# BEA eLink Information Integrator

## User Guide

BEA eLink Information Integrator Version 1.0
Document Edition 1.0
April 2000

**BEA eLink Information Integrator User Guide**

| Document Edition | Part Number | Date | Software Version |
|---|---|---|---|
| 1.0 | | April 2000 | BEA eLink Information Integrator 1.0 |

# Contents

# About This Document

This document explains what BEA eLink Information Integrator is and describes how to use the Server and the Formatter, Rules, and Tester GUIs.

This document covers the following topics:

- "Understanding the BEA eLink Solution" provides an overview of BEA eLink Information Integrator and the entire BEA eLink family of products.
- "Getting Started" describes the basic steps necessary to start using BEA eLink Information Integrator.
- "Building Format Definitions" describes the Formatter component of BEA eLink Information Integrator and provides instructions for its use.
- *"Defining Rules"* describes the Rules component of BEA eLink Information Integrator and provides instructions for its use.
- "Testing Message Parse and Reformat" describes the Tester component of BEA eLink Information Integrator and provides instructions for its use.
- *"Data Types"* describes the input format field types.
- *"MFL Document Type Definition"* describes the Message Format Language Document Type Definition (DTD) and provides an example Message Format Language document built using the DTD.
- *"Glossary"* provides definitions for terms that may be unfamiliar.

## What You Need to Know

This document is intended for application programmers 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/.

## 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, indicate that you are using the documentation for the BEA eLink Information Integrator 1.0 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 http://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 | 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:<br>*error_file_name*<br>The *when-return* value... |
| User input and screen output | 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:<br>**`dir c:\accounting\data`**<br>`Volume in drive C is WIN_NT_1`<br>`Volume Serial Number is 1234-5678`<br>`Directory of C:\BEADIR\DATA` |
| Syntax | Code samples can include the following elements:<br>■ Variable names can include hyphens or underscores (for example, *error_file_name)*<br>■ Optional items are enclosed in square brackets ([ ]). If you include an optional item, do not code the square brackets.<br>■ 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.<br>■ An ellipsis ( ... ) indicates that the preceding element can be repeated as necessary. |
| Omitted code | 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. |
| Environment variables | Environment variables are formatted in an uppercase font.<br>ENVFILE=${APPDIR} |
| Key names | Key names are presented in boldface type.<br>Press **Enter** to continue. |

| **Item** | **Examples** |
|---|---|
| Literals | Literals are formatted in a monospace font.<br>`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.<br>Type your password in the **Logon** window.<br>Select **Export** to make the service available to the client. |

# 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

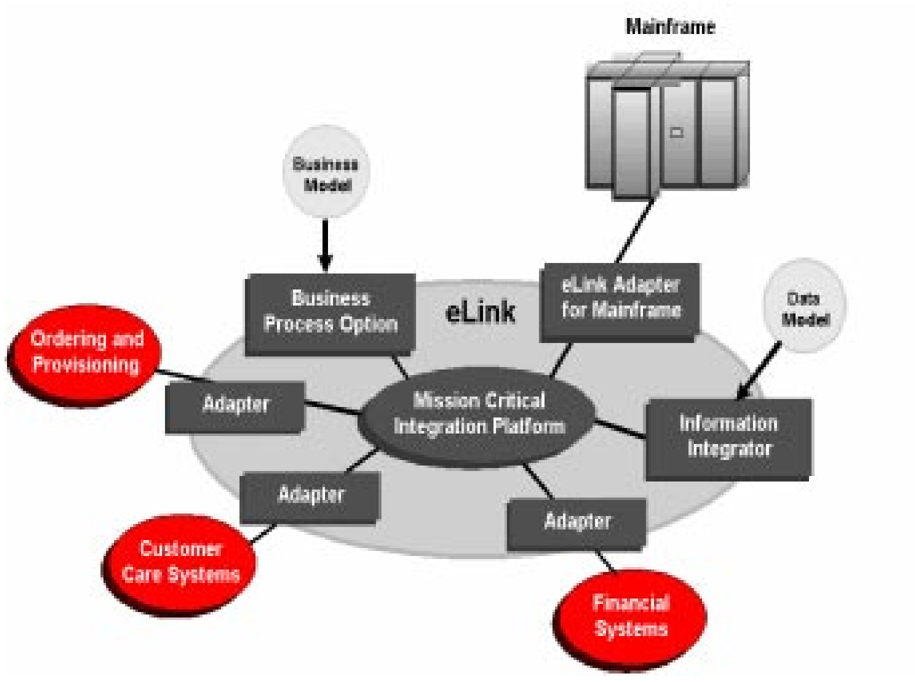
BEA eLink™ provides an open Enterprise Application Integration (EAI) solution that allows applications throughout organizations to communicate seamlessly. Using EAI, you gain the long-term flexibility and investment protection you need to keep up with today’s ever-changing business environment.

Typically, companies use packaged applications to automate many internal operations, such as financial, manufacturing, and 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 that leverage BEA’s transaction platform to integrate existing legacy applications with customer-focused and business-to-business e-commerce initiatives.

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 Information Integrator and 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 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. The terms, Tuxedo services/ servers and eLink services/servers can be used interchangeably. Because this document specifically addresses 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 queueing subsystem.

The following components operate with BEA eLink Platform:

- The Information Integrator translates data models used by different applications into a common data format. 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 Business Process Option (BPO) 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

The BEA eLink Solution combines key features from eLink Platform, eLink Information Integrator, and eLink Adapters. These key features include:

- Business process automation
- Message transport
- Data transformation
- Content-based routing rules
- Management tools

The BEA eLink Information Integrator provides three of the five key features: business process automation, data transformation, and content-based routing rules.

## Architectural Framework

The BEA eLink Information Integrator architecture is based on functional layers providing rules-based message recognition, evaluation, and dynamic formatting. The BEA eLink Information Integrator design permits convenient portability to a wide range of hardware, operating systems, and DBMSs.

The BEA eLink Information Integrator framework includes the following components:

- Formatter
- Rules
- Tester
- MsgDefAdmin
- iiServer

## Formatter

Formatter allows you to define a message format as a series of components. These components describe how to parse each piece of information in a message, how to output each piece of information, and how each piece of information relates to the message as a whole.

The Formatter components include the following:

- Literals
- Fields
- Input and output controls
- Flat and compound formats

For more information about Formatter, refer to “Building Format Definitions.”

## Rules

Rules allows you to define processing rules used by the iiServer to evaluate an input message. A processing rule consists of a Boolean expression that is evaluated against the input message and one or more subscriptions. Subscriptions contain actions that are performed if a Rules expression evaluates to TRUE.

For more information about Rules, refer to “Defining Rules.”

## Tester

Tester allows you to parse and reformat messages as a validation test. Tester uses the control tables built using Formatter to recognize and parse input and output messages. However, Tester parses and reformats messages directly on the desktop rather than adding them to the business process flow and parsing them using the iiServer.

For more information about Tester, refer to “Testing Message Parse and Reformat.”

## MsgDefAdmin

The `MsgDefAdmin` importer utility lets you add, retrieve, delete, and list message format definitions that are stored in the iiDatabase. `MsgDefAdmin` allows you to define message formats in an XML language called Message Definition Format Language

(MFL). You can then import these textual descriptions into the iiDatabase. Message definitions that exist within the iiDatabase can also be listed, retrieved, or deleted using this utility.

For more information about `MsgDefAdmin`, refer to “Using MsgDefAdmin to Build Definitions.”

## iiServer

The iiServer translates messages from one format to another, simplifying message exchange between different systems and applications. The iiServer breaks complex messages into simple messages for processing, handling anything from formats containing fixed, tagged, or delimited fields to multi-part nested formats.

The iiServer uses control tables within the iiDatabase to recognize and parse input and output messages. These tables describe message formats and format components. The iiServer uses this information to interpret values from incoming messages and dynamically construct outgoing messages.

The iiServer supports the following basic message-passing models:

- Request/Response translation
- Data enrichment
- Content-based routing
- Message explosion

**Note:** You can combine all or parts of these models into a single processing cycle.

## Request/Response Translation

In this model, a client/requesting 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 requestor.

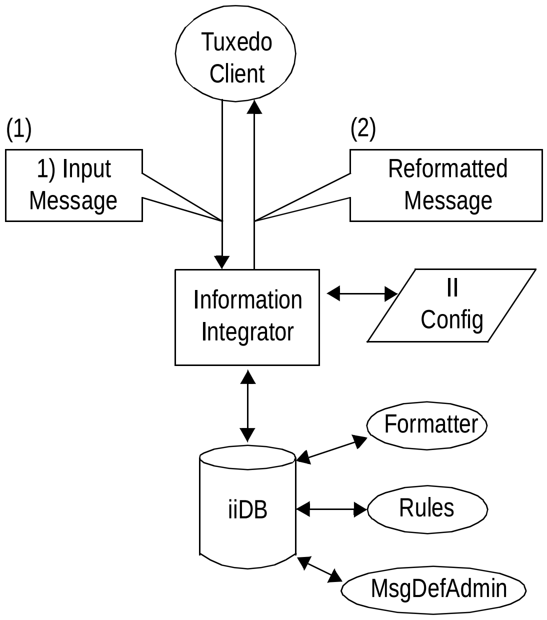
**Note:** In this context, client refers to the generator of a request, even if the generator is a Tuxedo server.

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 1-2.

**Figure 1-2 Request/response Translation Process**



## Data Enrichment

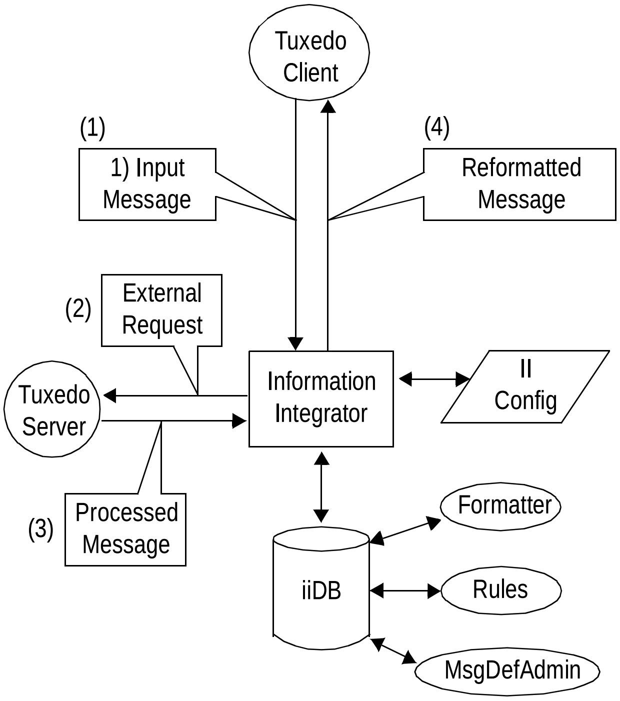
This processing model is similar to Request/Response Transformation. The difference in data enrichment is that the message can be processed by an external service to provide data enrichments that are not possible with the iiServer. This external enrichment of the message is transparent to the requesting 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)
- Reformat (optionally transform the message to meet the requestor’s requirements)
- Return (return the transformed message to the requestor)

The data enrichment process is shown in Figure 1-3.

**Figure 1-3 Data Enrichment Process**



## Content-based Routing

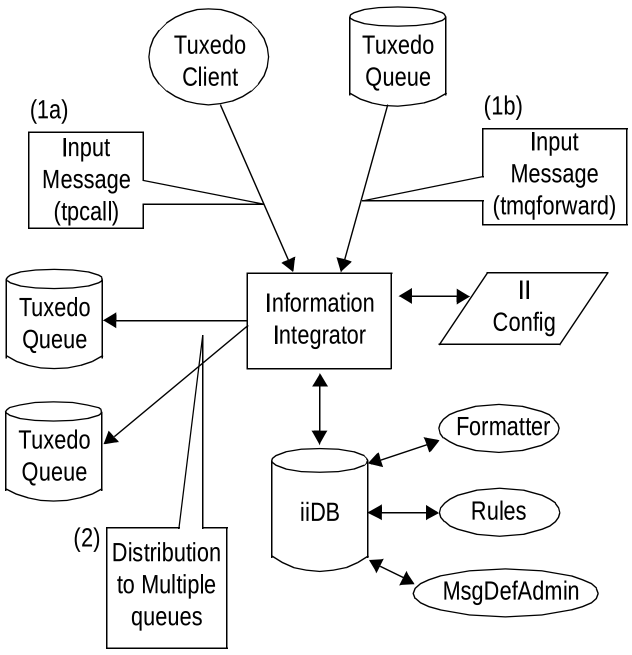
This processing model evaluates elements of the input message and determines how the message should be distributed.

Typical actions associated with this model include:

- Rule 1: IF Field 1 == SomeValue
  - Reformat (optionally transform message before distribution)
  - Enqueue (place message on a defined queue)
- Rule2: IF Field2 == AnotherValue
  - Reformat (optionally transform message before distribution)
  - Enqueue (place message on another defined queue)

The content-based routing process is shown in Figure 1-4.

**Figure 1-4 Content-based Routing Process**



## Message Explosion

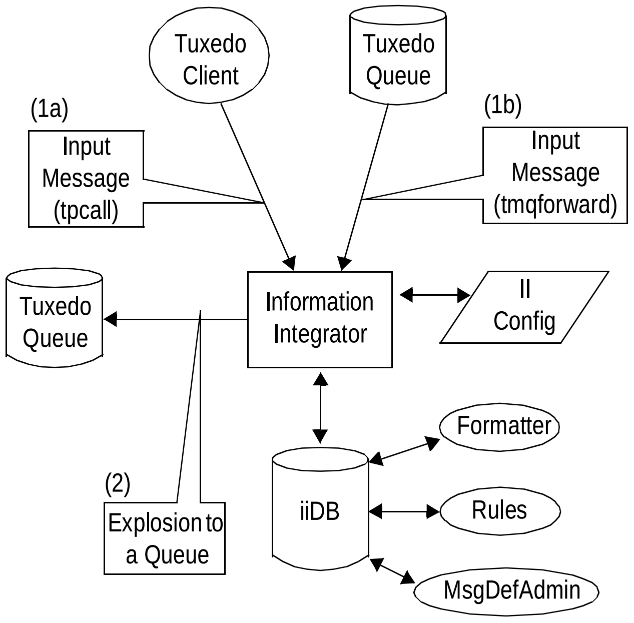
This model accepts a single input message, breaks it into component parts, and distributes selected components as individual messages.

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 1-5.

**Figure 1-5 Message Explosion Process**



# 2 Getting Started

This section provides an example of the Information Integrator transformation process. In order to use the example, you must have BEA eLink Information Integrator and eLink platform (BEA Tuxedo) already installed on your system.

This example illustrates how to perform data transformation. The example constructs an FML32 input buffer containing the three fields `STREET`, `CITY`, and `STATE`. The iiServer reformats these fields in the response buffer by concatenating them into a single output field called `ADDRESS`.

The files for this example are shipped with BEA eLink Information Integrator in the `C:\BEAII\InfoInt\sample` directory. The `sample` directory contains:

- `address.c`—Defines the source code that constructs the FML32 input buffer.
- `address.data`—Defines the FML32 input buffer.
- `build_sample.cmd`—Compiles `address.c` for the NT platform.
- `build_sample.sh`—Compiles `address.c` for the UNIX platform.
- `Fieldtable.txt`—Defines the Tuxedo FML buffer field table.
- `ii.cfg`—Defines configuration settings for the application service and the name of the application input message.
- `II_address_msg1.xml`—Specifies the contents of the message definition.
- `ud32.in`—Specifies the input data to run the example using the Tuxedo `ud32` utility.
- `sample.ubb`—Defines the Tuxedo UBB configuration settings.

This section describes the following topics, which you should perform in the order listed:

- Defining Message Formats
- Storing Formats in the iiDatabase
- Defining Rules
- Configuring the eLink Platform
- Running the Application

## Defining Message Formats

Message formats define the layout of data that is processed by the iiServer. This layout defines the actual 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 social security number, the message contains two fields.

For the example, you need to define a message format for the request buffer. The request buffer contains three fields: `STREET`, `CITY`, and `STATE`. The response buffer that the iiServer returns only contains the `ADDRESS` field. Because the response buffer is an FMLVO buffer (which is formatted by the iiServer), you do not need to define its format.

You can define the request buffer format using the steps described in the following procedure.

1. Open a DOS command-prompt window.
2. Use an XML language called Message Format Language (MFL) to define the buffer format. In the example, the request buffer format file is called `II_address_msg1.xml`, which is shown in Listing 2-1.

**Listing 2-1 `II_address_msg1.xml`**

```
<?xml version='1.0'?>
<!DOCTYPE MessageFormat SYSTEM 'mfl.dtd'>
<MessageFormat name="II_address_msg1">
    <FieldFormat name="STREET" type="String" delim="," />
    <FieldFormat name="CITY" type="String" delim="," />
    <FieldFormat name="STATE" type="String" />
</MessageFormat>
```

**Note:** The document type definition (DTD) for MFL is stored in `mfl.dtd`. Refer to *"MFL Document Type Definition"* for a listing of `mfl.dtd`.

For more information about defining message formats, refer to "Building Format Definitions."

## Storing Formats in the iiDatabase

Once you define the message format for the FML32 input buffer, you need to store it in the iiDatabase using the `MsgDefAdmin` utility.

**Note:** The `MsgDefAdmin` utility uses configuration settings specified in `sqlsvses.cfg`. Refer to *BEA eLink Information Integrator Installation and Administration Guide* for more information about the `sqlsvses.cfg` file syntax.

You can store format definitions by using the steps described in the following procedure.

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 2-2.

**Listing 2-2 `MsgDefAdmin` Importer Utility**

```
-  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.com/).

-  -  -    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 Definition
Q) Quit
>
```

4. Type **`1`** at the prompt and press **Enter**. `MsgDefAdmin` prompts you for the name of the file containing the XML message definition.
5. Type **`sample\II_address_msg1.xml`** and press **Enter**. `MsgDefAdmin` processes the file and returns a message that the definition was successfully created.
6. Type **`Q`** at the prompt and press **Enter** to quit `MsgDefAdmin`.

## Defining Rules

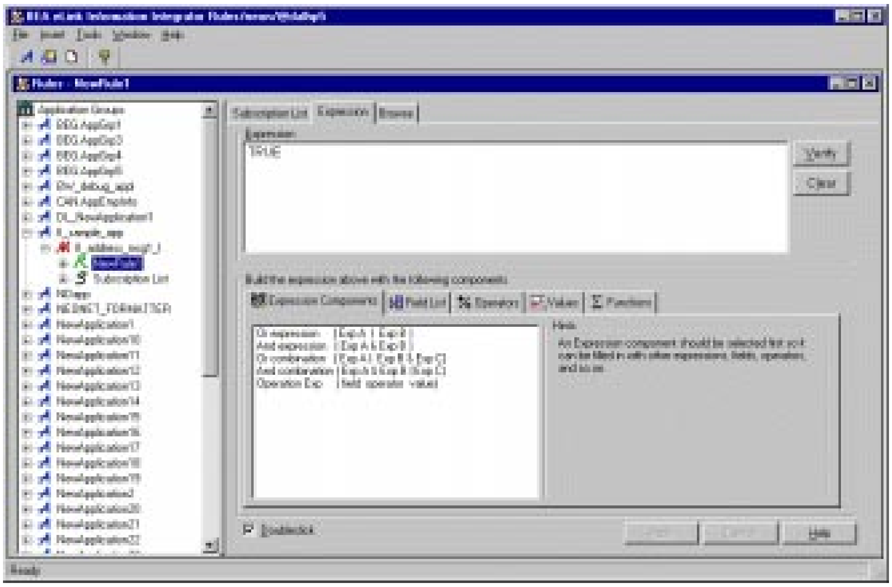
You now need to specify what actions the iiServer should take if a specific message type is received. Examples of general actions include reformatting a message, calling a Tuxedo service with a message, exploding a message into multiple messages, and enqueuing a message to a queue.

In the example, you need to define rules specifically for reformatting and returning a message using the Rules interface. This definition process includes the following steps:

- Define the rule
- Define the rule’s expression
- Define the rule’s subscriptions
- Define the actions to associate with a rule’s subscription

Figure 2-1 shows how the Rules interface should appear when defining the rule's expression. Note that for this example, the value in the `Expression` window must be set to `TRUE`.

**Figure 2-1 Defining a Rule's Expression**

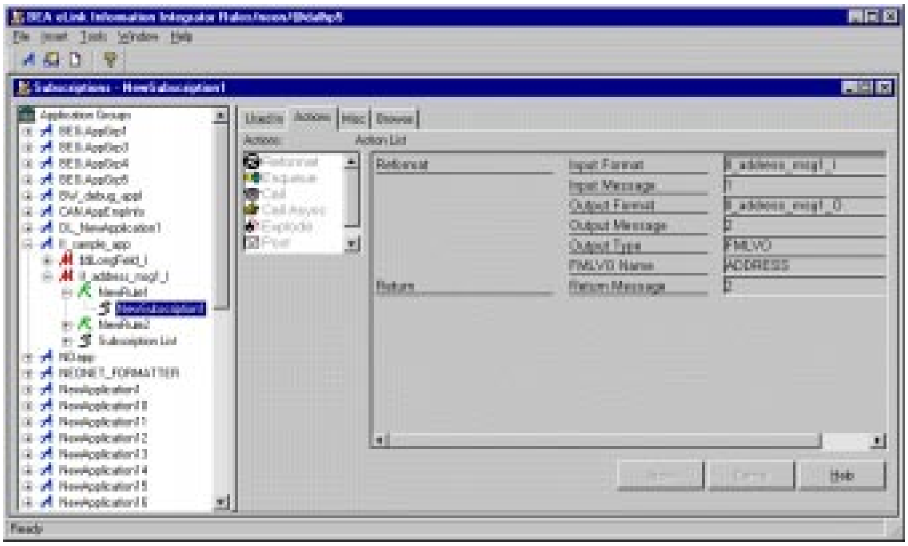


```
Or expression     | Exp A | Exp B |
And expression    | Exp A & Exp B |
Or combination    | Exp A | Exp B & Exp C|
And combination   | Exp A & Exp B | Exp C|
Operation Exp     | field operator value|
```

An Expression component should be selected first so it can be filled in with other expressions, fields, operators, and so on.

Figure 2-2 shows how the Rules interface should appear when defining the actions to associate with a rule’s subscription.

**Figure 2-2 Defining Actions**


BEA eLink Information Integrator Rules
File Insert Tools Window Help
Subscriptions - NewSubscription1
Application Groups
Actions
Action List
Reformat
Enqueue
Call
Call Async
Explode
Post
Help
Ready

| | |
|---|---|
| Input Format | ll_address_msgf_I |
| Input Message | 1 |
| Output Format | ll_address_msgf_O |
| Output Message | 2 |
| Output Type | FMLVO |
| FMLVO Name | ADDRESS |
| Return Message | 2 |

Once the subscription and its actions are defined, they must be associated with the rule. You can do this by dragging the subscription and dropping it on the rule in the left-hand pane..

Refer to “Building Rules” for more information about the rules definition process.

# Configuring the eLink Platform

Once you define appropriate rules, you need to configure your eLink platform. This includes the following tasks:

- Updating ii.cfg
- Setting Environment Variables
- Updating sample.ubb
- Creating a Tuxedo Field Table

## Updating ii.cfg

The `ii.cfg` file contains service definitions. For each message translation you want to do, you must enter an appropriate service definition in `ii.cfg`.

You can update this file by using the steps described in the following procedure.

1. Open a DOS command-prompt window or start a UNIX session.
2. Edit the `ii.cfg` file and specify an appropriate service definition. The `ii.cfg` file shipped with the example contains the service definition shown in Listing 2-3.

**Listing 2-3 `ii.cfg`**

```
*SERVICE
NAME=II_addr1
APPL_NAME=II_sample_app
INPUT_FORMAT=II_address_msg1_I
```

**Notes:** The `NAME` field specifies the name that Tuxedo clients must use to request services from the iiServer. The value of the `NAME` field is used when testing an application (in the `ud32.in` file) or running an application as a C source client (in the `address.c` file). You can choose any service name you like.

The `APPL_NAME` and the `INPUT_FORMAT` fields specify the application group name and the input message format defined in the Rules interface.

You must restart the iiServer after you change the `ii.cfg` file.

## Setting Environment Variables

You can set environment variables by using the steps described in the following procedure.

1. Open a DOS command-prompt window or start a UNIX session.
2. Edit the `setenv.bat` file on NT or the `setenv.sh` file on UNIX and specify the appropriate values for the environment variables. The `setenv.bat` file shipped with the example is shown in Listing 2-4.

**Listing 2-4 `setenv.bat`**

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

**Notes:** On NT, Tuxedo sets `TUXDIR` and adds `TUXDIR\bin` to the `PATH` environment variable. On UNIX, Tuxedo generates a TUX.ENV file that sets these variables for you.

You need to set the `INCLUDE`, `LIB`, `APPDIR`, `TUXCONFIG`, `FIELDTBLS32`, and `FLDTBLDIR32` variables on NT and UNIX, and the `SHLIB_PATH` (or `LD_LIBRARY_PATH` or `LIBPATH`) variable on UNIX.

`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`.

3. Type **`setenv.bat`** on NT or **`setenv.sh`** on UNIX and press **Enter** to complete the definition of environment variables.

## Updating sample.ubb

The `sample.ubb` file (Tuxedo UBB configuration file) defines the resources, machines, groups, servers, and services that Tuxedo should use.

You can update this file by using the steps described in the following procedure.

1. Open a DOS command-prompt window or start a UNIX session.
2. Edit `sample.ubb` and specify appropriate values for each variable. The `sample.ubb` file shipped with the example is shown in Listing 2-5.

**Listing 2-5 sample.ubb**

```
*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=<Location of the Tuxedo config file>
              TUXDIR=<Tuxedo directory>

<Machine Name>  LMID=II

*GROUPS
GROUP1
       LMID=IIGRPNO=1OPENINFO=NONE

*SERVERS
DEFAULT:
              CLOPT="-A"

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

*SERVICES
II_SERVICE
```

**Notes:** You need to set the `APPDIR`, `TUXDIR`, and `TUXCONFIG` variables independently of external variables.

Be sure that the variable values you set match the environment variables in `setenv.bat` or `setenv.sh`.

The value of the `Machine Name` field must be entered in upper case on NT.

3. Type **`tmloadcf -y sample.ubb`** and press **Enter** to convert `sample.ubb` to a binary TUXCONFIG file.

**Note:** You must restart Tuxedo after you change the `sample.ubb` file.

## Creating a Tuxedo Field Table

You must create a Tuxedo field table if your messages are contained in FML buffers. You can create this table by using the steps described in the following procedure.

1. Open a DOS command-prompt window or start a UNIX session.
2. Edit the `Fieldtable.txt` file and specify the appropriate table entries. The `Fieldtable.txt` file shipped with the example is shown in Listing 2-6.

**Listing 2-6 `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
```

**Note:** The `Fieldtable.txt` file 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 (these entries return errors if the test fails).

## Running the Application

You are now ready to run the application. You can test the application using the `ud32` utility or run the application as a C source client.

### Using the ud32 Utility

The `ud32` utility reads a tab-delimited text file and uses the information to construct an FML32 buffer. The ud32 client sends this buffer to the service defined in a ud32 input file called `ud32.in`.

You can create the `ud32.in` file using the steps described in the following procedure.

1. Open a DOS command-prompt window or start a UNIX session.
2. Edit the `ud32.in` file to specify the appropriate buffer information. The `ud32.in` file shipped with the example is shown in Listing 2-7.

**Listing 2-7 `ud32.in`**

```
STREET 2315 First North Street
CITY   San Jose
STATE  CA
SRVCNM II_addr1
```

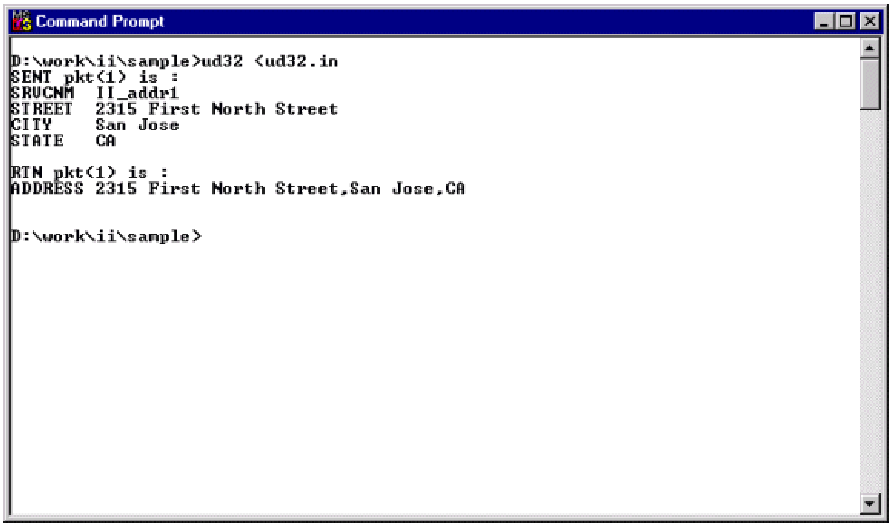
**Notes:** Make sure that there is a blank line at the end of the `ud32.in` file.

The value for the `SRVCNM` variable must match the value of the `NAME` variable in the `ii.cfg` file

3. Type **`tmboot -y`** and press **Enter** to start eLink platform and the iiServer.
4. Type **`ud32 <ud32.in`** and press **Enter**. `ud32` sends three input fields (`STREET`, `CITY`, and `STATE)` to the iiServer for reformatting and returns an output field (`ADDRESS).`

Figure 2-3 shows the invocation of ud32 and the accompanying system messages.

**Figure 2-3 Invocation of ud32**



```
D:\work\ii\sample>ud32 <ud32.in
SENT pkt(1) is :
SRVCNM  II_addr1
STREET  2315 First North Street
CITY    San Jose
STATE   CA

RTN pkt(1) is :
ADDRESS 2315 First North Street,San Jose,CA


D:\work\ii\sample>
```

## Running as the C Source Client

You can a create a C application to run as a source client. You can create this application and build it using the steps described in the following procedure.

1. Create a C application that sends an FML32 buffer to the iiServer for reformatting and returns an output field.

The `address.c` file shipped with the example is shown in Listing 2-8.

**Listing 2-8 `address.c`**

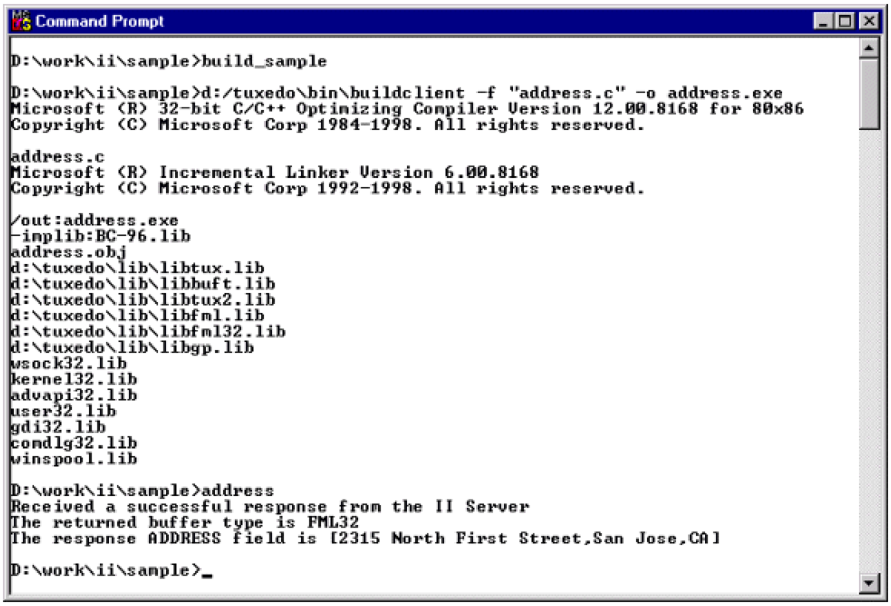
```
/*****************************************************************

 *  BEA eLink Information Integrator V1.0 - Sample Client
 *  Application
 *
 *              Copyright (c) 2000, BEA Systems, Inc.
 *                      All rights reserved
 *
 * This example constructs an FML32 buffer containing three fields
 * (STREET, CITY, STATE) and sends the buffer to the iiServer for
 * reformatting. The response buffer contains a single ADDRESS
 * field containing the concatenation of the three input fields.
 *
 *****************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>
#include <atmi.h>
#include <fml32.h>

#define MAXDIM(array)           (sizeof(array) / sizeof(array[0]))


static void         ErrorMsg(char* msg, ...);

int main(int argc, char** argv)

{
    auto     int i;
    auto     TPINIT*        initBuf;
    auto     FBFR32*        fmlBuf;
    auto     long           fmlLen = 0;
    auto     FLDID32        fldId;
    auto     char*          respVal;
    auto     char*          buf;
    auto     char           bufType[32] = "*Unknown*";
    static   struct
             {
                char*       inFldName;
                char*       inFldValue;

             }              inFields[] =
             {
```

```
            { "STREET",     "2315 North First Street"   },
            { "CITY",       "San Jose"                  },
            { "STATE",      "CA"                        }
        };

/* Allocate the TPINIT buffer */
initBuf = (TPINIT*) tpalloc("TPINIT", 0, 0);
if (NULL == initBuf)
{
    ErrorMsg("Error %d allocating TPINIT [%s]",
                tperrno, tpstrerror(tperrno));
}

/* Join Tuxedo */
if (tpinit(initBuf) < 1)
{
    ErrorMsg("Error %d on tpinit [%s]", tperrno,
    tpstrerror(tperrno));
}


/* Allocate a FML32 buffer for the request data */
fmlBuf = (FBFR32*) tpalloc("FML32", "", 0);
if (NULL == fmlBuf)
{
    ErrorMsg("Error %d allocating FML32 [%s]",
                tperrno, tpstrerror(tperrno));
}

/* Place input data values in the request FML32 buffer */
for (i = 0; i < MAXDIM(inFields); ++i)
{
    fldId = Fldid32(inFields[i].inFldName);
    if (BADFLDID == fldId)
        ErrorMsg("Unknown FML field name [%s]",
        inFields[i].inFldName);


    if (Fadd32(fmlBuf, fldId, inFields[i].inFldValue, 0) < 0)
    {
        ErrorMsg("Error %d adding field %s [%s]",
                    Ferror32, inFields[i].inFldName,
                    Fstrerror32(Ferror32));
    }
}


buf = (char*) fmlBuf;
```

```
    /* The value of the first tpcall argument, “II_addr1”, must
    match the value of the NAME parameter in the ii.cfg file.*/
    if (tpcall("II_addr1", buf, fmlLen,
                    &buf, &fmlLen, TPNOFLAGS) < 0)
    {
        ErrorMsg("Error %d on tpcall [%s]", tperrno,
         tpstrerror(tperrno));
    }


    fmlBuf = (FBFR32*) buf;
    fldId = Fldid32("ADDRESS");
    if (BADFLDID == fldId)
        ErrorMsg("Unknown FML field name [ADDRESS]");

    tptypes((char*) fmlBuf, bufType, NULL);
    printf("Received a successful response from the II Server\n");
    printf("The returned buffer type is %s\n", bufType);
    printf("The response ADDRESS field is [%s]\n",
            Fvals32(fmlBuf, fldId, 0));

    /* Free the FML32 buffer */
    tpfree((char*) fmlBuf);

    /* Leave Tuxedo */
    tpterm();
    return(EXIT_SUCCESS);
}


static void ErrorMsg(char* msg, ...)
{
    auto     va_list            ap;
    va_start(ap, msg);
    vprintf(msg, ap);
    printf("\n");
    va_end(ap);
    exit(EXIT_FAILURE);
}
```

2. Open a DOS command-prompt window.
3. Type **`sample\build_sample`** and press **Enter**. Figure 2-4 shows the build for `address.c` and the accompanying system messages.

**Figure 2-4 Building address.c**



```
D:\work\ii\sample>build_sample

D:\work\ii\sample>d:/tuxedo\bin\buildclient -f "address.c" -o address.exe
Microsoft (R) 32-bit C/C++ Optimizing Compiler Version 12.00.8168 for 80x86
Copyright (C) Microsoft Corp 1984-1998. All rights reserved.

address.c
Microsoft (R) Incremental Linker Version 6.00.8168
Copyright (C) Microsoft Corp 1992-1998. All rights reserved.

/out:address.exe
-implib:BC-96.lib
address.obj
d:\tuxedo\lib\libtux.lib
d:\tuxedo\lib\libbuft.lib
d:\tuxedo\lib\libtux2.lib
d:\tuxedo\lib\libfml.lib
d:\tuxedo\lib\libfml32.lib
d:\tuxedo\lib\libgp.lib
wsock32.lib
kernel32.lib
advapi32.lib
user32.lib
gdi32.lib
comdlg32.lib
winspool.lib

D:\work\ii\sample>address
Received a successful response from the II Server
The returned buffer type is FML32
The response ADDRESS field is [2315 North First Street,San Jose,CA]

D:\work\ii\sample>_
```

4. Type **`tmboot -y`** and press **Enter** to start eLink platform and the iiServer.
5. Type **`address`** and press **Enter** to run the application executable.
6. Type **`tmshutdown`** and press **Enter** to shutdown the Tuxedo eLink platform.

# 3 Building Format Definitions

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

- `MsgDefAdmin` importer utility—The importer utility is an easy-to-use command-line tool that allows you to store format definitions in the iiDatabase. You should use `MsgDefAdmin` to store formats whenever possible.

**Note:** You must use the Formatter interface when using output operations or mapping format fields.

- Formatter interface—The interface allows you to populate screens with format definition data and store the information in the iiDatabase.

This section describes the following topics:

- Using MsgDefAdmin to Build Definitions
- Starting Formatter
- Alternative Input and Output Formats

# Using MsgDefAdmin to Build Definitions

The `MsgDefAdmin` importer utility is an interpreter for four commands that let you add, retrieve, delete, and list message format definitions that are stored in the iiDatabase. `MsgDefAdmin` allows you to define message formats in an XML language called Message Definition Format Language (MFL). You can then import these textual descriptions into the iiDatabase. Message definitions that exist within the iiDatabase can also be listed, retrieved, or deleted using this utility.

## Creating and Storing MFL Documents

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`). `mfl.dtd` defines valid elements and attributes necessary to create message formats. The attributes in `mfl.dtd` are the same ones that you can define using Formatter. Refer to *“MFL Document Type Definition”* for a listing of `mfl.dtd` and an example MFL document.

As an example, suppose you are integrating a payroll application. This payroll application uses a message that has the following structure:

```
NUM_EMPLOYEES           4-byte Little Endian integer
EMPLOYEE_REC            Employee record
       EMPLOYEE_NAME    32 characters
       EMPLOYEE_SSN     11 characters
       EMPLOYEE_PAY     15 bytes of Packed Decimal
```

Listing 3-1 is an example instance of payroll data that contains data for one employee. In the example, binary data is represented with a `\x` prefix to indicate that it is hexadecimal. This is done for illustration purposes only.

**Listing 3-1 Instance of Payroll Data**

```
\x01\x00\x00\x00JohnDoe\xFF…......11122333300000000000000000000000
0063125F
```

The fields in this instance are defined as follows:

\x01\x00\x00\x00
: The NUM_EMPLOYEES field is one.

JohnDoe\xFF
: The EMPLOYEE_NAME field is delimited by `\xFF` byte.

111223333
: The EMPLOYEE_SSN field.

0000000000000000000000063125F
: The EMPLOYEE_PAY field is an unsigned packed field and has a value of $63,125.

You can create an MFL document for the payroll application by using the steps described in the following procedure.

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

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

3. Specify the message definition name in the `MessageFormat` element. In this example, name this message `PAYROLL_MSG` by setting the name attribute of the `MessageFormat` element.

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

4. Define the `NUM_EMPLOYEES` field by adding a `FieldFormat` element and setting the name and type attributes.

   ```
   <?xml version='1.0'?>
   <!DOCTYPE MessageFormat SYSTEM 'mfl.dtd'>
       <MessageFormat name='PAYROLL_MSG'>
           <FieldFormat name='NUM_EMPLOYEES'
            type='LittleEndian4'/>
   ```

5. Define the repeating `EMPLOYEE_REC` structure by adding a `StructFormat` element with the name attribute set to `EMPLOYEE_REC`. Because the structure repeats based on the `NUM_EMPLOYEES` field, set the `repeatField` attribute to point to the previously defined `NUM_EMPLOYEES` field.

```
<?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">
```

6. Add additional fields to `EMPLOYEE_REC` by defining `FieldFormat` elements. Set the type attribute according to each field's type.

   a. Specify a delimiter attribute of hexadecimal FF for the `EMPLOYEE_NAME` field. Also, specify the `EMPLOYEE_NAME` field with an `accessMode` of `Controlling`. This indicates that when the data is reformatted, each repetition of the repeating component (`EMPLOYEE_REC`) is detected by the presence or absence of an `EMPLOYEE_NAME` field.

   b. Specify a length of eleven characters for the `EMPLOYEE_SSN` field to indicate the exact size of the string.

   c. Specify a length of fifteen bytes for the `EMPLOYEE_PAY` field.

```
<?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'/>
```

7. Add the end tags for the `StructFormat` and `MessageFormat` elements.

```
<?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 iiDatabase.
9. Open a DOS command-prompt window.
10. Change to the directory where `MsgDefAdmin` is installed (for example, `C:\BEAII`)
11. 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 1.0    - - -
- - -      Message Definition Importer Tool      - - -

Enter choice:
1) Add Message Definition
2) Get Message Definition
3) Delete Message Definition
4) List Message Definitions
Q) Quit >
```

12. 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.

13. Type the name of the message file you created with your text editor and press **Enter**. `MsgDefAdmin` processes the file, stores the message definition in the iiDatabase, and returns a message that the definition was successfully created.

14. Type **`Q`** at the prompt and press **Enter** to quit `MsgDefAdmin`.

## Running MsgDefAdmin with command-line options

The `MsgDefAdmin` utility also supports execution in a non-interactive mode. Two command-line options are supported that allow you to import or export message definitions.

The `MsgDefAdmin` utility has the following command-line syntax:

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

where

`-i`

Denotes an import operation. `MsgDefAdmin` will import the message definitions in <*`XML filename`*> into the iiDatabase. 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 iiDatabase. 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 filename>`*
: Name of the XML file that contains or receives message definitions.

## 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.

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.

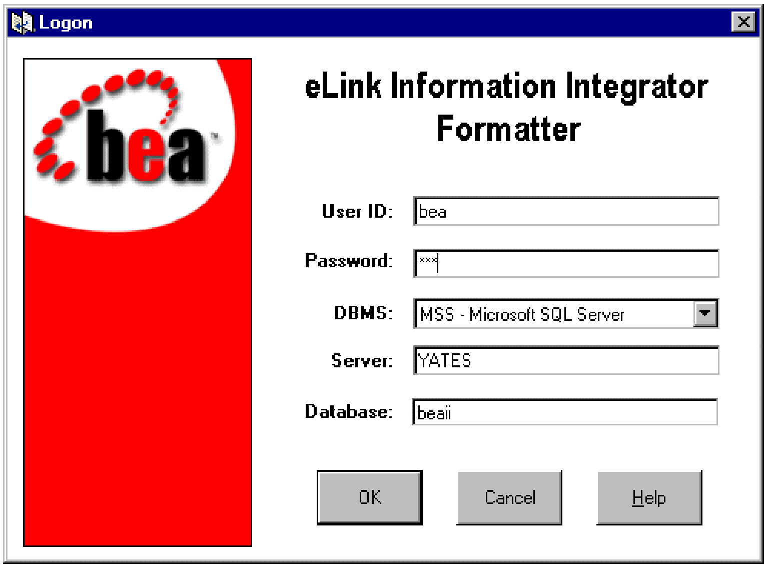
## Starting Formatter

You can launch the Formatter interface using the steps described in the following procedure.

1. Double-click the Formatter icon. The Formatter Logon dialog box appears as in Figure 3-1.

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

**Figure 3-1 Formatter Logon**



Logon

eLink Information Integrator Formatter

User ID: bea

Password: ***

DBMS: MSS - Microsoft SQL Server

Server: YATES

Database: beaii

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

3. Enter your password in the Password field and press **Tab**.

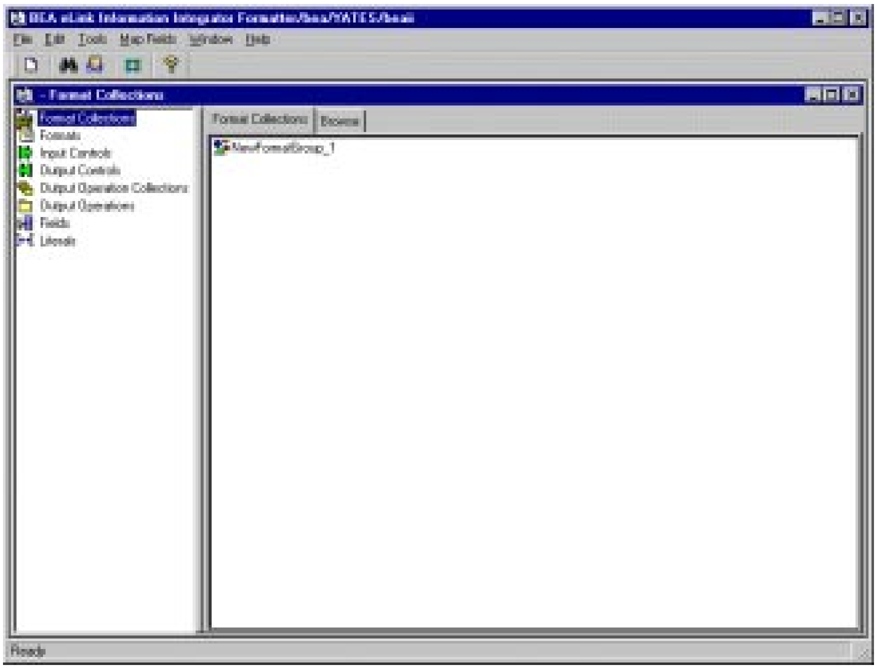
4. Select a database type from the DBMS drop-down list box. The Database field appears or disappears based on your selection.

   - 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 3-2.

**Figure 3-2 Formatter Main Window**



The 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 Formatter to Build Definitions

You can use the Formatter to edit or create message format definitions. If you want to create a message definition from scratch, follow the instructions in "Using MsgDefAdmin to Build Definitions." However, you must use Formatter to specify the mapping of data between message formats and any output operations performed on data items.

Using the Formatter, a message format definition is defined as a series of components. These components describe how to parse each piece of information in the message, how to output each piece of information, and how each piece of information relates to the message as a whole.

These components include:

- Literal—String of characters that exist in the message. For example, if a message always starts with a string of characters such as `START`, then this string is a literal.
- Field—Name for a piece of data.
- Input control—Describes how to parse an item of data. For example, an input control would describe that an item of data is an ASCII string that is delimited by a null character.
- Output control —Describes how to format output data. For example, an output control would describe that an item of data should be output as an EBCDIC string. Output operations associated with the output control could further describe that the EBCDIC string is right justified.
- Output operations—Describe an operation performed on the item of data when it is output. See the example given in the output control description.
- Flat format —Sequence of fields and controls that describe the format of a group of data items. For example, suppose three data items are a group. A flat format to describe how to parse this group would consist of three pairs of fields and input controls, one field and input control for each data item.
- Compound format—Sequence of flat formats.

To define a message format definition using Formatter, you must define the components of the message using a bottoms up strategy. A high-level approach is described in the following procedure.

1. Create the literals that are used in the message (delimiters, tags, and so forth)
2. Define the fields (data item names) of the message.
3. Define how to parse each data item of the message using an input control.
4. Define how to output each data item in an output control.
5. Define groups of data items by creating flat formats and adding fields and controls to these flat formats.
6. Define groups of flat formats by creating compound formats.

## Literals

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

- Delimiters
- Format terminators
- Repeating sequence terminators
- Default values
- Trim characters
- Prefixes and suffixes
- Pad characters
- Substitute values
- Comparison values
- Tag values
- Regular expressions

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; hex values that can include printable and non printable characters. Do not use single quotes in literal names. When literals are used as pad characters, only the first character is used.

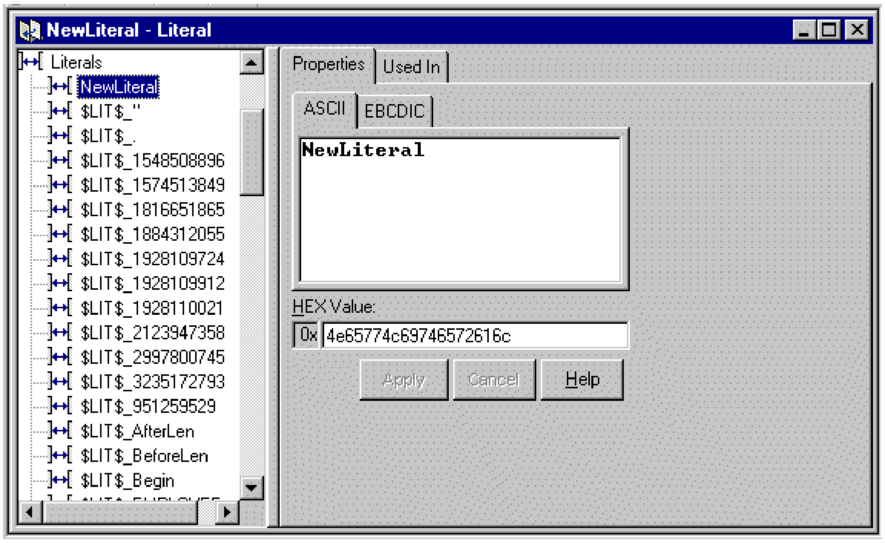
**Note:** The maximum character length of a literal is 127.

## Creating a Literal

You can create a literal using the steps described in the following procedure.

1. Select the Literals component category in the left pane of the Formatter window and right-click.
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 in Figure 3-3.

**Figure 3-3 Literal Property Sheet**



NewLiteral

HEX Value:

0x 4e65774c69746572616c

3. Type a unique name 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).

**Note:** You cannot drag and drop literals.

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

## Fields

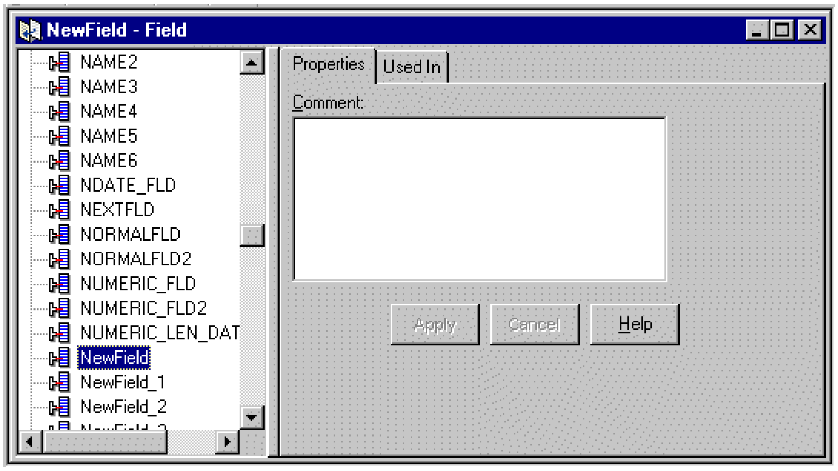
Fields are named items of message data and are the smallest possible container for information. In Formatter, the field name is the link between the input and output data. Each field can be used in multiple formats and associated with different input and output controls. To identify a field in a message,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

**Note:** Enter a unique name for each field. Field names can be up to 32 characters in length.

### Creating a Field

1. Select the Fields component category in the left pane of the Formatter window and right-click.
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. The cursor is positioned in the text box as in Figure 3-4.

**Figure 3-4 Fields Property Sheet**



3. Type a unique field name in the text box.

**Note:** The field name must be 32 characters or less. Field names with quotes may only have single or double quotes, not both. Field names with spaces or underscores must be enclosed by single or double quotes. A field with the name Field_1 could be used as “Field_1” or ‘Field_1’ (but not “Field_1’ or ‘Field_”).

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.

   The Comment field does not retain formatted text when imported or exported. Using carriage returns in the Comment field is not recommended.

**Note:** You can change the name of a field by double-clicking on the field name. Type the new name and press **Enter** to enter the name and position it in the list.

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

## 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.

After an input control is created, it can be saved as an output control. For more information, refer to “Saving an Input Control to an Output Control.”

### Creating an Input Control

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

1. Select the Input Controls component category in the left pane of the Formatter window and right-click.
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.
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. However, if using quotes, make sure they are not mismatched (for example, a single quote paired with a double quote).

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

5. Click the Control Type drop-down list and select an input control type. Input control types describe the type of field parsed by Formatter. The valid control-type values are described in Table 3-1.

**Table 3-1 Input Control Type Values**

| Input Control Type Value | Description |
|---|---|
| Data Only | Field has a data component only. Data is the value of the field. |
| Tag and Data | Field has a literal tag and data component (in this order). |
| Tag-Length and Data | Field has a tag, length, and data component (in this order). |
| Length and Data | Field has a length and data component (in this order). |
| Repetition Count | Field contains the count of a repeating component. |
| Literal | Field value is a literal. |
| Length-Tag and Data | Field has a length, tag, and data component (in this order). |
| Regular Expression | Compares input data to the value of the literal parse control using pattern matching. A regular expression requires an associated string data type and a literal value. |

6. Specify the values for the remaining fields on the Properties tab. Depending on the control type you selected, some fields are enabled or disabled. For example, you can define Data Only controls using the steps described in the following procedure.

   a. Select the input control type from the Type drop-down list in the Data section. The data types are described in “Data Types.”

   b. Select the input control termination type from the Termination drop-down list. The parse control termination type defines when Formatter should stop parsing for this field.

   c. Select one of the following options:

   - Enter the length of the data in the Length field if you selected Exact Length as your termination type.

- Enter the number of bytes in the Length field that Formatter skips before looking for the termination control if you selected Minimum Length + Delimiter as your termination type. You also need to specify a delimiter from the Delimiter drop-down list.
- Select a literal from the Delimiter drop-down list if you selected Delimiter as your termination type.

7. Select the Optional checkbox to make the input parse field optional. Optional means the parse continues even if parse parameters for the field are not found.
8. Click Apply to save your changes to the iiDatabase.

## 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.

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

1. Select the input control you want to save in the left pane of the Formatter window and right-click. 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.

## 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. Using the output control types of either `Input Field Value =` or `Input Field Exists` allows alternative output formatting. For more information on alternative output formatting, refer to “Alternative Input and Output Formats.”

The Output Control 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.

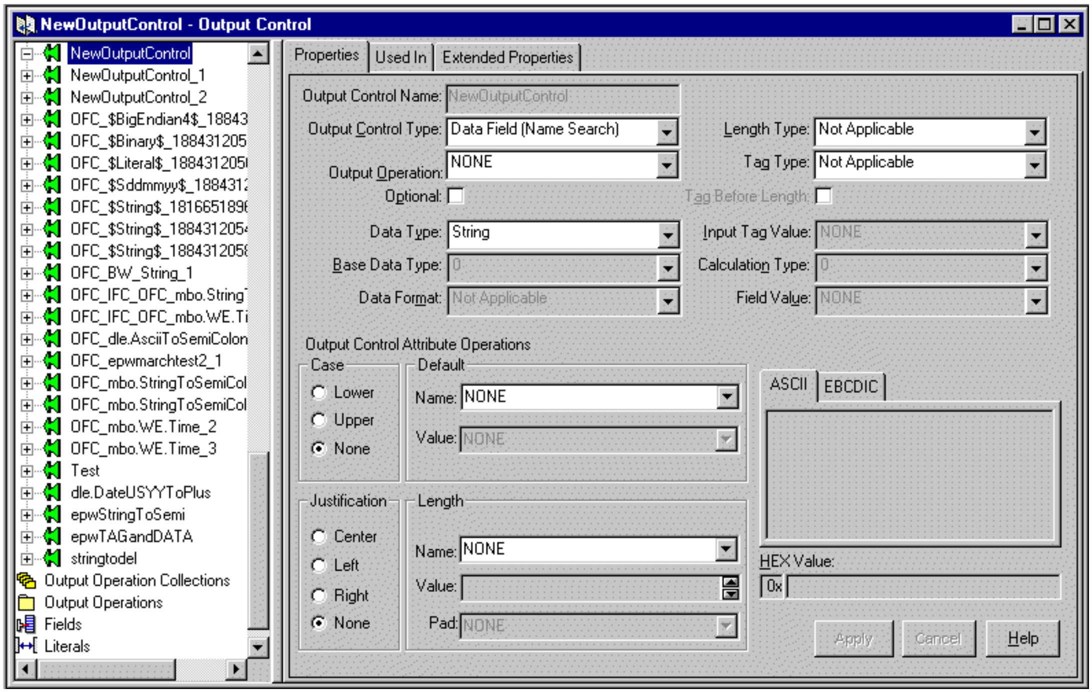
### Creating an Output Control

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

1. Select Output Controls from the Formatter window tree and click the right mouse button. A pop-up menu appears with the New menu item. The right pane displays the current list of output controls.

2. Select New to create a new output control entry in the left pane and open the Output Control property sheet in the right pane. The cursor is positioned in the output control text box as in Figure 3-5.

**Figure 3-5 Output Control Property Tab**


NewOutputControl - Output Control

NewOutputControl
NewOutputControl_1
NewOutputControl_2
OFC_$BigEndian4$_18843
OFC_$Binary$_188431205
OFC_$Literal$_188431205
OFC_$Sddmmyy$_188431
OFC_$String$_181665189
OFC_$String$_188431205
OFC_$String$_188431205
OFC_BW_String_1
OFC_IFC_OFC_mbo.String
OFC_IFC_OFC_mbo.WE.Ti
OFC_dle.AsciiToSemiColon
OFC_epwmarchtest2_1
OFC_mbo.StringToSemiCol
OFC_mbo.StringToSemiCol
OFC_mbo.WE.Time_2
OFC_mbo.WE.Time_3
Test
dle.DateUSYYToPlus
epwStringToSemi
epwTAGandDATA
stringtodel
Output Operation Collections
Output Operations
Fields
Literals

Properties | Used In | Extended Properties

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

Output Control Attribute Operations
Case: Lower Upper [✓ None]
Default: Name: NONE; Value: NONE
Justification: Center Left Right [✓ None]
Length: Name: NONE; Value: ; Pad: NONE
ASCII | EBCDIC
HEX Value: 0x

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. However, if using quotes, make sure the quotes are not mismatched (for example, a single quote with a double quote).

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.

5. Click the Control Type drop-down list and select an output control type. The valid control-type values are described in Table 3-2.

**Table 3-2 Output Control Types**

| Output Control Type | Description |
|---|---|
| Conditional Field | Mark field as output only if existence check field exists. |
| Data Field (Name Search) | Map the output field to the input field by field name. |
| Data Field (Tag Search) | Map the output field to the input field by tag value. |
| Existence Check Field | Mark field as an existence check field. |
| Input Field Exists | An input field exists. |
| Input Field Value = | The input field’s value equals a particular value. |
| Literal | Field value is a literal. |
| Rules Field | Output control is chosen based on rules evaluation. |

An output control defines how you want iiServer to output a message field. Output Control types identify how to map to the input data and the type of reformatting to use. The Data Field (Name Search) and Data Field (Tag Search) values map the output field to the input field by either name or tag. Other output control types define details concerning data mapping.

6. Specify the values for the remaining fields on the Properties tab. Depending on the control type you selected, some fields are enabled or disabled. For example, you can define Data Field (Name Search) output controls using the steps described in the following procedure.

   a. Select an operation to perform from the Output Operation drop-down list. The default is `None`. This value indicates that no operations are performed on the input data. In this case, the data is just copied.

   b. Select the Optional checkbox to make the output field optional. Optional means Formatter should continue with the output message if the field was not found in the input message or the input field was NULL.

      By default, the output control is mandatory. Mandatory means Formatter should fail with an error message if the field was not found in the input message or the input field was NULL, and there was no data-producing operation associated with the output control.

   c. Select a data type from the Data Type drop-down list if enabled.

   d. Choose from the following options:

   - Select a base data type from the Base Data Type drop-down list if you selected Date and Time, Date, Time, or Custom Date and Time as your data type.
   - Select a date/time format from the Data Format drop-down list if you selected Custom Date and Time as your data type.

7. Select the data type for the tag from the Tag Type drop-down list.

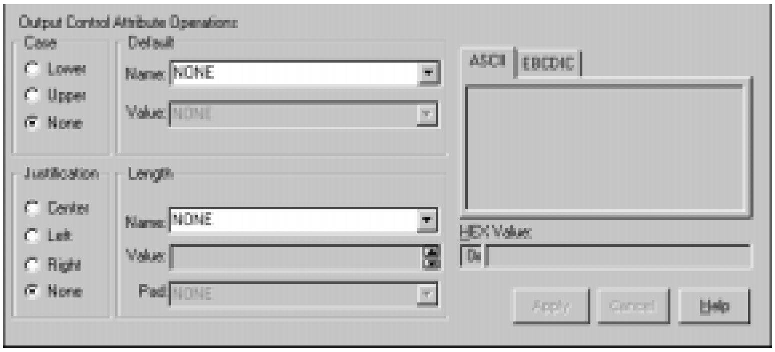
8. Click Apply to save your changes to the iiDatabase.

## Assigning and Defining Output Control Attribute Operations

The Output Control Attribute Operations section of the Properties tab allows you to assign existing or define new default and length output operations. You can also assign case and justification operations in this section. You cannot change existing operations from this section; however, you can change operations using the Extended Properties tab. See “Using the Extended Properties Tab” for more information.

The Output Control Attribute Operations section is shown in Figure 3-6.

**Figure 3-6 Output Control Attribute Operations Section**



Case: Lower ( ) Upper ( ) [✓ None]

Default — Name: NONE; Value: NONE

Justification: Center ( ) Left ( ) Right ( ) [✓ None]

Length — Name: NONE; Value:; Pad: NONE

HEX Value: 0x

You can define new default or length output operations by using the steps described in the following procedure.

1. Type a default output operations name in the Name field of the Default group. If there are matches for the characters you are typing, the matches scroll as you type to allow you to select an existing item. A non-matching name is added as a new name.
2. Click the Value field and type a new value or select an existing value.
3. Type the name of a length output operation in the Name field of the Length group to define a new length. If there are matches for the characters you are typing, the matches scroll as you type to allow you to select an existing item. A non-matching name is added as a new name.
4. Click the Value field and type a new value or select an existing value.
5. Click the Pad field and type a new value or select a value from the drop-down list of literals to assign a pad character.

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

You can assign existing output operations by using the steps described in the following procedure.

1. Select the appropriate case from the Case group.
2. Select an output operations name in the Name field of the Default group. If there are matches for the characters you are typing, the matches scroll as you type to allow you to select an existing item.
3. Type a name in the Name field of the Length group to define a new length. If there are matches for the characters you are typing, the matches scroll as you type to allow you to select an existing item.
4. Click the Pad field and type a new value or select a value from the drop-down list of literals to assign a pad character.
5. Click Apply to save your changes to the iiDatabase.

## 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. 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).

Selecting a case, justification, default, and length operation assigns them to a collection in the following order: default, case, length, and justify. This order is enforced by the initial assignment only.

The sections of the Extended Properties tab are labeled in Figure 3-7 and described in Table 3-3.

**Figure 3-7 Extended Properties Tab**

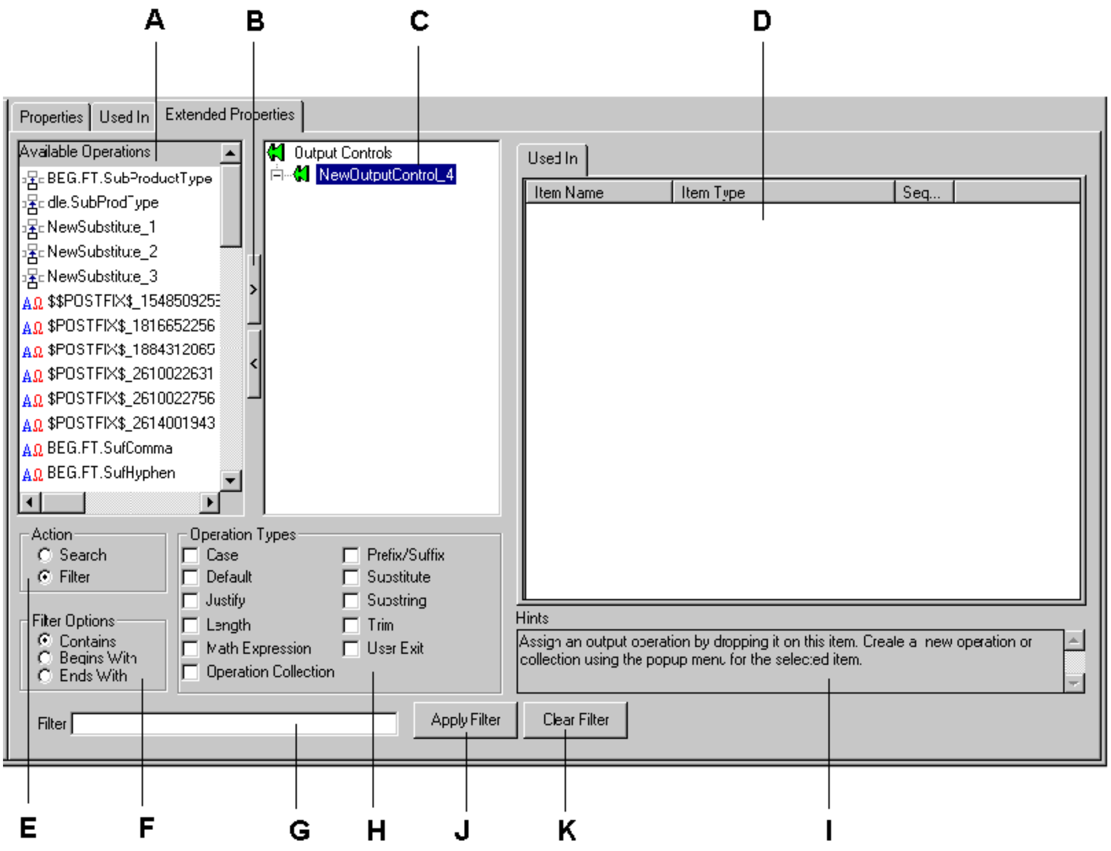


**Table 3-3 Extended Properties**

| 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 3-3 Extended Properties**

| Label | Section | Description |
|---|---|---|
| B | Add (>) and Remove (<) Buttons | The Add (< ) and Remove (>) buttons provide a fast way to perform specific tasks. The buttons behave differently, depending on what is selected.<br>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 >.<br>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 <.<br>To remove an output operation or output operation collection, select it and click <. A confirmation box appears asking you to confirm the removal. |
| C | Selected Operations | 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.<br>With a collection selected, you can reorder the sequence of output operations in the output operations collections tab.<br>Pop-up menus are available allowing you to create, remove, open, expand, and collapse items.<br>Creating a new item with an output control selected in the tree view assigns the new item to the control.<br>If an output operation or output operation collection is selected, you can access the pop-up menu and select Duplicate.<br>**Caution**: A change made to any assigned component of an output control affects all controls that use that component. |

**Table 3-3 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.<br>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. |

## 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.

# 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.

**Note:** You cannot create collections that contain themselves. If you see the Recursion icon, it is probable that something has corrupted the data in the iiDatabase.

You can define an output operation collection by using the steps described in the following procedure.

1. Select Output Operation Collection in the left pane of the Formatter window and right-click. A pop-up menu appears.

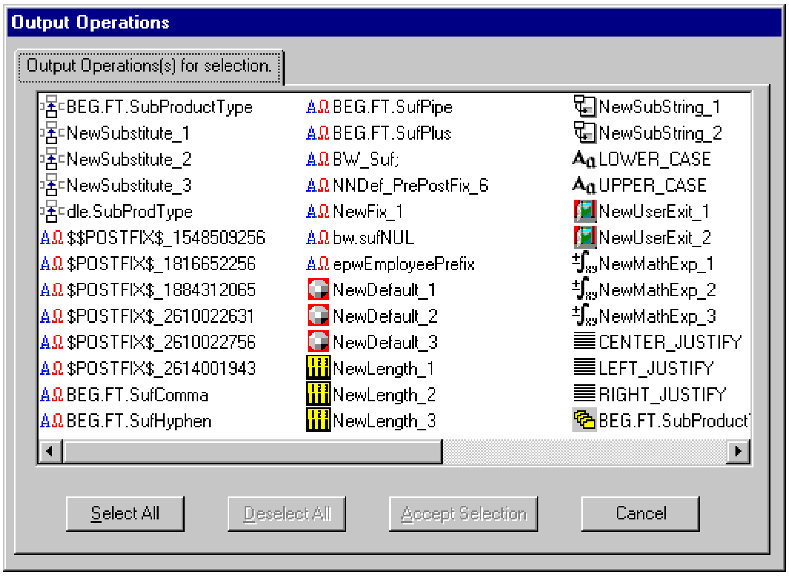
2. Select New. A new collection is added to the left pane.

   The cursor is positioned in the text box where you can type a unique Output Operation Collection name. Names must be 32 characters or less.

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

4. Right-click the new collection and choose Add Output Operations. The Output Operations window appears as in Figure 3-8.

**Figure 3-8 Output Operations**



BEG.FT.SubProductType
NewSubstitute_1
NewSubstitute_2
NewSubstitute_3
dle.SubProdType
$$POSTFIX$_1548509256
$POSTFIX$_1816652256
$POSTFIX$_1884312065
$POSTFIX$_2610022631
$POSTFIX$_2610022756
$POSTFIX$_2614001943
BEG.FT.SufComma
BEG.FT.SufHyphen
BEG.FT.SufPipe
BEG.FT.SufPlus
BW_Suf;
NNDef_PrePostFix_6
NewFix_1
bw.sufNUL
epwEmployeePrefix
NewDefault_1
NewDefault_2
NewDefault_3
NewLength_1
NewLength_2
NewLength_3
NewSubString_1
NewSubString_2
LOWER_CASE
UPPER_CASE
NewUserExit_1
NewUserExit_2
NewMathExp_1
NewMathExp_2
NewMathExp_3
CENTER_JUSTIFY
LEFT_JUSTIFY
RIGHT_JUSTIFY
BEG.FT.SubProduct

5. 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.

## Output Operations

Output operations provide the different actions that can be performed on an output field. Using operations, you can, for example, 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 operations are described in Table 3-4.

**Table 3-4 Output Operation Types**

| Output Operation Type | Description |
|---|---|
| Case | Case operations affect the case of the field data. The two defined case operations are `LOWER_CASE` and `UPPER_CASE`. |
| 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. |
| Prefix/Suffix | Prefix/Suffix operations enable you to attach literals to the beginning or end of the field data.<br>**Note**: The NULL value option forces the prefix/suffix value to be generated. |
| Justification | Justification operations justify field data to be left, center, or right within the length of the field. For pad characters, Justify operations must proceed a Length operation in a collection. To specify a pad character other than the default (a space), select the desired pad character in the associated Length operation. If Center Justify is specified, the data can be padded on both the left and right. The Justify and Length operations are related. |

**Table 3-4 Output Operation Types**

| Output Operation Type | Description |
|---|---|
| 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. Non-numeric data types are padded on the right; numeric data types are padded on the left. The Length and Justify operations are related. |
| Math Expression | Math Expression operations output a value resulting from an arithmetic expression. The expression can be built using arithmetic operators, constants, and input field values. For more information, see “Math Expression.” |
| 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, 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. For more information, see “Substitute.” |
| Substring | Substring operations enable you to extract a portion of an input string, defined by start byte position and length, and place it in the output field. |
| Trim | Trim operations remove a defined trim character to the right and left of the output data. |

## Default

You can define a default operation using the steps described in the following procedure.

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 iiDatabase.

## 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.

You can define a prefix/suffix operation using the steps described in the following procedure.

1. Select the Prefix/Suffix 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 Prefix/Suffix operation name.

**Note:** Prefix/Suffix Operation names have a 32-character maximum length.

3. Press **Enter** to finish defining the Prefix/Suffix 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 the type of Prefix/Suffix from the Prefix/Suffix 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. Check the Null Action box if the input field is not present in the input message.
8. Click Apply to save your changes to the iiDatabase.

## Length

You can define a Length operation by using the steps described in the following procedure.

1. Select the Length 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 the new, unique Length operation name.

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

3. Press **Enter** to finish defining the Length 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 pad character from the Pad Character drop-down list if the data length is less than the length specified for the output.
6. Use the Length field to specify the exact length for the output field.
7. Click Apply to save your changes to the iiDatabase.

## 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.

When input field names are used in math expressions, the math expression parser extracts the current value for the fields specified in the math expression. Alternatively, you can define a single math expression to use input fields that are currently mapped to the output for which the output math expression is applied. The string `$MAPPED_INFIELD` in a math expression is used in place of or in addition to field names. The value of the input field that is currently mapped to the output field replaces `$MAPPED_INFIELD`.

For example, suppose the following relationships exist.

- `inField1` has a value of 10 and maps to `OutField1`
- `inField2` has a value of 200 and maps to `OutField2`
- `inField3` has a value of 300 and maps to `OutField3`

If you apply the math expression `inField1 + $MAPPED_INFIELD` to `OutField2`, the result is 210. Likewise, if you apply the same math expression to `OutField3`, the result is 310.

The ability to substitute the value of the currently mapped input field is useful when the same calculation is applied to multiple input fields. Instead of defining a unique math expression with the same calculation for each field and explicitly naming fields in each math expression, define a single math expression containing `$MAPPED_INFIELD`. Use this math expression in all of the output controls associated with the output fields that are mapped to the input fields modifying the calculation.

**Note:** The math expression function ignores results and data from any previous operations.

For example, suppose you define an input message with fields `InF1`, `InF2`, and `InF3`, and an output message is defined with field `OutF1`. You could define a math expression as part of an output control associated with output message field

```
 OutF1 as: InF1 + InF2 * -InF3
```

This expression is evaluated as InF1 + (InF2 *(- InF3)) based on the precedence rules.

Other expression examples include:

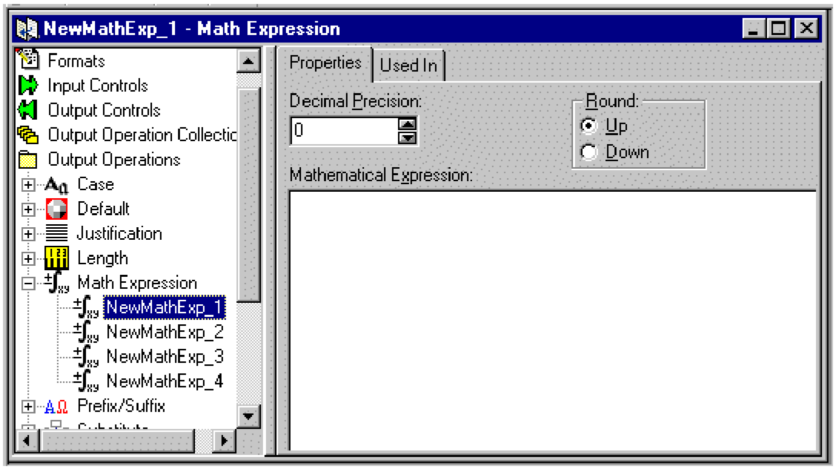
```
InF1 + -InF2
InF1 * 8
InF1 * 9.3
InF1 * -8
InF1 * -9.3
(InF1 + InF2) * 3/InF3
(InF1 * (InF2 + InF3) * 4)
```

You can define a math expression operation using the steps described in the following procedure.

1. Select the math expression output operation in the left pane of the Formatter window and right-click. A pop-up 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 math expression operation name.
3. Press `Enter`. The new entry is highlighted and moves to its alphabetical location in the list.
4. Click the Decimal Precision field as shown in Figure 3-9 and type the number of decimal points to which the expression result should be rounded.

**Figure 3-9 Math Expression Properties Tab**



Decimal Precision: 0

Round: [✓ Up] Down

5. Select whether the expression result should be rounded up or down in the Round group.
6. Type the expression in the Math Expression field. Fields defined in mathematical expressions must be in quotes (for example, "field1" = "field2"). A field name cannot contain a dash (for example, if the field name is abc-def, rename the name to abc_def).

7. Click Apply to save your changes to the iiDatabase.

**Note:** When multiplying fields in math expressions, multiply by 1.0 to avoid erroneous characters after the decimal. To set up an output control to handle a 6-digit ASCII number and insert a decimal, multiply the number by 1.000 where the number of zeroes after the decimal equals the amount of decimal precision required in the final number. Do not use terminating characters such as carriage returns or line feeds.

## Substitute

Substitute operations enable you to define a list of input strings to substitute and the output strings to replace them.

For example, each time Formatter receives the value `x` as input, you want to output the value as `xx`. You can change the value from the input of `x` to the output of `xx`.

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.

You can define a Substitute operation using the steps described in the following procedure.

1. Select Substitute operation in the left pane of the Formatter window and right-click. A pop-up 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 Substitute operation name.

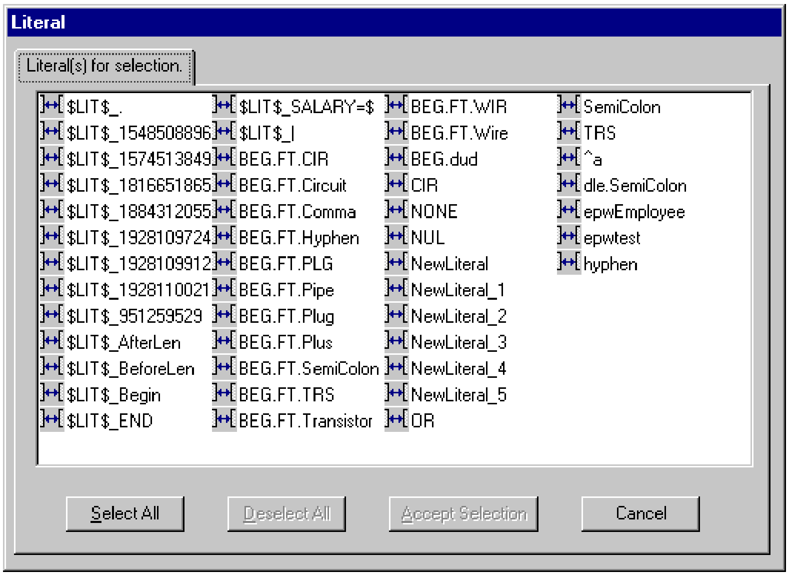
**Note:** Substitute operation names have a 32-character maximum length.

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

   The Substitute operation itself has no properties. You must add Substitute Items.
4. Select the substitute operation you want to add items to in the left pane of the Formatter window and right-click. A pop-up menu appears.

5. Select Add Substitute Items. The Literal dialog appears as in Figure 3-10.

**Figure 3-10 Literal**



| | | | |
|---|---|---|---|
| $LIT$_. | $LIT$_SALARY=$ | BEG.FT.WIR | SemiColon |
| $LIT$_1548508896 | $LIT$_I | BEG.FT.Wire | TRS |
| $LIT$_1574513849 | BEG.FT.CIR | BEG.dud | ^a |
| $LIT$_1816651865 | BEG.FT.Circuit | CIR | dle.SemiColon |
| $LIT$_1884312055 | BEG.FT.Comma | NONE | epwEmployee |
| $LIT$_1928109724 | BEG.FT.Hyphen | NUL | epwtest |
| $LIT$_1928109912 | BEG.FT.PLG | NewLiteral | hyphen |
| $LIT$_1928110021 | BEG.FT.Pipe | NewLiteral_1 | |
| $LIT$_951259529 | BEG.FT.Plug | NewLiteral_2 | |
| $LIT$_AfterLen | BEG.FT.Plus | NewLiteral_3 | |
| $LIT$_BeforeLen | BEG.FT.SemiColon | NewLiteral_4 | |
| $LIT$_Begin | BEG.FT.TRS | NewLiteral_5 | |
| $LIT$_END | BEG.FT.Transistor | OR | |

6. Select the literals you want to use as input strings and click Accept Selection.

7. Click the new substitute operation in the left pane. The corresponding Properties tab appears in the right pane.

8. Select for each substitute item:

   - A data type from the Output Data Type drop-down list.
   - A literal from the Output Value drop-down list. This value is substituted for the value specified in the Input Value drop-down list.

9. Click Apply to save your changes to the iiDatabase.

10. Repeat steps 8 and 9 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.

You can define a substring operation using the steps described in the following procedure.

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

   The cursor is positioned in the text box where you can type the new, unique substring name.

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

3. Press **Enter** to finish defining the substring. The new entry moves to its alphabetical location in the list and is highlighted.
4. Select the Properties tab in the right pane.
5. Specify the starting position for the substring in the Substring Start field.
6. Specify the length of the substring in the Substring Length field.
7. Specify the literal to use as a padding 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, padding the data on the right.
8. Click Apply to save your changes to the iiDatabase.

## Trim

You can define a Trim operation using the steps described in the following procedure.

1. Select the Trim 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 Trim operation name.

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

3. Press **Enter** to finish defining the Trim 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 to trim from the Trim Value drop-down list.
6. Select where you want to remove the Trim Value from the output field.
7. Click Apply to save your changes to the iiDatabase.

## Formats

Use the Formats function to build both input and output formats. The formats can be either flat or compound.

The component formats of a compound format are either optional or mandatory. An optional component can have missing components, and the parse or reformat will still succeed. A mandatory component must exist, and all its mandatory components must also exist, or the parse or reformat will fail. The component formats are mandatory by default.

Before creating flat or compound formats, define their component parts. The component parts for each format are described in Table 3-5.

**Table 3-5 Format Component Parts**

| Format | Components |
|---|---|
| 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 Format

You can create a new format using the steps described in the following procedure.

1. Select Formats from the tree in the Formatter window and click the right mouse button. A pop-up menu appears.
2. Select New. Hold down the right mouse button while navigating to the submenu to the right. This opens the submenu. Choose one of the following:
   - Select Compound to create a compound format, and select Input or Output to create an input format or output format respectively. A new format component is added to the left-hand pane.
   - Select Flat to create a flat format, and select Input or Output to create an input format or output format respectively. A new format component is added to the left-hand pane.
3. 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.
4. Click the Properties tab if it is not already selected. Depending on which format you selected (compound versus flat and input versus output), different fields will display on the Properties tab. For example, you can create a flat input format using the steps described in the following procedure.

   a. Select the Random Field Order checkbox to specify the component format order as Random (fields can appear in any order). The default order is ordinal (the fields appear in the specified order in the input message).

   b. Click the Format Termination field to open the drop-down list and select a termination type. Format Termination types are described in Table 3-6.

**Table 3-6 Format Termination Types**

| Format Termination Value | Description |
|---|---|
| Not Applicable | No data termination. Read to end of message. |
| Delimiter | The format is terminated by a delimiter. |
| Exact Length | The format has a fixed length. |
| White Space Delimited | The format is terminated by a white space. |

**Table 3-6 Format Termination Types**

| Format Termination Value | Description |
|---|---|
| Minimum Length + Delimiter | Parse a minimum number of characters and then look for a delimiter. |
| Minimum Length + White Space | Parse a minimum number of characters and then look for a white space. |

If the termination type is Delimiter or Minimum Length + Delimiter, select a literal from the Delimiter drop-down list.

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.

c. Select the format in the tree where you want to add fields and right-click. The pop-up menu appears.

d. Select Add Field Components from the pop-up menu. The Field dialog appears displaying a list of the fields.

e. Select the fields that you want to add and click Accept Selection. The fields are added to the format tree.

**Notes:** When selecting components for the format, make sure each component has associated input controls and fields on its property sheets.

To select fields used in another format, click the Format Filter drop-down list and select a format. The field list changes to list only fields that appear in the selected format. To return to the complete list of fields, click the Format Filter drop-down list and select Not Applicable.

f. Click Apply to save your changes to the iiDatabase.

### 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.

## Alternative Input and Output Formats

Alternative formats are a special form of compound format 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.

Exactly one of the alternatives must apply or the entire alternative compound 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.

## Optional Components and Fields

Optional components and fields are each characterized as optional within the context of a compound object.

For example, a compound format composed of component formats may have a mix of mandatory and optional component formats. A format may also have 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, Formatter continues to the next component. All component formats in an Alternative Compound should 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, then the parse for the entire alternative input format fails.

## Alternative Input Formats and Parsing

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.

## 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.

For information on using alternative formats in Formatter, refer to “Output Controls.”

**Note:** A component is not created if a mandatory component or field for the output format is not present in the incoming message.

## 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 with Alternative Component Example

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 Format Example

Creating an output format is conceptually clearer. You define a set of alternatives, then define an alternative compound having all of the alternatives as components, sequenced in order of desirability. As with input formats, it may be appropriate to order alternatives from specific to general. (This is not always the case. Many times, order is irrelevant on output because of mutually exclusive mandatory fields; there is only one format that applies.)

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.

# 4 Defining Rules

Rules are used by the iiServer to evaluate the contents of a message and perform actions based on the evaluation results. You use the Rules interface to define rules and associated subscriptions.

Rules are defined within an application group/message type pair. Rules are uniquely identified by the application group/message type/rule name triplet.

Rule application groups allow you to easily maintain rules associated with business needs. An application group is a logical grouping used to organize rules. For example, a company can split rules into groups by projects or split projects into logical subgroups.

Each application group can contain several message types, and a message type can be in more than one application group. You define message types through the Rules definition mechanism. A message type defines the layout of a string of data. When using Formatter, the message type is the same as the input format name.

Each rule has evaluation criteria (called an expression) that consists of fields from the message and associated Rules operators linked together with Boolean operators. You define field names through the Formatter definition mechanism. Fields can be compared against constant data or other fields within a message.

A rule can have multiple subscriptions, and each subscription can have multiple actions. A subscription is created in the Subscription list, then assigned to one or more rules within an application group/message type.

A subscription is defined by an application group/message type in the same way as a rule. Subscriptions are groupings of actions processed when a rule evaluates true. A subscription cannot be executed if it is not assigned to a rule.

This section discusses the following topics:

- Starting Rules
- Building Rules

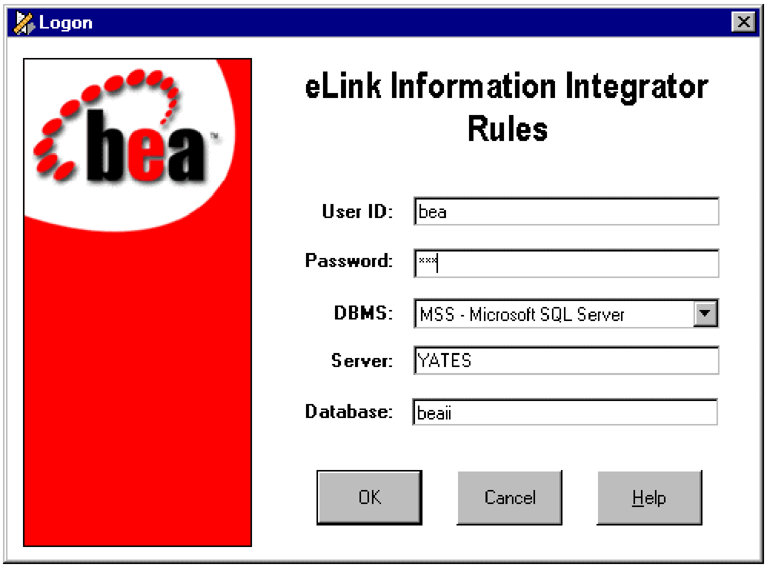
## Starting Rules

You can access the Rules application by using the steps described in the following procedure.

1. Double-click the Rules icon. The Rules Logon dialog box 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 Rules Logon**



Logon

eLink Information Integrator Rules

User ID: bea

Password: ***

DBMS: MSS - Microsoft SQL Server

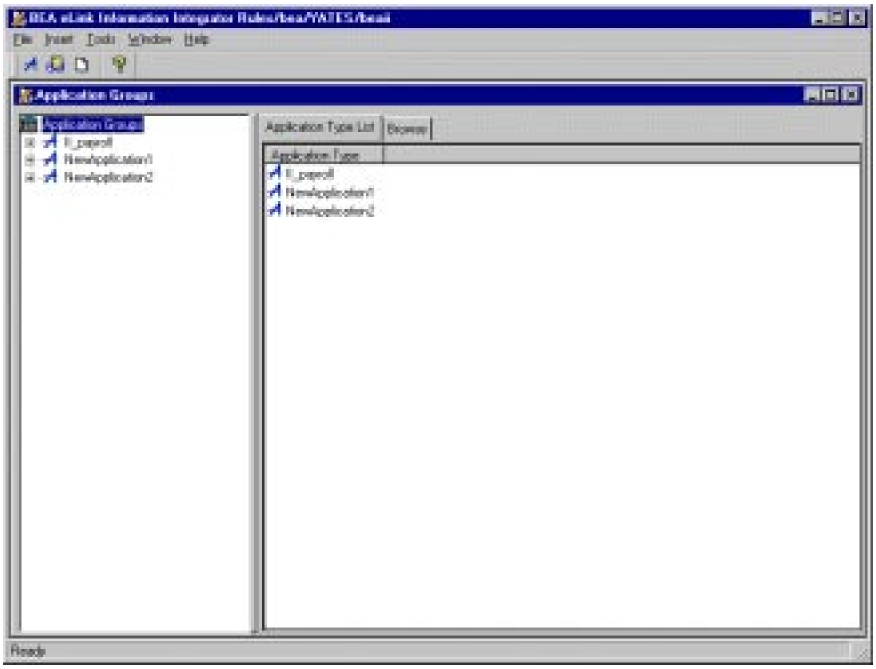
Server: YATES

Database: beaii

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 4-2.

**Figure 4-2 Rules Main Window**



The Rules main window is divided into two panes. Rules components are displayed in an hierarchical or tree-structured organization in the left pane, with Application Groups at the top level of the hierarchy. The right pane contains tabs that are associated with the selected object.

## Building Rules

This section describes how to build the following Rules components:

- Application groups
- Message types
- Rules
- Expressions
- Subscriptions
- Actions
- Conditional branching

## 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.

### Adding an Application Group

You can add an application group using the steps described in the following procedure.

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.

If you are using a case-insensitive database, you cannot use the same name with a change in case to identify components. For example, you cannot name one application group D1 and another d1. In a case-insensitive environment, make each item unique using something other than case differences.

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 “Adding a Message Type.”

## Copying an Application Group

You can copy an application group by using the steps in the following procedure.

1. Select the application group you want to duplicate.
2. Hold down the right mouse button to activate the pop-up menu and select Duplicate.

   The cursor is positioned in the application group text box. Type a unique application group name.
3. Press **Enter** to copy the application group and its components (messages, rules, and subscriptions) to the new application group name.

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

## Deleting an Application Group

You can delete an application group by using the steps described in the following procedure.

1. Select the application group you want to delete.
2. Hold down the right mouse button to activate the pop-up menu and select Delete. The delete confirmation box appears.
3. Click OK to delete the application group. The Delete box closes, and the application group and its components (messages, rules, and subscriptions) are deleted.

## Message Types

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

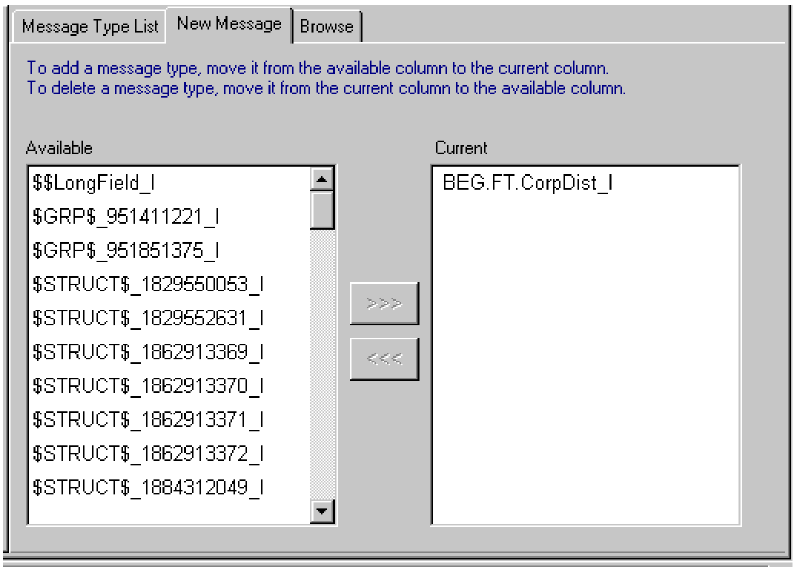
An application group should contain at least one message type. For the procedure to add an Application Group, see “Adding an Application Group.”

### Adding a Message Type

You can add a message type by using the steps described in the following procedure.

1. Select the application group that you want to add a message type to. 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 4-3.

**Figure 4-3 New Message Property Sheet**



To add a message type, move it from the available column to the current column.
To delete a message type, move it from the current column to the available column.

| Available | Current |
|---|---|
| $$LongField_I | BEG.FT.CorpDist_I |
| $GRP$_951411221_I | |
| $GRP$_951851375_I | |
| $STRUCT$_1829550053_I | |
| $STRUCT$_1829552631_I | |
| $STRUCT$_1862913369_I | |
| $STRUCT$_1862913370_I | |
| $STRUCT$_1862913371_I | |
| $STRUCT$_1862913372_I | |
| $STRUCT$_1884312049_I | |

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 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. Select the message type from the Current list box and double-click to delete a message type from the selected application group.

## Copying a Message Type

You can copy a message type to a different application group using the steps described in the following procedure.

1. Select the message type to duplicate in the left-hand pane.
2. Drag the message type to the application group you want to add it to in the left-hand pane.

## Deleting a Message Type

You can delete a message type using the steps described in the following procedure.

1. Select the message type you want to delete.
2. Hold down the right mouse button to activate the pop-up menu and select Delete. The Delete box appears.
3. Click OK to delete the message typ**e**. The Delete box closes and the message type is deleted.

## Rules

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

### Adding a Rule

You can add a rule using the steps described in the following procedure.

1. Select the message type that you want to define a rule for from the Rules window tree and click the right mouse button. 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.

   For the Rules engine to correctly process a rule, you must define the rule’s expression and subscription. 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.”

### Deleting a Rule

When you delete a rule, the expression and links to subscriptions belonging to the rule are also deleted. The subscriptions are not deleted.

You can delete a rule by using the steps described in the following procedure.

1. Select the rule that you want to delete.
2. Hold down the right mouse button to activate the pop-up menu and select Delete. The Delete box appears.
3. Click OK to delete the rule. The Delete box closes, and the rule is deleted.

## Copying a Rule

The Duplicate function lets you copy the selected rule, which includes the rule's associated expression and its links to subscriptions. The new rule is owned by the current user who has update permission.

You can duplicate a rule by using the steps described in the following procedure.

1. Select the rule that you want to duplicate.
2. Hold down the right mouse button to activate the pop-up menu and select Duplicate.

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

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

3. Press **Enter** to save the name.

   To change the rule's expressions, select the Expression tab and make the changes. For more information, refer to "Expressions."

   To change a rule's subscriptions, select the Subscription tab and make the changes. For more information, refer to "Subscriptions."

## Enabling a Rule

You can enable a disabled rule by using the steps described in the following procedure.

1. Select the rule that you want to enable.
2. Hold down the right mouse button to activate the pop-up menu.
3. Select Enable. The fields in the Expression tab are now active.

## Disabling a Rule

You can disable a rule by using the steps described in the following procedure.

1. Select the rule that you want to disable.
2. Hold down the right mouse button to activate the pop-up menu.
3. Select Disable. The fields in the Expression tab are now inactive.

## Expressions

The Expression tab is used to create, modify, or delete an expression for a selected rule.

The following tabs appear within the Expression tab:

- Expression Components—Combinations of expression strings used to automate typing for a user
- Field List—List of fields that appear in the format
- Operators—Comparison operator choices
- Values—Masked values for integers, strings, and so forth
- 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 |, expressions, and parentheses to control the order of evaluation. You can use | to explicitly direct what Boolean operations do together.

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.

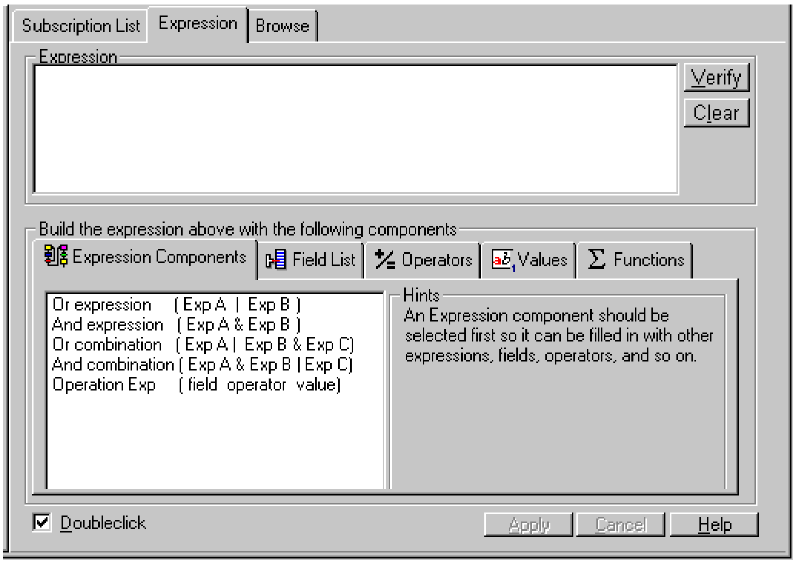
## Creating or Modifying an Expression

You can create or modify an expression using the steps described in the following procedure.

1. Select the appropriate rule.

2. Select the Expression tab. The tab appears as in Figure 4-4.

**Figure 4-4 Expression Tab**



```
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)
```

An Expression component should be selected first so it can be filled in with other expressions, fields, operators, and so on.

[✓] Doubleclick

3. Select the Expression Components tab. Select (by clicking or double-clicking) an Expression Component in the Rules tree to display it in the Expression tab.

   Use the Doubleclick checkbox to modify mouse action to a click or a double-click when selecting expression components.

4. Use the Field List, Operators, Values, and Functions tabs to add components to your expression.

**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, but AND will be changed to &, and OR will be changed to |.

You can type the expression directly into the Expression box without using the component tabs. You can modify the expression by right clicking within the Expression window to access the text edit menu. Use the following text edit functions to modify your expression: Undo your last change, Cut, Copy, Paste, Delete, and Select All of the text within the Expression window.

You can create an expression that defaults to true to define a rule that will always be processed. To do this, type `TRUE` in the Expression box. In this case, subscriptions associated with this rule will also always be processed.

5. Click Verify to validate the final expression. This tests the expression but does not save it to the iiDatabase.
6. Click Apply to either create or modify an expression.

   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 4-1.

**Table 4-1 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 Parens | (A \| ((B & (C)) \| D)) | (F1 STRING= Acct \| ((F2 INT= 100 & (F3 INT= 150)) \| F4 INT= 200)) |

### Clearing an Expression

You can clear an expression for a selected rule using the steps described in the following procedure.

1. Select the rule that contains the expression you want to clear. The expression is displayed in the Expression box.
2. Click Clear, and enter a new expression.

**Note:** Rules will not allow you to overwrite an existing expression with a blank expression.

3. Click Apply to update the iiDatabase.

## Subscriptions

After a rule is created, subscriptions that contain actions should be added to the rule's Subscription List. Each application/message group has its own Subscription List. You can drag and drop one or more subscriptions to any rule from the Subscription List or drag and drop into the Subscription List tab of a rule.

A subscription describes the actions that are performed if the rule's expressions pass evaluation. A subscription allows you to define message destination IDs, receiver locations, message formats, and any processes initiated upon message delivery.

A subscription can only be assigned to a rule within the same application group/message type.

**Note:** A subscription can be assigned to several rules if the rules are in the same application group and message type.

### Adding a Subscription

You can add a subscription by using the steps described in the following procedure.

1. Select the Subscription List for the Application Group/Message Type that you want to add a subscription to from the Rules window tree and click the right mouse button. 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.

   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.

   Each subscription should have at least one action. For the procedure to add an action, see “Adding an Action.”

## Duplicating a Subscription

You can copy a subscription by using the steps described in the following procedure.

1. Select the subscription that you want to duplicate from the Subscription List.
2. Hold down the right mouse button to activate the pop-up menu and select Duplicate. A text entry box appears at the top of the rules list.
3. Type a unique subscription name in the text box and press **Enter** to save a copy of this subscription with a new name.

## Deleting a Subscription

A subscription can only be deleted if it is not linked to any rules from the Subscription List.

If you own the subscription and the subscription is not linked to any rules, the subscription is deleted.

You can delete a subscription by using the steps described in the following procedure.

1. Select the subscription that you want to delete from the Subscription List.
2. Click the right mouse button to activate the pop-up menu and select Delete. The Delete box appears.
3. Click OK to delete the subscription. The delete box closes, and the subscription is deleted.

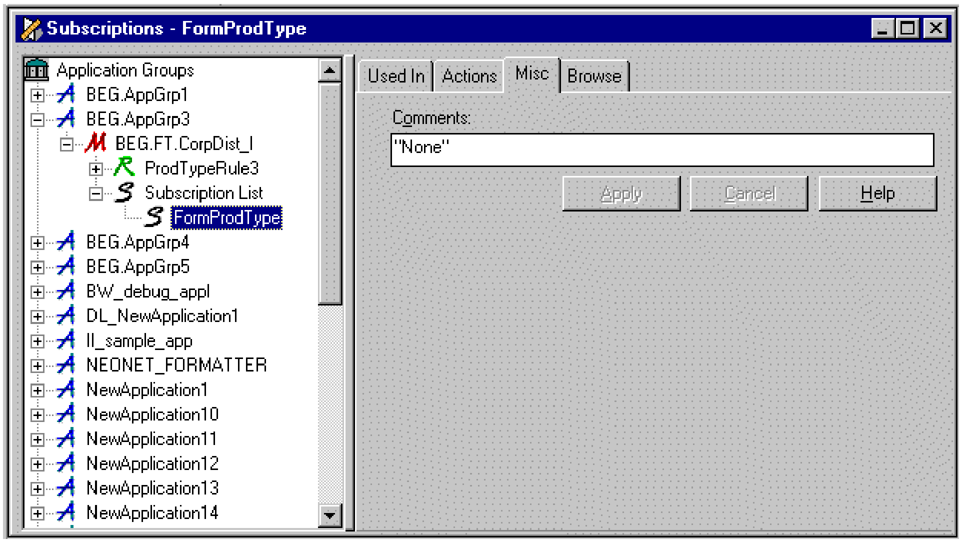
## Adding a Comment to a Subscription

You can associate a comment with a subscription by using the steps described in the following procedure. A comment must be 64 characters or less.

1. Select the subscription for which you want to add a comment from the Subscription List.

2. Select the Misc tab. A Comments text entry box appears containing the word “None” (if no comments were previously added) as in Figure 4-5.

**Figure 4-5 Misc Tab**



Comments: "None"

3. Select and delete the word None and type your comment.

4. Click Apply to save your comment.

## Removing a Subscription

Use the Remove function to remove a subscription from a rule. You cannot remove a subscription from the Subscription List, only from a rule. After you remove the subscription from the rule, the subscription will still be in the Subscription List.

You can remove a subscription by using the steps described in the following procedure.

1. Select the subscription you want to remove from the specific rule.
2. Click the right mouse button to activate the pop-up menu and select Remove. The Remove box appears.
3. Click OK to remove the subscription. The remove box closes, and the subscription is removed from the rule.

## Actions

Actions hold subscription instructions. The Actions affecting subscriptions include:

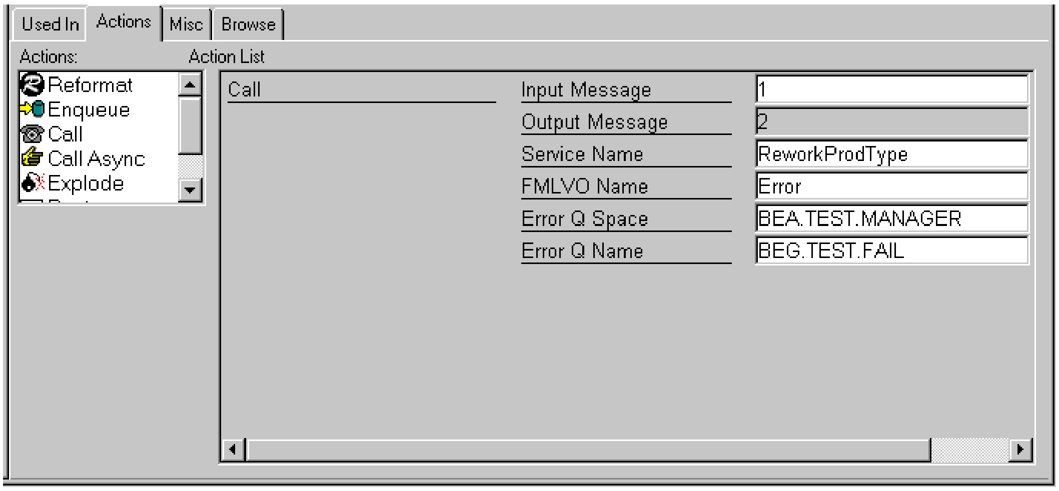
- Reformat
- Enqueue
- Post
- Call
- Call Async
- Explode
- Return

## Adding an Action

You can add an action to a subscription by using the steps described in the following procedure.

1. Select the subscription in which you want to add an action. The Actions tab appears as in Figure 4-6.

**Figure 4-6 Actions Tab**



| | | |
|---|---|---|
| Call | Input Message | 1 |
| | Output Message | 2 |
| | Service Name | ReworkProdType |
| | FMLVO Name | Error |
| | Error Q Space | BEA.TEST.MANAGER |
| | Error Q Name | BEG.TEST.FAIL |

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.

   The parameters you enter in the Action Values box should be based on the type of action selected. The action types are described in Table 4-2.

**Table 4-2 Subscription Actions**

| Action | Description |
|---|---|
| Reformat | Identifies the input and output message formats used when reformatting a message. In addition, the names of the input and output messages are assigned. These names may be used in other actions to identify the buffer to be used.<br><br>**Note:** The FMLVO Name parameter is only used when the output type is set to FMLVO. |
| 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. |
| Post | Permits entry of all the information required to post a typed buffer to the Tuxedo event broker. |
| Call | Obtains the necessary information to send a typed buffer to another Tuxedo service and accept a reply from that service.<br><br>**Note:** The FMLVO Name, Error Q Space, and Error Q Name parameters are optional. |
| Call Async | Gathers information required to send a typed buffer to another Tuxedo service.<br><br>**Note:** No reply is returned from the called service. |
| 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:<br>■ Enqueue—The enqueue action is performed for each matching subcomponent.<br>■ Post—The post action is performed for each matching subcomponent.<br>■ Call Async—The call async action is performed for each matching subcomponent.<br><br>**Note:** The Reformat, Call, Explode, and Return actions cannot be used after an Explode action. |
| Return | Identifies the buffer that is returned to the client. The return action must be the last action defined. |

3. Click Apply to save the actions.

### Deleting an Action

You can delete an action by using the steps described in the following procedure.

1. Select the action that you want to delete in the Action List.
2. Position the cursor on the action and press the right mouse button to open the pop-up menu.
3. Select Delete Action from the pop-up menu. The action is deleted from the Actions tab.
4. Click Apply to delete the action.

## Conditional Branching

If Rules is started from the Formatter interface, Rules opens in Conditional Branching mode. The only application group that appears in the Rules tree is II_FORMATTER.

You can add message types, rules, arguments, subscriptions, and actions to the II_FORMATTER application group using the procedures described earlier in this section. However, there is a difference in the Actions tab.

The Actions tab only contains the Rules Field action. When adding or inserting Rules Field actions, three options are necessary:

- `FIELD_NAME`
- `OUTPUT_FORMAT`
- `FORMAT_CONTROL`

See “Adding an Action” for details on adding and changing actions and options.

# 5 Testing Message Parse and Reformat

The Tester interface allows you to parse and reformat messages as a validation test. Tester uses control tables (built using `MsgDefAdmin` or Formatter) to recognize and parse input and output messages.

This section describes the following topics:

- Starting Tester
- Parsing and Reformatting Messages

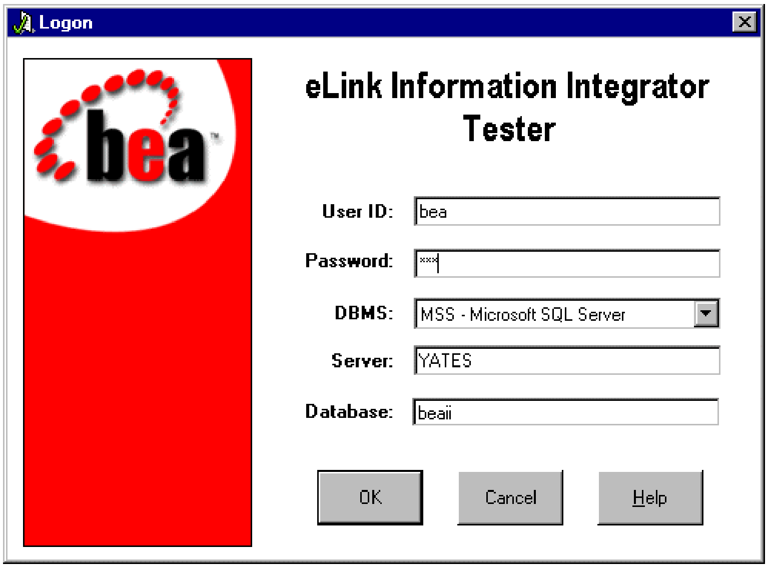
## Starting Tester

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**



Logon

eLink Information Integrator
Tester

User ID: bea

Password: ***

DBMS: MSS - Microsoft SQL Server

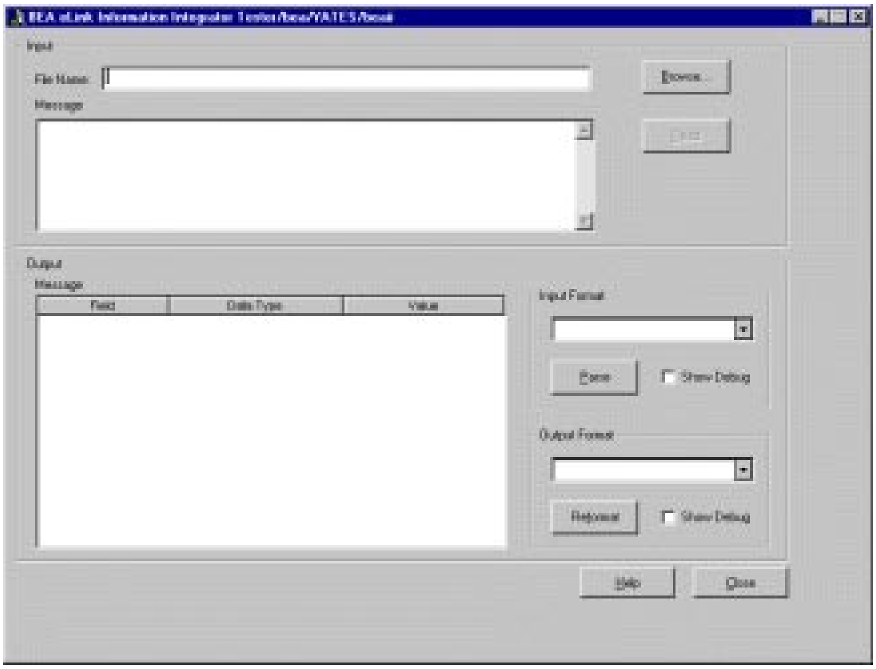
Server: YATES

Database: beaii

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 Tester main window appears as in Figure 5-2.

**Figure 5-2   Tester Main Window**



## Parsing and Reformatting Messages

You can parse and reformat messages by using the steps described in the following procedure.

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.

# A Data Types

Data types define the type of field in input and output formats.

## Understanding Data Types

The valid data types are described in Table A-1.

**Table A-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 A-1 Data Types**

| Data Type | Description |
|---|---|
| Binary Data | The Binary data type is used to parse any value and transform that value to an ASCII representation of the value internally in the Formatter. The internal representation takes each byte of the input value and converts it to a readable form. An example of this is parsing a byte whose value is hexadecimal 0x9C and transforming that to the internal ASCII representation of 9C, which is the hexadecimal value 0x3943. If this value is used in an output format with the output control's data type set to String, the value placed in the message is ASCII 0x9C. If this value is again placed in an output message with the data type Binary, the ASCII value is not printable and occupies one byte with the value of hexadecimal 0x9C.<br><br>Conversely, an input value of ASCII 3B7A parsed with the String data type can be output using the Binary data type. The output value is hexadecimal 0x37BA and occupies 2 bytes in the output message. Valid characters that can be converted to Binary from the String data type are 0 through 9 and A through F. All other characters are invalid. |
| EBCDIC Data | A string of characters encoded using the EBCDIC (Extended Binary Coded Decimal Interchange Code) encoding used on larger IBM computers. During a reformat from EBCDIC to ASCII, if a character being converted is not in the EBCDIC character set, the conversion results in a space hexadecimal 20. |
| IBM Packed Integer | Data type on larger IBM computers used to represent integers in compact form. Each byte represents two decimal digits, one in each nibble of the byte. The final nibble is always a hexadecimal F. For example, the number 1234 is stored as a 3-byte value: 01 23 4F (the number pairs show the hexadecimal values of the nibbles of each byte). The number 12345 is stored as a 3-byte value: 12 34 5F. There is no accounting for the sign of a number, so all numbers are assumed to be positive. |

**Table A-1 Data Types**

| Data Type | Description |
|---|---|
| IBM Signed Packed Integer | Data type on larger IBM computers used to represent integers in compact form. This data type takes into account the sign (positive or negative) of a number. Each byte represents two decimal digits, one in each nibble of the byte. The final nibble is a hexadecimal C if the number is positive and a hexadecimal D if the number is negative.<br>An example of how to generate a default value for an IBM Packed Integer is:<br>Data Type: IBM Signed Packed Decimal<br>Default Value: -12345 (default value in ASCII)<br>Data Length: (Null - use the numbers in this field.)<br>The control is optional and there is no corresponding field in the input message, so Formatter uses the default value, converts it to IBM Signed Packed Decimal, and generates the following output: 12 34 5D. Each pair of numbers represents the two nibbles of a byte. The result is three bytes long. |
| IBM Zoned Integer | Data type on larger IBM computers used to represent integers. Each decimal digit is represented by a byte. The left nibble of the byte is a hexadecimal F. The right nibble is the hexadecimal value of the digit. For example, 1234 is represented as F1 F2 F3 F4 (the number pairs show the hexadecimal values of the nibbles of each byte). |
| IBM Signed Zoned Integer | Data type on larger IBM computers used to represent integers. Each decimal digit is represented by a byte. The left nibble of each byte, except the last byte, is a hexadecimal F. The left nibble of the last byte is a hexadecimal C if the number is positive and a hexadecimal D if the number is negative. The right nibble of each byte is the hexadecimal value of the digit. For example, 1234 is represented as F1 F2 F3 C4 (the number pairs show the hexadecimal values of the nibbles of each byte). -1234 is represented as F1 F2 F3 D4. |

**Table A-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 A-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 A-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.<br>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 A-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.<br><br>Date/Time formats may include:<br>1) Variations in year (2- or 4-digit year representation: YY or YYYY).<br>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).<br>3) Variations in the day of the month—use of a day of the month number (01-31).<br>4) Variations in hour—12-hour or 24-hour representation, with or without a meridian indicator (AM or PM.)<br>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.

# B MFL Document Type Definition

The following sections contain the MFL document type definition (`mfl.dtd`) and an example MFL document built using `mfl.dtd`.

## mfl.dtd

`mfl.dtd` is shown in Listing B-1.

**Listing B-1 `mfl.dtd`**

```
<!--

Note: MessageFormatSet is not currently supported. All MFL docs must
begin with MessageFormat as the root element.

-->

<!ELEMENT MessageFormatSet ( MessageFormat+ ) >

<!ELEMENT MessageFormat    ( StructFormat | StructFormatRef |
FieldFormat )+ >

<!ELEMENT StructFormat     ( StructFormat | StructFormatRef |
FieldFormat )+ >
```

```
<!--

Note: StructFormatRef is not currently supported. If specified,
this element is ignored.

-->

<!ELEMENT StructFormatRef  EMPTY >

<!ELEMENT FieldFormat      ( (TagField?,LenField? ) |
(LenField?,TagField? ) ) >

<!ELEMENT TagField         EMPTY >

<!ELEMENT LenField         EMPTY >


<!ENTITY % dataTypes

    "(Literal

        | String | Numeric

        | Binary | EBCDIC

        | Packed | UPacked

        | ZonedDecimal | UZonedDecimal

        | BigEndian2 | BigEndian4

        | UBigEndian2 | UBigEndian4

        | LittleEndian2 | LittleEndian4

        | ULittleEndian2 | ULittleEndian4

        | BigSwapEndian2 | BigSwapEndian4

        | UBigSwapEndian2 | UBigSwapEndian4

        | LittleSwapEndian2 | LittleSwapEndian4

        | ULittleSwapEndian2 | ULittleSwapEndian4

        | Date | Time | DateTime

        | mmddyy | mmddyyyy | mmddyyhhmi | mmddyyhhmiss

        | Smmddyy | Smmddyyyy | SCmmddyyhhmi

        | SCmmddyyhhmipm | SCmmddyyhhmiss | SCmmddyyhhmisspm
```

```
            | ddmonyy | ddmonyyyy
            | Sddmmyy | Sddmmyyyy | SCddmmyyhhmi | SCddmmyyhhmipm
            | SCddmmyyhhmiss | SCddmmyyhhmisspm
            | Dddmonyy | Dddmonyyyy | Dmonyy
            | Dmonyyyy | monyy | monyyyy | monddyyyy
            | hhmiss | Chhmipm | Chhmi | Chhmisspm | Chhmiss
            )">

<!ENTITY % baseTypes "(String | Numeric | EBCDIC)">

<!ENTITY % lenTypes
    "(Literal
            | String | Numeric
            | Binary | EBCDIC
            | Packed | UPacked
            | ZonedDecimal | UZonedDecimal
            | BigEndian2 | BigEndian4
            | UBigEndian2 | UBigEndian4
            | LittleEndian2 | LittleEndian4
            | ULittleEndian2 | ULittleEndian4
            | BigSwapEndian2 | BigSwapEndian4
            | UBigSwapEndian2 | UBigSwapEndian4
            | LittleSwapEndian2 | LittleSwapEndian4
            | ULittleSwapEndian2 | ULittleSwapEndian4
            )">
```

```
<!ENTITY % boolean "(y | n | true | false | yes | no)">

<!ENTITY % accessModes "(Normal | Current | Next |

                         Controlling | Relative | Increment)" >

<!ATTLIST MessageFormatSet>

<!ATTLIST MessageFormat   name            NMTOKEN      #REQUIRED>

<!--

StructFormat Notes: If a structure is repeating, only one of
'repeat', 'repeatField', or 'repeatDelim' may be specified.

-->


<!ATTLIST StructFormat    name            CDATA      #REQUIRED

                          repeat          CDATA        '0'

                          repeatField     CDATA        #IMPLIED

                          repeatDelim     CDATA        #IMPLIED

                          delim           CDATA        #IMPLIED

                          alternative     %boolean;    'n'

                          optional        %boolean;    'n' >

<!--

Note: StructFormatRef is not currently supported. If specified,
this element is ignored.

-->

<!ATTLIST StructFormatRef messageFormat   CDATA      #REQUIRED

                          repeat          CDATA        '0'

                          repeatField     CDATA        #IMPLIED

                          repeatDelim     CDATA        #IMPLIED

                          optional        %boolean;    'n' >
```

```
<!--

FieldFormat Notes: The basetype and cutoff attributes are only valid
for date types. The value attribute is only valid for Literal types.
The length attribute is only valid for types 'String', 'Numeric',
'EBCDIC', and 'Binary'.

-->

<!ATTLIST FieldFormat    name           CDATA         #REQUIRED
                         type           %dataTypes;    'String'
                         basetype       %baseTypes;    'String'
                         cutoff         CDATA          '50'
                         length         CDATA          #IMPLIED
                         delim          CDATA          #IMPLIED
                         value          CDATA          #IMPLIED
                         optional       %boolean;      'n'
                         accessMode     %accessModes;  'Normal'
                         controlName    CDATA          #IMPLIED
                         source         CDATA          #IMPLIED >


<!ATTLIST TagField       type           %dataTypes;    #REQUIRED
                         value          CDATA          #REQUIRED>


<!ATTLIST LenField       type           %lenTypes;     #REQUIRED
                         length         CDATA          #IMPLIED>
```

## Example MFL Document

An example MFL document is shown in Listing B-2. This document was built using `mfl.dtd`. The example defines a message containing employee data.

**Listing B-2 Example MFL Document**

```
<?xml version='1.0'?>

<!DOCTYPE MessageFormat SYSTEM ‘mfl.dtd’>

<MessageFormat name='EMPLOYEES_MSG'>

    <StructFormat name='EMPLOYEE'>

        <FieldFormat name='EMP_ID' type='BigEndian4'/>

         <StructFormat name='NAME'>

           <FieldFormat name='LAST_NAME' type='String' delim=', ‘/>

          <FieldFormat name='FIRST_NAME' type='String'/>

         </StructFormat>

        <FieldFormat name='START_DATE' type='Sddmmyy'
         basetype='String' cutoff='70'/>

        <FieldFormat name='RATE' type='String' delim='.'>

          <TagField type='String' value='SALARY=$'/>

        </FieldFormat>

    <StructFormat name='EMP_PIC' optional='true'>

       <FieldFormat name='TITLE' type='Literal'
        value='--- Picture ---'/>

       <FieldFormat name='PICTURE' type='Binary'>

           <LenField type='BigEndian4'/>

       </FieldFormat>

    </StructFormat>
```

```
        <FieldFormat name='NUM_PHONES' type='BigEndian4'/>

         <StructFormat name='EMP_PHONES' repeatField="NUM_PHONES">

           <FieldFormat name='PHONE_NUMBER' type='String'
            length='10'/>

         </StructFormat>

     </StructFormat>

</MessageFormat>
```

**Notes:** The name of the message definition, `EMPLOYEES_MSG`, is given in the name attribute of the `MessageFormat` element. The field `EMP_ID` is defined as a BigEndian4 integer by the next `FieldFormat` element. The name of an employee is defined with two fields: `LAST_NAME` and `FIRST_NAME`. Note that a comma and a space delimit the `LAST_NAME` field (delim attribute of `LAST_NAME`).

The `START_DATE` field is a date string that is not Y2K compliant. The cutoff attribute of this field specifies that any two-digit year greater than 70 is prefixed with 19.

The field `RATE` defines a tag of `SALARY=$`, which will always precede `RATE` data. The field `PICTURE` defines a length field that precedes the `PICTURE` data. This length field specifies how many bytes the `PICTURE` data contains.

The field `NUM_PHONES` is used to specify how many instances of the field `PHONE_NUMBER` exist at run-time. This is done by specifying `NUM_PHONES` as the `repeatField` for the `StructFormat EMP_PHONES`.

Listing B-3 is an example instance of the message `EMPLOYEES_MSG`.

**Listing B-3 Instance of EMPLOYEES_MSG**

```
\x00\x00\x04\x20Doe, Jane\x0023/01/99SALARY=$56500.--- Picture
---\x00\x00\x00\x03:-}\x00\x00\x00\x0297294351019729435102
```

**Notes:** Each byte of the `EMP_ID` data is represented with the `\x` prefix to indicate it is hexadecimal. This is done for illustration purposes only. The value of the `EMP_ID` data is 1056.

The `NAME` data (Doe, Jane) follows `EMP_ID`. This data is terminated by the NULL character `\x00` (the default for String types).

The 23/01/99 string is the `START_DATE` for this employee.

The `RATE` field is preceded with the tag `SALARY=$`. The `RATE` data is 56000 and is delimited with a period.

The literal `‘--- Picture ---’` follows. The length field for the `PICTURE` data is represented with the \x prefix as before. The length of the `PICTURE` data is given as 3 bytes. The `PICTURE` data `:-}` then follows.

The field `NUM_PHONES` is given by the hexadecimal representation `\x00\x00\x00\x02`, which is a value of 2. This indicates that there are two instances of `PHONE_NUMBER` data: 972-943-5101 and 972-943-5102.

# 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