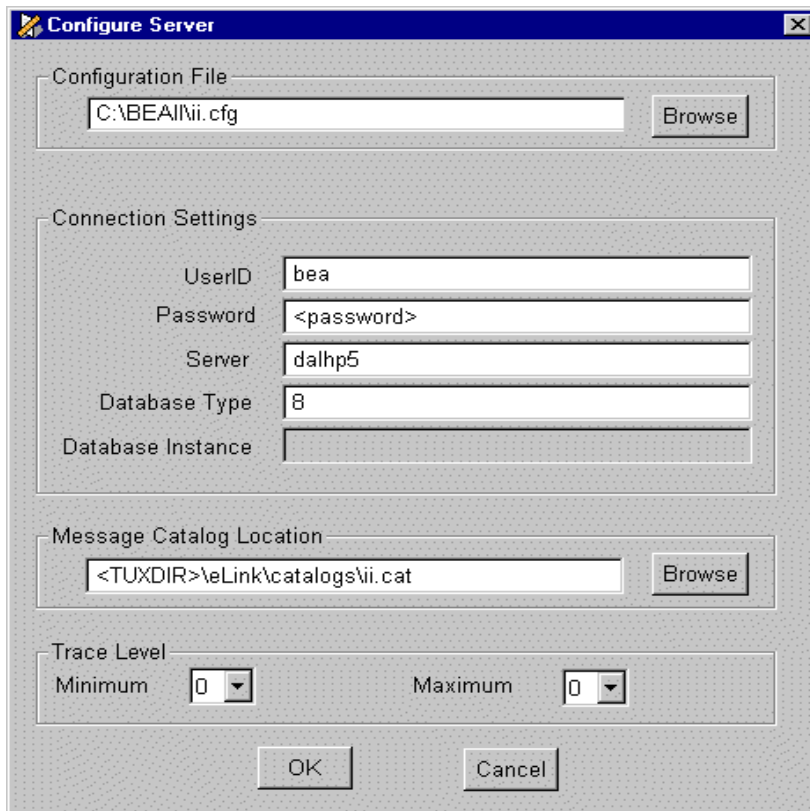
- Use the Configure Server window to create the configuration file and Server section for the configuration file.

- Use the Configure Service window to create one Service section for each service that is to be configured.

Using the Configure Server Window to Create a New Configuration File

1. From the Tools menu on the Rules main window, select Configure Server. The Configure Server window displays.

Figure 7-1 Configure Server Window



Note: The first time that the Configure Server window is used all text areas will be blank. After the initial use of the Configure Server window, it will display with the Server information from the configuration file that was most recently edited. This occurs even if the Rules application has been restarted.

2. Type the full path and file name of the new configuration file in the Configuration file text box, or click Browse to select the configuration file you want to edit.
3. Press Enter.

Note: When you enter a new configuration file name, the default values are filled in for the rest of the parameters. The default parameters for the Connection settings come from the connection information you entered when logging into Rules. However, for security the password is not displayed and must be edited manually.

4. Edit the remaining parameters as desired, using the Tab key to move between fields. The parameter definitions are as follows:

USERID -- User ID for logging into the database.

PASSWORD -- Password for logging into the database.

LOCATION -- Location of the database. On Oracle systems, this is the SQL*NET connection string. On SQL Server, this is the name of the database server.

DATABASE TYPE -- Numeric representation of the database type, as follows. May be "0" for default database type.

4 = MS SQL

8 = Oracle 7

9 = Oracle 8

MESSAGE CATALOG LOCATION -- Full path and file name of the TUXEDO catalog file. This file is installed as \$TUXDIR/eLink/catalogs/ii.cat.

DATABASE INSTANCE -- Indicates the instance of the database in MS SQL Server. Required if using an SQL Server database.

MAXIMUM -- Indicates the maximum trace level range to be used by the Information Integrator for logging trace messages. Possible values are 0-9.

MINIMUM -- Indicates the minimum trace level range to be used by the Information Integrator for logging trace messages. Possible values are 0-9.

Note: Note: If you change the Database Type field, press Enter after the change is made to indicate to Rules that it needs to enable or disable the Database Instance field. The Database Instance field is disabled if the Database Type is 8 or 9, for Oracle, and enabled if the Database Type is 4 for MS SQL.

5. When you complete the desired changes, click OK. The values you entered are validated and saved to the configuration file, and the Configure Server dialog closes.

Note: If at anytime you click Cancel before clicking OK, the Configure Server window closes, and none of the information entered is written to the configuration file.

The correspondence between the labels of the text fields in the Configure Server window and the names of the parameters in the configuration file is shown in Table 7-3.

Table 7-3 Configuration File Server Parameter Labels

Label in Configure Server window	Parameter Name in the Configuration File
UserID	DB_USERID
Password	DB_PASSWORD
Server	DB_LOCATION
Database Type	DB_TYPE
Database Instance	DB_INSTANCE
Message Catalog Location	MSG_CATALOG
Minimum	MINMSGLEVEL
Maximum	MAXMSGLEVEL

Editing an Existing Configuration File

To edit the Server section of an existing configuration file, follow the procedure for creating a new configuration file, using the Browse button in the configuration file group box to locate the desired configuration file. The parameter values are read from the file and displayed. After editing the desired text areas, click OK to save the data to the configuration file. Clicking Cancel at anytime before the clicking OK closes the Configure Server window without updating the configuration file.

Configuring Services

You use the Configure Service window (Figure 7-2) to create one Service section for each service that is to be configured. Prior to opening a Configure Service window, you must have used the Configure Server window to set the current configuration file (see “Using the Configure Server Window to Create a New Configuration File”).

To access the Configure Service window, follow these steps.

1. Right-click on a Message item in the left pane of the Rules window. A popup menu appears.
2. Choose Configure Service. The Configure Service window displays (see Figure 7-2).

Figure 7-2 Configure Service Window

The screenshot shows the 'Configure Service' dialog box. On the left, a list of services includes 'II_payroll' (selected) and 'IIRESET'. Below the list are 'New' and 'Delete' buttons. On the right, the configuration fields are as follows:

- Required**
 - Name: II_payroll
 - Appl Name: II_payroll_app
 - Input Format: II_payroll_msg1_IN
- FML Value Only**
 - FML32_VAL_FLD: II_payroll_msg1
 - FML32_DDR_FLD: II_payroll_target_ID
- Multiple Formats**
 - FML32_SELECT_FLD: II_payroll_STATE

Format Key	Format
CA	II_payroll_msg2_IN
NY	II_payroll_msg3_IN

At the bottom of the window are 'OK', 'Apply', and 'Cancel' buttons.

Notes: If you open a Configure Service window before setting up the configuration file in the Configure Server window, all editing, inserting, or deleting of services takes place in the last configuration file accessed, even if the Rules application has been restarted.

The name of the last configuration file edited is always saved between uses of the Rules application.

If the configuration file has never been set using the Configure Server window, you will receive an error when trying to open the Configure Service window.

For the IISERVER, configured services must have the Name, the Application, and the Input Format fields filled in. Table 7-4 shows the correspondence between these fields in the Configure Service window and the parameter names in the configuration file:

Table 7-4 Configuration File Service Parameter Labels

Label in Configure Service window	Parameter Name in the Configuration File
Name	NAME
Application	APPL_NAME
Input Format	INPUT_FORMAT

When the Configure Service window is displayed, the names of all of the Services that are currently configured appear in the Services data window. You can perform any of the following tasks from the Configure Service Window:

- Add a new service section to the configuration file.
- Edit the service section for an existing service.
- Delete a service section from the configuration file.

Adding a new service

1. Click the New button.
2. Edit the Required parameters and fill in any of the optional parameters necessary. The parameters are described below.

The following parameters are required:

NAME -- The service name being advertised. The service name must be 15 characters or less.

APPLICATION -- The application name associated with the service, as defined in the Rules application.

INPUT FORMAT -- The input format required by the application, as defined in the Rules application.

The following values are optional:

FML32_VAL_FLD -- Defines the field name that contains the message data. Used only if the input message is an FMLVO buffer. This means that the entire data for the message is contained in a single FML field. The value of this parameter gives the name of that FML field.

FML32_DDR_FLD -- Identifies the FML field name that contains data dependent routing information for the message data defined in FML32_VAL_FLD. Used only if the input message is an FMLVO buffer. If you want to take advantage of the data dependant routing feature of eLink, and you are using an FMLVO buffer for the message, this parameter should contain the name of the FML field that contains the data dependent routing information.

FML32_SELECT_FLD -- By setting this parameter, you can choose which message format will be used to process a particular message. If the field defined in the FML32_SELECT_FLD text area exists in a buffer, the contents of the field are compared to each of the entries in the Format Key column. If a match occurs, the message is parsed using the input format that appears in the corresponding row of the Format column. If no match occurs, the message is parsed using the input format in the Input Format text area. You can navigate through the Multiple Formats table using the Tab and arrow keys. To insert a new row in the table, select the item in the last row of the table and press the Enter key.

Note: The Format Key value must be in upper case characters.

3. Click Apply. The system validates the values you entered.

Editing an Existing Service

1. Select the service in the Service list.
2. Edit the appropriate parameters as described above.
3. Click Apply. The system validates the values you entered.

Deleting a Service

1. Select the service in the Service list.
2. Click Delete.

Note: Make sure you have selected the correct service prior to clicking Delete. If you selected an incorrect service, click Cancel to clear your selection PRIOR TO clicking Delete.

Saving the information to the configuration file

When you have finished adding, editing, and deleting services, click OK to save the information to the configuration file and close the dialog.

Notes: Choose names for your services that are related to their roles in the application.

When you access the Configure Service window through the popup menu associated with a Message item in the Rules window, the Application name and Input Format name are recorded. Any new service that is configured through this instance of the window will have, by default, the Application name and Input Format name (which is the same as the Message name) of this Message. Therefore, for each service that is to be configured, you should access the Configuration Service window through the popup menu associated with the appropriate Message item in the tree view.

You can also access the Configure Service window from the Tools menu on the Rules window. However, if accessed this way, no specific Application name or Input Format name appear by default.

The Rules application does not allow you to enter comments into the configuration file and will not preserve any comments were manually entered in the file.

Using a Text Editor

You can use any text editor to create the Information Integrator configuration file. A default configuration file (`ii.cfg`) is provided on the installation CD-ROM. You can use this file as a base and customize it to fit your particular needs.

The sections of the configuration file are and the parameters in each section are described below.

Defining the SERVER Section

The syntax for the SERVER section of the Information Integrator configuration file is as follows:

Listing 7-11 Syntax for SERVER section

```
*SERVER
  DB_USERID=userID
  DB_PASSWORD=password
  DB_LOCATION=location
  DB_INSTANCE=instance
  # MS SQL Server=4
  # Oracle 7.x=8
  # Oracle 8.x=9
  DB_TYPE=integer
  MSG_CATALOG=catalog file name
  MINMSGLEVEL=integer
  MAXMSGLEVEL=integer
```

Required Parameters

The following parameters must be defined in the SERVER section:

DB_USERID

User ID for logging into the database.

DB_PASSWORD

Password for logging into the database.

DB_LOCATION

Location of the database. On Oracle systems, this is the SQL*NET connection string. On SQL Server, this is the name of the database server.

DB_TYPE

Numeric representation of the database type. May be “0” for default database type.

MSG_CATALOG

Full path and file name of the TUXEDO catalog file. This file is installed as \$TUXDIR/eLink/catalogs/ii.cat.

Optional Parameters

The following parameters are optional in the SERVER section:

DB_INSTANCE

Indicates the instance of the database in MS SQL Server. Required if using an SQL Server database.

MAXMSGLEVEL

Indicates the maximum trace level range to be used by the Information Integrator for logging trace messages. Possible values are 0-9.

MINMSGLEVEL

Indicates the minimum trace level range to be used by the Information Integrator for logging trace messages. Possible values are 0-9.

Defining a SERVICE Section

The syntax for a SERVICE section of the Information Integrator configuration file is as follows:

Listing 7-12 Syntax for SERVICE section

```
*SERVICE
NAME=service name
APPL_NAME=application name
INPUT_FORMAT=input format
FML32_VAL_FLD=field name
FML32_DDR_FLD=field name
FML32_SELECT_FLD=field name
<select value>=<input format>
```

Required Parameters

The following parameters must be defined in the SERVER section:

NAME

The service name being advertised. The service name must be 15 characters or less.

APPL_NAME

The application name associated with the service, as defined in the Rules application (see “Defining Rules” for more information).

INPUT_FORMAT

The input format required by the application, as defined in the Rules application (see “Defining Rules” for more information).

7 Configuring Information Integrator and the IISERVER

Optional Parameters FML32_VAL_FLD
Defines the field name that contains the message data. Used only if the input message is an FMLVO buffer.

FML32_DDR_FLD
Identifies the FML field name that contains data dependent routing information for the message data defined in FML32_VAL_FLD. Used only if the input message is an FMLVO buffer.

FML32_SELECT_FLD
<SELECT VALUE>=<input format>
Identifies the FML field name that contains the information necessary for determining the message format to be used for processing. Associated with this parameter are one or more entries in the following format:

<SELECT VALUE>=<input format>

If the field defined in FML32_SELECT_FLD exists in the input buffer, the contents of this field are matched to each of the <SELECT VALUE> entries. If a match is found, the message is parsed using the associated <input format>. If no match is found, the message is parsed using the input format defined in the INPUT_FORMAT parameter.

Note: The <select value> must be in upper case characters.

Listing 7-13 shows an example of a SERVICE section of the `ii.cfg` file. An explanation of the processing defined by the sample follows the listing.

Listing 7-13 Sample SERVICE Section

```
*SERVICE
NAME=II_ADDRESS
APPL_NAME=II_ADDRESS_APPL
INPUT_FORMAT=II_ADDRESS_IN
FML32_VAL_FLD=ADDRESS1
FML32_DDR_FLD=ADDRESS1_TARGET_ID
FML32_SELECT_FLD=STATE
CA=II_ADDRESS2_IN
NY=II_ADDRESS3_IN
```

The parameters defined by this sample are as follows:

- The name of the service being advertised is II_ADDRESS.
- The associated application is II_ADDRESS_APPL.
- The default input format is II_ADDRESS_IN.
- The message to be processed is contained in the FML32 field ADDRESS1.
- The data-dependent routing information is contained in the FML32 field ADDRESS1_TARGET_ID.
- Before parsing the message, the buffer is examined to see if it contains a field named STATE.
- If not, the message is processed using the default format II_ADDRESS_IN.
- If so, the contents are compared and processed as follows:
 - If STATE=CA, use input format II_ADDRESS2_IN.
 - If STATE=NY, use input format II_ADDRESS3_IN.
 - Otherwise, use input format II_ADDRESS_IN

Sample Information Integrator Configuration File

This section contains a sample Information Integrator configuration file.

Listing 7-14 Sample configuration file

```
*SERVER
DB_USERID=bea
DB_PASSWORD=bea
DB_LOCATION=dall
DB_INSTANCE=dall
# MS SQL Server = 4
# Oracle 7.x = 8
# Oracle 8.x = 9
DB_TYPE=4
MSG_CATALOG=$TUXDIR/eLink/catalogs/ii.cat
MINMSGLEVEL=0
MAXMSGLEVEL=1

*SERVICE
NAME=II_ADDRESS
APPL_NAME=II_ADDRESS_APPL
INPUT_FORMAT=II_ADDRESS_IN
FML32_VAL_FLD=ADDRESS1
FML32_DDR_FLD=ADDRESS1_TARGET_ID
FML32_SELECT_FLD=STATE
CA=II_ADDRESS2_IN
NY=II_ADDRESS3_IN

*SERVICE
NAME=II_payroll
APPL_NAME=II_payroll_app
INPUT_FORMAT=II_payroll_msg1_IN
FML32_VAL_FLD=II_payroll_msg1
FML32_DDR_FLD=II_payroll_target_ID
FML32_SELECT_FLD=II_payroll_STATE
CA=II_payroll_msg2_IN
NY=II_payroll_msg3_IN

*SERVICE
NAME=IIRESET
APPL_NAME=DUMMY
INPUT_FORMAT=DUMMY
```

Creating the FML Field Table

You must create a field table if the messages you want to process with Information Integrator are contained in FML buffers. A sample field table (`Fieldtable.txt`) is included with Information Integrator. You can update this file to fit your particular environment by following these steps.

1. Open `Fieldtable.txt` in your preferred text editor.
2. Specify the appropriate table entries for your environment.

A sample `Fieldtable.txt` file is shown in Listing 7-15.

Listing 7-15 Sample `Fieldtable.txt`

```
*base 1000

STREET          1  string -   Street Address
CITY            2  string -   City Name
STATE          3  string -   State Name
ADDRESS        4  string -   Combined address
ELINK_ADAPTER_ERR 20 string -   eLink error message
ELINK_ADAPTER_ERR_CODE 21 string -   eLink error code
```

The sample `Fieldtable.txt` file shown above contains entries for the three input fields (`STREET`, `CITY`, and `STATE`), one entry for the reformatted output field (`ADDRESS`), and two entries for the eLink adaptor error message/code (`ELINK_ADAPTER_ERR` and `ELINK_ADAPTER_ERR_CODE`). These entries return error messages and codes if the application fails.

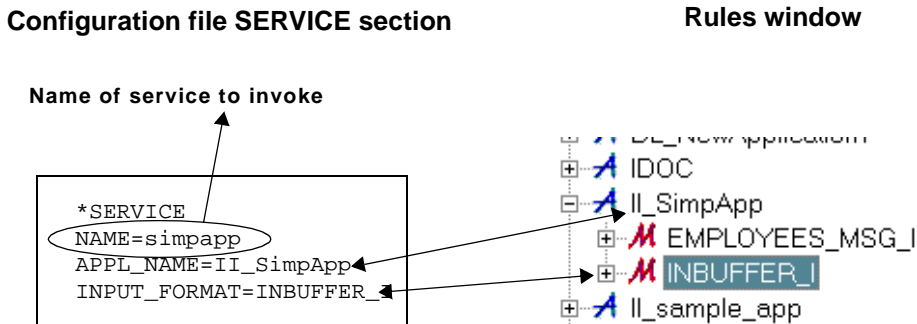
Understanding the IISERVER

The IISERVER translates messages from one format to another, simplifying message exchange between different systems and applications. The IISERVER breaks down complex messages into simple messages for processing. It can translate formats containing fixed, tagged, or delimited fields, as well as multi-part nested formats.

The IISERVER uses message format definitions within the database to recognize and parse input and output messages. You created these format definitions using either the Formatter GUI (see “Using Formatter to Build Definitions”) or the MsgDefAdmin tool (see “Using MsgDefAdmin”). The IISERVER uses this information to interpret values from incoming messages and dynamically construct outgoing messages. The IISERVER is run as an eLink server and is managed using eLink commands (see “BEA eLink Platform Reference” for more information).

The IISERVER determines its configuration from the Information Integrator configuration file. There is a direct correlation between the data entered in the Information Integrator Rules application (see “Defining Rules”) and the parameters you define in the SERVICE section of the configuration file. Figure 7-3 illustrates this relationship.

Figure 7-3 Relationship between Configuration File and Rules



In the above illustration, the APPL_NAME configuration parameter corresponds to the Application Group name in Rules, and the INPUT_FORMAT configuration parameter corresponds to the Message Type in Rules.

Understanding Message Processing

The IISERVER supports the following basic message-passing models:

- Request/Response Translation
- Data Enrichment
- Content-Based Routing
- Message Explosion

These models are briefly described in the following sections.

Request/Response Translation

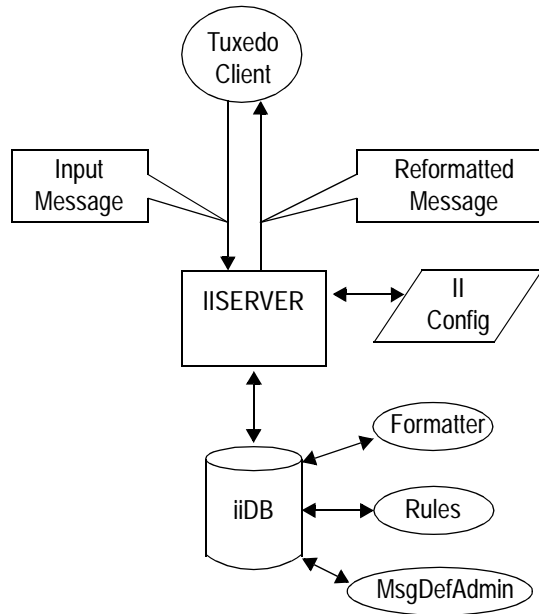
In this model, a client application sends a message to the IISERVER for some type of transformation. A series of actions associated with a business service advertised by the IISERVER is executed, and the resulting buffer is returned to the client.

The actions associated with this model include:

- Reformat (transforming the input message)
- Return (returning the transformed message)

The request/response translation process is shown in Figure 7-4.

Figure 7-4 Request/response Translation Process



The following actions take place in the process illustrated above:

1. The eLink client sends a message to the IISERVER.
2. The IISERVER obtains the necessary configuration information from the Information Integrator configuration file (see “Creating the Configuration File”).
3. The IISERVER obtains the necessary formatting and routing information from the database. This information is stored in the database when you create a Rule containing the Expression shown in Figure 7-5.
4. When translation is complete, the reformatted message is sent back to the eLink client.

Figure 7-5 Request/Response Translation Expression

Subscription List Expression Browse

Expression

TRUE

Verify

Clear

Build the expression above with the following components:

Expression Components Field List Operators Values Functions

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 can be filled in with other expressions, fields, operators, and so on.

Doubleclick

Apply Cancel Help

Notes: For more information on Rules and Expressions, refer to “Building Rules.”

This type of message routing is illustrated in the example described in “Using Information Integrator” and in the “Example” sections in each chapter.

Data Enrichment

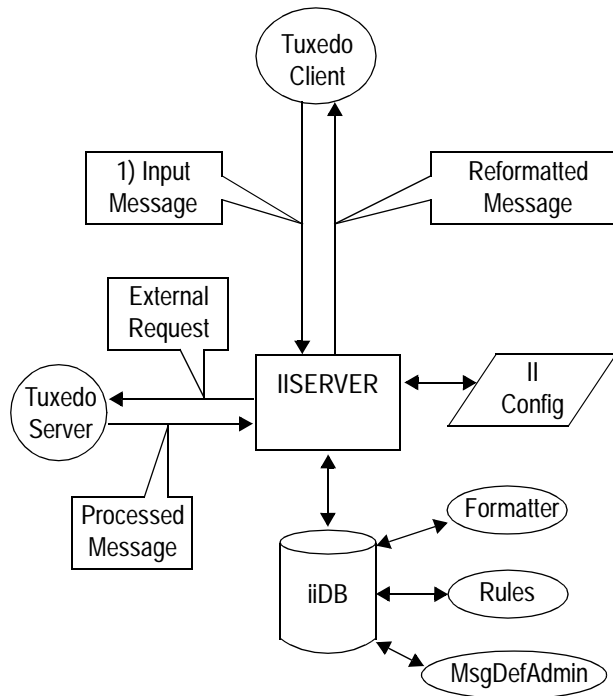
This processing model is similar to Request/Response Transformation, with the exception that the message can be processed by an external service (such as an eLink Third Party Adapter) to provide data enrichments that are not possible with the IISERVER. This external enrichment of the message is transparent to the requesting application.

The actions associated with this model include:

- Reformat (optionally transform the message to input for external services)
- Call (send message to external service and accept response)
- Return (return the transformed message to the requestor)

The data enrichment process is shown in Figure 7-6.

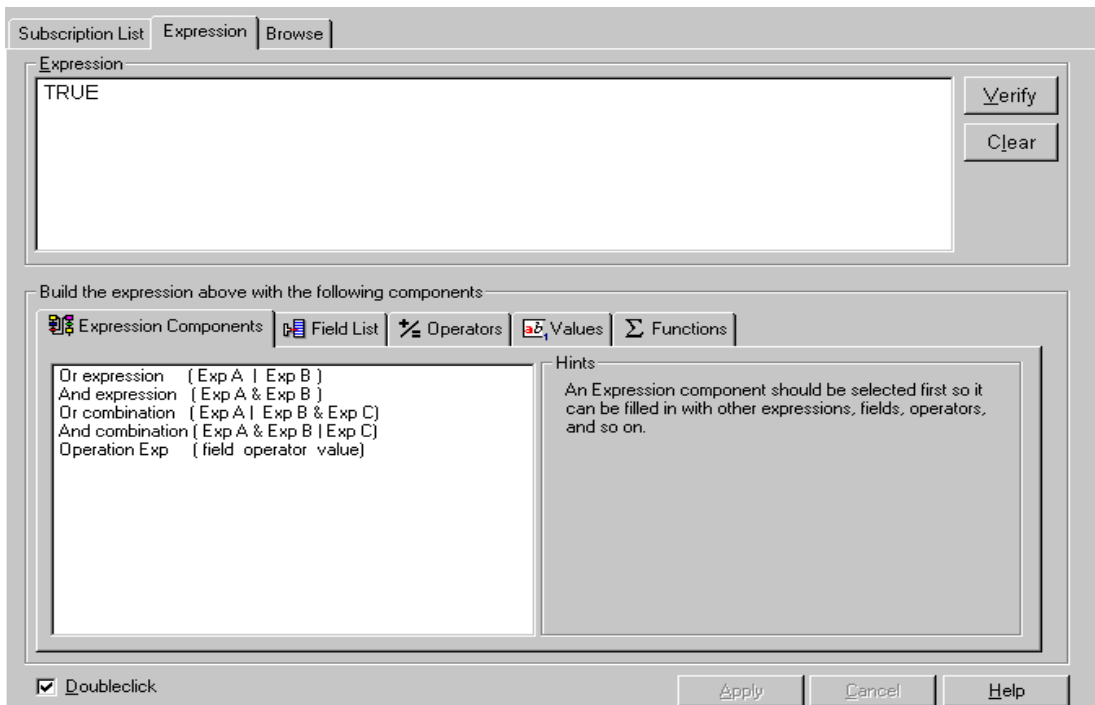
Figure 7-6 Data Enrichment Process



The following actions take place in the process illustrated above.

1. The eLink client sends a message to the IISERVER.
2. The IISERVER obtains the necessary configuration information from the Information Integrator configuration file (see “Creating the Configuration File”).
3. The message is sent to the external service for processing.
4. The external services sends the message back to the IISERVER.
5. The IISERVER obtains the necessary formatting and routing information from the database. This information is stored in the database when you create a Rule containing the Expression illustrated in Figure 7-7 and the Subscription illustrated in Figure 7-8.
6. When translation is complete, the reformatted message is sent back to the eLink client.

Figure 7-7 Data Enrichment Expression



Note: The Expression is TRUE because, in this example, we always want the Subscription to execute.

Figure 7-8 Data Enrichment Subscription

Action	Property	Value
Reformat	Input Format	II_address_msg1_I
	Input Message	1
	Output Format	II_address_msg1_O
	Output Message	2
	Output Type	STRING
	FMLVO Name	
	Return Message	
Call	Input Message	2
	Output Message	3
	Service Name	II_TOUPPER
	FMLVO Name	ADDRESS
	Error Q Space	
	Error Q Name	
	Return Message	3
Return	Return Message	3

Notes: The Input Format is the same as the Message Type. The original message buffer is 1. The buffer that is formatted in the Output Format II_address_msg1_O is buffer 2 (the output message number that is automatically generated). The output type is STRING.

The message that is the result of the original Reformat action will be the input buffer for the Call action to the eLink service II_TOUPPER (Call's Input Message = 2 = Output Message from Reformat). The return from the service call is stored in an FMLVO buffer named ADDRESS.

The buffer returned from the Call action to the service II_TOUPPER will be returned to the client (Return Message = 3 = Output Message from the Call).

For more information on Rules, Expressions, and Subscriptions, refer to “Building Rules.”

Content-Based Routing

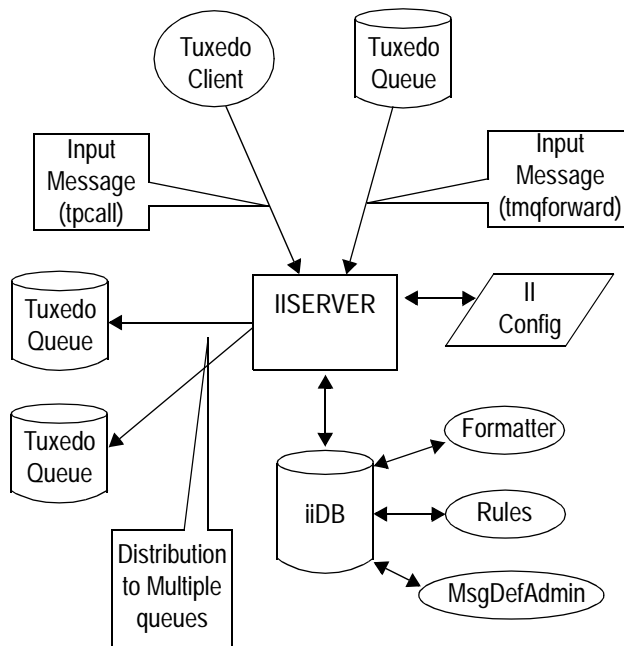
This processing model evaluates elements of the input message and determines how the message should be distributed.

Typical actions associated with this model include:

- Enqueue (place message on a defined queue for evaluation)
- Enqueue (place message on another defined queue, depending on results of evaluation)

The content-based routing process is shown in Figure 7-9.

Figure 7-9 Content-based Routing Process

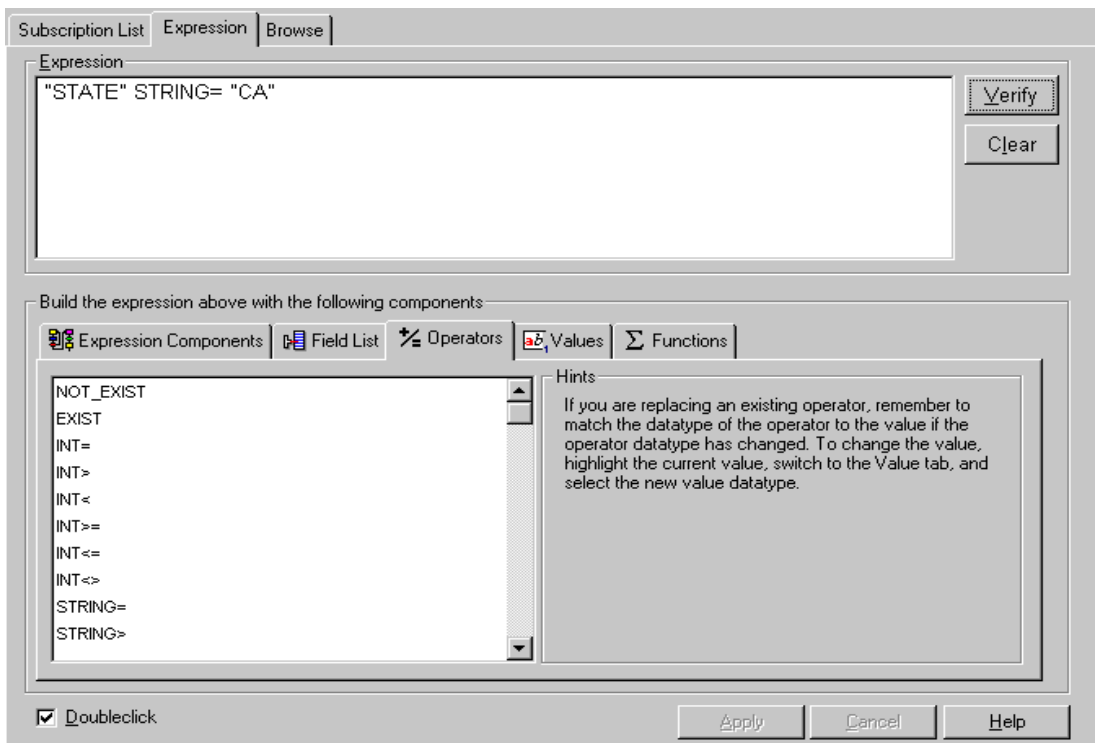


7 Configuring Information Integrator and the IISERVER

The following actions take place in the process illustrated above.

1. The eLink client sends a message to the IISERVER.
2. The IISERVER obtains the necessary configuration information from the Information Integrator configuration file (see “Creating the Configuration File”).
3. The IISERVER obtains the necessary formatting and routing information from the database. This information is stored in the database when you create a Rule containing the Expression illustrated in Figure 7-10.
4. The IISERVER evaluates the elements of the message to determine the proper distribution.
5. The IISERVER then distributes the message to the appropriate queues.

Figure 7-10 Content-Based Routing Expression



Note: The field “STATE” in the input buffer is compared to the string “CA.” If STATE = CA, the Subscription associated with the Rule will be executed.

For more information on Rules, Expressions, and Subscriptions, refer to “Building Rules.”

Message Explosion

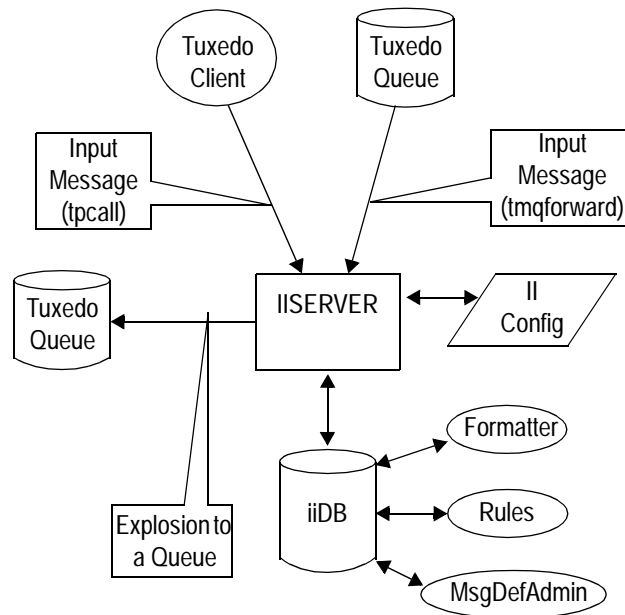
This model accepts a single input message, breaks it into component parts, and distributes selected components as individual messages.

Typical actions associated with this model include:

- Reformat (optionally transform the input message into component parts)
- Explode (parse message and execute the following action for each selected component)
- Enqueue (place the exploded message on a queue)

The message explosion process is shown in Figure 7-11.

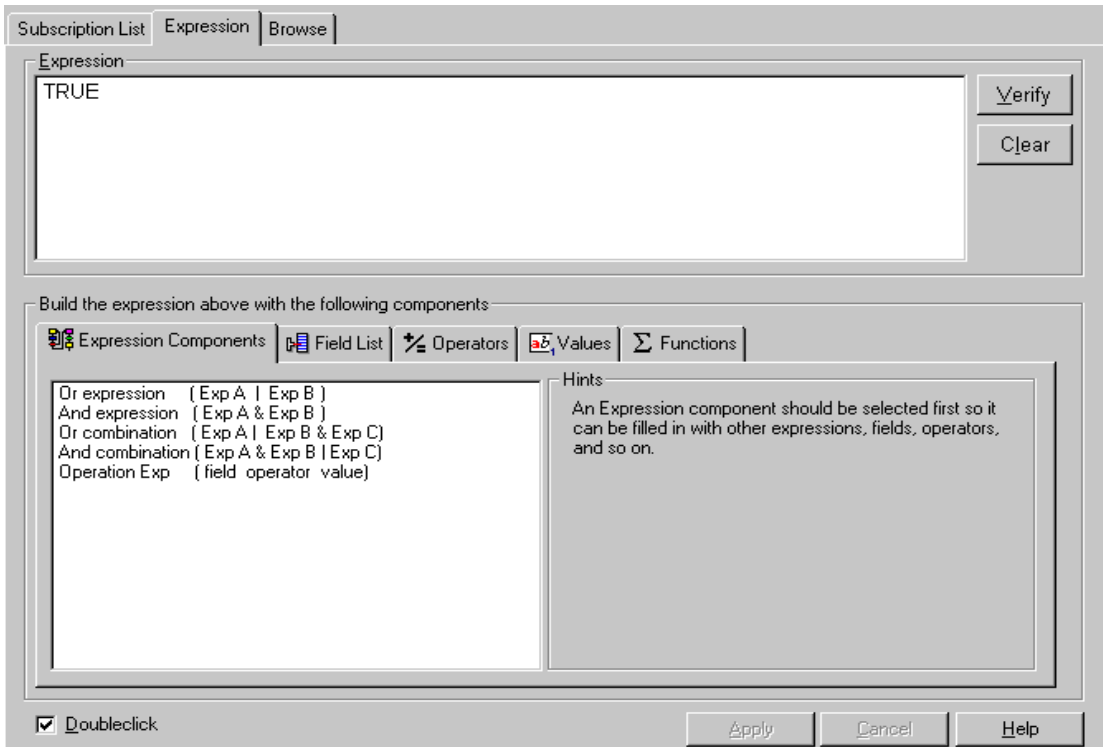
Figure 7-11 Message Explosion Process



The following actions take place in the process illustrated above.

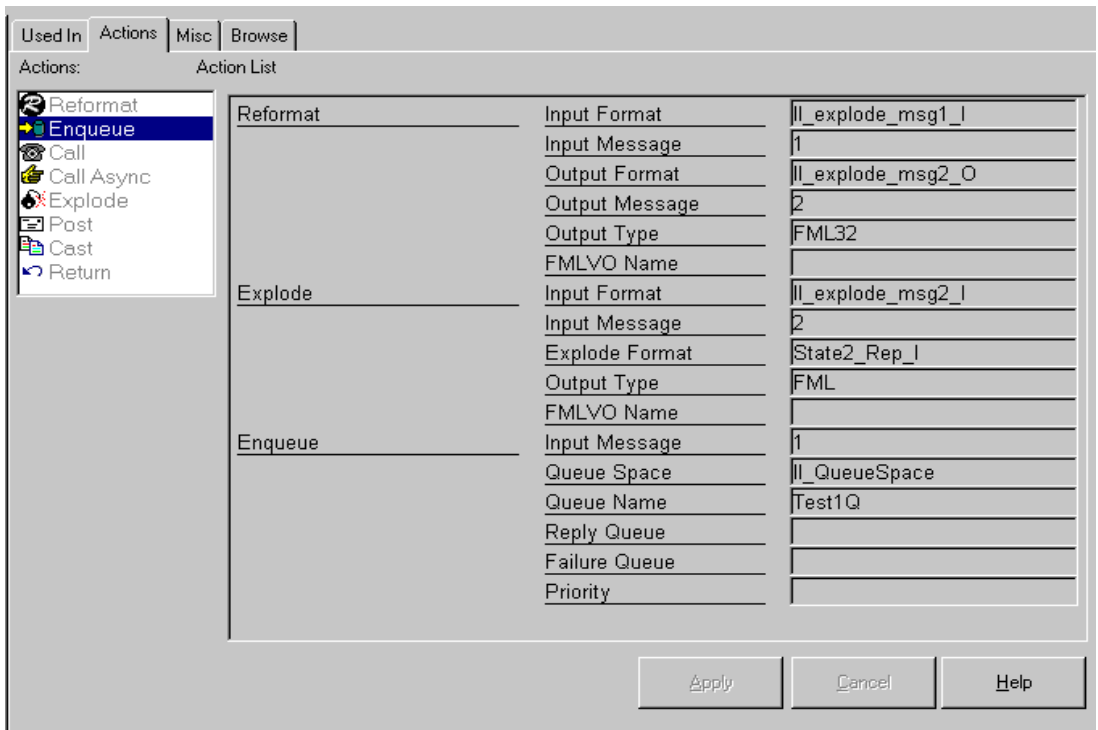
1. The eLink client sends a message to the IISERVER.
2. The IISERVER obtains the necessary configuration information from the Information Integrator configuration file (see “Creating the Configuration File”).
3. The IISERVER obtains the necessary formatting and routing information from the database. This information is stored in the database when you create a Rule containing the Expression illustrated in Figure 7-12 and the Subscription illustrated in Figure 7-13.
4. When translation is complete, the reformatted message is sent back to the IISERVER.
5. The IISERVER breaks the message into component parts.
6. The IISERVER then distributes the exploded messages to a queue.

Figure 7-12 Message Explosion Expression



Note: The Expression is TRUE because, in this example, we always want the Subscription to execute.

Figure 7-13 Message Explosion Subscription



Note: Refer to “Message Explosion Example” for a complete explanation of the Rules and Subscriptions used in this model.

Example – Step 2

This section outlines the procedure required to complete Step 2 in “Using Information Integrator.” In Step 2, you define your database.

1. Using your favorite text editor, open the file `\sample\payroll\sqlsvses.cfg`. This file is located in the `\sample\payroll` directory where you installed Information Integrator.
2. Scroll down to the following section in the file:

```
# Example SessionNames (uncomment the SessionNames needed):
#
#      MS SQL Server database:
# new_format_demo:Server:Userid:Password:Instance
# rules:Server:Userid:Password:Instance
# import:Server:Userid:Password:Instance
#      (change Server, Userid, Password, and Instance above)
#
#      Oracle database (final colon required on each line):
# new_format_demo:Service:Userid:Password:
# rules:Service:Userid:Password:
# import:Service:Userid:Password:
#      (change Service, Userid, Password above)
```

3. Find the section that corresponds to your database type (SQL Server or Oracle) and delete the “#” at the beginning of one of the lines.
4. Replace the word “Server or Service” with the name of your database server.
5. Replace the word “Userid” and “Password” with your user ID and password for the server.
6. If you are using SQL Server, replace the word “Instance” with your database instance.
7. Save the file.

Example – Step 7

This section outlines the procedure required to complete Step 7 in “Using Information Integrator.” In Step 7, you configure the IISERVER to advertise the services required to perform the data transformation. This involves editing several configuration files provided on the Information Integrator installation CD-ROM.

First, we will edit the `ii.cfg` file to set the correct user ID and password for the database, and to add the payroll sample as a service to be recognized by the IISERVER.

1. Copy the file `ii.cfg` from the directory where you installed Information Integrator to the `\sample\payroll` directory. This provides a copy you can edit while preserving the original configuration file.
2. Using your favorite text editor, open the file `\sample\payroll\ii.cfg`.
3. Edit the following lines with the appropriate information for your environment:

```
DB_USERID=userid
DB_PASSWORD=password
DB_LOCATION=location
DB_INSTANCE=instance
```

Note: The line `DB_INSTANCE` is only applicable for SQL Server installations. If you are running on Oracle, this line will be commented out.

4. Add a `*SERVICE` section for the payroll application as follows:

```
*SERVICE
NAME=II_payroll
APPL_NAME=II_Sample
INPUT_FORMAT=II_payroll_msg1_I
```

Note: The `APPL_NAME` is the name of the application group and the `INPUT_FORMAT` is the name of the input message. These two items were created in Rules (see the “Example” section in “Defining Rules” for more information).

5. Save the file.

Next, we will edit the `setenv.bat` file to set the correct locations for the eLink Platform and Information Integrator installations, and the payroll sample files.

1. Copy the file `setenv.bat` from the directory where you installed Information Integrator to the `\sample\payroll` directory. This provides a copy you can edit while preserving the original file.
2. Using your favorite text editor, open the file `\sample\payroll\setenv.bat`.
3. Edit the following lines with the appropriate information for your environment:

```
set TUXDIR=c:\tuxedo
set APPDIR=c:\beaii\sample\payroll
set TUXCONFIG=%APPDIR%\tuxconfig
```

where

TUXDIR

is the directory where eLink Platform is installed.

APPDIR

is the `\sample\payroll` subdirectory under the installation directory for Information Integrator.

TUXCONFIG

is the location and name of the binary version of the eLink Platform configuration file.

4. Save the file.

Next, we will edit the `sample.ubb` file to set the correct locations for the payroll sample files, the eLink Platform installation, and the binary version of the eLink Platform configuration file, as well as to set the correct machine name.

1. Copy the file `sample.ubb` from the directory where you installed Information Integrator to the `\sample\payroll` directory. This provides a copy you can edit while preserving the original file.
2. Using your favorite text editor, open the file `\sample\payroll\sample.ubb`.
3. Edit the following lines with the appropriate information for your environment:

```
*MACHINES
DEFAULT:
    APPDIR="c:\beaii\sample\payroll"
    TUXCONFIG="c:\beaii\sample\payroll\tuxconfig"
    TUXDIR="c:\tuxedo"
```

Machine Name LMID=II

where

APPDIR

is the `\sample\payroll` subdirectory under the installation directory for Information Integrator.

TUXCONFIG

is the name of the binary version of the eLink Platform configuration file, located in the `\sample\payroll` subdirectory under the installation directory for Information Integrator

TUXDIR

is the directory where eLink Platform is installed.

MACHINE NAME

is the name of the machine where Information Integrator is installed. If you are not sure of the machine name, check with your System Administrator.

Note: The values you enter in this file for the TUXDIR, APPDIR, and TUXCONFIG parameters must match the values you entered in the `setenv.bat` file.

4. Save the file.

Example – Step 8

This section outlines the procedure required to complete Step 8 in “Using Information Integrator.” In Step 8, the final step in the sample, you execute the files to perform the data transformation.

First, we will run the environment file to set the values we entered.

- Open a DOS window and type `setenv.bat` to set the values you entered (see “Example — Step 7” for more information).

Next, we create the binary version of the eLink Platform configuration file (see “Example — Step 7”).

- From the DOS window, type the following:

```
tmloadcf -y sample.ubb
```


Refer to “eLink Commands” for more information on the `tmloadcf` command.

Next, we will start the eLink Platform software.

- From the DOS window, type the following:

```
tmboot -y
```

Refer to “eLink Commands” for more information on the `tmloadcf` command.

Now, we will use the SendBuf utility to test our formats and configurations.

- From the DOS window, type the following:

```
sendbuf ii_payroll.data
```

Refer to “Using the sendBuf Utility” for more information on this tool.

A Additional Information Integrator Tools

This section describes additional command line tools that are available with BEA eLink Information Integrator. The following topics are covered:

- Defining the IIRESET Service
- Converting Data Integration Option Formats to MFL
- Using the sendBuf Utility
- Using the ud32 Utility
- Using the NNFie Utility
- Using APITEST
- Using MSGTEST
- Using the NNRIE Utility

Defining the IIRESET Service

BEA eLink Information Integrator includes a service called IIRESET. This service is used to remove all cached information and reload this information from the database.

The procedures for defining the IIRESET service are the same as defining any other service (see “Defining a SERVICE Section” for more information). The syntax for defining the IIRESET service in the Information Integrator configuration file is as follows:

Listing A-1 Syntax for the IIRESET service

```
*SERVICE
  NAME=IIRESET
  APPL_NAME=DUMMY
  INPUT_FORMAT=DUMMY
```

The following parameters must be defined in the SERVICE section for the IIRESET service:

NAME

IIRESET, to indicate that the IIRESET service is being advertised.

APPLICATION_NAME

The name of the application associated with this service.

INPUT_FORMAT

The input format required for this application.

Converting Data Integration Option Formats to MFL

Information Integrator provides a utility called `fgf2mfl` to convert Data Integration Option FML group formats and meta-type information formats to MFL. This allows you to use these formats with the `MsgDefAdmin` utility provided with Information Integrator (refer to “Using `MsgDefAdmin`” for more information).

The `fgf2mfl` utility has the following syntax:

```
fgf2mfl [-f | -m] <filename>
```

where

`-f`

Indicates that `<filename>` is a FML group format file.

`-m`

Indicates that `<filename>` is a meta-type information format file.

`<filename>`

Name of the Data Integration Option file that you want to convert to MFL.

If you invoke `fgf2mfl` without specifying a flag, `fgf2mfl` attempts to determine the file type from the filename extension. For example, an extension of `.fgf` specifies an FML group format, and an extension of `.mti` specifies a meta type format.

Using the sendBuf Utility

The `sendBuf` utility allows you to test Information Integrator by sending various typed buffers and displaying the returned value.

To use `sendBuf`, you create a script file defining the buffer and containing the commands you want to execute using any text editor. To define the buffer, you use the `$buffertype` parameter. The commands you can specify are as follows:

- `$invoke service` specifies a service to be executed
- `$enqueue` sends the buffer to the specified eLink Platform queue
- `$post event` posts the buffer to the eLink Platform event broker

Listing 7-16 shows a sample script file.

Listing 7-16 Sample sendBuf Script File

```
$buffertype=fml32
NAME=Pete
AGE=24
$invoke service=CalcBirthday
$enqueue qspace=QSPACE queue=TESTQ1
$post event=LowInventory
$buffer type=string value="Now is the time"
$invoke service=ProcString
$buffer type=carray value="@./myfile.data"
$invoke service=ProcCarray
```

The syntax for the script file is as follows:

```
$buffer type=buffer_type [value=buffer_value]
$invoke service=service_name
$enqueue qspace=qspace queue=queue_name
$post event=event_broker_name
```

where

buffer_type
is the buffer type (FML, FML32, STRING, CARRAY)

buffer_value

is the value to be placed in the buffer. This is only used for STRING and CARRAY buffer types.

service_name

is the name of the service you want to invoke

qspace

is the qspace where you want to place the buffer

queue_name

is the name of the eLink Platform queue where you want to place the buffer

event_broker_name

is the name of the eLink Platform event broker where you want to post the buffer

Note: If the buffer type is FML or FML32, you also need to define a name/value pair for the buffer. For example:

```
$buffer type=FML  
NAME="John Doe"  
RATE=22.66
```

To run the sendBuf utility, type the following at a command prompt:

```
sendbuf script_file_name
```

where

script_file_name

is the name of the script file you created.

Using the ud32 Utility

ud32 is a client program delivered with the BEA eLink Platform system that reads input consisting of text representation of FML buffers. You can use ud32 for ad hoc queries and updates to the IISERVER. It creates an FML32 buffer, makes a service call with the buffer, receives a reply (also in an FML32 buffer) from the service call, and displays the results on screen or in a file in text format. ud32 builds an FML32-type buffer with the FML fields and values that you represent in text format, makes a service call to the identified service in the buffer, and waits for the reply. The reply then comes back in FML32 format as a report.

For example, suppose you write a small file that contains the following text.

```
service name=.tmib and ta_operation=get, TACLASSES=T_SERVER
```

When you type this file into ud32, you receive an FML output buffer listing all the data in the system about the servers.

For more information on the ud32 utility, refer to the eLink Platform documentation.

Using the NNFie Utility

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

```
NNFie
```

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

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

```
NNFie.exe
```

The export file for NNFie is not interchangeable with the files created by the GUI. NNFie, NNRie, and sqlsvses.cfg must be in the same directory as NNFie.

Warning: Do not name components the same with only a change in case to identify them. For example, do not name one format “f1” and another format “F1”. You must make each item unique using something other than case differences.

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

Listing 7-17 shows the syntax for the NNFie command

Listing 7-17 Syntax for NNFie

```

NNFie
((-C <command file name>)
(-i <import file name> [-T] [ -o|-g|-n|-4]
[-s <session name>])

(-e <export file name> [-m <format name>+] [-q "comment"]
[-Q <Comment file name>] [-w <number>] [-s <session name>])
(-t <import file name> [-s <session name>])
(-I <import file name> [-s <session name>]))

```

In the above example:

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

You must keep the NNFie options in the correct position. For example, the following command is correct:

```
>nnfie -e myfile -m myformatname -s nnfie
```

The following command is incorrect because of the position of the options:

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

Table 7-5 lists the NNFie options and their definitions.

A Additional Information Integrator Tools

Table 7-5 NNFile Options

Name	Mandatory/Optional	Description
-C [<command file>]	Optional	Alternate command file name; default file is NNFile.cmd. If this option is provided, NNFile reads command line options from a file instead of the command line. Note: Command line option -C puts import/export command options in a text file. Do not use quotation marks around names (e.g., format name, session name, etc.) in the text file. Also, do not use back slashes in command lines.
-i [<import file>]	Mandatory for Import	This parameter is required to import data from the named file and is mutually exclusive from -e. The named file default is NNFile.exp. If you use the command line option -i, then the following options are available to you: [-T] [-o -g -n -4]. These additional options are described below the parameters table.
-e [<export file>]	Mandatory for Export	This parameter is required to export data from the named file and is mutually exclusive from -i. The named file default is NNFile.exp. If you use the command line option -e, then the following options are available to you: -q, -Q, -w, and -m. These additional options are described below the parameters table.
-s [<session name>]	Optional	Name of session in sqlsvses.cfg. Defaults to NNFile.
-I<import file name>	Mandatory	Writes description of all conflicts in import file to NNFile.log.
-t <import file name>	Mandatory	Writes an inventory of the import file to NNFile.log.

NNFile Import Syntax

Listing 7-18 shows the syntax for executing an import command.

Listing 7-18 Syntax for NNFile Import

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

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

Table 7-6 lists the NNFile import options and their definitions.

Table 7-6 NNFile Import Options

Import Options	Mandatory/Optional	Description
-T	Optional	Loads import file as one transaction. If an import failure for one component is detected, then the entire import is rolled back. The default behavior is a transaction boundary for each component.
-o	Optional	Overwrites all conflicts and replaces all components of same name with those in the export file.
-g	Optional	Ignores all conflicts and uses existing component definitions.
-n	Optional	Implements the interactive conflict resolution option. NNFile defaults to -n if no options are selected.
-4	Optional	Use R4_0 conflict resolution if a component in the export file conflicts with current data in the database. Do not import the new component but flag it in the error file and do not import any components that rely on the conflicting component.

NNFie Export Syntax

Listing 7-19 shows the syntax for executing the command to export an entire database.

Listing 7-19 Syntax for NNFie Export (entire database)

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

Listing 7-20 shows the syntax for executing the command to export a single format.

Listing 7-20 Syntax for NNFie Export (single format)

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

Listing 7-21 shows the syntax for executing the command to export multiple formats.

Listing 7-21 Syntax for NNFie Export (multiple formats)

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

Table 7-7 lists the NNFie export options and their definitions.

Table 7-7 NNFie Export Options

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

Using APITEST

The `apitest` executable outputs the structure and contents of a message parsed by the Information Integrator server.

The syntax for `apitest` is as follows:

Listing 7-22 Syntax for APITEST

```
apitest[-d[<filename>]]
-d :parse debug on
```

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

To run apitest:

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

Using MSGTEST

The `msgtest` executable uses input and output formats, delimiters, and other control information read from the database to parse and reformat an input message read from a file. The information needed by `msgtest` must be placed in the database using the Formatter graphical user interface.

Listing 7-23 Syntax for MSGTEST

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

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

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

To run msgtest:

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

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

Listing 7-24 shows an example command line using a msgtest text file.

Listing 7-24 Syntax for calling a MSGTEST text file

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

The file `myFormatterTest.txt` contains:

`ascii_string`

The input file name containing the message.

`output_AS1`

The output file name that will contain the translated message.

`AS_IF`

The input format to be read from the database.

`AS_NA1_OF`

The output format to be read from the database.

Configuration File

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

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

Listing 7-25 Syntax for `new_format_demo`

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

Using the NNRie Utility

NNRie is a command line tool that you can use to export rule definitions and orphan subscriptions (subscriptions that are not associated with a rule) from a database to a file and to import the exported file into a database. To use NNRie, Unix users must have write permissions to the current directory. Listing A-2 shows the syntax for NNRie.

Listing A-2 Syntax for NNRie

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

Table A-1 lists NNRie options and their definitions.

Table A-1 NNRie Options

Name	Mandatory/Optional	Description
-C [<command file>]	Optional	Alternate command file. The default is NNRie.cmd. If this option is provided, NNRie reads command line options from a file instead of a command line. If -C is present, NNRie expects the other parameters to be in the command file named in the same format as the command line.
-V (version)	Optional	Shows program version information only and does no processing.

A Additional Information Integrator Tools

Table A-1 NNRie Options

Name	Mandatory/Optional	Description
-i [<import file>]	Mandatory for Import	Indicates the program should import data from the named file. This parameter is required to import data and is mutually exclusive with -e. This parameter may be followed by the name of a file that contains the import data. The referenced file must have been created with the NNRie -e option. The default file name is NNRie.exp.
-e [<export file>]	Mandatory for Export	Indicates the program should export to the named file. This parameter is required to export data, and is mutually exclusive with -i. This parameter may be followed by the name of a file to hold the export data. The default file name is NNRie.exp.
-s <session name>	Optional	The session name corresponding to the session identifier in the configuration file (See the -c option below). The default session tag is "nnrmie".
-o (overwrite flag)	Optional	The default behavior is off (do not overwrite). If this parameter is present during export, it overwrites the export file. If this parameter is present during import, and a rule or subscription defined in the import file already exists in the importing database, the old rule is overwritten with the new definition if you have update permission. If you do not have update permission, an error is noted and the rule is replaced. If not overwriting rules, any rule that cannot be processed because it already exists in the importing database is noted.
-c <config file>	Optional	Indicates the name of the configuration file the program should read to load its session data for access to a database. The default configuration file is <code>sqlsvses.cfg</code> .
-a <application group>	Optional	Identifies the application group to export. If a value for this parameter is not identified, all application groups are exported. This parameter can be repeated as many times as necessary to define multiple application groups to export.
-m <message type>	Optional	Specifies the message type to export. This parameter also requires the -a parameter to be set. The default behavior is to export all message types within the specified application group. This parameter can be repeated as many times as necessary to define multiple message types within the same application group.

Table A-1 NNRie Options

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

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

NNRie Import Syntax

Listing 7-26 shows the syntax for importing a rule.

Listing 7-26 Syntax for importing a rule

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

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

NNRie Export Syntax

Listing 7-27 shows the syntax for exporting an entire database.

Listing 7-27 Syntax for exporting an entire database

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

Listing 7-28 shows the syntax for exporting a single application group.

Listing 7-28 Syntax for exporting a single application group

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

The application group name exports and then each message type within the application group exports. The message type export includes all subscriptions and rules in the specific application group/message type. This procedure is followed for each application group if multiple application group names are given.

Listing 7-29 shows the syntax for exporting a message type for an application group.

Listing 7-29 Syntax for exporting a message type for an application group

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

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

Listing 7-30 shows the syntax for exporting a single application group.

Listing 7-30 Syntax for exporting a single rule

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

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

Listing 7-31 shows the syntax for exporting a single application group.

Listing 7-31 Syntax for exporting more than one rule

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

Listing 7-32 shows the syntax for exporting a single subscription.

Listing 7-32 Syntax for exporting a single subscription

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

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

Command Line Functions

The NNRie command line functions are described below.

To overwrite component by component, enter the following syntax:

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

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

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

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

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

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

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

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

```
NNRie -i NNRie.exp -O
```

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

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

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

```
NNRIe -t NNRIe.exp (Writes to NNRIe.log)
>NNRIe -i NNRIe.exp -t inv.log (Writes to inv.log)
```

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

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

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

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

To implement the batch Overwrite conflict resolution, enter:

```
NNRIe -i <filename> -o
```

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

```
NNRIe -i <filename> -g
```

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

```
NNRIe -i <filename> -n
```

To implement the conflict report option, the syntax is:

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

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

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

```
NNRIe -i NNRIe.exp -O
```

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

B Message Explosion Example

This section illustrates a message explosion process. In this example, you need to take an input FML buffer containing a single city name and a variable number of state names and “explode” the buffer into the multiple buffers. Each of these buffers is an FML buffer containing one city name and one state name. These buffers will then be enqueued onto a queue named Test1Q.

The following topics are discussed:

- Input Buffer Contents
- MFL File Definitions
- Rule Definitions

Input Buffer Contents

The contents of the input buffer is shown in Table B-1.

Table B-1 Input Buffer Contents

FIELD	VALUE
CITY	San Jose
STATE	CA
STATE	TX
STATE	CO

The contents of the three buffers that would result from “exploding” the input message is shown in Table B-2.

Table B-2 Exploded Buffer Contents

FIELD	VALUE
Buffer1	
CITY	San Jose
STATE	CA
Buffer2	
CITY	San Jose
STATE	TX
Buffer3	
CITY	San Jose
STATE	CO

MFL File Definitions

This section describes the MFL files that need to be created for this message explosion example. Listing B-1 shows the MFL file required to create the appropriate input format to parse the input buffer.

Listing B-1 explode_msg1.xml

```
<?xml version='1.0'?>
<!DOCTYPE MessageFormat SYSTEM 'mfl.dtd'>
<MessageFormat name="II_explode_msg1">
  <FieldFormat name="CITY" type="String"/>
  <FieldFormat name="$RSTATE" type="BigEndian2" />
  <StructFormat name="State_Rep" repeatField="$RSTATE">
    <FieldFormat name="STATE" type="String" accessMode="Controlling" />
  </StructFormat>
</MessageFormat>
```

In the input buffer, the STATE field repeats a variable number of times. To account for this, a flat input format containing the single field STATE needs to be defined. The **StructFormat** statement in the MFL file creates this flat input format named **"State_Rep"**. The field \$RSTATE is the **repeatField** for this format; that is, it indicates the number of times that format appears in the message. The "\$R" prefix in the name of the field tells Information Integrator to populate the **repeatField** by using the FML function `Foccur()`. The `Foccur()` function determines the number of occurrences of the given field. Taking advantage of the functionality supplied by FML allows you the freedom to include an arbitrary number of occurrences of a given field in the input buffer, without having to explicitly assign a value to a field that indicates the number of occurrences of that field.

Because the input format **II_explode_msg1** defined in the above MFL file contains a flat input format, it must be a compound format. Only compound formats can contain other formats. Since compound formats cannot contain fields that are not included in other defined formats, `MsgDefAdmin` creates a flat input format with the fields CITY and \$RSTATE. The name of this flat input format begins with "\$STRUCT_", and it is visible only under the compound format **II_explode_msg1** in the Formatter window (see "Using the Formatter Window").

Now you need to define an output format into which the input message will be reformatted. This output format must allow for the fact that there will be a variable number of CITY/STATE pairs in the reformatted message. MFL file that could be used to create the appropriate output format.

Listing B-2 explode_msg2.xml

```
<?xml version='1.0'?>
<!DOCTYPE MessageFormat SYSTEM 'mfl.dtd'>
<MessageFormat name="II_explode_msg2">
<FieldFormat name="$RSTATE" type="BigEndian2" />
<StructFormat name="State2_Rep" repeatField="$RSTATE">
<FieldFormat name="CITY" type="String"/>
<FieldFormat name="STATE" type="String" accessMode="Controlling"/>
</StructFormat>
</MessageFormat>
```

As with the input format definition, there is a flat output format named "State2_Rep" which contains the fields CITY and STATE. This is created using the **StructFormat** statement in the MFL file. \$RSTATE is the **repeatField** for this structure. There are two reasons for this:

- The State2_Rep output format occurs exactly as many times as the STATE field occurs.
- Information Integrator maps fields with the same name to each other by default. The value in the \$RSTATE field in the input buffer should be exactly the same in the output buffer.

The format **II_explode_msg2** contains other formats, so it must be a compound format. Since compound formats cannot contain fields that are not included in other defined formats, MsgDefAdmin creates a flat output format with a name beginning with "\$STRUCT\$_" to contain the field \$RSTATE.

Note: You may want to use the Formatter application to view the format definitions created by **II_explode_msg1** and **II_explode_msg2**. For more information, refer to “Building Format Definitions.”

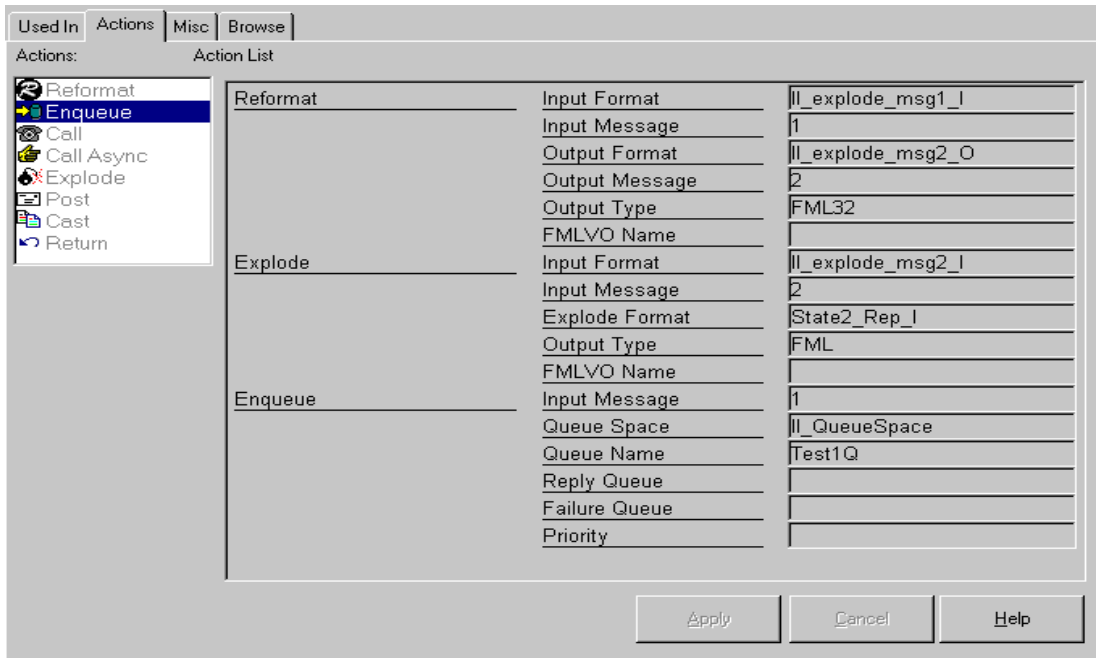
For more information on MFL files and the MsgDefAdmin utility, refer to “Using MsgDefAdmin.”

Rule Definitions

Now you need to define the necessary rules, following the steps below. For detailed instructions on performing these tasks, refer to “Building Rules.”

1. Create an Application Group named **ME**.
2. Add a new Message Type named **II_explode_msg1_I** to the Application Group.
Note: Remember that the Message Type is always the same as the Input Format used to parse the input message.
3. Add a Rule called **NewRule1** to the Message Type.
4. Define the Expression for the new Rule to be TRUE, because you always want **NewRule1** to “fire” when a request is made to a service with Application Group **ME** and Input Message **II_explode_msg1_I**.
5. Create a subscription that causes the IISERVER to perform the necessary Actions. This subscription is illustrated in Figure B-1.

Figure B-1 Figure 3: NewSubscription1



The above subscription contains three actions:

- **Reformat** — The Reformat takes the original input buffer passed in by the client and parses it using the **II_explode_msg1_I** compound input format. It reformats the message using the compound output format **II_explode_msg2_O** and produces an FML32 buffer. If the input message contained a CITY field and three STATE fields, after the reformat, the resulting buffer would contain three CITY/STATE pairs of fields.
- **Explode** — The result of the Reformat serves as input to the Explode action. This is because the Input Message for the Explode is defined as “2”, which is the Output Message for the Reformat. The Explode action separates the single message containing multiple CITY/STATE pairs into multiple messages containing one CITY/STATE pair each. This is accomplished by using **state2_Rep_I** as the Explode Format.
- **Enqueue** — Since the Enqueue action immediately follows the Explode action, each one of the separate buffers produced by the Explode action is enqueued on the queue **Test1Q** in the queue space **II_QueueSpace**.

C BEA eLink Platform Reference

This section provides you with basic information about BEA eLink Platform and FML buffers. The following topics are covered:

- BEA eLink Platform Architecture
- ATMI Runtime ServicesFML32
- eLink Commands

BEA eLink Platform Architecture

The eLink Platform communications application programming interface, Application to Transaction Monitor Interface (**ATMI**), is a collection of runtime services that can be called directly by a C (or COBOL) application. These runtime services provide support for communications, distributed transactions, and system management.

The Management Information Base (**MIB**) maintains a virtual repository of all the configuration and operational information for a runtime eLink environment. The eLink services are implemented using a shared bulletin board (BB) that contains configuration information. This is the dynamic part of the eLink. **Servers** advertise their **services** in the Bulletin Board. The Bulletin Board Liaison (BBL) is an administrative eLink server that is the keeper of the Bulletin Board. There is a BBL on every machine participating in the integration infrastructure; the BBL coordinates

changes to the local copy of the MIB. The Distinguished Bulletin Board Liaison (DBBL) is responsible for propagating global changes to the MIB and is the keeper of the static part of the MIB. The MASTER node is the computer where the DBBL runs.

Administrators use an ASCII file to specify eLink system configuration. This file, called the `UBBCONFIG` file, is used as input by the configuration loading utility, `tmloadcf`. The `tmloadcf` utility generates a binary version of the configuration called the `tuxconfig` file. This binary file is used by the system to construct the Bulletin Board and contains the persistent part of the MIB.

Servers are processes that provide one or more services. They continually check their message queue for service requests. A service is the name of a server interface. Many servers can support a single service, thereby providing for load balancing and a fail-safe mechanism. The mapping of services to servers is recorded in the Bulletin Board. When a service request is made, the Bulletin Board forwards the request to a server (eLink Adapter) that advertises that service. An eLink server advertises a service by posting its name in the Bulletin Board.

ATMI Runtime Services

The eLink Platform ATMI is a collection of runtime services that can be called directly by a C (or COBOL) application. The ATMI is a compact set of primitives used to open and close resources, begin and end transactions, allocate and free buffers, and provide the communication between adapters and other requestors or responders.

Following is a list of ATMI primitives for the C binding. See the *BEA Tuxedo Reference Guide* at <http://edocs.bea.com/tuxedo/tux65/index.htm> for detailed information on all the ATMI primitives.

Table 7-8 ATMI Primitives for the C Binding

API Group	C API Name	Description
Client Membership	tpchkauth	Check if authentication is needed
	tpinit	Used by a client to join an application
	tpterm	Used by a client to leave an application
Buffer Management	tpalloc	Create a message
	tprealloc	Resize a message
	tpfree	Free a message
	tptypes	Get a message type and subtype
Message Priority	tpgprio	Get the priority of the last request
	tpsprio	Set priority of the next request
Request/Response	tpcall	Synchronous request/response to service
	tpacall	Asynchronous request
	tpgetreply	Receive asynchronous response
	tpcancel	Cancel asynchronous request
Conversational	tpconnect	Begin a conversation with a service
	tpdiscon	Abnormally terminate a conversation
	tpsend	Send a message in a conversation
	tprecv	Receive a message in a conversation
Reliable Queueing	tpenqueue	Enqueue a message to an application queue
	tpdequeue	Dequeue a message to an application queue
Event-based	tpnotify	Send unsolicited message to a client
	tpbroadcast	Send message to several clients
	tpsetunsol	Set unsolicited message callback
	tpchkunsol	Check arrival of unsolicited message
	tppost	Post an event message
	tpsubscribe	Subscribe to event messages
	tpunsubscribe	Unsubscribe to event messages

Table 7-8 ATMI Primitives for the C Binding

API Group	C API Name	Description
Transaction Management	tpbegin	Begin a transaction
	tpcommit	Commit the current transaction
	tpabort	Rollback the current transaction
	tpgetlev	Check if in transaction mode
	tpsuspend	Suspend the current transaction
	tpresume	Resume a transaction
	tpscmt	Control commit return
Service Entry and Return	tpsvrinit	Server initialization
	tpsvrdone	Server termination
	tpreturn	End service function
	tpforward	Forward request
Dynamic Advertisement	tpadvertise	Advertise a service name
	tpunadvertise	Unadvertise a service name
Resource Management	tpopen	Open a resource manager
	tpclose	Close a resource manager

FML32

FML is a set of C language functions for defining and manipulating storage structures called fielded buffers, which contain attribute-value pairs called fields. The attribute is the field's identifier and the associated value represents the field's data content.

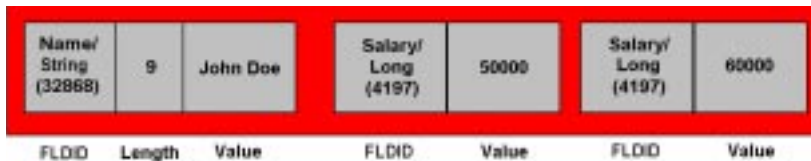
FML32 uses 32-bit values for the field lengths and identifiers. BEA eLink Adapters use FML32. FML32 allows for about 30 million fields, and field and buffer lengths of up to about 2 billion bytes. The definitions, types, and function prototypes for FML32 are located in `fm132.h` and functions are located in `-1fm132`. All definitions, types, and function names for FML32 have a "32" suffix (for example, `MAXFLEN32`, `FLDID32`, `Fchg32`). Also the environment variables are suffixed with "32" (for example, `FLDTBLDIR32`, `FIELDTBLS32`).

Note: FML has two interfaces. The original FML interface is based on 16-bit values for the length of fields and for containing information identifying fields. The original interface should not be used when creating eLink Adapters.

FML Buffers

A fielded buffer is composed of field identifier and field value pairs for fixed length fields (for example, long, short), and field identifier, field length, and field value triples for varying length fields.

Figure 7-14 Example of a Fielded Buffer



A field identifier is a tag for an individual data item in a fielded buffer. The field identifier consists of the name of the field number and the type of data in the field. The field number must be in the range 1 to 33,554,431 inclusive for FML32, and the type definition for a field identifier is `FLDID32`.

Field numbers 1 to 100 are reserved for system use and should be avoided. The field types can be any of the standard C language types: `short`, `long`, `float`, `double`, and `char`. Two other types are also supported: `string` (a series of characters ending with a null character) and `carray` (character arrays). These types are defined in `fm132.h` as `FLD_SHORT`, `FLD_LONG`, `FLD_CHAR`, `FLD_FLOAT`, `FLD_DOUBLE`, `FLD_STRING`, and `FLD_CARRAY`.

For FML32, a fielded buffer pointer is of type `FBR32 *`, a field length has the type `FLDLEN32`, and the number of occurrences of a field has the type `FLDOCC32`.

Fields are referred to by their field identifier in the FML32 interface. However, it is normally easier to remember a field name. There are two approaches to mapping field names to field identifiers. One is a compile-time mapping, the other is a run-time mapping.

Mapping Field Names to Field Identifiers

To avoid naming conflicts, BEA eLink Adapters must use the following run-time mapping method. Field name/identifier mappings can be made available to **FML32** programs at run-time through field table files. Field data types must be specifiable within field table files.

The FML32 interface uses the environment variables, `FLDTBLDIR32` to specify a list of directories where field tables can be found and `FIELDTBLS32` to specify a list of the files that are to be used from the table directories.

Note: The environment variables, `FLDTBLDIR32` and `FIELDTBLS32`, must be set prior to using FML32.

Within application programs, the FML32 function, `FLdid32`, provides for a run-time translation of a field name to its field identifier, and `Fname32` translates a field identifier to its field name. Type conversion should be performed implicitly via FML library functions. Implicit type conversion facilitates component reuse.

Use FML32 symbolic names and retrieve their values using `FLDID32`. The `Mkfldhdr32` function must not be used to build an eLink server because it may cause conflicts with other field IDs.

Any field in a fielded buffer can occur more than once. Many FML32 functions take an argument that specifies which occurrence of a field is to be retrieved or modified. If a field occurs more than once, the first occurrence is numbered 0 and additional occurrences are numbered sequentially. The set of all occurrences make up a logical sequence, but no overhead is associated with the occurrence number (that is, it is not stored in the fielded buffer). If another occurrence of a field is added, it is added at the end of the set and is referred to as the next higher occurrence. When an occurrence other than the highest is deleted, all higher occurrences of the field are shifted down by one (for example, occurrence 6 becomes occurrence 5, 5 becomes 4, etc.).

FML32 Primitives

Following is a summary of some of the FML32 primitives that are used for all eLink programs including general eLink services and adapters. This subset of FML32 primitives should be sufficient to create most eLink clients and servers. For more complete details and code examples, see the *BEA Tuxedo Reference Guide* at <http://edocs.beasys.com/tuxedo/tux65/index.htm>

Table 7-9 FML32 Primitives

FML Primitive	Description
Fadd32	Add new field occurrence
Fchg32	Change field occurrence value
Ffind32	Find field occurrence in buffer
Fget32	Get copy and length of field occurrence
Fielded32	Return true if buffer is fielded
Finit32	Initialize fielded buffer
Fldid32	Map field name to field identifier
Fneeded32	Compute size needed for buffer
Fsizeof32	Returns the size of an FML32 buffer

Warning: The `Falloc` function allocates FML buffers; however, buffers allocated using `Falloc` cannot be passed in a `tpcall`. **FML32** buffers that will be passed using the `tpcall` or `tpacall` **ATMI** primitives should be allocated by using a `tpalloc` with `type` parameter set to `FML32`.

Use FML32 symbolic names and retrieve their values using `Fldid32`. Field IDs must be determined dynamically at runtime or during initialization at boot time. The `Mkfldhdr32` function must not be used to build the adapter because it may cause conflicts with other field IDs.

eLink Commands

Commands are used to configure and administer the eLink runtime environment. Refer to *Administering the BEA Tuxedo System* for procedures and administrative tasks that are based on the command-line interface. For details about individual commands, refer to the *BEA Tuxedo Reference Manual*. Both documents may be found online at <http://edocs.beasys.com/tuxedo/tux65/index.htm>

Commonly Used Tuxedo Commands

Following is a list of the **Tuxedo** commands that are most commonly used.

Table 7-10 Commonly Used Tuxedo Commands

Tuxedo Commands	Description
<code>buildclient</code>	Constructs a BEA Tuxedo client module. This command combines the files supplied by the <code>-f</code> and <code>-l</code> options with the standard BEA Tuxedo libraries to form a load module and invokes the platform's default compiler to perform the build.
<code>buildserver</code>	Constructs a BEA Tuxedo server load module. This command generates a stub file containing a <code>main()</code> function and invokes the platform's default compiler to perform the build.
<code>tmadmin</code>	Invokes the BEA Tuxedo bulletin board command interpreter.
<code>tmboot</code>	Invokes a BEA Tuxedo application with a configuration defined by the options specified.
<code>tmloadcf</code>	Parses a <code>UBBCONFIG</code> file and load binary <code>TUXCONFIG</code> configuration file.
<code>tmshutdown</code>	Shuts down a set of BEA Tuxedo servers.

Table 7-10 Commonly Used Tuxedo Commands

Tuxedo Commands	Description
ud32	Runs the BEA Tuxedo ud32 client that reads a tab delimited text file, produces an FML32 buffer, and uses the buffer to make a request to a specified service.

Commonly Used tadmin Commands

The `tadmin` command allows you to inspect and dynamically configure your eLink application. There are many commands that can be invoked from `tadmin`, probably the most important being `help`. Several of the most useful commands are summarized in the following table.

Table 7-11 Commonly Used tadmin Commands

Command	Description
<code>help</code>	Prints help messages.
<code>quit</code>	Terminates the session.
<code>pclt</code>	Prints information for the specified set of client processes.
<code>psr</code>	Prints information for application and administrative servers.
<code>psc</code>	Prints information for application and administrative services.
<code>susp</code>	Suspends services.

For details about `tadmin` commands, refer to the *BEA Tuxedo Reference Manual* at <http://edocs.beasys.com/tuxedo/tux65/index.htm>.

D Data Types

Data types define the type of field in input and output formats. The valid data types for Information Integrator are described in Table D-1.

Table D-1 Data Types

Data Type	Description
Not Applicable	No data type is assumed.
String	A string of standard ASCII characters. Note that non-printable characters are valid as long as they are in the ASCII character set. (EBCDIC characters outside the valid ASCII range are not valid ASCII characters. During a reformat from ASCII to EBCDIC, if a character being converted is not in the EBCDIC character set, the conversion results in a EBCDIC space (hexadecimal 40)).
Numeric	A string of standard ASCII numeric characters.

Table D-1 Data Types

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

Table D-1 Data Types

Data Type	Description
IBM Signed Packed Integer	<p>Data type on larger IBM computers used to represent integers in compact form. This data type takes into account the sign (positive or negative) of a number. Each byte represents two decimal digits, one in each nibble of the byte. The final nibble is a hexadecimal C if the number is positive and a hexadecimal D if the number is negative.</p> <p>An example of how to generate a default value for an IBM Packed Integer is:</p> <p>Data Type: IBM Signed Packed Decimal Default Value: -12345 (default value in ASCII) Data Length: (Null - use the numbers in this field.)</p> <p>The control is optional and there is no corresponding field in the input message, so Formatter uses the default value, converts it to IBM Signed Packed Decimal, and generates the following output: 12 34 5D. Each pair of numbers represents the two nibbles of a byte. The result is three bytes long.</p>
IBM Zoned Integer	<p>Data type on larger IBM computers used to represent integers. Each decimal digit is represented by a byte. The left nibble of the byte is a hexadecimal F. The right nibble is the hexadecimal value of the digit. For example, 1234 is represented as F1 F2 F3 F4 (the number pairs show the hexadecimal values of the nibbles of each byte).</p>
IBM Signed Zoned Integer	<p>Data type on larger IBM computers used to represent integers. Each decimal digit is represented by a byte. The left nibble of each byte, except the last byte, is a hexadecimal F. The left nibble of the last byte is a hexadecimal C if the number is positive and a hexadecimal D if the number is negative. The right nibble of each byte is the hexadecimal value of the digit. For example, 1234 is represented as F1 F2 F3 C4 (the number pairs show the hexadecimal values of the nibbles of each byte). -1234 is represented as F1 F2 F3 D4.</p>

Table D-1 Data Types

Data Type	Description
Little Endian 2	Two-byte integer where the bytes are ordered with the rightmost byte being the high order or most significant byte. For example, the hexadecimal number 0x0102 is stored as 02 01 (where the number pairs show the hexadecimal values of the nibbles of a byte).
Little Swap Endian 2	Two-byte integer where the two bytes are swapped with respect to a Little Endian 2 value. For example, the hexadecimal number 0x0102 is stored as 01 02.
Little Endian 4	Four-byte integer where the bytes are ordered with the rightmost byte being the high order or most significant byte. For example, the hexadecimal number 0x01020304 is stored as 04 03 02 01 (where the number pairs show the hexadecimal values of the nibbles of a byte).
Little Swap Endian 4	Four-byte integer where the two bytes of each word are swapped with respect to a Little Endian 4 value. For example, the hexadecimal number 0x01020304 is stored as 03 04 01 02.
Big Endian 2	Two-byte integer where the bytes are ordered with the leftmost byte being the high order or most significant byte. For example, the hexadecimal number 0x0102 is stored as 01 02 (where the number pairs show the hexadecimal values of the nibbles of a byte).
Big Swap Endian 2	Two-byte integer where the two bytes are swapped with respect to a Big Endian 2 value. For example, the hexadecimal number 0x0102 is stored as 02 01.
Big Endian 4	Four-byte integer where the bytes are ordered with the leftmost byte being the high order or most significant byte. For example, the hexadecimal number 0x01020304 is stored as 01 02 03 04 (where the number pairs show the hexadecimal values of the nibbles of a byte).
Big Swap Endian 4	Four-byte integer where the two bytes of each word are swapped with respect to a Big Endian 4 value. For example, the hexadecimal number 0x01020304 is stored as 02 01 04 03.

Table D-1 Data Types

Data Type	Description
Decimal, International	Data type where every third number left of the decimal point is preceded by a period. The decimal point is represented by a comma. Numbers right of the decimal point represent a fraction of one unit. For example, the number 12345.678 is represented as 12.345,678. Decimal international data types can contain negative values.
Decimal, U.S.	Data type where every third number left of the decimal point is preceded by a comma. The decimal point is represented by a period. Numbers right of the decimal point represent a fraction of one unit. For example, the number 12345.678 is represented as 12,345.678. Decimal US data types can contain negative values.
Unsigned Little Endian 2	Like Little Endian 2 except that the value is interpreted as an unsigned value.
Unsigned Little Swap Endian 2	Like Little Swap Endian 2 except that the value is interpreted as an unsigned value.
Unsigned Little Endian 4	Like Little Endian 4 except that the value is interpreted as an unsigned value.
Unsigned Little Swap Endian 4	Like Little Swap Endian 4 except that the value is interpreted as an unsigned value.
Unsigned Big Endian 2	Like Big Endian 2 except that the value is interpreted as an unsigned value.
Unsigned Big Swap Endian 2	Like Big Swap Endian 2 except that the value is interpreted as an unsigned value.
Unsigned Big Endian 4	Like Big Endian 4 except that the value is interpreted as an unsigned value.
Unsigned Big Swap Endian 4	Like Big Swap Endian 4 except that the value is interpreted as an unsigned value.

Table D-1 Data Types

Data Type	Description
Date and Time*	Based on the international ISO-8601:1988 standard datetime notation: YYYYMMDDhhmmss. See the first paragraph of each of the Date and Time type descriptions for details on representing Date and Time components. Combined dates and times may be represented in any of the following list of base data types. For some data types, a minimum of 8 bytes is required. The list includes: Numeric, String, and EBCDIC.
Time*	Based on the international ISO-8601:1988 standard time notation: hhmmss where hh represents the number of complete hours that have passed since midnight (between 00 and 23), mm is the number of minutes passed since the start of the hour (between 00 and 59), and ss is the number of seconds since the start of the minute (between 00 and 59). Times are represented in 24-hour format. Times may be represented in any of the following list of base data types. For some data types, a minimum of 4 bytes is required. The list includes: Numeric, String, and EBCDIC.
Date*	Based on the international ISO-8601:1988 standard date notation: YYYYMMDD where YYYY represents the year in the usual Gregorian calendar, MM is the month between 01 (January) and 12 (December), and DD is the day of the month with a value between 01 and 31. Dates may be represented in any of the following list of base data types. For some data types, a minimum of 4 bytes is required. The list includes: Numeric, String and EBCDIC.

Table D-1 Data Types

Data Type	Description
Custom Date and Time*	Custom Date and Time enables users to specify different formats of dates, times, and combined dates and times. Date/Time formats may include: 1) Variations in year (2- or 4-digit year representation: YY or YYYY). 2) Variations in month—use of a month number (01-12) or three-letter abbreviation (JAN, FEB, MAR, APR, MAY, JUN, JUL, AUG, SEP, OCT, NOV, DEC). 3) Variations in the day of the month—use of a day of the month number (01-31). 4) Variations in hour—12-hour or 24-hour representation, with or without a meridian indicator (AM or PM.) 5) Custom date/time formats are available in the Format drop-down list. Custom date/time formats must have a base data type of Numeric, String, or EBCDIC.

* If you use Date and Time, Date, Time, or Custom Date Time as the Data Type, select a Base Data Type of ASCII String, Numeric, or EBCDIC. If you select Custom Date Time, also select a format string from the Format drop-down list and a Year Cutoff.

A 16-byte length limit applies to IBM platforms using packed data types (IBM Packed Integer and IBM Signed Packed Integer). However, the 16-byte limit has been removed for the IBM zoned data types (IBM Zoned Integer and IBM Signed Zoned Integer) to reflect the way the data is actually handled by IBM platform assemblers.

E Operators

This section describes the operators that can be used in expressions. Operators are defined by type. These types are:

- Existence Operators — Determine if a field exists in a message.
- Integer, String, Float, Date, Time, and DateTime Operators — Evaluate a message field against a static value using the specified operator symbol.
- Field-to-Field Operators — Compare two groups of data (fields) within a message.

Specific operators only apply to a certain data types. For example, there are no operators that compare integer and float data types. Operators are executed against a field name and a value or between two field names (each of the field names must be of the same type, as defined by the Message Type).

The tables that follow describe the valid operator values for the various operator types.

Table E-1 Existence Operators

Operator	Description
EXIST	Checks to see if the field is present in the message.
NOT_EXIST	Checks to see if the field is not present in the message.

Table E-2 String Operators

Operator	Description
STRING=	Checks to see if the field is equal to the defined literal string.
STRING<>	Checks to see if the field is not equal to the defined literal string.
STRING<	Checks to see if the field value (string) is less than the defined literal string.
STRING>	Checks to see if the field value (string) is greater than the defined literal string.
STRING<=	Checks to see if the field value (string) is less than or equal to the defined literal string.
STRING>=	Checks to see if the field value (string) is greater than or equal to the defined literal string.

Note: String operators do not check for variations in case. “TEST” is the same as “test” or “Test.”

Table E-3 Field-to-Field String Operators

Operator	Description
F2FSTRING=	Checks to see if the two field values (strings) are equal.
F2FSTRING<>	Checks to see if the two field values (strings) are not equal.
F2FSTRING<	Checks to see if the contents of Field1 (string) are less than the contents of Field2 (string).
F2FSTRING>	Checks to see if the contents of Field1 (string) are greater than the contents of Field2 (string).
F2FSTRING<=	Checks to see if the contents of Field1 (string) are less than or equal to the contents of Field2 (string).
F2FSTRING>=	Checks to see if the contents of Field1 (string) are greater than or equal to the contents of Field2 (string).

Note: Field-to-Field String operators do not check for variations in case. “TEST” is the same as “test” or “Test.”

Table E-4 Case Sensitive String Operators

Operator	Description
CS STRING=	Checks to see if the field is equal to the defined case-sensitive literal string.
CS STRING<>	Checks to see if the field is not equal to the defined case-sensitive literal string.
CS STRING<	Checks to see if the field value (string) is less than the defined case-sensitive literal string.
CS STRING>	Checks to see if the field value (string) is greater than the defined case-sensitive literal string.
CS STRING<=	Checks to see if the field value (string) is less than or equal to the defined case-sensitive literal string.
CS STRING>=	Checks to see if the field value (string) is greater than or equal to the defined case-sensitive literal string.

Notes: Case-sensitive string operators do check for variations in case. “TEST” is not the same as “test” or “Test.”

Case-sensitive operators may not work correctly on case-insensitive databases

Table E-5 Case Sensitive Field-to-Field String Operators

Operator	Description
F2FCSSTRING=	Checks to see if the two field values (case-sensitive strings) are equal.
F2FCSSTRING<>	Checks to see if the two field values (case-sensitive strings) are not equal.
F2FCSSTRING<	Checks to see if the case-sensitive contents of Field1 (string) are less than the case-sensitive contents of Field2 (string).

Operator	Description
F2FCSSTRING>	Checks to see if the case-sensitive contents of Field1 (string) are greater than the case-sensitive contents of Field2 (string).
F2FCSSTRING<=	Checks to see if the case-sensitive contents of Field1 (string) are less than or equal to the case-sensitive contents of Field2 (string).
F2FCSSTRING>=	Checks to see if the case-sensitive contents of Field1 (string) are greater than or equal to the case-sensitive contents of Field2 (string).

Notes: Case-sensitive Field-to-Field String operators do check for variations in case. “TEST” is not the same as “test” or “Test.”

Case-sensitive Field-to-Field String operators may not work correctly on case-insensitive databases.

Table E-6 Integer Operators

Operator	Description
INT=	Checks to see if the field is equal to the defined integer value.
INT<>	Checks to see if the field is not equal to the defined integer value.
INT<	Checks to see if the field value (integer) is less than the defined integer value.
INT>	Checks to see if the field value (integer) is greater than the defined integer value.
INT<=	Checks to see if the field value (integer) is less than or equal to the defined integer value.
INT>=	Checks to see if the field value (integer) is greater than or equal to the defined integer value.

Table E-7 Field to Field Integer Operators

Operator	Description
F2FINT=	Checks to see if the two field values (integers) are equal.
F2FINT<>	Checks to see if the two field values (integers) are not equal.
F2FINT<	Checks to see if the contents of Field1 (integer) are less than the contents of Field2 (integer).
F2FINT>	Checks to see if the contents of Field1 (integer) are greater than the contents of Field2 (integer).
F2FINT<=	Checks to see if the contents of Field1 (integer) are less than or equal to the contents of Field2 (integer).
F2FINT>=	Checks to see if the contents of Field1 (integer) are greater than or equal to the contents of Field2 (integer).

Table E-8 Float (Decimal) Operators

Operator	Description
Float=	Checks to see if the field is equal to the defined float value.
Float<>	Checks to see if the field is not equal to the defined float value.
Float<	Checks to see if the field value (float) is less than the defined float value.
Float>	Checks to see if the field value (float) is greater than the defined float value.
Float<=	Checks to see if the field value (float) is less than or equal to the defined float value.
Float>=	Checks to see if the field value (float) is greater than or equal to the defined float value.

E Operators

Table E-9 Date and Time Operators

Operator	Operator Type	Data Type	Description
DATE=	Date Operator	DATE	Checks to see if the field is equal to the defined date value.
DATE<>	Date Operator	DATE	Checks to see if the field is not equal to the defined date value.
DATE<	Date Operator	DATE	Checks to see if the field value (date) is less than the defined date value.
DATE>	Date Operator	DATE	Checks to see if the field value (date) is greater than the defined date value.
DATE<=	Date Operator	DATE	Checks to see if the field value (date) is less than or equal to the defined date value.
DATE>=	Date Operator	DATE	Checks to see if the field value (date) is greater than or equal to the date integer value.
F2FDATE=	Field to Field Date Operator	DATE	Checks to see if the two field values (dates) are equal.
F2FDATE<>	Field to Field Date Operator	DATE	Checks to see if the two field values (dates) are not equal.
F2FDATE<	Field to Field Date Operator	DATE	Checks to see if the contents of Field1 (date) are less than the contents of Field2 (date).
F2FDATE>	Field to Field Date Operator	DATE	Checks to see if the contents of Field1 (date) are greater than the contents of Field2 (date).
F2FDATE<=	Field to Field Date Operator	DATE	Checks to see if the contents of Field1 (date) are less than or equal to the contents of Field2 (date).
F2FDATE>=	Field to Field Date Operator	DATE	Checks to see if the contents of Field1 (date) are greater than or equal to the contents of Field2 (date).
TIME=	Time Operator	TIME	Checks to see if the field is equal to the defined time value.
TIME<>	Time Operator	TIME	Checks to see if the field is not equal to the defined time value.

Operator	Operator Type	Data Type	Description
TIME<	Time Operator	TIME	Checks to see if the field value (time) is less than the defined time value.
TIME>	Time Operator	TIME	Checks to see if the field value (time) is greater than the defined time value.
TIME<=	Time Operator	TIME	Checks to see if the field value (time) is less than or equal to the defined time value.
TIME>=	Time Operator	TIME	Checks to see if the field value (time) is greater than or equal to the time integer value.
F2FTIME=	Field to Field Time Operator	TIME	Checks to see if the two field values (times) are equal.
F2FTIME<>	Field to Field Time Operator	TIME	Checks to see if the two field values (times) are not equal.
F2FTIME<	Field to Field Time Operator	TIME	Checks to see if the contents of Field1 (time) are less than the contents of Field2 (time).
F2FTIME>	Field to Field Time Operator	TIME	Checks to see if the contents of Field1 (time) are greater than the contents of Field2 (time).
F2FTIME<=	Field to Field Time Operator	TIME	Checks to see if the contents of Field1 (time) are less than or equal to the contents of Field2 (time).
F2FTIME>=	Field to Field Time Operator	TIME	Checks to see if the contents of Field1 (time) are greater than or equal to the contents of Field2 (time).
DATETIME=	DATETIME Operator	DATETIME	Checks to see if the field is equal to the defined DATETIME value.
DATETIME<>	DATETIME Operator	DATETIME	Checks to see if the field is not equal to the defined DATETIME value.
DATETIME<	DATETIME Operator	DATETIME	Checks to see if the field value (DATETIME) is less than the defined DATETIME value.
DATETIME>	DATETIME Operator	DATETIME	Checks to see if the field value (DATETIME) is greater than the defined DATETIME value.
DATETIME<=	DATETIME Operator	DATETIME	Checks to see if the field value (DATETIME) is less than or equal to the defined DATETIME value.

E Operators

Operator	Operator Type	Data Type	Description
DATETIME>=	DATETIME Operator	DATETIME	Checks to see if the field value (DATETIME) is greater than or equal to the DATETIME integer value.
F2FDATETIME=	Field to Field DATETIME Operator	DATETIME	Checks to see if the two field values (DATETIMES) are equal.
F2FDATETIME<>	Field to Field DATETIME Operator	DATETIME	Checks to see if the two field values (DATETIMES) are not equal.
F2FDATETIME<	Field to Field DATETIME Operator	DATETIME	Checks to see if the contents of Field1 (DATETIME) are less than the contents of Field2 (DATETIME).
F2FDATETIME>	Field to Field DATETIME Operator	DATETIME	Checks to see if the contents of Field1 (DATETIME) are greater than the contents of Field2 (DATETIME).
F2FDATETIME<=	Field to Field DATETIME Operator	DATETIME	Checks to see if the contents of Field1 (DATETIME) are less than or equal to the contents of Field2 (DATETIME).
F2FDATETIME>=	Field to Field DATETIME Operator	DATETIME	Checks to see if the contents of Field1 (DATETIME) are greater than or equal to the contents of Field2 (DATETIME).

F eLink Information Integrator COBOL Copybook Importer

This section provides an overview of eLink Information Integrator COBOL Copybook Importer (hereafter referred to as eLink II CCBI). The following topics are discussed:

- BEA eLink II CCBI Overview
- Using eLink II CCBI
- Importing the Converted File
- Supported COBOL Data Types

BEA eLink II CCBI Overview

The eLink II CCBI is a command-line utility that you can use to parse COBOL copybook files to Extensible Markup Language (XML) files. Used in conjunction with the BEA eLink Information Integrator, eLink II CCBI enables you to perform data translation and integrate the result into your eLink application development project.

The process includes two steps. The first step is to convert the COBOL copybook into a Message Formatting Language (MFL) document. (This is a type of XML document.) The second step is to load the XML document into the eLink Information Integrator database.

The following definitions give you a basic understanding of the components with which you will be working:

ccbparsE

This is the eLink II CCBI parsing command that transforms the COBOL copybook input into an XML document. Endian order, data type, and data size are handled by command line options. The ccbparsE command has the following format:

```
ccbparsE [options] inFileNamE [options] [outFileNamE]
[options]
```

where:

ccbparsE

Is the parsing command.

[options]

Define integer order, data type, and data size. The options can be entered anywhere on the command line and have the following values:

`BigEndian` specifies that the highest order bit is stored first (default).

`LittleEndian` specifies that the lowest order bit is stored first.

`EBCDIC` specifies that the host data type is EBCDIC (default).

`ASCII` specifies that the host data type is ASCII (default).

inFileNamE

Is the name of the input COBOL copybook file (required).

outFileNamE

Is the name of the output XML file (default is `stdout`).

Message Formatting Language (MFL)

This is a specific type of XML format containing instruction that can be used by eLink Information Integrator for converting data. These MFL documents can be imported into eLink Information Integrator using the `MsgDefAdmin` utility. The eLink II CCBI generates MFL documents from COBOL copybooks, thus eliminating the need to manually design the data conversion formats with the eLink Information Integrator GUI.

MsgDefAdmin

This is a utility that loads an MFL file into the eLink Information Integrator database.

Using eLink II CCBI

The following steps demonstrate how to use eLink II CCBI. The Java runtime environment must be available and you must know the location of the installed Information Integrator files.

1. Change to your <installation directory>\bin directory.
2. Set a path to your Java runtime environment, for example:

```
set path = jdk1.2.2\bin;%path%
```
3. Convert the COBOL copybook file. For example, to convert the copybook named *cobol2* from an EBCDIC host with BigEndian order into an MFL file named *star*, enter the following:

```
ccbpars cobol2 star
```
4. Import the converted file into the eLink Information Integrator database. Refer to “Importing the Converted File” for more information.

Importing the Converted File

The following steps import a converted MFL file into the eLink Information Integrator database. The eLink Information Integrator must be running and the imported file must be available in the local directory structure. It may be necessary to transfer the imported file from the directory where it was created in the previous procedure. Refer to the *BEA eLink Information Integrator User Guide* and the *BEA eLink Information Integrator Installation and Administration Guide* for details about installing and using that product.

1. Open a DOS command-prompt window.
2. Change to the directory where the eLink Information Integrator `MsgDefAdmin` utility is installed (for example, `C:\BEAII`). Type `MsgDefAdmin` and press Enter. The `MsgDefAdmin` importer utility appears as shown in Listing 7-33.

Listing 7-33 Listing 3-2 MsgDefAdmin

```
- Copyright (c) 2000 BEA Systems, Inc.  
All Rights Reserved.  
Distributed under license by BEA Systems, Inc.  
BEA eLink Information Integrator is a registered trademark.  
This product includes software developed by the Apache Software  
Foundation (http://www.apache.org/).  
- - - BEA eLink Information Integrator 1.0 - - -  
- - - Message Definition Importer Tool - - -  
Enter choice:  
1) Add Message Definition  
2) Get Message Definition  
3) Delete Message Definition  
4) List Message Definitions  
Q) Quit >
```

3. Type **1** to select **Add Message Definition** and press Enter. An additional prompt displays asking you to specify the name of the file containing the MFL message definition.
4. Type the name of the message file you created in the previous procedure and press Enter. `MsgDefAdmin` processes the file, stores the message definition in the `iiDatabase`, and returns a message that the definition was successfully created.
5. Type **Q** at the prompt and press Enter to quit `MsgDefAdmin`.
6. Use the imported MFL file in your application program as needed.

Supported COBOL Data Types

The COBOL Copybook Importer does not recognize all extensions to standard (ANSI) COBOL and data type size information. As a result, Copybooks that make use of non-supported extensions may not get translated correctly.

In addition, Information Integrator does not support 8 byte integer or floating point data types, and the MFL parser identifies some date fields as character. For this reason, the MFL parser includes both the original copybook entry and the results of the XML parsing in the source element for each field. You can use this information to guide you in editing the MFL data to better represent the original Copybook.

For example, if your original copybook entry appears as follows:

```
05  birth-date           pic xx/xx/xx.
```

The XML translation appears as follows:

```
<field name="birth-date"  
type="alphanumeric"  
length="8"  
picture="xx/xx/bx" align="n"/>
```

The MFL translation appears as follows:

```
<FieldFormat name='birth-date' type='EBCDIC' length='8'  
source='pic xx/xx/xx.(alphanumeric)'/>
```

If you review the source element in the MFL document, you will find that the appropriate Information Integrator data type is either Smmddy or Sddmmy. You can decide which data type is most appropriate for the specific data.

Table F-1 lists the COBOL data types supported by the parser. Unsupported data types cause an error in the parser. All errors encountered by the parser are noted with a message that includes the line numbers in question. Since the parser generates an empty MFL document on any error, the lines containing unsupported types should either be commented out or deleted prior to running the parser.

Table F-1 COBOL Data Types

COBOL Type	Support
COMP, COMP-4, BINARY (integer)	supported
COMP, COMP-4, BINARY (fixed)	supported
COMP-3, PACKED-DECIMAL	supported
COMP-5, COMP-X	supported (note 1)
DISPLAY numeric (zoned)	supported
BLANK WHEN ZERO (zoned)	supported
SIGN IS LEADING (zoned)	supported
SIGN IS LEADING SEPARATE (zoned)	supported
SIGN IS TRAILING (zoned)	supported
SIGN IS TRAILING SEPARATE (zoned)	supported
edited numeric	supported
COMP-1, COMP-2 (float)	supported
edited float numeric	supported
DISPLAY (alphanumeric)	supported
edited alphanumeric	supported
INDEX	supported
POINTER	supported
PROCEDURE-POINTER	supported
JUSTIFIED RIGHT	ignored
SYNCHRONIZED	ignored
REDEFINES	supported
66 RENAMES	not supported

Table F-1 COBOL Data Types

COBOL Type	Support
66 RENAMES THRU	not supported
77 level	supported
88 level (condition)	ignored
group record	supported
OCCURS (fixed array)	supported
OCCURS DEPENDING (variable-length)	supported
OCCURS INDEXED BY	ignored
OCCURS KEY IS	ignored

Support for the following data types is limited. The following formats will be converted to an unsigned 4 byte integer type:

```
05 pic 9(5) comp-5
```

```
05 pic 9(5) comp-x
```

The following formats will generate errors:

```
05 pic X(5) comp-5
```

```
05 pic X(5) comp-x
```

In these samples, pic 9(5) could be substituted for pic x(5).

Note: Some unsupported types can be changed to a supported type with little loss of accuracy. In the formats shown above, you could change a pic X() statement to pic 9() statement.

Glossary

accelerator keys

A combination of keystrokes that bypasses the menu system to carry out an action.

action

An action specifies a subscription function that is performed when a message evaluates as true. Actions are defined in the Rules interface. Each action type has its own set of option name-value pairs.

administration workstation

A form of client from which a user carries out BEA eLink Information Integrator management duties.

alternative format

A special form of compound format where one format in a set of alternatives applies to a message. For example, if the alternative format is named A, it may contain component formats B, C, and D. A message of format A may actually be of variation of only B, C, or D.

application group

A logical grouping of applications used to organize rules.

argument

In Rules, an argument is evaluation criteria made up of fields from a message and associated operators.

asynchronous

In electronic messaging, a method of operation in which receiving applications are loosely coupled and independent. The receiver need not respond immediately to a message, and the sender does not have to wait for a response before proceeding with the next operation. Compare to synchronous.

cross-platform

Used to describe programs that can execute in dissimilar computing environments.

Data Type

Describes how to interpret the data.

delimiter

One or more characters marking either the end or beginning of a piece of data.

embedded length

A piece of data associated with a field indicating the length of the data in the field. For example, the data may appear in a message as 3.DOG, where the 3 indicates that the field data (DOG) is three bytes in length.

field

The smallest possible container for information. You can use a field in more than one message. If the first_name field exists in one message, for example, you can use the same field in other messages.

flat format

A format only containing fields and associated controls. Flat input formats are composed of fields with associated controls.

format

Formats describe how messages are constructed. Input formats describe how to separate input messages into their component parts. Output formats describe how to build output messages from the parsed components of an input message.

input control

In Formatter, an input control is used to parse input data. The input control is used to determine how to find the beginning and end of the data in a field.

literal

One or more symbols or letters in data that represents itself.

message

A packet of data both sent and received with a definite beginning and end.

message type

A message type defines the layout of a string of data. The message type name in Rules is the same as the input format name in Formatter.

nesting level

Level within the hierarchy of a specific component. A repeating child format and the fields within it may have a nesting level one greater than that of the parent format and any non-repeating components of the parent format. The nesting level of the root format is one.

option

An option consists of a name-value pair of data related to an action.

parent/child

Compound formats contain other flat and compound formats. If you have a compound format X that contains a repeating format Y, X is the parent to child Y.

parse

To analyze a message by breaking it down into its component fields.

persistence

The ability of a computerized system to remember the state of data or objects between runs.

protocol

A set of rules that govern the transmission and reception of data.

queue

A simple data structure for managing the time-staged delivery of requests to servers. Queued elements may be sorted in some order of priority. Clients insert items in the queue and servers remove items from the queue, as soon as possible, in batch, or periodically.

regular expression

Strings which express rules for string pattern matching.

repeating component

A component (either a field or format) that may appear multiple times in an input or output message.

rule

A rule is uniquely defined by its application group, message type, and rule name. It contains evaluation criteria (a rules expression) and is associated with subscriptions to perform if the rule evaluates to true.

shared subscription

A subscription that is associated with more than one rule. This subscription will only be retrieved once by the Rules APIs even if multiple associated rules evaluate to true.

subscription

A subscription is uniquely identified by its application group, message type, and subscription name. It contains actions with options and can be associated with one or more rules.

synchronous

In electronic messaging, a method of operation in which sender and receiver applications are tightly coupled and dependent. The receiver must answer the sender's message immediately with a well-defined response; the sender must wait for the receiver's response before proceeding to the next operation. Compare to asynchronous.

table

A database remembers relationships between pieces of information by storing the information in tables. The columns and rows in each table define the relationships in a highly structured way. Tables are classified by function into two types: support tables and data tables. Most tables fit into only one category, but some can serve as support and data tables.

A support table stores information that changes infrequently and functions as a list from which you make selections.

tag

A set of bits of characters that identifies various conditions about data in a file. In Formatter, a standard value indicating the field name.

Index

A

actions

- Generic action 6-16
- Put Message action 6-16
- Reformat action 6-16

adding parse controls 4-39

apitest A-11

application groups 6-9

assigning users to a database 7-8

C

COBOL Copybook Importer F-1

comparison values 4-9

configuration files

- sqlsvses.cfg 7-3

creating

- formats 4-54, 4-55, 4-56, 4-57
- literals 4-10
- output controls 4-39

customer support contact information v

D

default values 4-9

defining user groups 7-9

delimiters 4-9

documentation, where to find it iv

E

eLink Information Integrator COBOL
Copybook Importer F-1

eLink Platform Reference C-1

Expression property sheet 6-12, 6-13

expressions 6-12

- creating 6-13

modifying 6-13

F

field names 4-11

fields 4-11

- creating 4-11

format terminators 4-9

formats

- NNFie A-6

Formats function 4-54

Formatter

alternative formats

- output formats 4-63

apitest executable A-11

creating

- fields 4-11
- formats 4-54, 4-55, 4-56, 4-57
- literals 4-10

fields 4-11

Formats function 4-54

input controls 4-22

- saving as output controls 4-53, 4-58

input formats

- tagged 4-62

literals 4-9

msgtest executable A-12

output controls 4-39

- creating 4-39

Math Expression operations 4-15

operation substitute strings 4-17

output operation collections 4-20

output operations 4-13

sqlsvses.cfg 7-3

G

Generic action 6-16

I

input controls 4-22
input formats 4-62

L

literals 4-10
Login accounts
 creating 7-7

M

mandatory data 4-39
Math Expression operations 4-15
message types 6-9
modifying
 expressions 6-13
MsgDefAdmin 3-1
msgtest A-12

N

NNFie A-6
notation conventions vi

O

operation substitute strings 4-17
optional data 4-39
Oracle system enhancements 7-5
 granting roles to users 7-6
output controls
 formatting output data 4-39
 mandatory data 4-39
 Math Expression operations 4-15
 optional data 4-39
 output operation collections 4-20
 output operations 4-13
 saving input controls 4-53, 4-58
output formats 4-63
output operation collections 4-20

output operations 4-13

P

pad character 4-9
Platform C-1
prefixes 4-9
printing product documentation iv
processing literals 4-9
Put Message action 6-16

R

Reformat action 6-16
related information v
repeating sequence terminators 4-9
Rules

- application groups 6-9
- building Rules 6-8
- creating
 - expressions 6-13
- Expression property sheet 6-12
- expressions 6-12
- message types 6-9
- Oracle system enhancements 7-5
- starting Rules 6-3
- subscriptions 6-15
 - actions 6-16
- Sybase/SQL Server system
 - enhancements 7-7

S

saving input controls as output controls 4-53,
 4-58
SQL Server system enhancements 7-7
sqlsvses.cfg

- configuring 7-3

starting Rules 6-3
Subscription List 6-15
subscriptions 6-15

substitute operations 4-17
substitute values 4-9
suffixes 4-9
support
 technical v
Sybase/SQL Server system enhancements 7-7
Sybase/SQL system enhancements
 assigning users to a database 7-8
 creating Login accounts 7-7
 defining user groups 7-9
system enhancements
 Oracle 7-5
 Sybase/SQL Server 7-7

T

tag values 4-9
tagged input format 4-62
Tester 5-1
trim characters 4-9

U

users
 assigning to a database 7-8
 defining groups 7-9
 granting roles 7-6

