SeeBeyond™ eBusiness Integration Suite

e*Insight Business Process Manager Implementation Guide

Release 4.5.2

SEEBEYOND™

Version 20020220105939.

Contents

Chapter 1

Introduction

This guide provides comprehensive information on implementing eBusiness solutions using the e*Insight Business Process Manager. It discusses the essentials of implementing e*Insight, and the components used in a complete e*Insight implementation. This guide also provides detailed information on the e*Insight architecture and its core components, as well as the e*Gate schema components that make up an e*Insight implementation. Finally, it discusses how e*Insight and e*Gate work together to provide a comprehensive toolset for designing, creating, and maintaining a fully functional eApplication.

1.1 Document Purpose and Scope

This guide explains how to use the e*Insight Business Process Manager. This user guide includes information on the following topics:

- Understanding the e*Insight schema components.
- Functions and methods available to the user.

This guide gives the e*Insight implementor the necessary background and methodology for getting an e*Insight system up and running in a real-world situation. To do this, it provides detailed information on the e*Gate schema that e*Insight uses as its back end and explains the various areas requiring configuration. This guide also contains several detailed case studies showing how to implement various features built into e*Insight, such as automatic undo of failed business processes.

1.2 Intended Audience

The reader of this guide is presumed to be a developer or system administrator with responsibility for developing or maintaining the e*Insight system. The implementor should have experience of Windows NT and UNIX operations and administration, and should be thoroughly familiar with Windows-style GUI operations.

Since most of the work in an e*Insight implementation involves setting up the e*Gate components that send data into and out of the e*Insight system, the implementor should also have experience implementing e*Gate.

1.3 Writing Conventions

The writing conventions listed in this section are observed throughout this document.

Hypertext Links

When you are using this guide online, cross-references are also hypertext links and appear in **blue text** as shown below. Click the **blue text** to jump to the section.

For information on these and related topics, see **"Supporting Documents" on page 19**.

Command Line

Text to be typed at the command line is displayed in a special font as shown below.

```
java -jar ValidationBuilder.jar
```

Variables within a command line are set in the same font and bold italic as shown below.

```
stcregutil -rh host-name -un user-name -up password -sf
```

Code and Samples

Computer code and samples (including printouts) on a separate line or lines are set in the command-line font as shown below.

```
Configuration for BOB_Promotion
```

However, when these elements (or portions of them) or variables representing several possible elements appear within ordinary text, they are set in *italics* as shown below.

path and *file-name* are the path and file name specified as arguments to **-fr** in the **stcregutil** command line.

Notes and Cautions

Points of particular interest or significance to the reader are introduced with ***Note***, ***Caution***, or ***Important***, and the text is displayed in *italics*, for example:

Note: *The Actions menu is only available when a Properties window is displayed.*

User Input

The names of items in the user interface such as icons or buttons that you click or select appear in **bold** as shown below.

Click **Apply** to save, or **OK** to save and close.

File Names and Paths

When names of files are given in the text, they appear in **bold** as shown below.

Use a text editor to open the **ValidationBuilder.properties** file.

When file paths and drive designations are used, with or without the file name, they appear in **bold** as shown below.

In the **Open** field, type **D:\setup\setup.exe** where **D:** is your CD-ROM drive.

Parameter, Function, and Command Names

When names of parameters, functions, and commands are given in the body of the text, they appear in **bold** as follows:

The default parameter **localhost** is normally only used for testing.

The Monk function **iq-put** places an Event into an IQ.

After you extract the schema files from the CD-ROM, you must import them to an e*Gate schema using the **stcregutil** utility.

1.4 Supporting Documents

The following SeeBeyond documents provide additional information about e*Insight and e*Gate:

- *SeeBeyond eBusiness Integration Suite Deployment Guide*
- *SeeBeyond eBusiness Integration Suite Primer*
- *e*Insight Business Process Manager User's Guide*
- *e*Insight Business Process Manager Installation Guide*
- *e*Gate Integrator Alert Agent User's Guide*
- *e*Gate Integrator Alert and Log File Reference Guide*
- *e*Gate Integrator Collaboration Services Reference Guide*
- *e*Gate Integrator Intelligent Queue Services Reference Guide*
- *e*Gate Integrator SNMP Agent User's Guide*
- *e*Gate Integrator System Administration and Operations Guide*
- *e*Gate Integrator User's Guide*
- *Monk Developer's Reference*
- *Standard e*Way Intelligent Adapters User's Guide*

1.5 SeeBeyond Web Site

The SeeBeyond Web site is your best source for up-to-date product news and technical support information. The site's URL is

http://www.SeeBeyond.com

Chapter 2

Introduction to the SeeBeyond eBI Suite

This chapter provides an overview of the SeeBeyond eBusiness Integration Suite, and explains how the e*Insight Business Process Manager fits into the Suite.

2.1 SeeBeyond eBusiness Integration Suite

This section provides an overview of the SeeBeyond eBusiness Integration Suite and its parts. It also provides a detailed overview of the e*Insight Business Process Manager (e*Insight) components.

Complex and dynamic partner relationships, and the management of various processes, present a tremendous challenge in eBusiness. Organizations and their trading partners are both faced with the problem of managing disparate component applications and aligning proprietary software requirements. In addition, organizations and their trading partners must agree on data exchange and security standards.

The SeeBeyond eBusiness Integration Suite merges traditional Enterprise Application Integration (EAI) and Business-to-Business (B2B) interactions into a multi-enterprise eBusiness Integration (eBI) product suite. This suite allows you to:

- Leverage your existing technology and applications.
- Create an eApplication consisting of component applications that are managed by your organization or your trading partners.
- Rapidly execute eBusiness strategies.
- Create and manage virtual organizations across the entire value chain.
- Rapidly implement industry standard business protocols.
- Quickly and easily establish new business partners, or update existing ones.
- Automatically secure transmissions sent over the public domain.

This suite also provides:

- Extensive and flexible back-office connectivity.
- Powerful data transformation and mapping facilities.
- Content-based routing.
- Unparalleled scalability based on a fully distributed architecture.

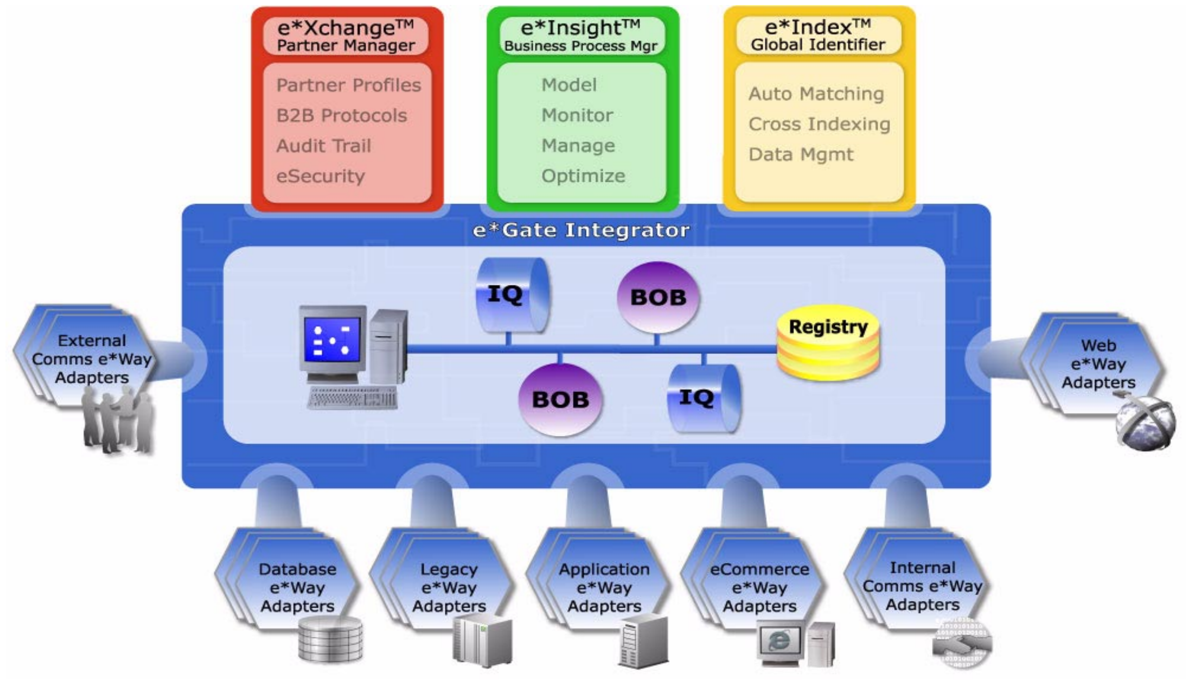
2.1.1 SeeBeyond eBusiness Integration Suite Components

The SeeBeyond eBusiness Integration Suite includes the following components and sub-components:

- eBusiness integration applications:
  - e*Insight™ Business Process Manager
  - e*Xchange™ Partner Manager
  - e*Index Global Identifier
- e*Gate™ Integrator:
  - e*Way™ Intelligent Adapters
  - Intelligent Queues (IQ™)
  - Business Object Brokers (BOBs)

See Figure 1 for a graphical representation of the SeeBeyond eBusiness Integration Suite and its components.

Figure 1 SeeBeyond eBusiness Integration Suite



e*Insight Business Process Manager

The e*Insight Business Process Manager facilitates the automation and administration of business process flow across eBusiness activities. Through graphical modeling and monitoring, business analysts can instantly assess the detailed state of a business process instance and identify bottlenecks in the process.

e*Xchange Partner Manager

The e*Xchange Partner Manager manages trading partner profiles and supports standard eBusiness message format and enveloping protocols, including RosettaNet, UN/EDIFACT, ASC X12, and BizTalk. The e*Xchange Partner Manager includes a Validation Rules Builder to aid in the creation of X12 and UN/EDIFACT message validation based on industry implementation guides.

The eSecurity Manager authenticates and ensures full integrity of message data sent to and from trading partners, which is imperative when conducting eBusiness over the public domain. The eSecurity Manager uses public key infrastructure (PKI) to ensure origin authentication of the sender.

e*Index Global Identifier

e*Index Global Identifier (e*Index) is a global cross-indexing application that provides a complete solution for automated person-matching across disparate source systems, simplifying the process of sharing member data between systems.

e*Index centralizes information about the people who participate throughout your business enterprise. The application provides accurate identification and cross-referencing of member information in order to maintain the most current information about each member. e*Index creates a single, consistent view of all member data by providing an automatic, common identification process regardless of the location or system from which the data originates.

e*Gate Integrator Components

e*Gate Integrator enables the flow of information across an extended enterprise by providing comprehensive connectivity to applications and datastores across a network. e*Gate is based on a distributed architecture with an open design that deploys flexible load balancing options. e*Gate processes Events according to user-defined business logic and integrates business processes between applications, ensuring end-to-end data flow into back-office systems.

e*Way Intelligent Adapters

e*Way Intelligent Adapters provide specialized application connectivity and also provide support for robust data processing such as business Collaborations, transformation logic, and publish/subscribe relationships. e*Way adapters are multi-threaded to enable high-performance distributed processing capabilities. This multi-threaded processing allows for ultimate deployment flexibility and load balancing.

Intelligent Queues

Intelligent Queues (IQs) are open-queue services for SeeBeyond or third-party queuing technology that provide robust data transport.

In conjunction with Java-enabled Collaborations, SeeBeyond JMS IQs can provide guaranteed exactly once delivery of messages.

Business Object Brokers

A BOB component is similar to an e*Way in the sense that it establishes connectivity and is capable of data transformation. BOBs use Collaborations to route and transform data within the e*Gate system. They have the following properties:

- They only communicate with IQs within e*Gate. They do not communicate with external applications as e*Ways do.
- They are optional by design. You can add them to an environment to remove some load from your e*Ways, either to set up easily maintainable data processing or to enable multiple internal processes.

2.2 Introducing e*Insight Business Process Manager (e*Insight)

The e*Insight Business Process Manager (e*Insight) is the component within the SeeBeyond eBusiness Integration Suite that facilitates the automation of the business process flow of eBusiness activities. The functions of e*Insight include business process model design, monitoring, and execution as well as the ability to analyze historical performance.

Using e*Insight, business analysts are able to design eBusiness process models through a user-friendly, fully graphical tool. The e*Insight GUI provides the appropriate graphical tools for an analyst to define all types of business models, from simple to very complex.

Once a business flow is modeled, the business analyst has the capability to instantly assess the detailed state of a business process instance through a color-coded graphical representation of the model. This way, the user can identify the processes that need intervention, repair, or authorization. The e*Insight GUI provides the appropriate facilities for the business analyst to examine the attributes of the business process instance (as defined by the business process analyst, during the design of the model), and—with the appropriate security privileges—modify their values.

For example, the business analyst can examine both syntactically and semantically the contents of a purchase order that failed to be processed, modify (repair) the purchase order, and then restart the failed business process instance, taking into account the modified purchase order.

In addition to the capability of monitoring the state of a given business process instance, e*Insight provides the business analyst with a complete historical picture, by tracking and storing all instances and the associated attributes of the business process model. The analyst has access to each one of the instances and can assess the performance of each through examining the values of the model's attributes as instantiated in the specific instance in review.

e*Insight provides the capability to analyze the performance of a business process model on a historical basis, so that trends can be determined and possible bottlenecks identified. The analyst can create charts on the performance of the business process model against an array of system attributes (such as "duration" and "state"), and user-defined attributes (for example, “order amount” or “PO source”). Charting the data in this way makes it easy to discern areas where the model needs re-design.

2.3 Building an eApplication

An eApplication is an integrated collection of software that enables you to model and manage an eBusiness. The SeeBeyond eBusiness Integration Suite provides the glue and essential building blocks that allow you to create a composite eApplication for running your eBusiness.

Implementing e*Insight involves the following steps:

1. Install and learn the basics of e*Insight.

   Use the *e*Insight Business Process Manager Installation Guide* to help you install the e*Insight software. See the *e*Insight Business Process Manager User’s Guide* for overview information and details on using the e*Insight GUI.

2. Obtain a working knowledge of e*Insight.

   Read chapters 1 through 3 of this Guide to comprehend the technical architecture of e*Insight, its components, and how they work together with e*Gate back-end components. This provides the foundation for implementing a working end-to-end eBusiness scenario.

3. Create an implementation plan.

   Use this manual as a guide for preparing a step-by-step roadmap of your implementation. This book describes several different types of e*Insight implementations. Use these as the basis for planning the e*Insight implementation best suited to your business needs.

2.4 Basic Information

Implementing an e*Insight system is the process of translating the vision of the business analyst into a functioning system. Once the analyst has determined that a certain business task must be accomplished with e*Insight, it is the job of the implementor to make this a reality.

You implement e*Insight by using the e*Insight GUI to enter the relevant data into the e*Insight database. Then you combine the generated e*Gate components with other e*Gate components you add to create a complete e*Insight schema. The e*Insight components are mostly pre-configured and do not require any (or very slight) modification by the implementor. The components that you add are completely user-defined. However, the e*Insight GUI and this guide provide a framework for integrating these user-defined components into a working e*Insight system.

Chapter 3

Implementation Overview

3.1 Basic Information

Implementing an e*Insight system is the process of translating the vision of the business analyst into a functioning system. Once the analyst has determined that a certain business task must be accomplished with e*Insight, it is the job of the implementor to make this a reality.

You implement e*Insight by using the e*Insight GUI to enter the relevant data into the e*Insight database. Then you combine the generated e*Gate components with other e*Gate components you add to create a complete e*Insight schema. The e*Insight components are mostly pre-configured and do not require any (or very slight) modification by the implementor. The components that you add are completely user-defined. However, the e*Insight GUI and this guide provide a framework for integrating these user-defined components into a working e*Insight system.

3.2 Implementation Road Map

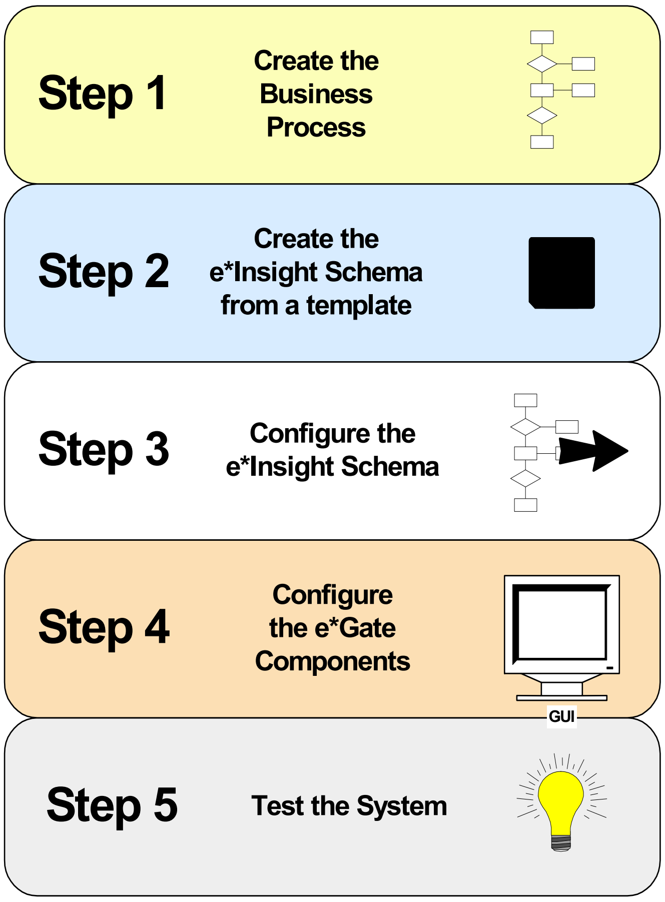
While each type of implementation involves a different approach. However, at a high level, there are certain similarities.

In general, the work of implementing an end-to-end scenario with e*Insight involves taking what is created in e*Insight and integrating it into a working e*Gate schema. e*Gate powers every e*Insight scenario, and a successful e*Insight implementation is dependent on a successful e*Gate implementation.

To give you an overview of the complete process, the following implementation road map contains high-level steps for a full e*Insight implementation. The road map is further refined and given more detail in the case study chapters that immediately follow this one.

Figure 2, illustrates the major steps in the integration process for an e*Insight implementation.

Figure 2 Integration Road Map


Step 1
Create the Business Process
Step 2
Create the e*Insight Schema from a template
Step 3
Configure the e*Insight Schema
Step 4
Configure the e*Gate Components
GUI
Step 5
Test the System

Step 1: Create a Business Process

Use the e*Insight GUI to design the business process.

For information on creating the business process, see the *e*Insight Business Process Manager User's Guide*.

Step 2: Copy the e*Insight Schema

Use a copy of the e*Insight schema as your starting point in e*Gate for supporting e*Insight.

For information on creating a copy of the e*Insight schema, see **"Copy the e*Insight Schema" on page 85**.

Step 3: Configure the e*Insight Schema Based on the Business Process

After you create the business process in e*Insight you must configure the e*Gate schema created in step 3 that supports your business process. Use the e*Gate schema configuration utility in the e*Insight GUI for this step. Complete your configuration using e*Gate Enterprise Manager.

Step 4: Configure the e*Gate Components

Configuring the e*Gate components forms the majority of the integration work done in e*Insight implementations. In this step, you:

- configure the e*Ways that send data into and out of the e*Insight system
- make all user-configurable associations in the e*Gate GUI

Step 5: Test and Tune the System

It is a good idea to test the system in stages. For example, make sure that one activity works properly before you try to run the entire business process. One good approach is to start with the "upstream" activities at the beginning of the business process, and work your way down to the last activity.

Once you have the entire system working, make adjustments as necessary to improve performance.

3.3 The e*Insight Schema

The e*Insight Schema is the e*Gate schema that implements a particular e*Insight installation. The starting point for a working e*Gate schema for e*Insight are the e*Gate schemas provided with the product. These schemas are:

- **eIJSchema** (Java)
- **eISchema** (Classic)

These schemas contains a number of pre-configured and partially pre-configured e*Gate components used by e*Insight. In addition to the components that are provided, a complete e*Insight implementation requires several other e*Gate components that are added to the e*Insight schema during the implementation process. The pre-configured components that are used, as well as the additional e*Gate components that are added to make up the final working e*Insight schema, depends entirely on the specifics of the implementation.

3.3.1 The eIJSchema (Java)

This is a new schema, introduced for 4.5.2. It is designed specifically to be used in a Java environment and all the components provided are Java based. The following sections in this guide provide information for the **eIJSchema**:

- Configuration Information
  - **"e*Insight Schema Components (eIJSchema)" on page 29**
  - **"e*Insight ETD for Java — eI_StandardEvent.xsc" on page 69**
  - **"Common Configuration Tasks" on page 83**
  - **"e*Insight Performance" on page 262**

    - "Troubleshooting" on page 275
    - "Java Helper Methods" on page 303
- Sample Implementations
    - "e*Insight Implementation (eIJSchema)" on page 96
    - "e*Insight Authorization Activity Implementation (eIJSchema)" on page 126
    - "e*Insight User Activity Implementation" on page 181
    - "e*Insight Sub-Process Implementation (eIJSchema)" on page 193
    - "e*Insight Remote Sub-Process Implementation" on page 219
    - "Active and Passive Modes" on page 236

3.3.2 The eISchema (Classic)

This schema can be used in a combined Monk and Java environment. The following sections in this guide provided information for the **eISchema**.

- Configuration Information
    - "e*Insight Schema Components (eISchema)" on page 48
    - "e*Insight ETD for Monk—eX_Standard_Event.ssc" on page 74
    - "Common Configuration Tasks" on page 83
    - "e*Insight Performance" on page 262
    - "Troubleshooting" on page 275
    - "e*Insight Helper Monk Functions" on page 284
- Sample Implementations
    - "e*Insight Implementation (eIJSchema)" on page 96
    - "e*Insight Authorization Activity Implementation (eISchema)" on page 170
    - "e*Insight User Activity Implementation" on page 181
    - "e*Insight Sub-Process Implementation (eISchema)" on page 205
    - "e*Insight Remote Sub-Process Implementation" on page 219
    - "Active and Passive Modes" on page 236

Chapter 4

e*Insight Schema Components (eIJSchema)

The purpose of this chapter is to describe the e*Gate components provided with the eIJSchema as well as those that are added in the implementation process, and discuss how each fits into and supports a working e*Insight implementation. For each component there is a detailed drawing showing the other components with which it interacts as well as the publication and subscription information for its Collaborations. In addition, for each component we discuss: the type of component it is, its function in e*Insight, any configuration the implementor must perform, the Collaborations it uses, and what is contained in the Events it processes.

4.1 The Purpose of the e*Gate Schema for e*Insight

The purpose of the e*Gate Schema for e*Insight is to provide the working portion of e*Insight. Whereas the e*Insight GUI is primarily used to configure and monitor the e*Insight system, the Schema components actually move and transform the data handled by e*Insight.

4.1.1 e*Insight Components

The e*Insight components start, run and implement business processes. The e*Insight components that start and implement business processes are user-defined and must be added to the e*Insight schema. The components that run business processes are provided by the e*Insight installation and require only a small amount of configuration on the part of the user.

4.2 e*Insight Schema Components Overview

Table 1 lists all of the module types used by e*Insight. It lists the components that are provided as part of the e*Insight schema (eIJSchema) installation, and also the components that the user adds in the implementation process. The meaning of the column headings is as follows.

- **Component**—The e*Gate logical name for the component. *Italics* indicates that the name varies by association or is user-defined.
- **Description**—A brief description of what the component does in e*Insight.
- **In Default eIJSchema**—Whether or not this component is provided as part of the schema installation of e*Insight.
- **Configuration Required**—Most of the modules in the default eIJSchema require little configuration on the part of the implementor. Table 1 uses the following terms to describe the level of configuration required:
  - **No—**The component does not require any configuration or programming on the part of the implementor.
  - **Minor—**You must add the e*Insight database connection information to the configuration file.
  - **Some—**You must make additional changes to the configuration file beyond providing the e*Insight database connection information.
  - **Yes—**The component is mostly or entirely user-defined and must be configured and programmed by the implementor.
- **More Information**—A cross reference to the section that describes this component in detail.

Table 1 e*Insight Schema Component Types

Component	Description	In Default eISchema?	Configuration Required?	More Information
e*Insight Engine	This is a specially configured Multi-Mode e*Way that runs business processes, using the e*Insight e*Way Connection.	Yes	Some	4.3.1 on page 33
eI_Resubmitter	A placeholder component used in the e*Insight Event failure handling process.	Yes	Yes	4.3.2 on page 38
Start BP Component	Either an e*Way or BOB that sends the Event that starts a business process instance.	No	Yes	4.3.4 on page 40
eX_*Activity* e*Way	Implements an e*Insight activity that connects to an external system.	No	Yes	4.3.5 on page 41
eX_*Activity* BOB	Implements an e*Insight activity that does not connect to an external system.	No	Yes	ß on page 44

4.2.1 e*Insight Schema Component Relationships Diagram

Figure 4 on the next page illustrates the relationships among the e*Insight schema components. Not every e*Insight implementation uses all of these components. Some of the components shown are not provided as part of the e*Gate Schema for e*Insight installation from the CD. These components are shown in light blue and must be added to the base e*Insight schema by the implementor as needed.

Figure 3 e*Insight Overview Legend

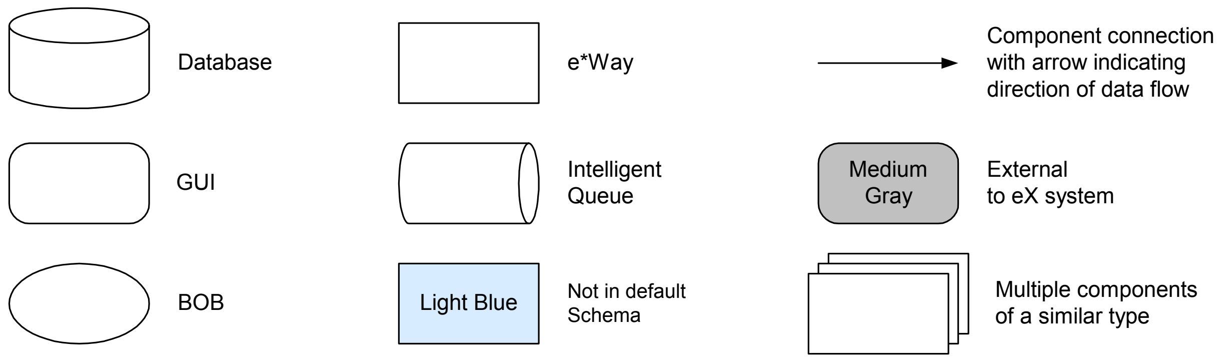
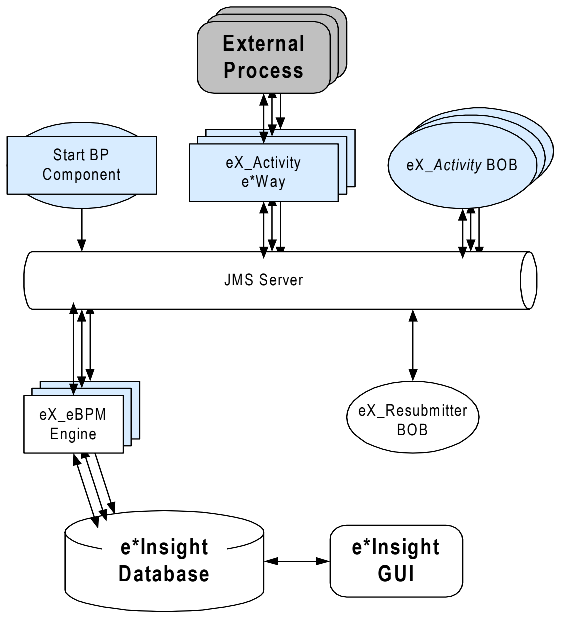

Database
e*Way
Component connection with arrow indicating direction of data flow
GUI
Intelligent Queue
Medium Gray
External to eX system
BOB
Light Blue
Not in default Schema
Multiple components of a similar type

Figure 4 e*Insight 4.5.2 Components



4.3 e*Insight Business Process Manager Components

The e*Insight components start, run and implement the businesses processes created in the e*Insight GUI. The components that run the business processes are supplied in the e*Insight installation, while those that implement a business process are user defined and must be added to the e*Insight e*Gate schema.

Components That Run Business Processes

The two component types dedicated to running and managing business processes are:

- One or more e*Insight engines
- The **eI_Resubmitter** e*Way

The e*Insight engine manages and runs business processes in e*Insight. One e*Insight engine is required, but more can be added to provide additional processing capacity when handling a large number of transactions.

For information on using multiple engines, see **"Using Multiple e*Insight Engines" on page 263**.

The **eI_Resubmitter** e*Way is used in e*Insight Event failure handling.

Components that Start Business Processes

The component that starts a business process is:

- The ***START_BP*** component (the exact name can be chosen by the user).

This is a user-defined component that creates the e*Insight Event that begins a business process instance.

Components that Implement Business Process Activities

The components that implement business process activities are:

- **eX_*Activity*** e*Ways

By default the activity components are named after the activity they implement with the prefix "eX_" added to the activity name found in the e*Insight GUI. They are added to the e*Insight schema when you use the e*Insight GUI to configure that schema for a particular business process. The e*Way may connect to an external system if required by the business activity. The user must also supply the programming to carry out the business logic of the activity and return an activity completed message (the "Done" Event) to the e*Insight engine

Note: *In addition to Collaborations running in e*Gate components, activities can also be implemented using Java scripts that run within the e*Insight engine. See the* ***e*Insight Business Process Manager User's Guide*** *for more information.*

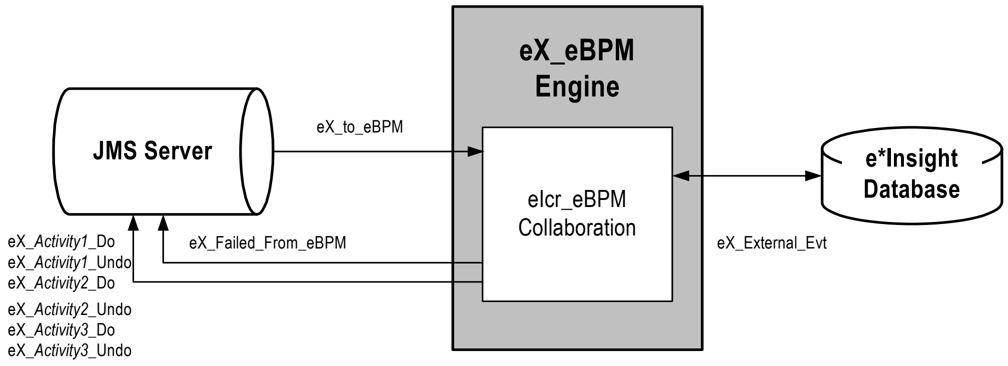
4.3.1 e*Insight Engine

The e*Insight engine manages and runs business processes. One e*Insight engine is required to operate, but more can be added to provide additional processing capacity when handling a large number of transactions.

For more information on using multiple engines, see **"e*Insight Engine Affinity (eIJSchema)" on page 266**.

An e*Insight engine is comprised of a specially configured Collaboration (eIcol_eBPM) and the e*Insight e*Way Connection (eIcp_eInsightEngine). The e*Insight engine runs within a Multi-Mode e*Way (eX_eBPM), which is referred to as the e*Insight Engine Container. An e*Insight engine allows an e*Way to communicate with the e*Insight database as shown in Figure 5.

Figure 5 The e*Insight Engine (Java)



Configuring the e*Insight Engine

The configuration file for the e*Insight engine allows you to set the JVM parameters available for the Multi-Mode e*Way. You can change the default configuration if required.

Note: *The e*Insight engine configuration file does not exist by default. You must create a new configuration file.*

Configuring the e*Insight Engine Connection

The e*Insight engine connection requires only minimal configuration on the part of the user. Table 2 lists those parameters in the engine's configuration file that the user can change.

Table 2 e*Insight Engine Connection Configuration Settings

Screen	Parameter	Setting
eBPM Settings	Database Type	Specifies the type of e*Insight database. Select one of the following: ▪ **Oracle** when using an Oracle 8i (8.1.6 and 8.1.7) ▪ **SQL Server** when using SQL Server 7.0 or SQL Server 2000 ▪ **Sybase** when using Sybase 11.9 or 12.
	JDBC URL String	This is the connection string used by the e*Insight engine to communicate with the e*Insight database. Use the connection string that is appropriate for the database client setup on the machine running the e*Insight engine. (Refer to the relevant driver documentation for more details on configuring your system).

Table 2 e*Insight Engine Connection Configuration Settings (Continued)

Screen	Parameter	Setting
eBPM Settings	JDBC URL String	An Oracle database connection might use the string: **jdbc:oracle:*thin*:@*machine_name*:*port*:*db*** where ▪ ***thin*** is the type of oracle client interface. See your e*Insight database administrator for more information. ▪ ***machine_name*** is the network name of the computer running the e*Insight database. If the database is on the same machine as the e*Insight engine you can use "localhost" instead of the machine's network name. ▪ ***port*** is correct port for communicating with the e*Insight database (1521 is the default). ▪ ***db*** is the service name used to communicate with e*Insight Oracle database from the local machine. If you are using XML data and a Model Specific database, then you would use the OCI8 driver. See the e*Insight Business Process Manager Installation Guide for information on installing the driver. You might use the string: **jdbc:oracle:*oci8*:@*db*** where ▪ ***oci8*** is the type of oracle client interface. ▪ ***db*** is the service name used to communicate with e*Insight Oracle database from the local machine. A SQL Server database connection might use the string: **jdbc:SeeBeyond:sqlserver://<server>:<port#>;DatabaseName=<dbname>;embedded=true;SelectMethod=cursor** ▪ ***server*** is the network name of the computer running the e*Insight database. If the database is on the same machine as the e*Insight engine you can use "localhost" instead of the machine's network name. ▪ ***dbname*** is the name of the e*Insight SQL Server database. ▪ ***port*** is correct port for communicating with the e*Insight database. A Sybase database connection might use the string: **jdbc:sybase:Tds:<server>:<port>** ▪ ***server*** is the network name of the computer running the e*Insight database. If the database is on the same machine as the e*Insight engine you can use "localhost" instead of the machine's network name. ▪ ***port*** is correct port for communicating with the e*Insight database.

Table 2 e*Insight Engine Connection Configuration Settings (Continued)

Screen	Parameter	Setting
eBPM Settings	JDBC Driver Class	Enter name of JDBC Driver Class which interprets the JDBC URL String specified previously to gain access to the e*Insight database. For example, **oracle.jdbc.driver.OracleDriver** can be used with an ORACLE database, and **sun.jdbc.odbc.JdbcOdbcDriver** can be used with a SQL Server database or a Sybase database.
	Database User name	Determines the database user name under which the e*Insight engine accesses the e*Insight database. The user should have the same rights as the administrator user (default is **ex_admin**) created by the e*Insight database schema creation scripts.
	Encrypted Password	Determines the password associated with the name the e*Insight engine uses to access the e*Insight database. The default password used by e*Insight database creation scripts is **ex_admin**.
	Maximum Business Process Cache Size	This is the number of business processes that the e*Insight engine can hold in memory at one time. If the cache is full and another business process needs to be loaded, the least recently used (LRU) business process in the cache is replaced with the new business process. The default is 1024 business processes. The size of the business processes does not matter. Entering the special value of zero (0) implies that caching of Business Process definitions is NOT desired, and thus the eBPM Engine ALWAYS reloads the Business Process definition from the database for EVERY Activity event of a Business Process Instance. Note, this feature severely impacts performance
	Maximum Instance Cache Size	Enter the maximum number of Business Process Instances that the e*Insight engine caches in memory. When the maximum size is reached, the Engine first removes the Least Recently Used (LRU) Instance from the cache. Entering a value of -1 means that there is no limit to the number of Instances kept in memory. The value entered for this parameter effects the total amount of memory used by the engine. Limit the number of instances if you start getting out of memory messages when running the engine. Note: A value of zero (0) should NOT be used.

Table 2 e*Insight Engine Connection Configuration Settings (Continued)

Screen	Parameter	Setting
eBPM Settings	Auto Model Reload	Determines if the engine dynamically loads an enabled Business Process Version if the enabled/ disabled status of Business Process Version changes. If the value is set to YES then Business Process Versions that are enabled or disabled while the engine is running are immediately recognized. However, setting this value to YES may degrade performance.
	Instance Caching	Instance Caching is the most efficient way to process Business Process Instances. Setting this value to YES keeps a cache of the instance information throughout the life span of the Business Process Instance. Setting this value to NO retrieves the information from the database instead. This allows more flexibility and fault tolerance at the cost of performance. To improve performance it is recommended to set this parameter to YES and use multiple e*Insight engines. To use both instance caching and multiple engines it is necessary to ensure that a single instance is always processed by the same engine. This is achieved by using engine affinity. For information using Instance Caching with multiple engines see **"e*Insight Engine Affinity (eIJSchema)" on page 266**.
	Brand By Collaboration Name	This parameter allows you to name Event Types based on the Collaboration rather than the e*Way name. This is used for engine affinity when multiple e*Insight engine e*Way Connections are used in a single e*Way. For more information, see **"e*Insight Engine Affinity (eIJSchema)" on page 266**.
	Business Processes to Preload	This parameter allows you to load all or a subset of all the business processes stored in the e*Insight database. The default is ALL.
	Using e*Xchange with e*Insight	If an Event Type Definition is used that contains both e*Insight and e*Xchange sections, setting this to NO causes the e*Xchange section to be ignored. This reduces the time taken by the e*Insight engine at runtime to parse the Event.
	(Others)	(Default)
connector	ALL	(Default)

eIcr_eBPM Collaboration

The **eIcr_eBPM** Collaboration is not user-configurable. It provides the **eX_*Activity*_Do** and **eX_*Activity*_Undo** Events to the e*Gate layer components that carry out those activities. It also publishes failed Events to the JMS Server. This Collaboration is also used to retrieve Events that require processing by the e*Insight engine. For example, it would retrieve "Done" Events from an activity Collaboration.

Subscribed Event Type:

- **eX_External_Evt**—This Event carries the data retrieved from the e*Insight database to the e*Insight engine.
- **eX_to_eBPM**—This Event Type carries all Events intended for e*Insight. These include Start BP Events, "Done" Events, and other Events that must be processed by the e*Insight engine. The corresponding ETD is **eI_StandardEvent.xsc**. This is the only Event Type to which the e*Insight engine subscribes; all data sent to the e*Insight engine must use this Event Type.

Note: *The **eI_StandardEvent.xsc** does not contain a section for the e*Xchange Partner Manager information. If your implementation also uses e*Xchange Partner Manager then you may need to use **eIX_StandardEvent.xsc** as this contains the TP_EVENT location.*

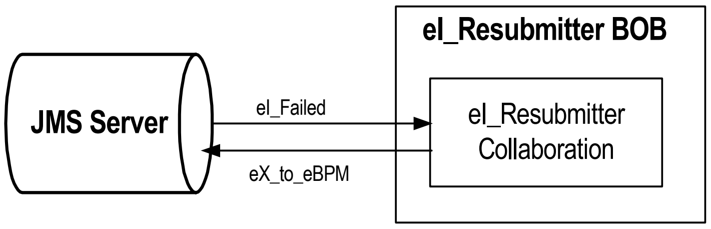
Published Event Types:

- **eX_DynamicET**—This Event is used to enable the Collaboration to automatically generate one of the following Events:
  - **eX_*Activity*_Do**—This Event causes the subscribing Collaboration to execute the "Do" logic of the activity with the same name in the business process. See **"Subscribed Event Type: eX_Activity_Do" on page 43** for more information.
  - **eX_*Activity*_Undo**—This Event causes the subscribing Collaboration to execute the "Undo" logic of the activity with the same name in the business process. See **"Subscribed Event Type: eX_Activity_Undo" on page 43** for more information.
- **eX_Failed_From_eBPM**—This Event contains the failed Event along with error information.
- **eX_External_Evt**—This Event carries the data that is written to the e*Insight database.

4.3.2 eI_Resubmitter BOB

The eI_Resubmitter BOB is a placeholder component that the implementor can use to resubmit failed Events back to the JMS Server after repairing them. The Event Repair logic in the eI_Resubmitter BOB's Collaboration must be supplied by the implementor.

Figure 6 eI_Resubmitter Detail (Java)



Configuring the eI_Resubmitter BOB

The user must fill in the **eI_Resubmitter** Collaboration with the logic to repair and resubmit Events retrieved from the JMS Server.

eI_Resubmitter Collaboration

The **eI_Resubmitter** Collaboration is a placeholder Collaboration that the implementor can use as a starting point to add logic that repairs and resubmits Events that have failed to be processed by the e*Insight engine due to data errors.

Subscribed Event Type: eX_Failed_From_eBPM

This Event Type contains the e*Insight Event that failed to process correctly at the e*Insight engine level due to a data error, along with the error information.

Published Event Type: eX_to_eBPM

This Event Type contains the repaired version of the failed Event to be reprocessed by the e*Insight engine.

4.3.3 Failed Event Handling by the e*Insight Engine

How the e*Insight engine handles errors generated when processing Events, depends on the type of error.

Error Types

Connection errors

Connection errors are errors that e*Insight receives because of a faulty connection to the e*Insight database.

Data errors

Data errors are exceptions that e*Insight generates because it cannot process an Event that is sent to it. Also in this class of errors are those generated by e*Insight because of a faulty business process configuration.

Error Handling

Connection Failure Handling

The normal handling of Events that can't be processed due to a connection error is to make a note of the error in the e*Insight engine's log file and retry processing the Event until a connection is made.

Data Failure Handling

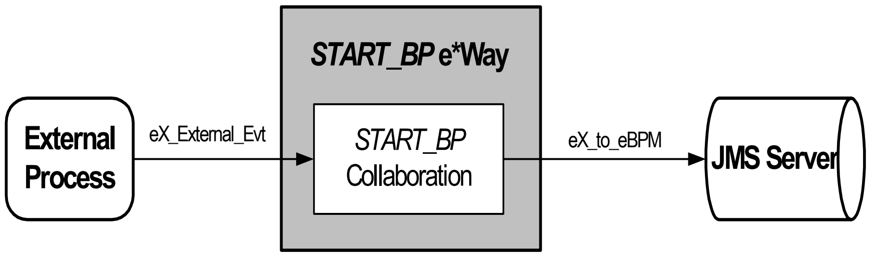
Events that fail to process due to data errors are not retried, but a notation is made in the e*Insight log file and the Event itself is published to a special location. This method allows the e*Insight engine to move on to other processing and not spend time attempting to resend failed Events.

The e*Insight engine publishes the failed Event to the JMS Server under the **eX_FailedEvent** Event Type. The **eI_Resubmitter** BOB subscribes to this Event Type and you can use it to repair the Event and republish it to the e*Insight Engine. The **eX_FailedEvent.xsc** ETD associated with the **eX_FailedEvent** Event Type has two major node structures. Once contains error information, and the other contains the failed Event.

4.3.4 START_BP Component

The START_BP component is the e*Gate component that sends the "Start" Event which initiates a business process instance (BPI). This component does not have a corresponding activity in the business process model.

Figure 7 START_BP e*Way Detail



Typically, an e*Way used to start the BPI connects to a business application, which in turn provides the data used by the business process. The type of e*Way connection chosen depends on the type of business application or external system to which the e*Way must connect in order to bring in the data. For example, if the business application is Siebel, then the e*Way Connection used is the Siebel e*Way Connection. If the data is held internally to e*Gate then a JMS Connection is used. Alternatively, it is possible to retrieve data from a JMS IQ.

The Collaboration must put the data they receive into the standard format used throughout e*Insight.

Configuring the START_BP Component

Configuring the START_BP Component depends on where the data is retrieved from. See the appropriate e*Way User Guide for information on configuring a specific e*Way Connection. In addition, the implementor must create a Collaboration Rules Script for the START_BP component that constructs the inbound e*Insight Event that starts a business process instance.

START_BP Collaboration

This Collaboration, used by the **START_BP** component, prepares the **eX_to_eBPM** Event. This Event is sent to the e*Insight engine in order to start an instance of the business process. The Collaboration must do two things:

- populate the three nodes required to start a BPI in the e*Insight standard Event
- place the data it receives into one or more global attributes of the business process

See **"Starting a Business Process (eIJSchema)" on page 87** for more information on how to start a BPI.

Subscribed Event Type: eX_External_Evt

When using a **START_BP** e*Way, this Event carries the data from the external application to which the **START_BP** e*Way connects. When the data is held internally to e*Gate, this Event carries data from a JMS Server.

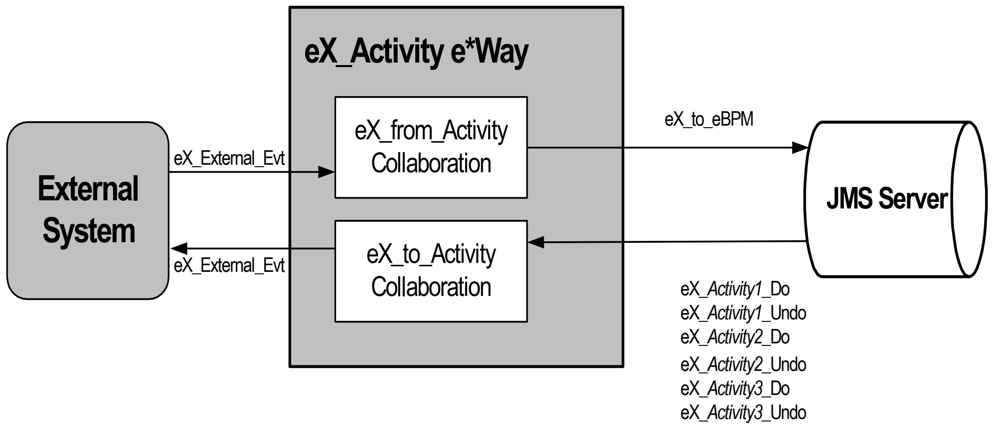
Published Event Type: eX_to_eBPM

This "Start" Event carries the data to begin an instance of a particular business process to the e*Insight engine.

4.3.5 Single-Mode Activity e*Way

An activity e*Way implements an e*Insight activity that requires a connection to a system outside of e*Gate.

Figure 8 Activity e*Way Detail



When you use the e*Insight GUI to configure the e*Gate schema supporting the e*Insight implementation, each activity in the business process becomes either a pair of Collaborations or a single Collaboration. The choice to use s single Collaboration or multiple Collaborations depends on the preference of the implementor.

For an activity e*Way that connects to an external system, the two Collaborations that are created are named **eX_from_*Activity*** and **eX_to_*Activity***, where ***Activity*** is replaced with the **Activity Name** from the associated business process in e*Insight. In addition to the Collaborations, the corresponding Collaboration Rules and the Event Types subscribed to and published by the Collaborations are also named after the activity name.

The Collaboration Rule created in this process is only a placeholder. Implementors must configure the Collaboration Rules by writing the Collaboration Rules Script and choosing the service under which this script runs. The CRS contains the programming that implements the business logic for the corresponding activity in the business process.

The type of e*Way Connection used to implement a particular activity depends on what the activity is supposed to accomplish in the business process. For example, an SAP e*Way Connection could be used to connect to an ERP system to look up the credit standing of a customer, or an Oracle e*Way Connection could be used to look up the mailing address of a prospective client in a marketing database.

Configuring the eX_to_*Activity* e*Way

Some of the configuration for the activity e*Way is done for you when you use the e*Insight GUI to configure the schema. This includes setting up the component relationships and Event Type routing in the e*Insight schema, but not the actual business logic programming or the type of e*Way that is used. The business logic programming must be done by the implementor in Collaboration Rules Scripts used by the activity Collaborations, and the e*Way's configuration file must be defined based on the type of e*Way chosen to implement the activity. See the appropriate e*Way Users Guide for information on how to set up the e*Way chosen.

eX_to_*Activity* Collaboration

The **eX_to_*Activity*** Collaboration receives the Event that carries the data used in the activity. It receives the Event from the e*Insight Engine, and passes it to the external process that implements the activity's business logic. The Collaboration must be configured to convert the data into whatever format is required by the external system to which the activity e*Way connects.

Important: *In addition to passing the attribute data it receives from the e*Insight engine to the external system, the **eX_to_Activity** Collaboration must preserve the e*Insight Business Process Instance tracking information contained in the **eX_Activity_Do** or **eX_Activity_Undo** Events. This information is used to send the return or "Done" Event back to the e*Insight engine, when the activity completes. See* **Sending the "Done" Event Back to e*Insight (eIJSchema)** *on page 88 for more information on what information is required in the "Done" Event.*

Do and Undo logic in an Activity Collaboration

The **eX_to_*Activity*** Collaboration in an activity e*Way connecting to an external system and the **eX_Activity** Collaboration in an Activity connecting internally to e*Gate both subscribe to two Event Types: **eX_*Activity*_Do** and **eX_*Activity*_Undo**. When the activity Collaboration picks up a "Do" Event Type, it carries out a "positive" instance of the activity. When the activity Collaboration picks up an "Undo" Event Type from the JMS Server, it carries out a "negative" or compensating version of the activity—in other words, an activity that cancels out a previously completed "Do" instance of the activity for the current business process instance.

By default, the eX_to_Activity and eX_Activity Collaboration in an activity e*Way subscribe to both the "Do" and "Undo" Events. Consequently the CRS must contain logic to handle both the activity and the compensating transaction for the activity. Of course, the implementor is also free to place the "Undo" logic in a separate Collaboration as long as the **eX_*Activity*_Undo** Event Type is subscribed to and the proper Event is returned to the e*Insight engine.

The e*Insight engine provides local attributes only available to a particular activity. It uses them for holding values set by the "Do" portion of the activity Collaboration. These values can then be used in the "Undo" logic portion of the activity Collaboration to carry out the compensating transaction. That is, these attributes can be set by the "Do" portion of the CRS and then recalled by the "Undo" portion of the CRS in order to cancel out the "Do" when necessary.

For more information on local attributes and where to set them in the e*Insight Standard ETD, see **"Local Attributes" on page 68**.

Subscribed Event Type: eX_*Activity*_Do

This Event causes the subscribing Collaboration to execute the "Do" logic of the corresponding activity in the business process. This Event Type is in standard e*Insight format and contains the current values of any global variables designated as "Input" by the activity in the appropriate location in the **eI_StandardEvent.xsc** ETD.

Subscribed Event Type: eX_*Activity*_Undo

This Event causes the subscribing Collaboration to execute the "Undo" logic of the corresponding activity in the business process. That is, it causes a compensating transaction to occur that "undoes" the completed activity within a BPI (see**"Do and Undo logic in an Activity Collaboration" on page 43** for an explanation of "undoing" an activity). This Event Type is in standard e*Insight format, and contains the current values of any global variables designated as "Input" by the activity in the appropriate location in the **eI_StandardEvent.xsc** ETD. Also, the **eX_*Activity*_Undo** Event contains any local variables set by the Collaboration executing the "Do" logic associated with this activity.

Published Event Type: eX_External_Evt

This Event carries data to the external process that executes the activity. It must be in a form compatible with the external system to which the e*Way connects.

eX_from_*Activity* Collaboration

This Collaboration returns the "Done" Event to the e*Insight engine. To do this, the **eX_from_*Activity*** Collaboration must take the data it receives from the external system and use it to populate the required nodes in the Event returned to the e*Insight engine. Specifically, it must do the following:

- Populate the activity status node on the **eI_StandardEvent.xsc** ETD with the value "SUCCESS" or "FAILURE" depending on whether or not the activity completes successfully.
- Set the values of any global variables designated as "Output" or "Input/Output" by the business process activity.
- Return the e*Insight BPI tracking information included in the Event (either **eX_*Activity*_Do** or **eX_*Activity*_Undo**) that initiated this activity.
- Set the values of any local variables used by the activity.

See **Sending the "Done" Event Back to e*Insight (eIJSchema)** on page 88 for more information.

Subscribed Event Type: eX_External_Evt

This Event contains the result of the completed activity from the external process that executed the activity's business logic.

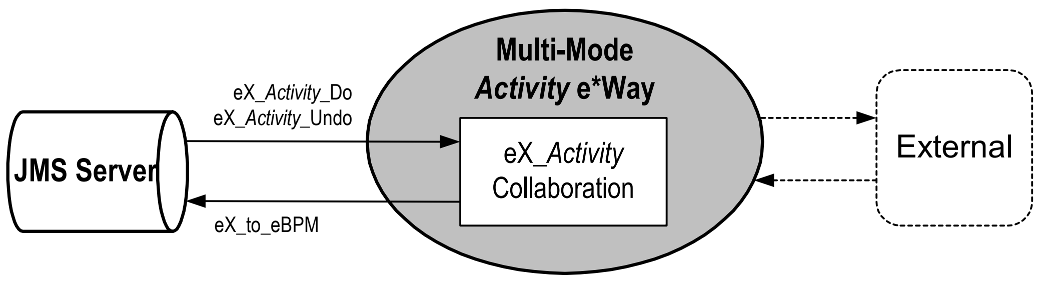
Published Event Type: eX_to_eBPM

- This is the "Done" Event that carries the data from a completed activity back to the e*Insight engine.

4.3.6 Multi-Mode Activity e*Way

The Multi-Mode Activity e*Way implements an e*Insight activity that can have multiple connections either inside or outside of e*Gate. A Multi-Mode Activity e*Way only needs one Collaboration to process the data and return it to the JMS Server. The CRS associated with the Multi-Mode e*Way's Collaboration Rule carries out the business logic of the activity to which it corresponds. This Rule could be a Monk script, a Java program, or any other script or application supported by the Collaboration Service under which the CRS runs.

Figure 9 Multi-Mode Activity e*Way Detail



eX_*Activity* Collaboration

This BOB activity Collaboration fulfills all of the functions that were split into a "to" and a "from" Collaboration in the case of an Single-Mode activity e*Way. In other words, it must:

- copy the e*Insight BPI tracking information to the destination Event in the CRS
- use the values of the "Input" attributes provided by the e*Insight engine in the **eX_*Activity*_Do** (or **eX_*Activity*_Undo**) to complete the business logic for this activity
- implement both the "Do" and "Undo" logic for the activity
- populate the status node (with "SUCCESS" or "FAILURE") depending on the outcome of the activity
- set the values for any "Output" or "Input/Output" attributes
- set the values for any local attributes

Subscribed Event Types:

- **eX_*Activity*_Do**—This Event causes the subscribing Collaboration to execute the "Do" logic of the activity with the same name in the business process. It is the same Event Type as that subscribed to by the **eX_to_*Activity* Collaboration**. See **"Subscribed Event Type: eX_Activity_Do" on page 43** for more information.
- **eX_*Activity*_Undo**—This Event causes the subscribing Collaboration to execute the "Undo" logic of the activity with the same name in the business process. It is the same Event Type as that subscribed to by the **eX_to_*Activity* Collaboration**. See **"Subscribed Event Type: eX_Activity_Undo" on page 43** for more information.

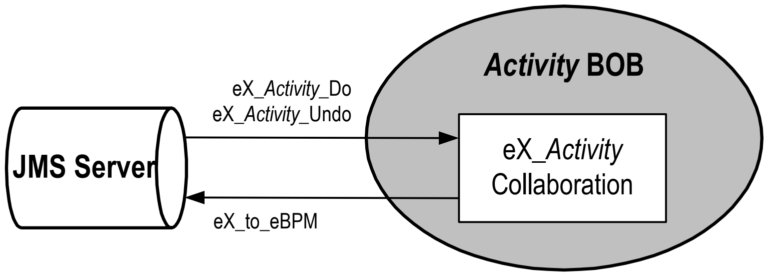
Published Event Type: eX_to_eBPM

This "Done" Event carries the data from a completed activity back to the e*Insight engine. It is the same as that published by the **eX_from_Activity** Collaboration. See **"Published Event Type: eX_to_eBPM" on page 44** for more information.

4.3.7 Activity BOB

The Activity BOB implements an e*Insight activity that does not require a connection to a system outside of e*Gate. A BOB only needs one Collaboration to process the data and return it to the JMS Server. The CRS associated with the BOB's Collaboration Rule carries out the business logic of the activity to which it corresponds. This Rule could be a Monk script, a Java program, or any other script or application supported by the Collaboration Service under which the CRS runs.

Figure 10 Activity BOB Detail



eX_*Activity* Collaboration

This BOB activity Collaboration fulfills all of the functions that were split into a "to" and a "from" Collaboration in the case of an activity e*Way. In other words, it must:

- copy the e*Insight BPI tracking information to the destination Event in the CRS
- use the values of the "Input" attributes provided by the e*Insight engine in the **eX_*Activity*_Do** (or **eX_*Activity*_Undo**) to complete the business logic for this activity
- implement both the "Do" and "Undo" logic for the activity
- populate the status node (with "SUCCESS" or "FAILURE") depending on the outcome of the activity
- set the values for any "Output" or "Input/Output" attributes
- set the values for any local attributes

Unlike the e*Way activity Collaborations, the BOB Collaboration does *not* need to reformat the data for an external system. The data remains in the standard e*Insight ETD.

Subscribed Event Types:

- **eX_*Activity*_Do**—This Event causes the subscribing Collaboration to execute the "Do" logic of the activity with the same name in the business process. It is the same Event Type as that subscribed to by the **eX_to_*Activity* Collaboration**. See **"Subscribed Event Type: eX_Activity_Do" on page 43** for more information.
- **eX_*Activity*_Undo**—This Event causes the subscribing Collaboration to execute the "Undo" logic of the activity with the same name in the business process. It is the same Event Type as that subscribed to by the **eX_to_*Activity* Collaboration**. See **"Subscribed Event Type: eX_Activity_Undo" on page 43** for more information.

Published Event Type: eX_to_eBPM

This "Done" Event carries the data from a completed activity back to the e*Insight engine. It is the same as that published by the **eX_from_Activity** Collaboration. See **"Published Event Type: eX_to_eBPM" on page 44** for more information.

4.4 Using Monk in eIJSchema

Although eIJSchema is primarily designed to use Java, you can manually configure a schema based on eIJSchema to use Monk. If you use a Monk component, then you introduce the following limitations:

- the engine can no longer publish a binary XML string
- the JMS connection cannot be used to connect to the Monk component
- the Event Type published to the Monk component must be specifically named in the engine's publications

4.4.1 Updating an eISchema to use the eIJSchema engine

1. Backup your schema.
2. Open eIJSchema and export the **eX_eBPM** module.
3. Open the schema that you are updating.
4. Rename **eX_eBPM** to **eX_eBPM_old**.
5. Import the eX_eBPM module that you exported in step 2.
6. Configure e*Insight engine connection.
7. Open the **eIcr_eBPM** Collaboration Rule properties.
   A. Delete initialization string.
8. Open the **eIcol_eBPM** Collaboration properties.
   A. Change the subscription source to <ANY>.
   B. Delete the publication of Event Type eX_DynamicET.
   C. Add a publication for every activity in your e*Gate schema as described in Table 3.

Table 3 Publication for eIJSchema update

Instance Name	Event Type	Destination
EIStandardInOut	eX_<Activity Name>_Do	eX_eBPM
EIStandardInOut	eX_<Activity Name>_Undo	eX_eBPM

9. Open the IQ Manager properties. Change the **IQ Manager Type** to **SeeBeyond JMS**.
10. Delete eX_eBPM_old.

Chapter 5

e*Insight Schema Components (eISchema)

The e*Insight Schema is the e*Gate schema that implements a particular e*Insight installation. The starting point for a working e*Gate Schema for e*Insight is the e*Gate schema called **eISchema** created when you install the e*Gate Schema for e*Insight from the installation CD. This schema contains a number of pre-configured and partially pre-configured e*Gate components used by e*Insight. In addition to the components that are provided on the CD, a complete e*Insight implementation requires several other e*Gate components that are added to the e*Insight schema during the implementation process. The pre-configured components that are used, as well as the additional e*Gate components that are added to make up the final working e*Insight schema, depends entirely on the specifics of the implementation.

The purpose of this chapter is to describe the e*Gate components provided with the eISchema as well as those that are added in the implementation process, and discuss how each fits into and supports a working e*Insight implementation. For each component there is a detailed drawing showing the other components with which it interacts as well as the publication and subscription information for its Collaborations. In addition, for each component we discuss: the type of component it is, its function in e*Insight, any configuration the implementor must perform, the Collaborations it uses, and what is contained in the Events it processes.

5.1 The Purpose of the e*Gate Schema for e*Insight

The purpose of the e*Gate Schema for e*Insight is to provide the working portion of e*Insight. Whereas the e*Insight GUI is primarily used to configure and monitor the e*Insight system, the Schema components actually move and transform the data handled by e*Insight.

5.1.1 e*Insight Components

The e*Insight components start, run and implement business processes. The e*Insight components that start and implement business processes are user-defined and must be added to the e*Insight schema. The components that run business processes are provided by the e*Insight installation and require only a small amount of configuration on the part of the user.

5.2 e*Insight Schema Components Overview

Table 4 lists all of the component types used by e*Insight. It lists the components that are provided as part of the e*Insight schema (eISchema) installation, and also the components that the user adds in the implementation process. The meaning of the column headings is as follows.

- **Component**—The e*Gate logical name for the component. *Italics* indicates that the name varies by association or is user-defined.
- **Description**—A brief description of what the component does in e*Insight.
- **In Default eISchema**—Whether or not this component is provided as part of the schema installation of e*Insight.
- **Configuration Required**—Most of the components in the default eISchema require little configuration on the part of the implementor. Table 4 uses the following terms to describe the level of configuration required:
  - **No—**The component does not require any configuration or programming on the part of the implementor.
  - **Minor—**You must add the e*Insight database connection information to the configuration file.
  - **Some—**You must make additional changes to the configuration file beyond providing the e*Insight database connection information.
  - **Yes—**The component is mostly or entirely user-defined and must be configured and programmed by the implementor.
- **More Information**—A cross reference to the section that describes this component in detail.

Additional Components

There are a number of components in eISchema that are not used in a standard implementation. These include:

- **eI_DynamicET** Event Type
- **eIcr_BuiltForSuccess** Collaboration Rule
- **eIcr_eBPM** Collaboration Rule
- **<host_name>_jmsserver** IQ Manager

The above components are provided to enable you to upgrade your schema to use the engine provided with eIJSchema. For more information, see **"e*Insight Schema Components (eIJSchema)" on page 29**. You can delete these components if you do not want to use them.

Important: *It is recommended that you use the eIJSchema base e*Gate schema initially, rather than upgrade an e*Gate schema based on eISchema to use the components provided in eIJSchema.*

Table 4 e*Insight Schema Component Types

Component	Description	In Default eISchema?	Configuration Required?	More Information
eX_eBPM Engine	This is a specially configured Java e*Way Connector that runs business processes.	Yes	Some	5.3.1 on page 52
eI_Resubmitter BOB	A placeholder component used in the e*Insight Event failure handling process.	Yes	Yes	5.3.2 on page 57
Start BP Component	Either an e*Way or BOB that sends the Event that starts a business process instance.	No	Yes	5.3.4 on page 59
eX_*Activity* e*Way	Implements an e*Insight activity that connects to an external system.	No	Yes	5.3.5 on page 62
eX_*Activity* BOB	Implements an e*Insight activity that does not connect to an external system.	No	Yes	5.3.6 on page 64

5.2.1 e*Insight Schema Component Relationships Diagram

Figure 12 on the next page illustrates the relationships among the e*Insight schema components. Not every e*Insight implementation uses all of these components. Some of the components shown are not provided as part of the e*Gate Schema for e*Insight installation. These components are shown in light blue and must be added to the base e*Insight schema by the implementor as needed.

Figure 11 e*Insight Overview Legend

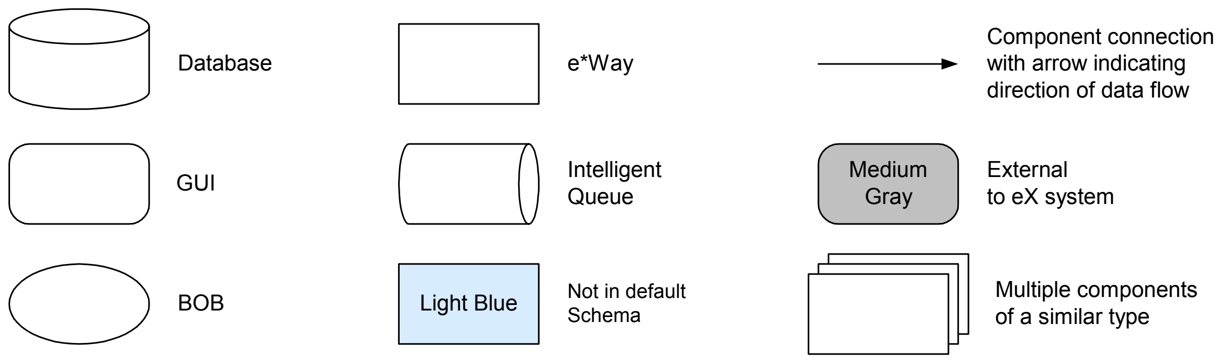
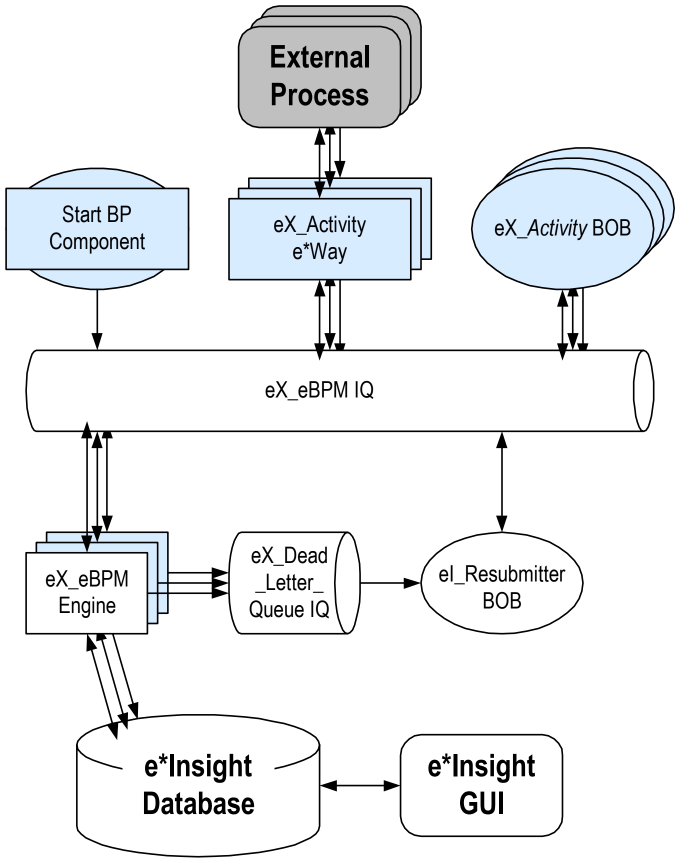

Database
e*Way
Component connection with arrow indicating direction of data flow
GUI
Intelligent Queue
Medium Gray
External to eX system
BOB
Light Blue
Not in default Schema
Multiple components of a similar type

Figure 12 e*Insight 4.5.2 Components



5.3 e*Insight Business Process Manager Components

The e*Insight components start, run and implement the businesses processes created in the e*Insight GUI. The components that run the business processes are supplied in the e*Insight installation, while those that implement a business process are user defined and must be added to the e*Insight e*Gate schema.

Components That Run Business Processes

The two component types dedicated to running and managing business processes are:

- One or more e*Insight engines
- The **eI_Resubmitter** BOB

The e*Insight engine manages and runs business processes in e*Insight. One e*Insight engine is required, but more can be added to provide additional processing capacity when handling a large number of transactions.

The **eI_Resubmitter** BOB is used in e*Insight Event failure handling.

Components that Start Business Processes

The component that starts a business process is:

- The ***START_BP*** component (the exact name can be chosen by the user).

This is a user-defined component that creates the e*Insight Event that begins a business process instance. It can be a BOB or an e*Way depending on the requirements of the scenario.

Components that Implement Business Process Activities

The components that implement business process activities are:

- **eX_*Activity*** e*Ways
- **eX_*Activity*** BOBs

By default the activity components are named after the activity they implement with the prefix "eX_" added to the activity name found in the e*Insight GUI. They are added to the e*Insight schema when you use the e*Insight GUI to configure that schema for a particular business process. They can be either an e*Way or a BOB depending on whether the business activity involves a connection to an external system. The user must also supply the programming to carry out the business logic of the activity and return an activity completed message (the "Done" Event) to the e*Insight engine

Note: *In addition to Collaborations running in e*Gate components, activities can also be implemented using Java scripts that run within the e*Insight engine. See the* ***e*Insight Business Process Manager User's Guide*** *for more information.*

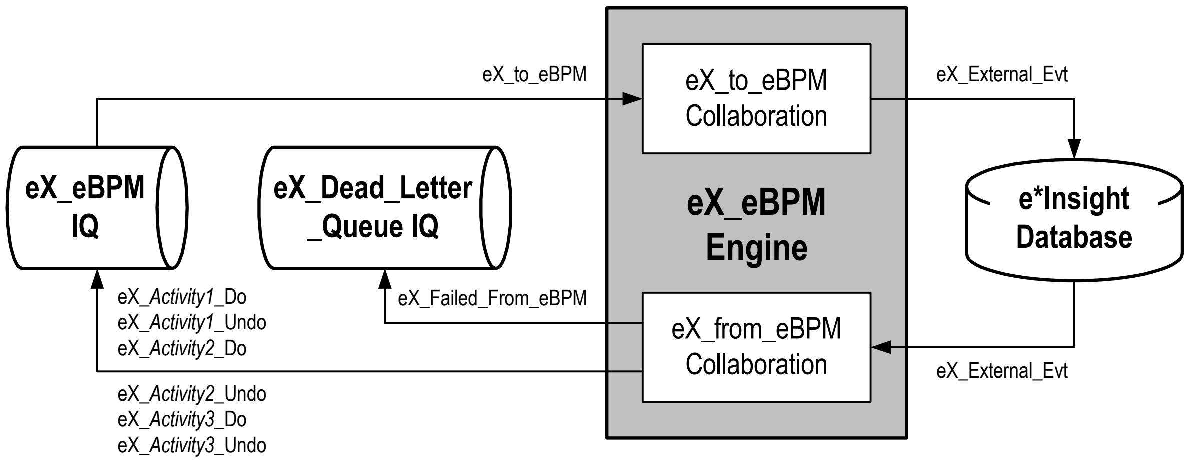
5.3.1 e*Insight Engine

The e*Insight engine manages and runs business processes. One e*Insight engine is required to operate, but more can be added to provide additional processing capacity when handling a large number of transactions.

For more information on using multiple engines, see **"Using Multiple e*Insight Engines (eISchema)" on page 271**.

The e*Insight engine, **eX_eBPM**, is a specially configured Java e*Way that is used to run business processes. An e*Insight engine communicates with both the **eX_eBPM** and **eX_Dead_Letter_Queue** IQs, as well as the e*Insight database as shown in Figure 13.

Figure 13 eX_eBPM Engine Detail



Configuring the e*Insight Engine

The e*Insight engine requires only minimal configuration on the part of the user. Table 5 lists those parameters in the engine's configuration file that the user can change.

Table 5 e*Insight Engine Configuration Settings

Screen	Parameter	Setting
General Settings	(All)	(Default)
Communication Setup	(All)	(Default)
Java VM Configuration	JNI DLL	This is the path to the java virtual machine used by the e*Insight engine. If necessary, replace the default value, **C:\eGate\Client\JRE\1.3\bin\hotspot\jvm.dll**, with the fully qualified path to the JNI DLL file on the Participating Host that is running the e*Insight engine. The default location given above is where the public installation of java 1.2.2 places this file on a Windows NT machine. ***Note:*** *In a UNIX environment the name and location of this file is different; for example,* ***/usr/Solaris_JDK_1.2.2_05a/jre/sparc/libjvm.sol***
	Enable Custom Error Handling	The default value is "YES". Change the value to "NO" if you do not want custom error handling enabled. See the **eI_Resubmitter** BOB below for more information.
	(Others)	(Default)

Table 5 e*Insight Engine Configuration Settings (Continued)

Screen	Parameter	Setting
eBPM Settings	JDBC URL String	This is the connection string used by the e*Insight engine to communicate with the e*Insight database. Use the connection string that is appropriate for the database client setup on the machine running the e*Insight engine. (Refer to the relevant driver documentation for more details on configuring your system). An Oracle database connection might use the string: **jdbc:oracle:*thin*:@*machine_name:port:db*** where ▪ ***thin*** is the type of oracle client interface. See your e*Insight database administrator for more information. ▪ ***machine_name*** is the network name of the computer running the e*Insight database. If the database is on the same machine as the e*Insight engine you can use “localhost” instead of the machine’s network name. ▪ ***port*** is correct port for communicating with the e*Insight database (1521 is the default). ▪ ***db*** is the service name used to communicate with e*Insight Oracle database from the local machine. If you are using XML data and a Model Specific database, then you would use the OCI8 driver. See the e*Insight Business Process Manager Installation Guide for information on installing the driver. You might use the string: **jdbc:oracle:*oci8*:@*db*** where ▪ ***oci8*** is the type of oracle client interface. ▪ ***db*** is the service name used to communicate with e*Insight Oracle database from the local machine. A SQL Server database connection might use the string: **jdbc:SeeBeyond:sqlserver://<server>:<port#>;DatabaseName=<dbname>;embedded=true** ▪ ***server*** is the network name of the computer running the e*Insight database. If the database is on the same machine as the e*Insight engine you can use “localhost” instead of the machine’s network name. ▪ ***dbname*** is the name of the e*Insight SQL Server database. ▪ ***port*** is correct port for communicating with the e*Insight database.

Table 5 e*Insight Engine Configuration Settings (Continued)

Screen	Parameter	Setting
eBPM Settings	JDBC URL String	A Sybase database connection might use the string: **jdbc:sybase:Tds:<server>:<port>** ▪ ***server*** is the network name of the computer running the e*Insight database. If the database is on the same machine as the e*Insight engine you can use "localhost" instead of the machine's network name. ▪ ***port*** is correct port for communicating with the e*Insight database.
	Database Type	Specifies the type of e*Insight database. Select one of the following: ▪ **Oracle** when using an Oracle 8i (8.1.5 and above) ▪ **SQL Server** when using SQL Server 7.0 or SQL Server 2000 ▪ **Sybase** when using Sybase 11.9
	User name	Determines the database user name under which the e*Insight engine accesses the e*Insight database. The user should have the same rights as the administrator user (default is **ex_admin**) created by the e*Insight database schema creation scripts.
	Password	Determines the password associated with the name the e*Insight engine uses to access the e*Insight database. The default password used by e*Insight database creation scripts is **ex_admin**.
	JDBC Driver Class	Enter name of JDBC Driver Class which interprets the JDBC URL String specified previously to gain access to the e*Insight database. For example, **oracle.jdbc.driver.OracleDriver** can be used with an ORACLE database, and **sun.jdbc.odbc.JdbcOdbcDriver** can be used with a SQL Server database or a Sybase database.
	Maximum Business Process Cache Size	This is the number of business processes that the e*Insight engine can hold in memory at one time. If the cache is full and another business process needs to be loaded, the least recently used (LRU) business process in the cache is replaced with the new business process. The default is 1024 business processes. The size of the business processes does not matter. Entering the special value of zero (0) implies that caching of Business Process definitions is NOT desired, and thus the e*Insight engine ALWAYS reloads the Business Process definition from the database for EVERY Activity event of a Business Process Instance. Note, this feature severely impacts performance.

Table 5 e*Insight Engine Configuration Settings (Continued)

Screen	Parameter	Setting
	Maximum Instance Cache Size	Enter the maximum number of Business Process Instances that the e*Insight engine caches in memory. When the maximum size is reached, the Engine first removes the Least Recently Used (LRU) Instance from the cache. Entering a value of -1 means that there is no limit to the number of Instances kept in memory. The value entered for this parameter effects the total amount of memory used by the engine. Limit the number of instances if you start getting out of memory messages when running the engine. Note: A value of zero (0) should NOT be used.
	Instance Caching	Instance Caching is the most efficient way to process Business Process Instances.Setting this value to YES keeps a cache of the instance information throughout the life span of the Business Process Instance. Setting this value to NO retrieves the information from the database instead. This allows more flexibility and fault tolerance at the cost of performance. To improve performance it is recommended to set this parameter to YES and use multiple e*Insight engines. To use both instance caching and multiple engines it is necessary to ensure that a single instance is always processed by the same engine. This is achieved by using engine affinity. For information using Instance Caching with multiple engines see **"e*Insight Engine Affinity (eISchema)" on page 272**.
	Business Processes to Preload	This parameter allows you to load all or a subset of all the business processes stored in the e*Insight database. The default is ALL.
	Use Default Locale Encoding	Setting this value to YES specifies that all data received by and sent from this e*Insight Engine is encoded in the same language as the default Locale setting of this Participating Host. Setting this value to NO specifies that data is being passed to and from this e*Insight Engine in an UTF-8 encoding format.
	(Others)	(Default)

eX_from_eBPM Collaboration

The **eX_from_eBPM** Collaboration is *not* user-configurable. It provides the **eX_*Activity*_Do** and **eX_*Activity*_Undo** Events to the e*Gate layer components that carry out those activities. It also publishes failed Events to the **eX_Dead_Letter_Queue** IQ.

Subscribed Event Type: eX_External_Evt

This Event carries the data retrieved from the e*Insight database to the e*Insight engine.

Published Event Types:

- **eX_*Activity*_Do**—This Event causes the subscribing Collaboration to execute the "Do" logic of the activity with the same name in the business process. See **"Subscribed Event Type: eX_Activity_Do" on page 63** for more information.
- **eX_*Activity*_Undo**—This Event causes the subscribing Collaboration to execute the "Undo" logic of the activity with the same name in the business process. See **"Subscribed Event Type: eX_Activity_Undo" on page 64** for more information.
- **eX_Failed_From_eBPM**—This Event contains the failed Event along with error information.

eX_to_eBPM Collaboration

The **eX_to_eBPM** Collaboration is *not* user-configurable. The e*Insight engine uses this Collaboration to retrieve from the **eX_eBPM** IQ Events that require processing by the e*Insight engine. For example, it would retrieve "Done" Events put there by activity Collaborations in the e*Gate layer.

Subscribed Event Type: eX_to_eBPM

This Event Type carries all Events intended for e*Insight. These include Start BP Events, "Done" Events, and other Events that must be processed by the e*Insight engine. The corresponding ETD is **eX_Standard_Event.ssc**. This is the only Event Type to which the e*Insight engine subscribes; all data sent to the e*Insight engine must use this Event Type.

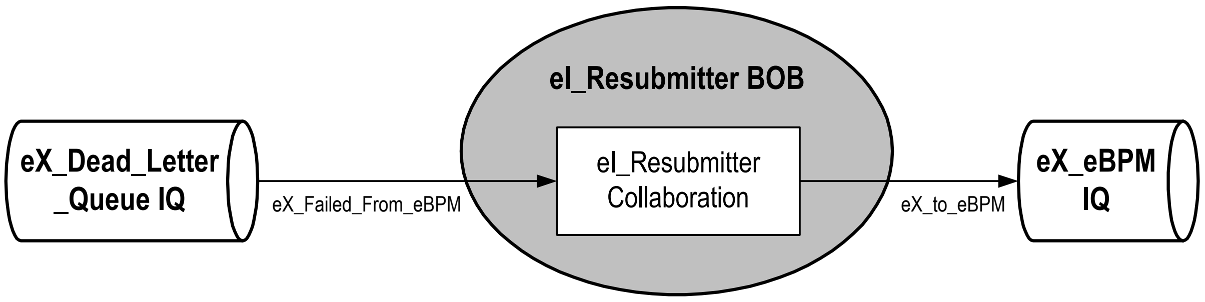
Published Event Type: eX_External_Evt

This Event carries the data that is written to the e*Insight database.

5.3.2 eI_Resubmitter BOB

The eI_Resubmitter BOB is a placeholder component that the implementor can use to resubmit failed Events back to the e*Insight IQ after repairing them. The Event Repair logic in the eI_Resubmitter BOB's Collaboration must be supplied by the implementor.

Figure 14 eI_Resubmitter BOB Detail



Configuring the eI_Resubmitter BOB

The user must fill in the **eI_Resubmitter** Collaboration with the logic to repair and resubmit Events retrieved from the **eX_Dead_Letter_Queue**.

eI_Resubmitter Collaboration

The **eI_Resubmitter** Collaboration is a placeholder Collaboration that the implementor can use as a starting point to add logic that repairs and resubmits Events that have failed to be processed by the e*Insight engine due to data errors.

Subscribed Event Type: eX_Failed_From_eBPM

This Event Type contains the e*Insight Event that failed to process correctly at the e*Insight engine level due to a data error, along with the error information.

Published Event Type: eX_to_eBPM

This Event Type contains the repaired version of the failed Event to be reprocessed by the e*Insight engine.

5.3.3 Failed Event Handling by the e*Insight Engine

How the e*Insight engine handles errors generated when processing Events, depends on the type of error and on whether custom error handling is enabled in the e*Insight engine's configuration file.

Error Types

Connection errors

Connection errors are errors that the e*Insight receives because of a faulty connection to the e*Insight database.

Data errors

Data errors are exceptions that the e*Insight generates because it cannot process an Event that is sent to it. Also in this class of errors are those generated by the e*Insight because of a faulty business process configuration.

Error Handling

Normal Event Failure Handling

The normal handling of Events that can't be processed due to a connection error is to make a note of the error in the e*Insight engine's log file and retry processing the Event until a connection is made.

The normal e*Insight engine handling of Events that can't be processed due to a data error is, in addition to generating a log entry, to retry processing the Event that generated the error up to a maximum value and then count the Event as failed. Once a certain number of failed Events have been processed by the engine it shuts down. Both the maximum number of resends per Event and the maximum number of failed Events allowed by the e*Insight engine are set in the e*Insight engine's configuration file in the **General Settings** section.

Special Event Failure Handling

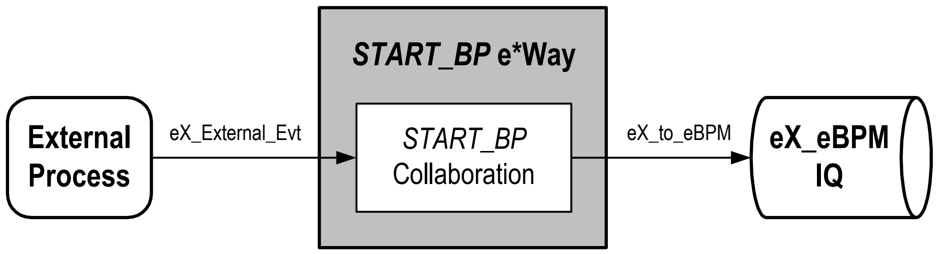
When custom error handling is enabled in the e*Insight engine's configuration file (as it is by default) Events that fail to process due to data errors are handled in a special way. Events that fail to process due to communication errors are not affected by custom error handling. When the e*Insight engine has custom error handling enabled, an Event is not retried, but a notation is made in the e*Insight log file and the Event itself is published to a special IQ. This method allows the e*Insight engine to move on to other processing and not spend time attempting to resend failed Events.

The e*Insight engine publishes the failed Event to the **eX_Dead_Letter_Queue** IQ under the **eX_Failed_From_eBPM** Event Type. The **eI_Resubmitter** BOB subscribes to this Event Type and you can use it to repair the Event and republish it to the **eX_eBPM** IQ. The **eX_Failed_Event.ssc** ETD associated with the **eX_Failed_From_eBPM** Event Type has two major node structures. Once contains error information, and the other contains the failed Event. The failed Event is placed in the CDATA leaf node that can contain XML data.

5.3.4 START_BP Component (e*Way or BOB)

The START_BP component is the e*Gate component that sends the “Start” Event which initiates a business process instance (BPI). This component does not have a corresponding activity in the business process model. The start BP component can be either an e*Way or a BOB depending on the availability of the data within the e*Insight system. If the data required to start a BPI is available within the system, then you can use a BOB to start the BPI. Otherwise, you must use an e*Way to bring the data into the e*Gate environment before it can be used to send the “Start” Event.

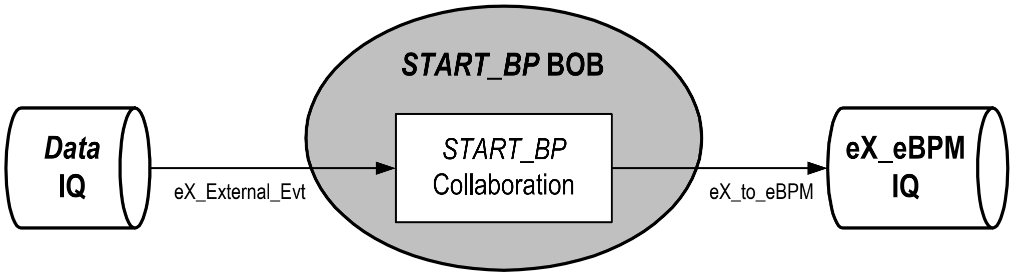
Figure 15 START_BP e*Way Detail



Typically, an e*Way used to start the BPI connects to a business application, which in turn provides the data used by the business process. The type of e*Way chosen depends on the type of business application or external system to which the e*Way must connect in order to bring in the data. For example, if the business application is Siebel, then the e*Way used is the Siebel e*Way.

Using a BOB to start the business process is almost the same as using an e*Way; the only difference is where these two component types get their data. Unlike an e*Way, a BOB gets its data directly from an e*Gate IQ without having to connect to an external system, as shown in Figure 16.

Figure 16 START_BP BOB Detail



Both the BOB and the e*Way starting the BP must put the data they receive into the standard format used throughout e*Insight.

Configuring the START_BP Component

Configuring the START_BP Component depends on the type of component it is. For example, a BOB has no configuration file, and an e*Way's configuration file is different depending on the type of e*Way that is used. See the appropriate e*Way User Guide for information on configuring a specific e*Way. In addition, the implementor must create a Collaboration Rules Script for the START_BP component that constructs the inbound e*Insight Event that starts a business process instance.

START_BP Collaboration

This Collaboration, used by the **START_BP** component, prepares the **eX_to_eBPM** Event. This Event is sent to the e*Insight engine in order to start an instance of the business process. The Collaboration must do two things:

- populate the three nodes required to start a BPI in the e*Insight standard Event
- place the data it receives into one or more global attributes of the business process

Start BP Nodes in the e*Insight Standard Event

The following three nodes in the **eX_Standard_Event.ssc** ETD must be populated in the Event sent to the e*Insight engine in order to start a BPI:

- **BP_EVENT.AS.NAME.Value** must be filled with the exact name of the business process as it appears in the e*Insight GUI
- **BP_EVENT.AS.ID.Value** must be filled with a unique ID (for example, a timestamp)
- **BP_EVENT.AS.TYPE.Value** must be filled with the string "START_BP"

Converting Input Data to e*Insight Format

Data required by the business process must be placed in one or more global attributes of the business process by the **START_BP** Collaboration. In addition, the Collaboration must also convert any data it receives to the XML format used by the e*Insight system.

If any global attribute data contains characters that conflict with the XML structure of the e*Insight Event, then this data must be converted to base 64 encoding prior to sending it into the e*Insight system. You can convert the data in the **START_BP** Collaboration by using the Monk function **raw->base64**.

Note: *Make sure that the **stc_monkutils.dll** that contains the function **raw->base64** is loaded before using **raw->base64** in a Collaboration Rules Script. For example, you may use the command: **load-extension "stc_monkutils.dll"** in the CRS itself or you may put path to a file that loads in the **initialization file** box in the Collaboration Rule that uses the CRS.*

See **"Starting a Business Process (eIJSchema)" on page 87** for more information on how to start a BPI.

Published Event Type: eX_to_eBPM

This "Start" Event carries the data to begin an instance of a particular business process to the e*Insight engine.

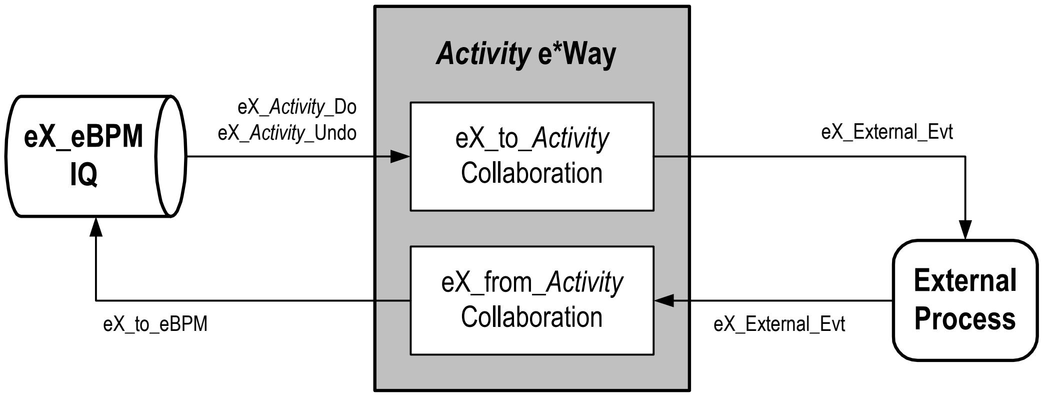
Subscribed Event Type: eX_External_Evt

When using a **START_BP** e*Way, this Event carries the data from the external application to which the **START_BP** e*Way connects. In the case of a **START_BP** BOB, this Event carries data from an e*Gate IQ.

5.3.5 Activity e*Way

An activity e*Way implements an e*Insight activity that requires a connection to a system outside of e*Gate.

Figure 17 Activity e*Way Detail



When you use the e*Insight GUI to configure the e*Gate schema supporting the e*Insight implementation, each activity in the business process becomes either a pair of Collaborations in an e*Way or a single Collaboration in a BOB. The choice to use an e*Way or a BOB to hold the activity Collaboration or Collaborations depends on the preference of the implementor.

For an activity e*Way, the two Collaborations that are created are named **eX_from_*Activity*** and **eX_to_*Activity***, where ***Activity*** is replaced with the **Activity Name** from the associated business process in e*Insight. In addition to the Collaborations, the corresponding Collaboration Rules and the Event Types subscribed to and published by the Collaborations are also named after the activity name.

The Collaboration Rule created in this process is only a placeholder. Implementors must configure the Collaboration Rules by writing the Collaboration Rules Script and choosing the service under which this script runs. The CRS contains the programming that implements the business logic for the corresponding activity in the business process.

The type of e*Way used to implement a particular activity depends on what the activity is supposed to accomplish in the business process. For example, an SAP e*Way could be used to connect to an ERP system to look up the credit standing of a customer, or an Oracle e*Way could be used to look up the mailing address of a prospective client in a marketing database.

Configuring the eX_to_*Activity* e*Way

Some of the configuration for the activity e*Way is done for you when you use the e*Insight GUI to configure the schema. This includes setting up the component relationships and Event Type routing in the e*Insight schema, but not the actual business logic programming or the type of e*Way that is used. The business logic programming must be done by the implementor in Collaboration Rules Scripts used by the activity Collaborations, and the e*Way's configuration file must be defined based on the type of e*Way chosen to implement the activity. See the appropriate e*Way Users Guide for information on how to set up the e*Way chosen.

eX_to_*Activity* Collaboration

The **eX_to_*Activity*** Collaboration receives the Event that carries the data used in the activity. It receives the Event from the e*Insight Engine, and passes it to the external process that implements the activity's business logic. The Collaboration must be configured to convert the data into whatever format is required by the external system to which the activity e*Way connects.

Important: *In addition to passing the attribute data it receives from the e*Insight engine to the external system, the* ***eX_to_Activity*** *Collaboration must preserve the e*Insight Business Process Instance tracking information contained in the* ***eX_Activity_Do*** *or* ***eX_Activity_Undo*** *Events. This information is used to send the return or "Done" Event back to the e*Insight engine, when the activity completes. See* **Sending the "Done" Event Back to e*Insight (eIJSchema)** *on page 88 for more information on what information is required in the "Done" Event.*

Do and Undo logic in an Activity Collaboration

The **eX_to_*Activity*** Collaboration in an activity e*Way and the **eX_Activity** Collaboration in an Activity BOB both subscribe to two Event Types: **eX_*Activity*_Do** and **eX_*Activity*_Undo**. When the activity Collaboration picks up a "Do" Event Type from the **eX_eBPM** IQ, it carries out a "positive" instance of the activity. When the activity Collaboration picks up an "Undo" Event Type from the IQ, it carries out a "negative" or compensating version of the activity—in other words, an activity that cancels out a previously completed "Do" instance of the activity for the current business process instance.

By default, the eX_to_Activity Collaboration in an activity e*Way (eX_Activity in a BOB) subscribe to both the "Do" and "Undo" Events. Consequently the CRS must contain logic to handle both the activity and the compensating transaction for the activity. Of course, the implementor is also free to place the "Undo" logic in a separate Collaboration as long as the **eX_*Activity*_Undo** Event Type is subscribed to and the proper Event is returned to the e*Insight engine.

The e*Insight engine provides local attributes only available to a particular activity. It uses them for holding values set by the "Do" portion of the activity Collaboration. These values can then be used in the "Undo" logic portion of the activity Collaboration to carry out the compensating transaction. That is, these attributes can be set by the "Do" portion of the CRS and then recalled by the "Undo" portion of the CRS in order to cancel out the "Do" when necessary.

For more information on local attributes and where to set them in the e*Insight Standard ETD, see **"Local Attributes" on page 68**.

Subscribed Event Type: eX_*Activity*_Do

This Event causes the subscribing Collaboration to execute the "Do" logic of the corresponding activity in the business process. This Event Type is in standard e*Insight format and contains the current values of any global variables designated as "Input" by the activity in the appropriate location in the **eX_Standard_Event.ssc** ETD.

Subscribed Event Type: eX_*Activity*_Undo

This Event causes the subscribing Collaboration to execute the "Undo" logic of the corresponding activity in the business process. That is, it causes a compensating transaction to occur that "undoes" the completed activity within a BPI (see**"Do and Undo logic in an Activity Collaboration" on page 63** for an explanation of "undoing" an activity). This Event Type is in standard e*Insight format, and contains the current values of any global variables designated as "Input" by the activity in the appropriate location in the **eX_Standard_Event.ssc** ETD. Also, the **eX_*Activity*_Undo** Event contains any local variables set by the Collaboration executing the "Do" logic associated with this activity.

Published Event Type: eX_External_Evt

This Event carries data to the external process that executes the activity. It must be in a form compatible with the external system to which the e*Way connects.

eX_from_*Activity* Collaboration

This Collaboration returns the "Done" Event to the e*Insight engine. To do this, the **eX_from_*Activity*** Collaboration must take the data it receives from the external system and use it to populate the required nodes in the Event returned to the e*Insight engine. Specifically, it must do the following:

- Populate the activity status node on the **eX_Standard_Event.ssc** ETD with the value "SUCCESS" or "FAILURE" depending on whether or not the activity completes successfully.
- Set the values of any global variables designated as "Output" or "Input/Output" by the business process activity.
- Return the e*Insight BPI tracking information included in the Event (either **eX_*Activity*_Do** or **eX_*Activity*_Undo**) that initiated this activity.
- Set the values of any local variables used by the activity.

See **Sending the "Done" Event Back to e*Insight (eIJSchema)** on page 88 for more information.

Subscribed Event Type: eX_External_Evt

This Event contains the result of the completed activity from the external process that executed the activity's business logic.

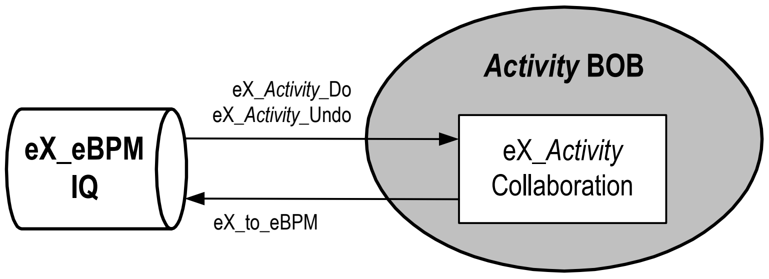
Published Event Type: eX_to_eBPM

This is the "Done" Event that carries the data from a completed activity back to the e*Insight engine.

5.3.6 Activity BOB

The Activity BOB implements an e*Insight activity that does not require a connection to a system outside of e*Gate. A BOB only needs one Collaboration to process the data and return it to the **eX_eBPM** IQ. The CRS associated with the BOB's Collaboration Rule carries out the business logic of the activity to which it corresponds. This Rule could be a Monk script, a Java program, or any other script or application supported by the Collaboration Service under which the CRS runs.

Figure 18 Activity BOB Detail



eX_*Activity* Collaboration

This BOB activity Collaboration fulfills all of the functions that were split into a "to" and a "from" Collaboration in the case of an activity e*Way. In other words, it must:

- copy the e*Insight BPI tracking information to the destination Event in the CRS
- use the values of the "Input" attributes provided by the e*Insight engine in the **eX_*Activity*_Do** (or **eX_*Activity*_Undo**) to complete the business logic for this activity
- implement both the "Do" and "Undo" logic for the activity
- populate the status node (with "SUCCESS" or "FAILURE") depending on the outcome of the activity
- set the values for any "Output" or "Input/Output" attributes
- set the values for any local attributes

Unlike the e*Way activity Collaborations, the BOB Collaboration does *not* need to reformat the data for an external system. The data remains in the standard e*Insight ETD.

Subscribed Event Types:

- **eX_*Activity*_Do**—This Event causes the subscribing Collaboration to execute the "Do" logic of the activity with the same name in the business process. It is the same Event Type as that subscribed to by the **eX_to_*Activity* Collaboration**. See **"Subscribed Event Type: eX_Activity_Do" on page 63** for more information.
- **eX_*Activity*_Undo**—This Event causes the subscribing Collaboration to execute the "Undo" logic of the activity with the same name in the business process. It is the same Event Type as that subscribed to by the **eX_to_*Activity* Collaboration**. See **"Subscribed Event Type: eX_Activity_Undo" on page 64** for more information.

Published Event Type: eX_to_eBPM

This “Done” Event carries the data from a completed activity back to the e*Insight engine. It is the same as that published by the **eX_from_Activity** Collaboration. See **“Published Event Type: eX_to_eBPM” on page 64** for more information.

Chapter 6

Understanding the e*Insight ETD

e*Insight uses a single Event Type Definition (ETD) to define Events as they move from one component to another in the e*Insight system. The ETD is named **eI_StandardEvent.xsc** for Java and **eX_Standard_Event.ssc** for Monk. It is an XML DTD in SeeBeyond's proprietary messaging format. For a description of the XML DTD see **Appendix A**.

All data going into and coming out of the e*Insight components is parsed according to the e*Insight ETD. Understanding this ETD is the key to creating the Collaboration Rules scripts necessary to process the data according to the rules determined by the business process.

Note: *The **eI_StandardEvent.xsc** does not contain a section for the e*Xchange Partner Manager information. If your implementation also uses e*Xchange Partner Manager then you may need to use **eIX_StandardEvent.xsc** as this contains the TP_EVENT location.*
*The TP_EVENT location in the **eX_Standard_Event.ssc** contains information for e*Xchange Partner Manager. You can ignore this section if your implementation does not use e*Xchange Partner Manager.*
*For more information on the TP_EVENT location, refer to the e*Xchange Partner Manager Implementation Guide.*

6.1 Using the ETD with e*Insight

The e*Insight engine uses the e*Insight ETD to carry out the business process. The **BP_EVENT** location in the e*Insight ETD contains data the e*Insight engine uses to track the business process instance. **BP_EVENT** also contains global and local attribute data.

6.1.1 About Business Process Attributes

Business process attributes are defined in the e*Insight GUI and can be global or local.

Global attributes are available to be used and shared by any of the activities in the business process. Local attributes can only be used within a specific activity.

The e*Insight engine uses attributes defined in the business process to send data to Collaborations associated with each business activity and receive data from them.

Global Attributes

You define global attributes as part of creating the business process in the e*Insight GUI. These global attributes make it possible to share data between activities in a business process as well as move data to and from the e*Gate components that implement those activities. The e*Insight GUI allows you to designate whether a particular global attribute is used by a particular activity and whether it is an “Input,” “Output,” or “Input/Output” attribute.

Input Attributes

The current values of global attributes designated as “Input,” for a particular activity in a business process, are included in the e*Insight portion of the **eX_*Activity*_Do** (or Undo) Event.

Important: *The e*Insight engine does **not** send a value for every one of the global attributes in a business process to the current activity. The e*Insight engine **only** sends values for the global attributes designated as “Input” or “Input/Output,” for the current activity, with the **eX_Activity_Do** Event.*

Output Attributes

The e*Insight engine expects global attributes designated as “Output,” to be provided in the “Done” Event it receives from the activity component once the activity has completed. The activity Collaboration that publishes this Event must set the values for the “Output” attributes in its Collaboration Rules script prior to sending the “Done” Event.

Input/Output Attributes

The e*Insight engine populates the **eX_*Activity*_Do** (or Undo) Event with the current values of the global attributes designated as “Input/Output,” for a particular activity in the business process. In addition, the e*Insight engine expects to receive values for global attributes designated as “Input/Output,” in the “Done” Event it receives from a completed activity. The activity Collaboration that publishes this Event must set the values for the “Input/Output” attributes in its Collaboration Rules script.

Local Attributes

Unlike global attributes that can be used by any of the activities within a business process, local attributes are defined only for a specific activity and cannot be used outside that activity. Also, you do *not* need to specify whether they are “Input,” “Output,” or “Input/Output.” In practice they behave as “Output” attributes, because they can be set by the activity Collaboration that publishes the “Done” Event.

You cannot use local attributes to pass information from the **eX_*Activity*_Do** Event to the **eX_*Activity*** Collaboration, because they are empty (null) until they have been set. This is because while local attributes are defined (a placeholder is set up) in the e*Insight GUI, they are not set (given an actual value) until they are set by an activity Collaboration in the “Done” Event that is sent to the e*Insight engine when an activity completes. On the other hand, you can use local attributes to pass information from the **eX_*Activity*_Undo** Event to the **eX_*Activity*** Collaboration. In this case, the value for the attribute could have been previously set by the activity Collaboration in the "Done" Event it sent when it completed the "Do" portion of the activity.

Use local attributes to implement undo logic

Because local attributes are set in the "Done" Event that is sent back to e*Insight after the activity completes, local attributes can be used to store prior state information that would be used by the "Undo" portion of the activity, should a completed "Do" activity need to be undone.

6.2 e*Insight ETD for Java — eI_StandardEvent.xsc

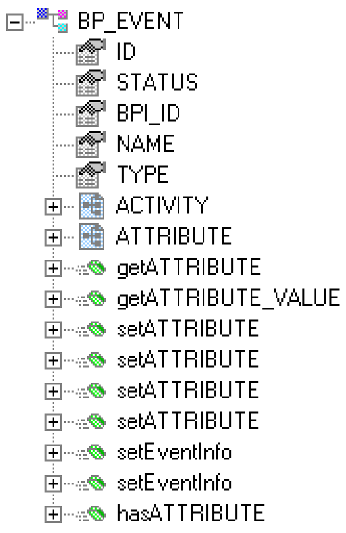
This section describes how the Java Event Type Definition is structured.

Note: *When you install the e*Gate Schema for e*Insight,* ***eIX_StandardEvent.xsc*** *is also created. This ETD has a TP_EVENT location that is used for e*Xchange Partner Manager. You should use this ETD if your implementation requires both e*Insight and e*Xchange. For more information on the TP_EVENT location, refer to the e*Xchange Partner Manager Implementation Guide.*

6.2.1 BP_EVENT

All data relevant to e*Insight processing is contained in the **BP_EVENT** branch of the ETD. The structure is shown in Figure 19.

Figure 19 BP_EVENT



The information is held in three different locations within the Event Type Definition. Each of these nodes contains a different type of information pertinent to e*Insight.

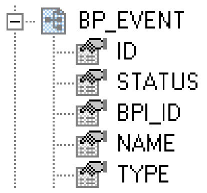
- **BP_EVENT** contains information about the business process. It also contains the child elements ACTIVITY and ATTRIBUTE.

- **ACTIVITY** is an optional element that contains information about the current business process activity and any local attributes that have been defined for that activity.
- **ATTRIBUTE** is a repeating element that contains information about the global attributes for the business process.

BP_EVENT Element

This location in the e*Insight ETD contains general information about the current business process in the five nodes as shown in Figure 31.

Figure 20 BP_EVENT Element



ID

This node must contain a user-assigned unique identifier for the business process instance. This ID could be a time stamp, a document number, or some other ID string.

This node must be populated in the Event that starts a business process instance as well as in the “Done” Event sent back to the e*Insight engine.

STATUS

This node can contain one of the values shown in the following table.

Value	Purpose
“SUCCESS”	Indicates that the current activity completed successfully.
“FAILURE”	Indicates that the current activity did not complete successfully.

The activity Collaboration must set the value of this node in the “Done” Event sent to the e*Insight engine.

BPI_ID

This node contains a value assigned by the e*Insight engine. When e*Insight is running in active mode, including the BPI_ID value in “Done” Event returned to the e*Insight engine after an activity completes speeds up the time it takes for the engine to process the Event.

NAME

This node must contain the name of the business process, exactly (including case) as it appears in the e*Insight GUI.

TYPE

This node must contain one of the values shown in the following table.

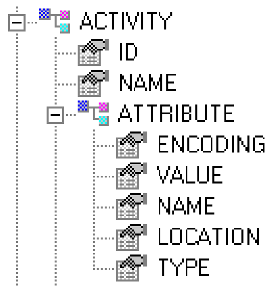
Value	Purpose
"START_BP"	Indicates to the e*Insight engine that this Event starts a BPI.
"DO_ACTIVITY"	Indicates that this is a "Do" Event for the current activity.
"UNDO_ACTIVITY"	Indicates that this is an "Undo" Event for the current activity.

This node must be populated with the string "START_BP" in the Event that starts a BPI.

BP_EVENT.ACTIVITY Nodes

This location in the e*Insight ETD contains information about the current activity. The **ACTIVITY** node contains information of a general nature about the current activity. The **ACTIVITY.ATTRIBUTE** node contains information about any local attributes that have been defined for the current activity. Figure 21 shows the location of these nodes in the e*Insight ETD.

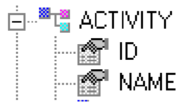
Figure 21 BP_EVENT.ACTIVITY



ACTIVITY Element

This location in the e*Insight ETD contains ID information about the current activity in two nodes as shown in Figure 22.

Figure 22 ACTIVITY Element



ID

This node contains a number assigned by the e*Insight engine for the current activity within a BPI. The e*Insight engine uses this number to speed up processing.

NAME

This node contains the name of the current activity. It must match exactly, including case, the name as it appears in the e*Insight GUI.

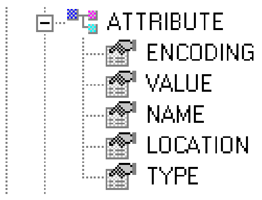
ACTIVITY.ATTRIBUTE Element

This repeating node structure contains the local attribute information defined for the current activity. The structure itself is exactly the same as the global attribute node structure, and holds exactly the same types of data. The only difference is the location in the ETD structure. The following section describes the node structure in the e*Insight ETD used by both global and local attributes.

BP_EVENT.ATTRIBUTE Nodes

This is a repeating node structure that contains the global business process attribute information in five fields as shown in Figure 23.

Figure 23 ATTRIBUTE Element



ENCODING

Describes the type of encoding used to safely convert XML data to an ASCII format. Currently only base 64 encoding is supported.

VALUE

This node contains the current value of the attribute. Events sent to an activity Collaboration have this node populated by the e*Insight engine for attributes designated as “Input” or “Input/Output” in the e*Insight GUI for the current activity. This node must be filled in the “Done” Event sent back to the e*Insight engine by the activity Collaboration, for attributes designated as “Output” or “Input/Output.”

NAME

This node must contain the name of the global attribute. It must match exactly the name as it appears in the e*Insight GUI.

LOCATION

The value in this node describes where the attribute value is located.

Setting this node to a value other than “EMBEDDED” indicates that the data in the **VALUE** field is a pointer (for example, the path to a file) to where the e*Insight engine can *find* the value for the attribute, but *not* actual value itself.

If a value for the **LOCATION** node is not provided (left out of the Event), the e*Insight engine assumes the value is “EMBEDDED”.

This node can contain one of the values from the following table.

SeeBeyond Proprietary and Confidential

Value	Purpose
“FILE”	Indicates that the value for the attribute can be found in the file at the location specified in the **VALUE** field.
“DB”	Indicates that the value for the attribute can be found in the e*Insight database at the location specified in the **VALUE** field.
“URL”	Indicates that the value for the attribute can be found at the URL location specified in the **VALUE** field.
“EMBEDDED”	Indicates that the value for the attribute is contained in the current e*Insight Event in the **VALUE** field. This is the default value.
“AUTO”	Indicates that the value for the attribute is actual data but storage in e*Insight is automatically determined.

TYPE

The value in this node describes the data type of attribute value. This field must contain one of the values from the following table.

Value	Purpose
“BIN”	Indicates that the data in the **VALUE** field is base 64 encoded binary data and is not interpreted as XML by the e*Insight engine.
“XML”	Indicates that the data in the **VALUE** field is XML data that has been encoded using the scheme described in the **ENCODING** field. Currently only base 64 encoding is supported.
“STRING”	Indicates that the data in the **VALUE** field is clear ASCII data, with no characters that could be interpreted as XML tags.
“TRANSIENT”	Indicates that the data in the **VALUE** field is string data that is not stored in the e*Insight database. The e*Insight engine uses special global attributes with this data type to increase its processing speed.
“NUMBER”	Indicates that the data in the **VALUE** field is interpreted as a number. The data is interpreted as a decimal number, however, it must be given as a string.
“BOOLEAN”	Indicates that the data in the **VALUE** field is interpreted as boolean.

6.3 e*Insight ETD for Monk—eX_Standard_Event.ssc

The first step in using the ETD is understanding the structure of the nodes in the context of the XML message being created. This section describes how the Monk Event Type Definition is structured.

6.3.1 ETD Structure

The ETD contains a number of nodes that do not explicitly correlate to the XML DTD but are required by the Monk engine to parse the XML data correctly. Each level is structured in the same way.

Table 6 lists these *facilitator* nodes.

Table 6 Facilitator Nodes in the ETD

Name	Description
CT	A container node for an XML element. This node allows the short and long forms of XML tags to coexist in the structure.
DSN	Identifies a data section within an XML element. This is the long form of the XML tag.
DS	Identifies a data set within an XML element. The sub-elements within a data set can occur in any order.
Empty	The short form of the corresponding DSN node XML tag.
CM	XML comment.
Data	Holds the data for the element.
AS	Identifies an XML attribute set within an XML element.
EQ	The equals sign ("=") within an XML attribute.
Value	Holds the value for the XML attribute.

The facilitator nodes always occur in a set order and define the structure of the XML message. In the e*Insight ETD, the facilitator nodes define three types of branches:

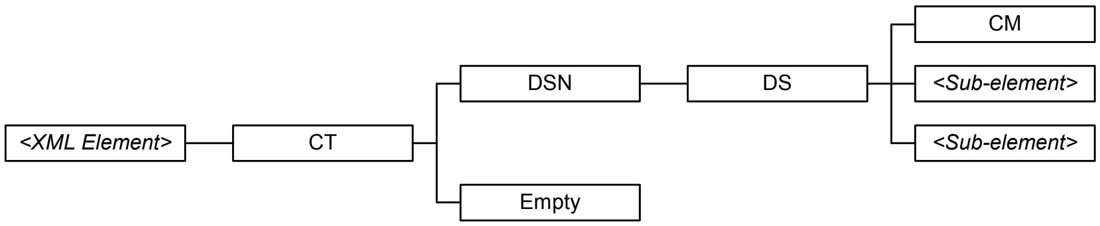
- XML element with sub-elements
- XML element without sub-elements
- XML attribute

6.3.2 XML Element with Sub-elements

The following diagram illustrates the ETD structure for an XML element that has sub-elements.

SeeBeyond Proprietary and Confidential

Figure 24 XML Element with Sub-elements



Each XML element contains one child node, **CT**. **CT** identifies the parent node as an XML element. The **CT** node contains two child nodes: **DSN** and **Empty**. **DSN** is the long form of the XML tag (</*tag*>) and **Empty** is the short form (</>).

The **DSN** and **DS** nodes always occur as parent-child pairs. In this type of branch, **DS** is the parent node for two types of child nodes:

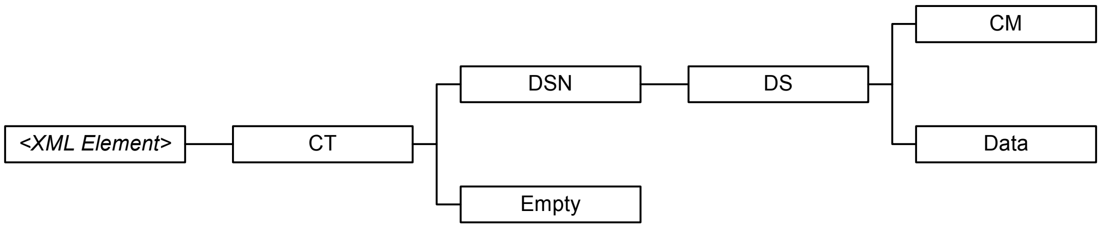
- **CM**, which holds XML comments for the element
- ***<sub-element>***, the name of a sub-element of the parent element

The **DS** node always contains a **CM** child node to hold XML comments. Each ***<sub-element>*** node contains an ETD structure of its own, with the ***<sub-element>*** node as the parent node for the branch.

6.3.3 XML Element without sub-elements

The following diagram illustrates the ETD structure for an XML element that does not have sub-elements.

Figure 25 XML Element without sub-elements

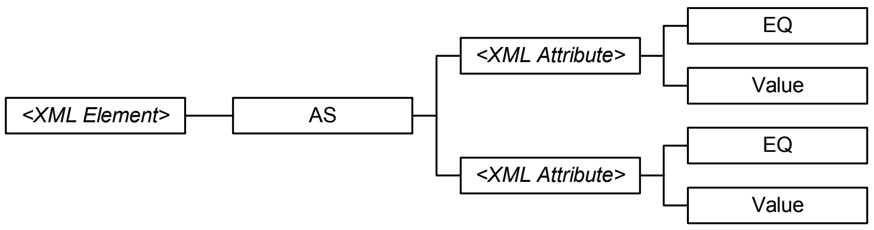


Notice that the only difference between this diagram and the previous diagram is a **Data** child node in place of the ***<sub-element>*** child nodes above. The **Data** node contains the actual data for the XML element that is defined. When creating Collaboration Rules scripts, you must map the XML element data to the **Data** nodes at the terminal end of the element's branch.

6.3.4 XML Attribute

The following diagram illustrates the ETD structure for an XML attribute.

Figure 26 XML Attribute

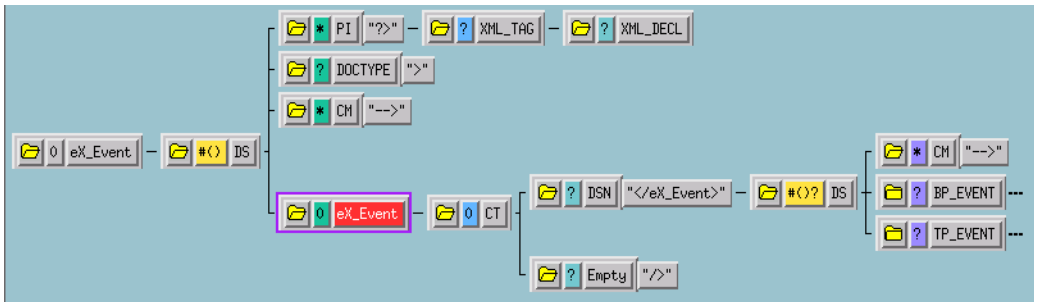


In this case, the XML element contains one child node, **AS**, which identifies the branch as XML attributes of the parent element. The **AS** node contains the ***<XML Attribute>*** nodes as child nodes. Each ***<XML Attribute>*** node has two child nodes: **EQ** to represent the equal sign (=) in the attribute and **Value** which holds the actual value for the attribute. When creating Collaboration Rules scripts, you must map the XML attribute value to the **Value** nodes at the terminal end of the attribute's branch.

6.3.5 Element Overview

The following diagram illustrates the entire e*Insight ETD tree. Note that this is only a diagrammatic representation of the tree, since the actual tree conforms to the node structure described in **"e*Insight ETD for Monk—eX_Standard_Event.ssc" on page 74**.

Figure 27 The e*Insight ETD



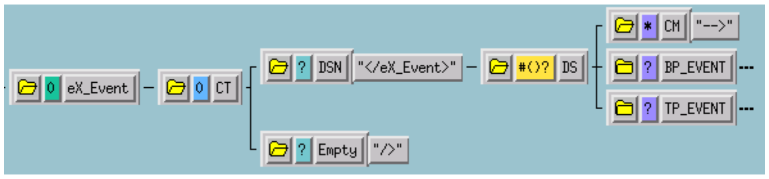
All data pertinent to e*Insight is contained in the XML element **eX_Event**. **eX_Event** contains two distinct "trees": **BP_EVENT** and **TP_EVENT**. **BP_EVENT** contains all of the information pertaining to e*Insight. **TP_EVENT** contains all of the information pertaining to e*Xchange. Both **BP_EVENT** and **TP_EVENT** are optional nodes in the ETD. So if you use e*Insight to track business process activities but do not use e*Xchange to send data to and from trading partners, you do not need to populate the **TP_EVENT** element. Conversely, if you use e*Xchange to send data to and from trading partners but do not track business process activities in e*Insight, you do not need to populate the **BP_EVENT** element in your Collaboration Rules scripts.

Example: XML Element with Sub-elements

eX_Event is an example of a top-level XML element.

In this example, the **CT**, **DSN**, **DS**, **Empty**, and **CM** facilitator nodes describe the top-level XML element **eX_Event**. Figure 28 shows the ETD structure for this element.

Figure 28 XML Element **eX_Event**



The **eX_Event** parent node contains one child node, **CT**. **CT** identifies **eX_Event** as an XML element. The **CT** node contains two child nodes: **DSN** and **Empty**. **DSN** is the long form of the XML tag (</eX_Event>) and **Empty** is the short form (</>).

The **DSN** and **DS** nodes always occur as parent-child pairs. **DS** is the parent node for three child nodes:

- A **CM** node to hold XML comments for the element.
- **BP_EVENT**, a sub-element of **eX_Event**.
- **TP_EVENT**, a sub-element of **eX_Event**.

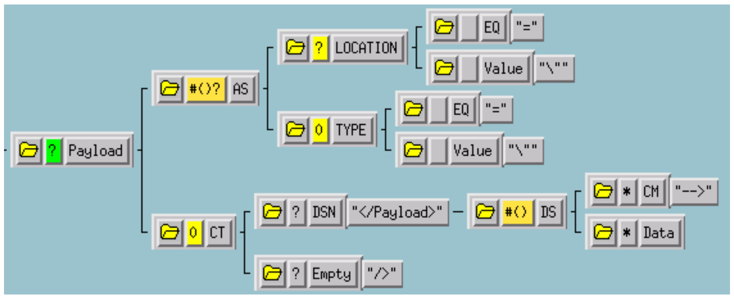
The **DS** node always contains a **CM** child node to hold XML comments. In this example, the eX_Event element does not hold data directly, but contains two sub-elements—**BP_EVENT** and **TP_EVENT**—which have similar facilitator node branches associated with them.

The following example explains the structure of XML attributes.

Example: XML Element with Attributes

In this example, the **AS** and **EQ** facilitator nodes describe the XML attributes **TYPE** and **LOCATION**. Both are XML attributes of the **Payload** element. Figure 29 shows the ETD structure for these attributes.

Figure 29 XML Attribute **Type**



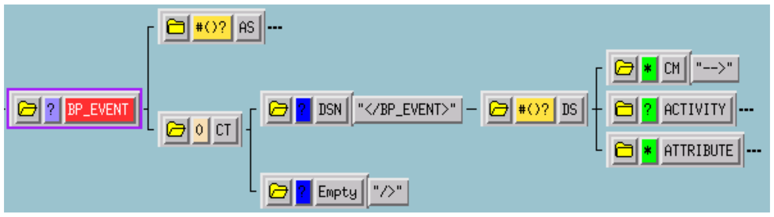
6.4 Using eX_Standard_Event.ssc

This section describes how **eX_Standard_Event.ssc** is populated.

6.4.1 BP_EVENT

All data relevant to e*Insight processing is contained in the **BP_EVENT** branch of the ETD.

Figure 30 BP_EVENT



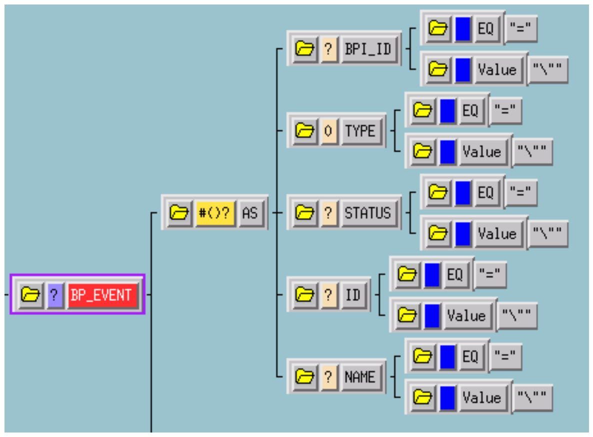
Three of the nodes shown in Figure 30 are collapsed (followed by three dashes) indicating there are additional nodes underneath these nodes. Each of these nodes contains a different type of information pertinent to e*Insight.

- **BP_EVENT.AS** contains information about the business process.
- **BP_EVENT.CT.DSN.DS.ACTIVITY** is an optional node that contains information about the current business process activity and any local attributes that have been defined for that activity.
- **BP_EVENT.CT.DSN.DS.ATTRIBUTE** is a repeating node that contains information about the global attributes for the business process.

BP_EVENT.AS Nodes

This location in the e*Insight ETD contains general information about the current business process in the five nodes as shown in Figure 31.

Figure 31 BP_EVENT.AS



BPI_ID

This node contains a value assigned by the e*Insight engine. When e*Insight is running in active mode, including the BPI_ID value in "Done" Event returned to the e*Insight engine after an activity completes speeds up the time it takes for the engine to process the Event.

TYPE

This node must contain one of the values shown in the following table.

Value	Purpose
"START_BP"	Indicates to the e*Insight engine that this Event starts a BPI.
"DO_ACTIVITY"	Indicates that this is a "Do" Event for the current activity.
"UNDO_ACTIVITY"	Indicates that this is an "Undo" Event for the current activity.

This node must be populated with the string "START_BP" in the Event that starts a BPI.

STATUS

This node can contain one of the values shown in the following table.

Value	Purpose
"SUCCESS"	Indicates that the current activity completed successfully.
"FAILURE"	Indicates that the current activity did not complete successfully.

The activity Collaboration must set the value of this node in the "Done" Event sent to the e*Insight engine.

ID

This node must contain a user-assigned unique identifier for the business process instance. This ID could be a time stamp, a document number, or some other ID string.

This node must be populated in the Event that starts a business process instance as well as in the "Done" Event sent back to the e*Insight engine.

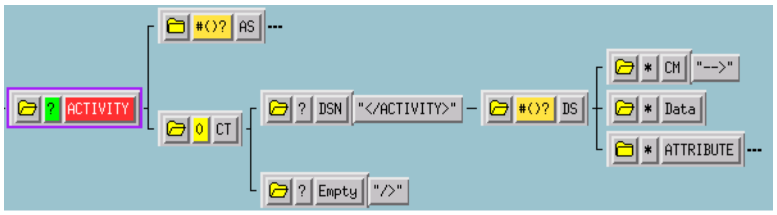
NAME

This node must contain the name of the business process, exactly (including case) as it appears in the e*Insight GUI.

BP_EVENT.CT.DSN.DS.ACTIVITY Nodes

This location in the e*Insight ETD contains information about the current activity. The **ACTIVITY.AS** node contains information of a general nature about the current activity. The **ACTIVITY.CT.DSN.DS.ATTRIBUTE** node contains information about any local attributes that have been defined for the current activity. Figure 32 shows the location of these nodes in the e*Insight ETD.

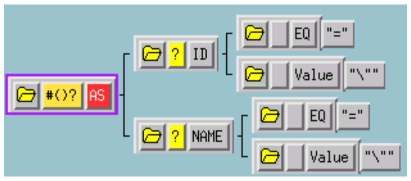
Figure 32 BP_EVENT.CT.DSN.DS.ACTIVITY



ACTIVITY.AS Nodes

This location in the e*Insight ETD contains ID information about the current activity in two nodes as shown in Figure 33.

Figure 33 ACTIVITY.AS



ID

This node contains a number assigned by the e*Insight engine for the current activity within a BPI. The e*Insight engine uses this number to speed up processing.

NAME

This node contains the name of the current activity. It must match exactly, including case, the name as it appears in the e*Insight GUI.

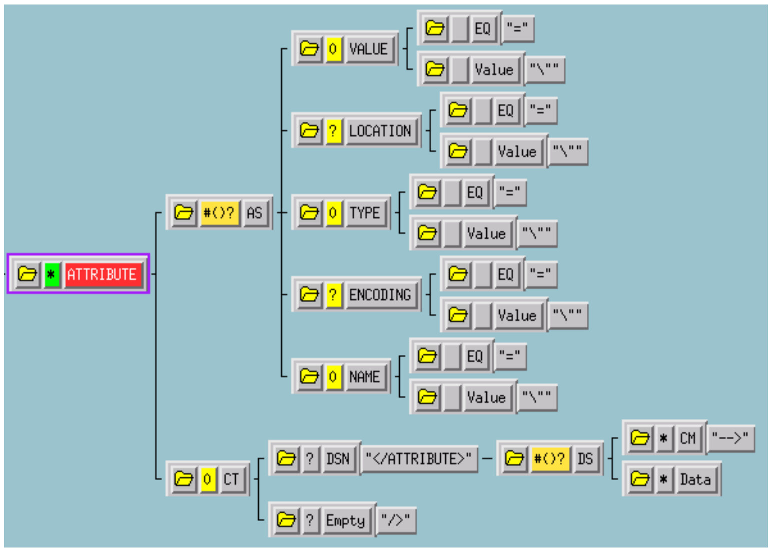
ACTIVITY.CT.DSN.DS.ATTRIBUTE Nodes

This repeating node structure contains the local attribute information defined for the current activity. The structure itself is exactly the same as the global attribute node structure, and holds exactly the same types of data. The only difference is the location in the ETD structure. The following section describes the node structure in the e*Insight ETD used by both global and local attributes.

BP_EVENT.CT.DSN.DS.ATTRIBUTE.AS Nodes

This is a repeating node structure that contains the global business process attribute information in five sub-node locations as shown in Figure 34:

Figure 34 BP_EVENT.CT.DSN.DS.ATTRIBUTE



*Note: The **ATTRIBUTE.CT** node structure is not used in e*Insight processing, but is needed in the e*Insight ETD for correct XML parsing.*

VALUE

This node contains the current value of the attribute. Events sent to an activity Collaboration have this node populated by the e*Insight engine for attributes designated as “Input” or “Input/Output” in the e*Insight GUI for the current activity. This node must be filled in the “Done” Event sent back to the e*Insight engine by the activity Collaboration, for attributes designated as “Output” or “Input/Output.”

LOCATION

The value in this node describes where the attribute value is located.

Setting this node to a value other than “EMBEDDED” indicates that the data in the **VALUE.Value** node is a pointer (for example, the path to a file) to where the e*Insight engine can *find* the value for the attribute, but *not* actual value itself.

If a value for the **LOCATION** node is not provided (left out of the Event), the e*Insight engine assumes the value is “EMBEDDED”.

This node can contain one of the values from the following table.

Value	Purpose
"FILE"	Indicates that the value for the attribute can be found in the file at the location specified in the **VALUE.Value** node.
"DB"	Indicates that the value for the attribute can be found in the e*Insight database at the location specified in the **VALUE.Value** node.
"URL"	Indicates that the value for the attribute can be found at the URL location specified in the **VALUE.Value** node.
"EMBEDDED"	Indicates that the value for the attribute is contained in the current e*Insight Event in the **VALUE.Value** node. This is the default value.
"AUTO"	Reserved for future use.

TYPE

The value in this node describes the data type of attribute value. This node must contain one of the values from the following table.

Value	Purpose
"BIN"	Indicates that the data in the **VALUE.Value** node is base 64 encoded binary data and is not interpreted as XML by the e*Insight engine.
"XML"	Indicates that the data in the **VALUE.Value** node is XML data that has been encoded using the scheme described in the **ENCODING** node. Currently only base 64 encoding is supported.
"STRING"	Indicates that the data in the **VALUE.Value** node is string data.
"TRANSIENT"	Indicates that the data in the **VALUE.Value** node is string data that is not stored in the e*Insight database. The e*Insight engine uses special global attributes with this data type to increase its processing speed.
"NUMBER"	Indicates that the data in the **VALUE.Value** node is interpreted as a number. The data is interpreted as a decimal number, however, it must be given as a string.
"BOOLEAN"	Indicates that the data in the **VALUE.Value** node is interpreted as boolean.

ENCODING

Describes the type of encoding used to safely convert XML data to an ASCII format. Currently only base 64 encoding is supported.

NAME

This node must contain the name of the global attribute. It must match exactly the name as it appears in the e*Insight GUI.

Chapter 7

Common Configuration Tasks

This chapter provides a configuration information for common implementation tasks. The chapter starts with a review of the e*Insight implementation road map and then looks in detail at certain tasks that are performed within an e*Insight implementation.

The tasks covered in this chapter include copying the e*Insight Schema and creating messages to send to the e*Insight engine.

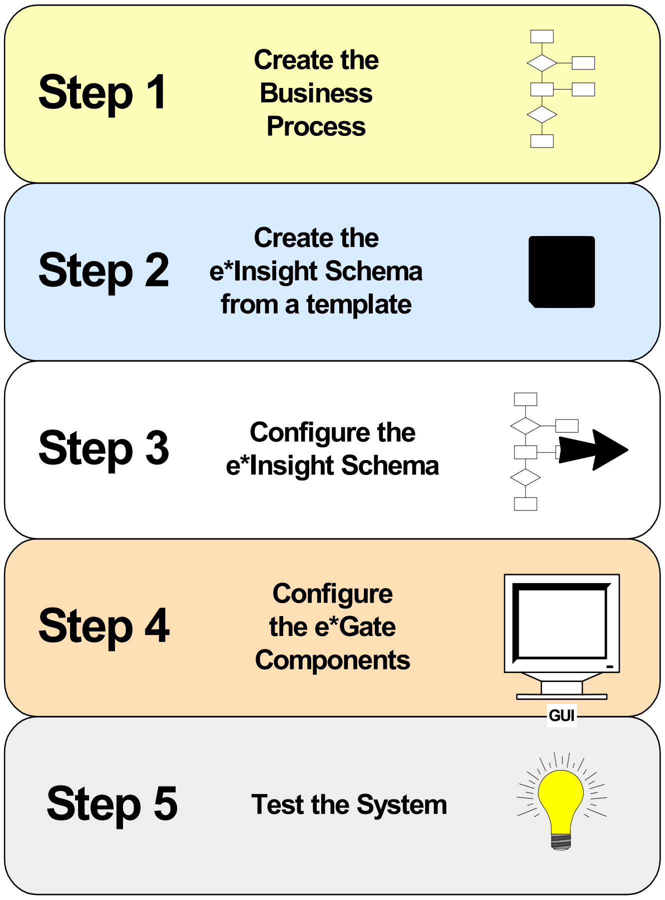
This chapter also contains instructions for deleting business processes and business process instances from the database.

7.1 Implementation Road Map

Before considering individual tasks, we will first review the implementation road map for an e*Insight implementation.

Figure 35, illustrates the major steps in the integration process for an e*Insight implementation.

Figure 35 Integration Road Map


Step 1
Create the Business Process
Step 2
Create the e*Insight Schema from a template
Step 3
Configure the e*Insight Schema
Step 4
Configure the e*Gate Components
GUI
Step 5
Test the System

7.2 Common Configuration Tasks

The following are common tasks that an implementor must perform in the course of setting up the e*Gate components, both to carry out the business process and to interact with the e*Insight system.

7.2.1 Copy the e*Insight Schema

When beginning an integration project, make a copy of the e*Insight schema, that is installed from the CD. Don't make any modifications to e*Insight Schema itself; keep it as a template. Make changes to the copy of the e*Insight Schema that you create. Use this copy as your starting point in e*Gate for supporting e*Insight.

When you install the e*Gate Schema for e*Insight from the CD a number of different components are installed on your Registry Host. Firstly, you install the eIJSchema and/or eISchema that can be seen in the e*Gate Enterprise Manager GUI. Secondly, you install the support files that are used by the Schema into the Default repository, for example the configuration files, Event Type Definitions, and Collaboration Rules Scripts. You also install some files required for e*Gate to communicate with e*Insight, for example, workflow.jar.

There are three ways to copy the e*Insight Schema; you can use the e*Insight GUI, create a copy of the schema from e*Gate Enterprise Manager, or install from the CD. These methods are described below. The method that you choose depends on your requirements.

Using the e*Insight GUI

When you install the e*Insight GUI a text file is created on your local machine containing e*Insight schema information. This text file is used to create your schema on the Registry host. This process does not effect any of the files associated with the e*Insight schemas.

Note: *If you have changed the eIJSchema or eISchema on the Registry host, these changes do not appear in your new schema since the new schema is based on the text file installed with the e*Insight GUI.*

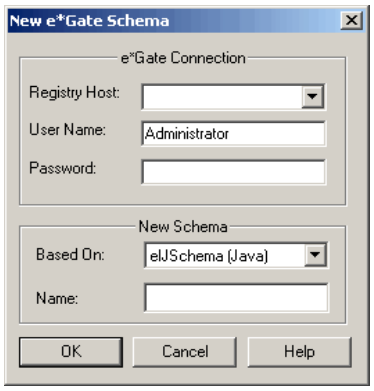
To create a new e*Insight Schema using the e*Insight GUI

1 Open the e*Insight GUI.

2 From the File menu, select Create New e*Gate Schema.

3 The New e*Gate Schema dialog appears. See Figure 36.

Figure 36 New e*Gate Schema Dialog



Registry Host:
User Name: Administrator
Password:
Based On: eIJSchema (Java)
Name:

4 Enter or select the name of the Registry Host where you want to create the schema.

5 Enter a user name and password that are valid on the Registry Host.

6 From the Based On drop-down list, select either **eIJSchema** (Java) or **eISchema** (Classic).

7 Enter a name for the schema in the **Name** box.

8 Click **OK**.

Copying the Schema from the Registry Host

This process makes an exact copy of the current eIJSchema or eISchema held on the Registry Host. You may want to use this method if you have modified the eIJSchema or eISchema since the original installation from the CD. This process does not effect any of the files associated with the e*Insight schemas.

To create a copy of the e*Insight Schema

1 Open the eISchema in the e*Gate Enterprise Manager GUI.

   A Start the e*Gate Enterprise Manager.

   B Log in to eIJSchema or eISchema.

2 Export the schema to a file **c:\eGate\client\<*e*Insight schema backup file name*>**.

   A Select **Export Schema Definitions to File ...** from the **File** pull-down menu.

   B In the **Select archive File** dialog box enter <*e*Insight schema backup file name*> in the **File name** text box, and then click **Save**.

3 Create a new schema using the export file as a template.

   A Select **New Schema** from the **File** pull-down menu.

   B Enter <*new e*Insight schema name*> in the text box.

   C Mark the **Create from export** check box.

   D Click **Find** and browse for the <*e*Insight schema backup file name*> file created in step 2 above.

   E Click **Open**.

   The Enterprise Manager creates a copy of the eIJSchema or eISchema with the schema name entered in step 3B above.

Installing from the CD

You can install a new schema from the CD. Refer to the *e*Insight Business Process Manager Installation Guide* for installation instructions. You should define a unique name for your schema so it does not overwrite an existing schema.

Note: *Installing from the CD installs the eIJSchema and/or eISchema that can be seen in the e*Gate Enterprise Manager GUI. It also re-installs the support files that are used by the Schema into the Default repository and the files required for e*Gate to* *communicate with e*Insight, for example, workflow.jar. If you have changed these files, either manually or via an ESR, then these changes are overwritten.*

7.3 Sending Messages to the e*Insight Engine (eIJSchema)

7.3.1 Starting a Business Process (eIJSchema)

The Collaboration in the e*Way (or BOB) that feeds data into the business process must publish an **eX_to_eBPM** Event Type to the **eIcp_eInsightJMS** connection point. This "Start" Event must include the following:

- the name of the business process
- a unique ID for the business process instance
- an event type of "START_BP"
- all the input global attributes required for the event

When the e*Insight engine receives this Event, it creates a new instance of the business process.

The business process name, unique ID, and "START_BP" event type are set by setting a value in the relevant node. The three nodes required to start the BPI populated in the **eI_StandardEvent.xsc** ETD are described in Table 7.

Table 7 Configuring a start message

eI_StandardEvent	How to populate
BP_EVENT.NAME	Must be filled with the exact name of the BP as it appears in the e*Insight GUI.
BP_EVENT.ID	Must be filled with a unique ID (for example, a timestamp).
BP_EVENT.TYPE	Must be filled with the string "START_BP".

For information on setting global attribute values, see **"Setting Attributes (eIJSchema)" on page 87**.

7.3.2 Setting Attributes (eIJSchema)

You must set the value of a global attribute for a business process that has been designated as an "Output" attribute for that activity in the e*Insight GUI, or designated as an "Input" attribute when starting a business process. To do this you create an activity Collaboration that sends the **eX_to_eBPM** Event back to the e*Insight engine.

You can use the setATTRIBUTE method in your Collaboration Rules script to set the value of a global attribute. The syntax is:

```
setATTRIBUTE(java.lang.String name, java.lang.String type,
java.lang.String value)
```

- replace **java.lang.String name** with the exact name of the attribute as it appears in the e*Insight GUI
- replace **java.lang.String type** with one of the strings "STRING", "XML", "BIN", "NUMBER", or "BOOLEAN" as appropriate
- replace **java.lang.String value** with the attribute value

7.3.3 Getting Attributes (eIJSchema)

You must get the value of a global attribute for a business process that has been designated as an "Input" attribute for that activity in the e*Insight GUI. To do this, create an activity Collaboration that receives the **eX_Activity_Do (or Undo)** Event sent from the e*Insight engine.

You can use the **getATTRIBUTE_VALUE** helper function to retrieve the value of an attribute. The syntax is:

```
void getATTRIBUTE_VALUE(java.lang.String name)
```

- replace **java.lang.String name** with the exact name of the attribute as it appears in the e*Insight GUI

7.3.4 Sending the "Done" Event Back to e*Insight (eIJSchema)

When an activity completes (successfully or not), a "Done" Event must be sent back to the e*Insight engine carrying the status of the activity. To do this, the activity Collaboration sending this Event must publish an **eX_to_eBPM** Event Type to the **eIcr_eInsightJMS** connection point.

Note: *The User Activity, and Authorization Activity do not require a "Done" Event to be sent back to the e*Insight engine.*

The "Done" Event must have the following nodes in **eI_StandardEvent** (Java) ETD populated. Table 8 describes the required nodes when an activity is set to Active control and Table 9 describes the required nodes when an activity is set to Passive control.

Table 8 Done Event in Active control mode

Java Node Location	How to populate
BP_EVENT.BPI_ID	Copy from source to destination.
BP_EVENT.ID	Copy from source to destination.
BP_EVENT.NAME	Copy from source to destination.
BP_EVENT.TYPE	Copy from source to destination.
BP_EVENT.STATUS	Copy either the string "SUCCESS" or "FAILURE", depending on whether or not the activity completed successfully.
BP_EVENT.ACTIVITY.ID	Copy from source to destination. (Not absolutely required, but recommended to speed processing.)
BP_EVENT.ACTIVITY.NAME	Copy from source to destination.

Java Node Location	How to populate
BP_EVENT.ATTRIBUTE[i].VALUE	Use the e*Insight helper function **getATTRIBUTE_VALUE** to determine the value of the machine-assigned attribute, **eX_eBPMServer**, in the source ETD. Copy this value to the destination ETD. Make sure the instance [i] to which you copy does not overwrite or append to an existing iteration of the destination ETD's repeating node. ***(Active mode only)***
BP_EVENT.ATTRIBUTE[i].TYPE	Copy the string "TRANSIENT". Make sure the instance [i] to which you copy does not overwrite or append to an existing iteration of the destination repeating node. ***(Active mode only)***
BP_EVENT.ATTRIBUTE[i].NAME	Copy the string "eX_eBPMServer". Make sure the instance [i] to which you copy does not overwrite or append to an existing iteration of the destination repeating node. ***(Active mode only)***

Note: *Make sure that setting the "Output" or "Input/Output" attributes does not conflict with setting the e*Insight engine assigned* ***eX_eBPMServer*** *attribute, set in the same repeating node.*

Using a copy of the entire BP_Event to send the "Done" Event

You can copy the entire BP_EVENT source node to the BP_EVENT destination node instead of copying each field individually. In addition to the necessary information for the "Done" Event, you are also copying the attributes that have a direction defined in the activity of **Input** or **Input/Output**. You can send attributes that are only defined as **Input** attributes in the "Done" event, but since they are not defined as **Output**, the engine cannot update the database. The activity works correctly, but the engine has to perform unnecessary processing when it tries to update the database.

Alternatively, you can use the **removeATTRIBUTE** method in a Collaboration to remove attributes from the "Done" Event that is sent back to the e*Insight Engine. The following example code copies everything in the BP_EVENT node to the output, removes the "Customer_Name" global attribute, and then sets the value of the BP status node to "SUCCESS".

```
getEIStandardInOut().setBP_EVENT(getEIStandardInOut().getBP_EVENT())
getEIStandardInOut().getBP_EVENT().removeATTRIBUTE("Customer_Name")
getEIStandardInOut().getBP_EVENT().setSTATUS("SUCCESS")
```

Table 9 Done Event in Passive control mode

Java Node Location	How to populate
BP_EVENT.ID	Copy from source to destination. It is the responsibility of the developer to ensure that this value is stored within the event as it is passed through e*Gate.

Java Node Location	How to populate
BP_EVENT.NAME	The business processes name is hard-coded and is the name of the business process for the immediate parent of this activity (this is very important for sub-processes).
BP_EVENT.TYPE	Set the string to "DO_ACTIVITY".
BP_EVENT.STATUS	Set the string to either "SUCCESS", or "FAILURE", depending on whether or not the activity completed successfully.
BP_EVENT.ACTIVITY.ID	Leave this blank. In Passive mode, the source may not contain the correct ID.
BP_EVENT.ACTIVITY.NAME	Set the string to contain the activity name.

In addition, the "Done" Event must carry with it the values for any attributes (global or local) specified as "Output" or "Input/Output". These can be set as shown in **"Setting Attributes (eIJSchema)" on page 87**.

7.4 Sending Messages to the e*Insight Engine (eISchema)

7.4.1 Starting a Business Process (eISchema)

The Collaboration in the e*Way (or BOB) that feeds data into the business process must publish an **eX_to_eBPM** Event Type. This "Start" Event must include the following:

- the name of the business process
- a unique ID for the business process instance
- an event type of "START_BP"
- all the input global attributes required for the event

When the e*Insight engine receives this Event, it creates a new instance of the business process.

The business process name, unique ID, and "START_BP" event type are set by setting a value in the relevant node. The three nodes required to start the BPI populated in the **eX_Standard_Event.ssc** (Monk), or **eI_StandardEvent.xsc** (Java) ETDs are described in Table 7.

Table 10 Configuring a start message (eISchema)

Monk Node Location	Java Node Location	How to populate
BP_EVENT.AS.NAME.Value	BP_EVENT.NAME	Must be filled with the exact name of the BP as it appears in the e*Insight GUI.
BP_EVENT.AS.ID.Value	BP_EVENT.ID	Must be filled with a unique ID (for example, a timestamp).

Table 10 Configuring a start message (eISchema)

Monk Node Location	Java Node Location	How to populate
BP_EVENT.AS.TYPE.Value	BP_EVENT.TYPE	Must be filled with the string "START_BP".

For information on setting global attribute values, see **"Setting Attributes (eIJSchema)" on page 87**.

7.4.2 Setting Attributes (eISchema)

You must set the value of a global attribute for a business process that has been designated as an "Output" attribute for that activity in the e*Insight GUI, or designated as an "Input" attribute when starting a business process. To do this you create an activity Collaboration that sends the **eX_to_eBPM** Event back to the e*Insight engine. The methods of achieving this in both a Monk and Java Collaboration are discussed below.

Setting Attributes in a Monk Collaboration

The Monk Collaboration must do one of the following:

- populate the correct nodes in the **eX_Standard_Event.ssc**
- use the e*Insight helper function **eX-set-attribute**

Setting Attributes by Populating Nodes in the e*Insight Standard ETD

Your Collaboration must populate three required nodes in the **eX_Standard_Event.ssc** ETD for each attribute that must be set. Set them as follows:

- **BP_EVENT.CT.DSN.DS.ATTRIBUTE[n].AS.VALUE.Value**, with the attribute value
- **BP_EVENT.CT.DSN.DS.ATTRIBUTE[n].AS.NAME.Value**, with the exact name of the attribute as it appears in the e*Insight GUI
- **BP_EVENT.CT.DSN.DS.ATTRIBUTE[n].AS.TYPE.Value**, with one of the strings "STRING", "XML", or "BIN" as appropriate

Note: *In the above list,* ***n*** *= 0 for the first attribute that you set, 1 for the next, and so on. Also, be sure to increment the index to prevent overwriting data in the destination Event. If an attribute already exists (for example, eX_eBPMServer) then you should ensure that this does not get overwritten.*

Setting Attributes by Using the eX-set-attribute Helper Function

Rather than use three COPY statements in your Collaboration Rules script, you can use the **eX-set-attribute** helper function instead. The syntax is:

```
eX-set-attribute <root-path> <attribute> <value> <type>
```

- replace ***<root-path>*** with **~input%eX_Event** or **~output%eX_Event**

- replace ***<attribute>*** with the exact name of the attribute as it appears in the e*Insight GUI
- replace ***<value>*** with the attribute value
- replace ***<type>*** with one of the strings "STRING", "XML", or "BIN" as appropriate

Important: *Make sure that the Monk file **eX-eBPM-utils.monk**, containing the e*Insight helper functions, are loaded before calling them in a Collaboration Rules Script. You can do this in several ways, by putting them in the root of the **monk_library** directory, loading them explicitly in your CRS, or using the **eX-init-eXchange** bootstrap file to load them via the Collaboration Rule.*

Setting Attributes in a Java Collaboration

You can use the setATTRIBUTE method in your Collaboration Rules script to set the value of a global attribute. The syntax is:

```
setATTRIBUTE(java.lang.String name, java.lang.String type,
java.lang.String value)
```

- replace **java.lang.String name** with the exact name of the attribute as it appears in the e*Insight GUI
- replace **java.lang.String type** with one of the strings "STRING", "XML", "BIN", "NUMBER", or "BOOLEAN" as appropriate
- replace **java.lang.String value** with the attribute value

7.4.3 Getting Attributes (eISchema)

You must get the value of a global attribute for a business process that has been designated as an "Input" attribute for that activity in the e*Insight GUI. To do this, create an activity Collaboration that receives the **eX_Activity_Do (or Undo)** Event sent from the e*Insight engine. The methods of achieving this in both a Monk and Java Collaboration are discussed below.

Getting Attributes in a Monk Collaboration

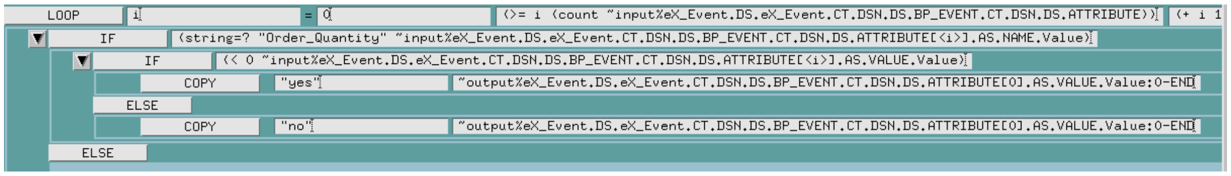
This Collaboration must do one of the following:

- retrieve the value from the correct node in the **eX_Standard_Event.ssc**
- use the e*Insight helper function **eX-get-attribute**

Getting Attributes by Copying from Nodes in the e*Insight Standard ETD

To get the value from the correct node, your Collaboration must systematically search through all of the attribute nodes until it finds the one containing the value for the required attribute. Figure 37, on the next page, shows an example of a LOOP rule that does this.

Figure 37 Get Attribute Loop



```
LOOP  i  =  0  (>= i (count ~input%eX_Event.DS.eX_Event.CT.DSN.DS.BP_EVENT.CT.DSN.DS.ATTRIBUTE))  (+ i 1
  IF  (string=? "Order_Quantity" ~input%eX_Event.DS.eX_Event.CT.DSN.DS.BP_EVENT.CT.DSN.DS.ATTRIBUTE[<i>].AS.NAME.Value)
    IF  (< 0 ~input%eX_Event.DS.eX_Event.CT.DSN.DS.BP_EVENT.CT.DSN.DS.ATTRIBUTE[<i>].AS.VALUE.Value)
      COPY  "yes"  ~output%eX_Event.DS.eX_Event.CT.DSN.DS.BP_EVENT.CT.DSN.DS.ATTRIBUTE[0].AS.VALUE.Value:0-END
    ELSE
      COPY  "no"  ~output%eX_Event.DS.eX_Event.CT.DSN.DS.BP_EVENT.CT.DSN.DS.ATTRIBUTE[0].AS.VALUE.Value:0-END
  ELSE
```

Using an IF statement inside the LOOP, this CRS checks the repeating node **BP_EVENT.CT.DSN.ATTRIBUTE** for the specified attribute. When it finds the attribute (in the above example, "Order_Quantity"), the node **ATTRIBUTE[<i>].AS.VALUE.Value** contains the attributes value.

Once found, the script can use the value of the attribute to carry out the business logic of the Collaboration. In the above example, a check is made to see if the value in question is greater than zero and if it is to set the value of an output attribute to "yes".

Getting Attributes by Using the eX-get-attribute Helper Function

Rather than use a LOOP in your Collaboration to obtain the value of an Input attribute you can use the **eX-get-attribute** helper function instead. The syntax is:

```
eX-get-attribute <root-path> <attribute>
```

- replace ***<root-path>*** with **~input%eX_Event** or **~output%eX_Event**
- replace ***<attribute>*** with the exact name of the attribute as it appears in the e*Insight GUI

Getting Attributes in a Java Collaboration

You can use the **getATTRIBUTE_VALUE** helper function to retrieve the value of an attribute. The syntax is:

```
void getATTRIBUTE_VALUE(java.lang.String name)
```

- replace **java.lang.String name** with the exact name of the attribute as it appears in the e*Insight GUI

7.4.4 Sending the "Done" Event Back to e*Insight (eISchema)

When an activity completes (successfully or not), a "Done" Event must be sent back to the e*Insight engine carrying the status of the activity. To do this, the activity Collaboration sending this Event must publish an **eX_to_eBPM** Event Type to the **eX_eBPM** IQ.

Note: *The User Activity, and Authorization Activity do not require a "Done" Event to be sent back to the e*Insight engine.*

The "Done" Event must have the following nodes in either the **eX_Standard_Event.ssc** (Monk), or **eI_Standard_Event** (Java) ETD populated. Table 8 describes the required nodes when an activity is set to Active control and Table 9 describes the required nodes when an activity is set to Passive control.

Table 11 Done Event in Active control mode

Monk Node Location	Java Node Location	How to populate
BP_EVENT.AS.BPI_ID.Value	BP_EVENT.BPI_ID	Copy from source to destination.
BP_EVENT.AS.ID.Value	BP_EVENT.ID	Copy from source to destination.
BP_EVENT.AS.NAME.Value	BP_EVENT.NAME	Copy from source to destination.
BP_EVENT.AS.TYPE.Value	BP_EVENT.TYPE	Copy from source to destination.
BP_EVENT.AS.STATUS.Value	BP_EVENT.STATUS	Copy either the string "SUCCESS" or "FAILURE", depending on whether or not the activity completed successfully.
BP_EVENT.CT.DSN.DS.ACTIVITY.AS.ID.Value	BP_EVENT.ACTIVITY.ID	Copy from source to destination. (Not absolutely required, but recommended to speed processing.)
BP_EVENT.CT.DSN.DS.ACTIVITY.AS.NAME.Value	BP_EVENT.ACTIVITY.NAME	Copy from source to destination.
BP_EVENT.DS.ATTRIBUTE[i].AS.VALUE.Value	BP_EVENT.ATTRIBUTE[i].VALUE	Use a loop or the e*Insight helper function **eX-get-attribute**, or **getATTRIBUTE_VALUE** to determine the value of the machine-assigned attribute, **eX_eBPMServer**, in the source ETD. Copy this value to the destination ETD. Make sure the instance [i] to which you copy does not overwrite or append to an existing iteration of the destination ETD's repeating node. ***(Active mode only)***
BP_EVENT.DS.ATTRIBUTE[i].AS.TYPE.Value	BP_EVENT.ATTRIBUTE[i].TYPE	Copy the string "TRANSIENT". Make sure the instance [i] to which you copy does not overwrite or append to an existing iteration of the destination repeating node. ***(Active mode only)***
BP_EVENT.DS.ATTRIBUTE[i].AS.NAME.Value	BP_EVENT.ATTRIBUTE[i].NAME	Copy the string "eX_eBPMServer". Make sure the instance [i] to which you copy does not overwrite or append to an existing iteration of the destination repeating node. ***(Active mode only)***

Note: *Make sure that setting the "Output" or "Input/Output" attributes does not conflict with setting the e*Insight engine assigned **eX_eBPMServer** attribute, set in the same repeating node.*

Using the helper function eX-copy-no-attribute to send the "Done" Event

The **eX-copy-no-attribute** e*Insight helper function copies the entire source node to the destination node, except for the user defined attributes. You can use this function in a Collaboration that sends the "Done" Event back to the e*Insight Engine, as shown in the following example.

```
eX-copy-no-attribute ~input%eX_Event ~output%eX_Event
eX-set-BP_EVENT ~output%eX_Event "STATUS" "SUCCESS"
```

The second command sets the value of the BP status node to "SUCCESS" after all the required information is copied to the destination Event. To report the failure of the activity, replace "SUCCESS" with "FAILURE" in the second command.

Table 12 Done Event in Passive control mode

Monk Node Location	Java Node Location	How to populate
BP_EVENT.AS.ID.Value	BP_EVENT.ID	Copy from source to destination. It is the responsibility of the developer to ensure that this value is stored within the event as it is passed through e*Gate.
BP_EVENT.AS.NAME.Value	BP_EVENT.NAME	The business processes name is hard-coded and is the name of the business process for the immediate parent of this activity (this is very important for sub-processes).
BP_EVENT.AS.TYPE.Value	BP_EVENT.TYPE	Set the string to “DO_ACTIVITY”.
BP_EVENT.AS.STATUS.Value	BP_EVENT.STATUS	Set the string to either “SUCCESS”, or “FAILURE”, depending on whether or not the activity completed successfully.
BP_EVENT.CT.DSN.DS.ACTIVITY.AS.ID.Value	BP_EVENT.ACTIVITY.ID	Leave this blank. In Passive mode, the source may not contain the correct ID.
BP_EVENT.CT.DSN.DS.ACTIVITY.AS.NAME.Value	BP_EVENT.ACTIVITY.NAME	Set the string to contain the activity name.

In addition, the “Done” Event must carry with it the values for any attributes (global or local) specified as “Output” or “Input/Output”. These can be set as shown in **“Setting Attributes (eIJSchema)” on page 87**.

Chapter 8

e*Insight Implementation (eIJSchema)

This chapter discusses the steps involved to create an e*Insight Business Process Manager implementation using the eIJSchema base schema.

The case study in this chapter was designed primarily to illustrate the functionality of e*Insight. In addition to showing a working example of a business process implementation, the following e*Insight features are demonstrated:

- Attribute value correction and business process restart
- Undoing a partially completed business process

This case study is extended in later chapters to include authorization and user activities, and local, dynamic, and remote sub-processes.

8.1 Overview

The major tasks in the implementation are shown in Table 13.

Table 13 Overview of implementation tasks

	Task	Section
1	Create the business process (BP) in the e*Insight GUI	**"Create the Payroll BP in e*Insight" on page 100**
2	Use the e*Insight GUI to configure the e*Gate schema that supports e*Insight	**"Configure the Integration Schema (e*Insight)" on page 103**
3	Configure the e*Insight Engine	**"Configure the e*Insight Engine" on page 107**
4	Add and configure the user-defined e*Gate components	**"Configure User-defined e*Gate Components" on page 108**
5	Run and test the scenario	**"Run and Test the e*Insight scenario" on page 117**

The chapter begins with a description of the scenario and then shows how to set it up.

8.1.1 Case Study: Payroll Processing

The case study discussed in this chapter illustrates a simplified implementation of payroll processing. In this case, e*Insight receives payroll data as a delimited text file. Once e*Insight has received the data, a check is made to see if the employee is eligible for a bonus, if they are the bonus is calculated. Finally, the payroll is processed and a message added to the payslip indicating whether a bonus has been paid.

Figure 39 shows the components involved in the business process implementation. The diagram is then separated into two sections and there is a description of how the data flows between these components.

Figure 38 e*Insight Data Flow Diagram

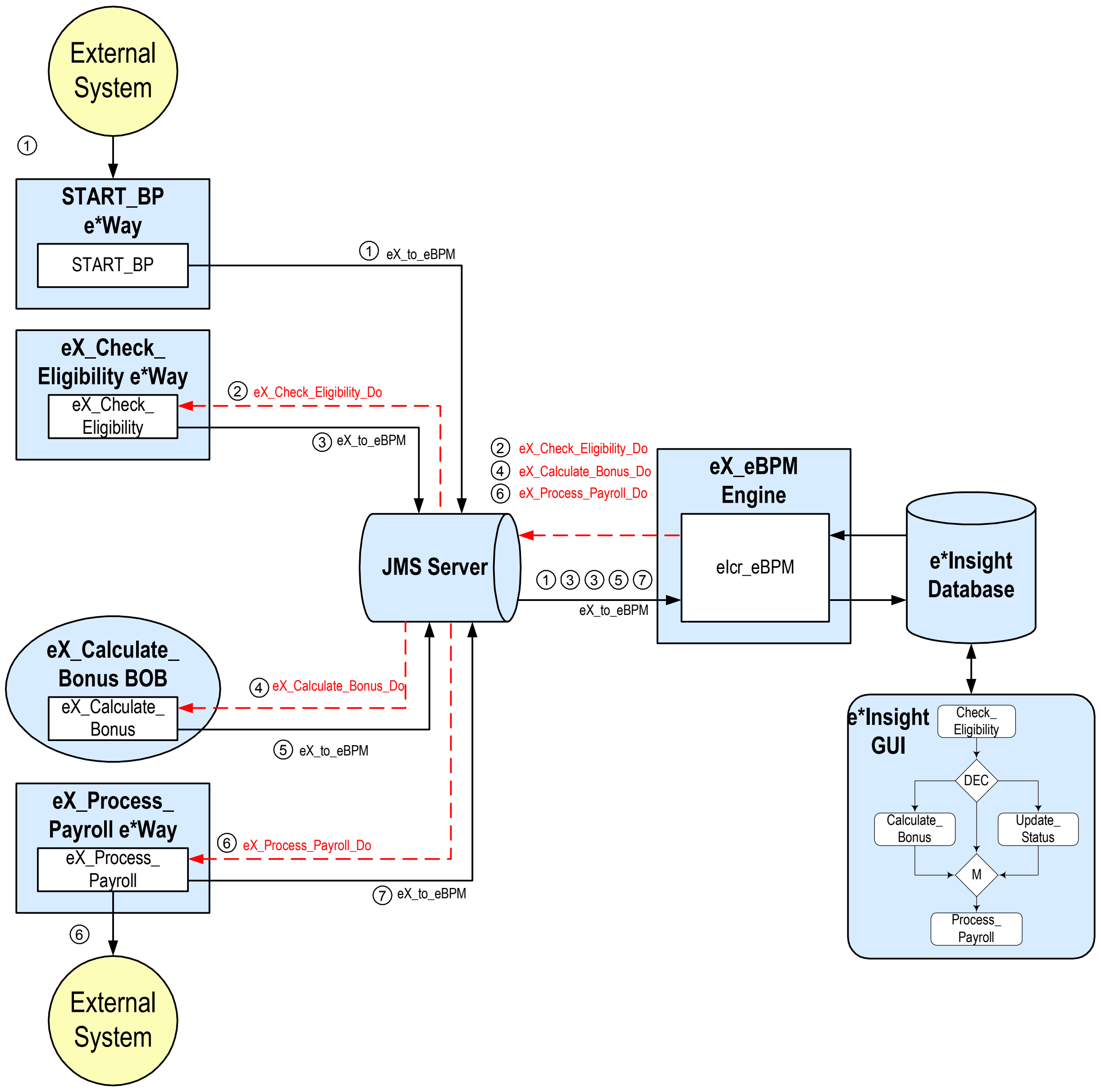
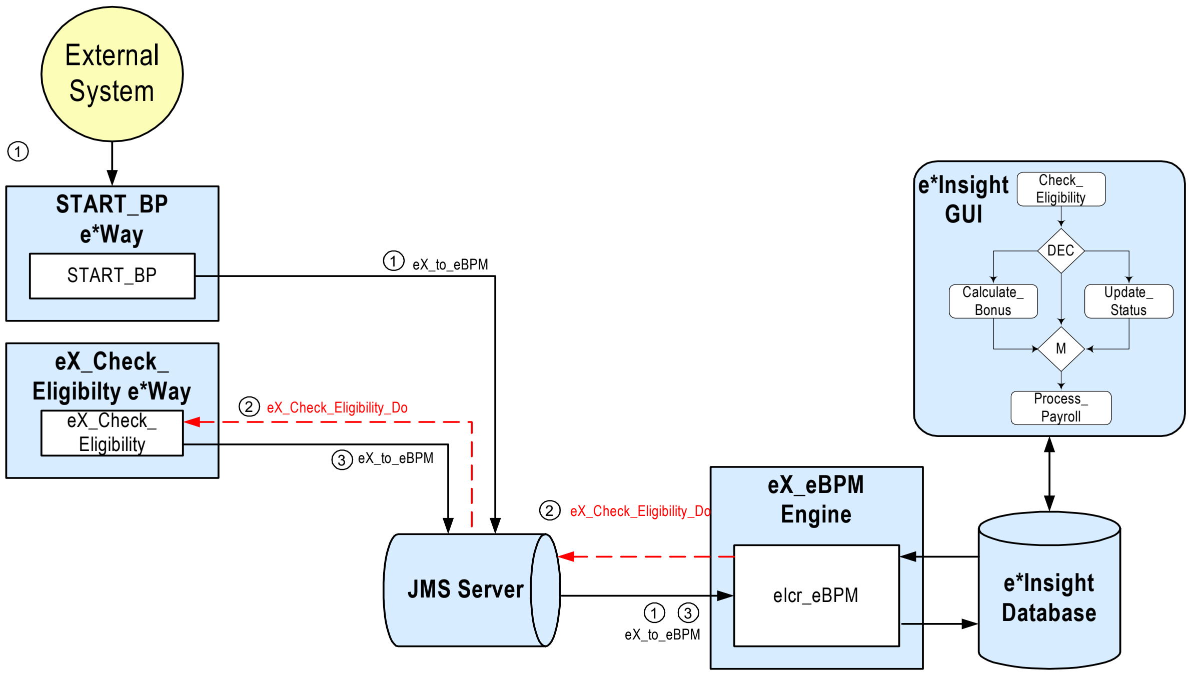


Figure 39 e*Insight Data Flow Diagram (Part 1)



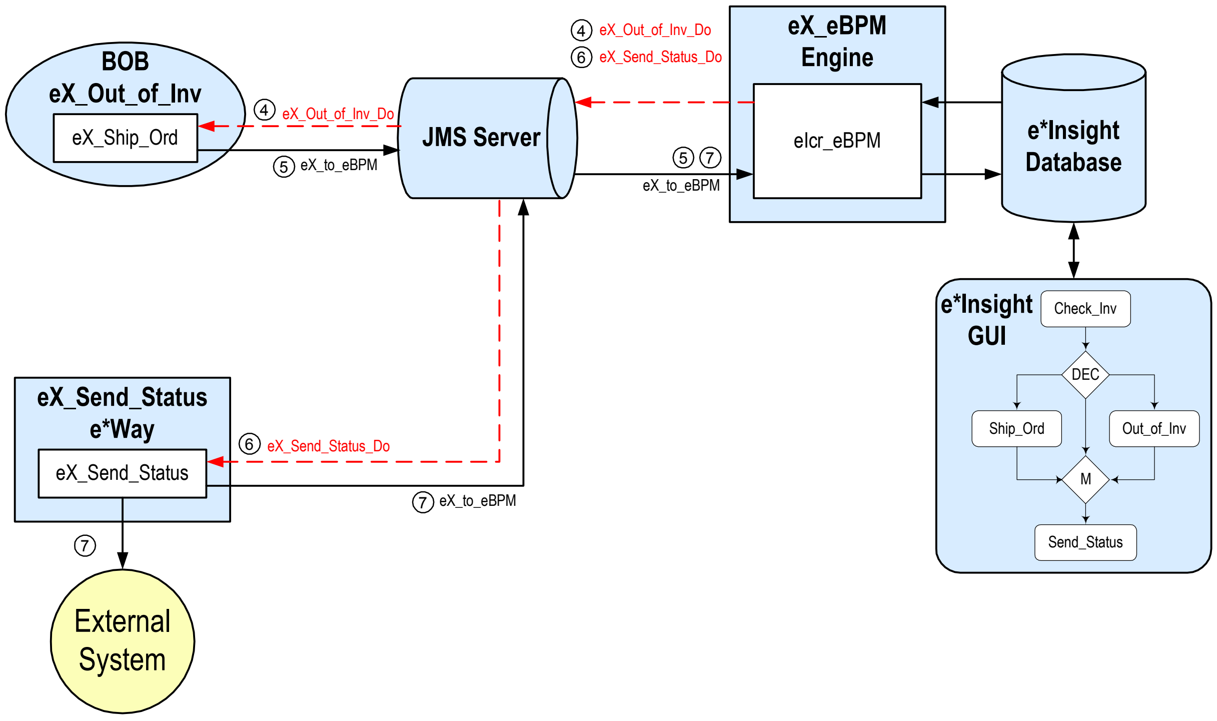
① The user-defined **START_BP** e*Way picks up the text file containing the order information from a shared location on the network, uses the order information to create the e*Insight Event that causes the e*Insight engine to start a business process instance, and publishes it using the **eX_to_eBPM** Event Type to the JMS Server. The e*Insight engine retrieves the Event and uses the information it contains to start the BPI.

② The e*Insight engine publishes a "Do" Event (**eX_Check_Eligibility_Do**) for the first activity in the business process (**Check_Eligibility**). **eX_Check_Eligibility** e*Way, the e*Gate component that corresponds to this activity in the business process, retrieves this Event from the JMS Server and uses the information it contains to check the availability of the items ordered.

③ When the **Check_Eligibility** activity is finished, the **eX_Check_Eligibility** BOB publishes a "Done" Event using the **eX_to_eBPM** Event Type. The e*Insight engine retrieves the "Done" Event, updates the BPI to reflect whether the employee is eligible for a bonus, and then moves forward to the next activity in the business process based on the result of a decision gate. If the employee is eligible for a bonus, the next activity is **Calculate_Bonus**; if the employee is in the sales department then the next activity is **Process_Payroll**, if the employee has been employed for less than three months then the next activity is **Update_Status**.

Let's assume the employee has been employed for less than three months. The e*Insight engine processes the e*Insight script corresponding to the **Update_Status** activity in the business process, and then moves forward to the next activity in the business process—**Process_Payroll**.

Figure 40 e*Insight Data Flow Diagram (Part 2)



④ Let's assume the employee is eligible for a bonus. The e*Insight engine publishes a "Do" Event (**eX_Calculate_Bonus_Do**) corresponding to the **Calculate_Bonus** activity in the business process. The **eX_Calculate_Bonus** BOB retrieves this Event and uses the information to calculate the bonus.

⑤ When the **Calculte_Bonus** activity is finished, the **eX_Calculate_Bonus** BOB publishes a "Done" Event indicating that the bonus has been calculated. The e*Insight engine retrieves this Event, updates the BPI, and then moves forward to the next activity in the business process—**Process_Payroll**.

⑥ The e*Insight engine publishes a "Do" Event (**eX_Process_Payroll_Do**) corresponding to the **Process_Payroll** activity in the business process. The **eX_Process_Payroll** e*Way retrieves this Event and uses the information it contains to send a order status report to the payroll system.

⑦ The **eX_Process_Payroll** e*Way publishes two Events: one containing the status report to be sent to the payroll system, and also the "Done" Event. The e*Insight engine retrieves the "Done" Event and uses the information it contains to update the BPI to indicate that the final activity in the business process has completed successfully.

8.2 Create the Payroll BP in e*Insight

The following is a summary of the procedure for creating a BP in the e*Insight GUI.

1 Create a Business Process named **Payroll**.

1 Add the activities.

2 Add the decision gates.

3 Make the connections between the activities and gates.

4 Add all the global attributes.

5 Assign global attributes to activities.

6 Add the logic to the decision gates.

7 Configure the properties for the activities.

For more information on creating this business process, see the *e*Insight Business Process Manager User's Guide*.

Use the diagram shown in Figure 41 and the following tables to create the BP in e*Insight.

Important: *Mark the check box for* ***Manual Restart*** *on the General tab of the Properties dialog box for each activity.*

Figure 41 Payroll Business Process Model

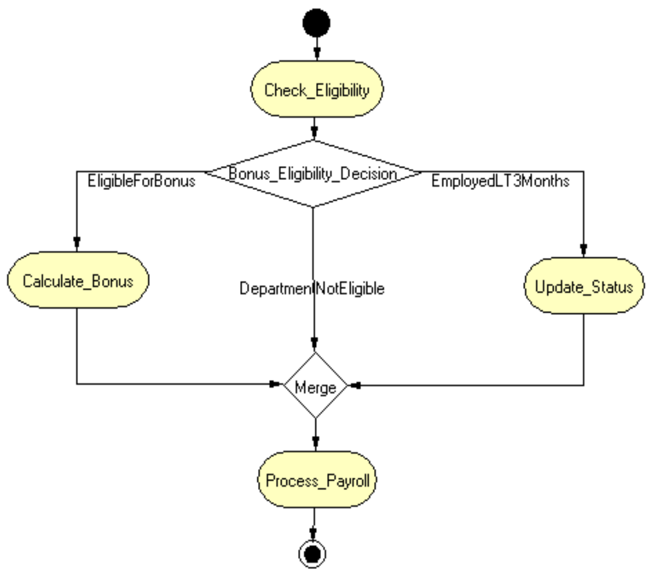


Table 14 BP Global Attributes

Attribute	Type	Data Direction	Default Value
First_Name	String	Input	
Last_Name	String	Input	
Department	String	Input	
Salary	Number	Input	
Start_Date	String	Input	
Pay_Period	String	Input	
Grade	String	Input/Output	
Probation	Boolean	Input	
Eligibility	String	Internal	null
Comments	String	Internal	No comment
Bonus	Number	Internal	0

Table 15 Activity Attributes

Activity	Attribute(s)	Input/Output
Check_Eligibility	Department	Input
	Probation	Input
	Eligibility	Output
Calculate_Bonus	Salary	Input
	Grade	Input
	Bonus	Output
Update_Status	Comments	Output
Process_Payroll	First_Name	Input
	Last_Name	Input
	Department	Input
	Salary	Input
	Pay_Period	Input
	Bonus	Input
	Comments	Input

Table 16 Decision Gate

Link	Target Activity	Expression
EmployedLT3Months	Update_Status	Probation==true
DepartmentNotEligible	Merge	Department==″sales″
EligibleForBonus	Calculate_Bonus	(Default)

8.2.1 Creating the processes performing the Activities

The activities in our scenario can either be performed by e*Gate or an e*Insight Script. Three of the activities (**Check_Eligibility**, **Calculate_Bonus**, and **Process_Payroll**) use e*Gate. This is described in **"Configure the Integration Schema (e*Insight)" on page 103**.

The **Update_Status** activity is performed by an e*Insight Script. This is described below.

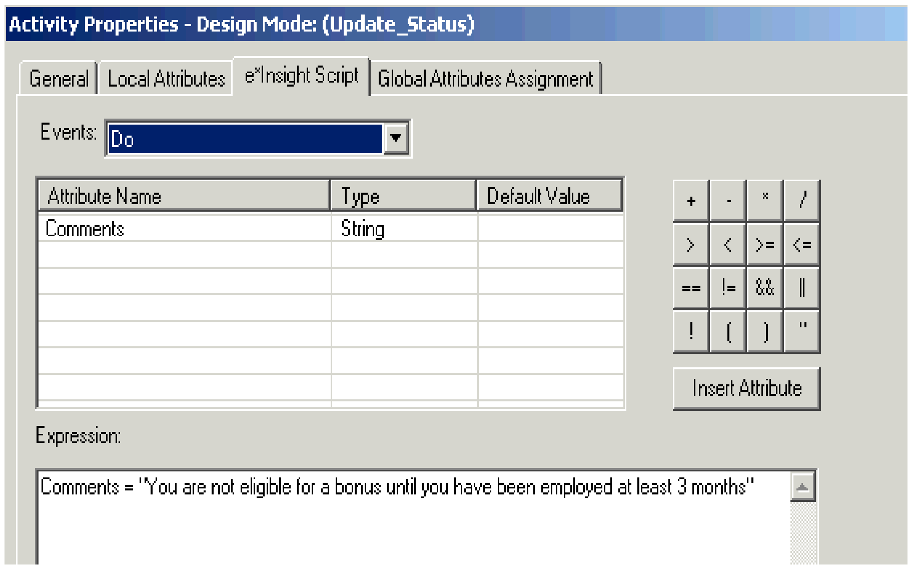
Configuring the e*Insight Script for Update_Status

This script defines a message that appears on the pay slip. It sets the value of the **Comments** attribute to a short message indicating that the employee has not been with the company long enough to receive a bonus.

To configure the e*Insight Script for Update_Status

1. From the **Update_Status** properties, **Activity Performed by** area, select **e*Insight Script**.
2. Select the **e*Insight Script** tab.
3. Configure the script as shown in Figure 42.

Figure 42 Update_Status e*Insight Script Tab


Activity Properties - Design Mode: (Update_Status)
General
Local Attributes
e*Insight Script
Global Attributes Assignment
+
-
*
/
>
<
>=
<=
==
!=
&&
||
!
(
)
"
Insert Attribute

Events: Do

Attribute Name	Type	Default Value
Comments	String	

Expression:

```
Comments = "You are not eligible for a bonus until you have been employed at least 3 months"
```

8.3 Configure the Integration Schema (e*Insight)

All the activities in this example, except **Update_Status** are carried out using e*Gate components. You must first create a Schema (a copy of eIJSchema) with the basic components required for e*Insight. You then configure these components for your environment and create additional components for the activities.

To create a copy of eIJSchema

1. From the e*Insight GUI **File** menu, select **New e*Gate Schema**.
2. Enter or select a **Registry Host** on which to create the schema.
3. Enter a **Username** and **Password** that is valid on the Registry Host.
4. From the **Based on** list, select **eIJSchema (Java)**.
5. In the **Name** box, enter **Payroll**.
6. Click **OK**.

After creating the business process, you must configure the e*Gate Registry schema that supports the e*Insight system.

e*Insight allows you to specify the type of component (e*Way or BOB) associated with a particular activity and where it runs.

Integration Schema Activity Components Summary

The information in Table 17 shows a summary of the e*Gate components that support this example.

Table 17 Integration Schema Activity Components

Name	Type	Participating Host	Configuration Instructions
eX_Check_Eligibility	Multi-Mode e*Way	localhost	**"Creating the eX_Check_Eligibility Multi-Mode e*Way" on page 104**
eX_Calculate_Bonus	BOB	localhost	**"Creating the eX_Calculate_Bonus BOB" on page 106**
eX_Process_Payroll	Single-Mode e*Way	localhost	**"Process_Payroll e*Way Configuration" on page 107**

For information on how to use the e*Insight GUI to configure the e*Gate Registry see the *e*Insight Business Process Manger User's Guide*.

Note: *This example runs all software components on a single machine (named "localhost"). In an actual implementation, these components could be distributed throughout a network, depending on the requirements of the system.*

Creating the eX_Check_Eligibility Multi-Mode e*Way

The eX_Check_Eligibility Collaboration runs the Check_Eligibility CRS. This checks to see what type of Event it has received, either a “Do” Event or an “Undo” Event. If it is an “Undo” Event the Comments attribute is populated with the string “The Check_Eligibility activity has been reversed” simulating the case of executing a compensating transaction for the activity.

If the Event is a “Do” Event, then the values for the **Department** and **Probation** are checked. Depending on what this number is the following happens.

If the department is sales, this indicates that the employee is not eligible for a bonus since the department they work for is not eligible. The CRS sets the value of the **Eligible** attribute to “department not eligible” and sends a “SUCCESS” Event back to e*Insight indicating that the activity has completed successfully.

If the employee is still on probation, this indicates that the employee is not eligible for a bonus since they have not been working for the company for the required three months. The CRS sets the value of the **Eligible** attribute to “probation” and sends a “SUCCESS” Event back to e*Insight indicating that the activity has completed successfully.

Any other employee is treated as being eligible for a bonus and the CRS set the value of **Eligible** to “yes” and sends “SUCCESS”.

To configure the Check_Eligibility activity

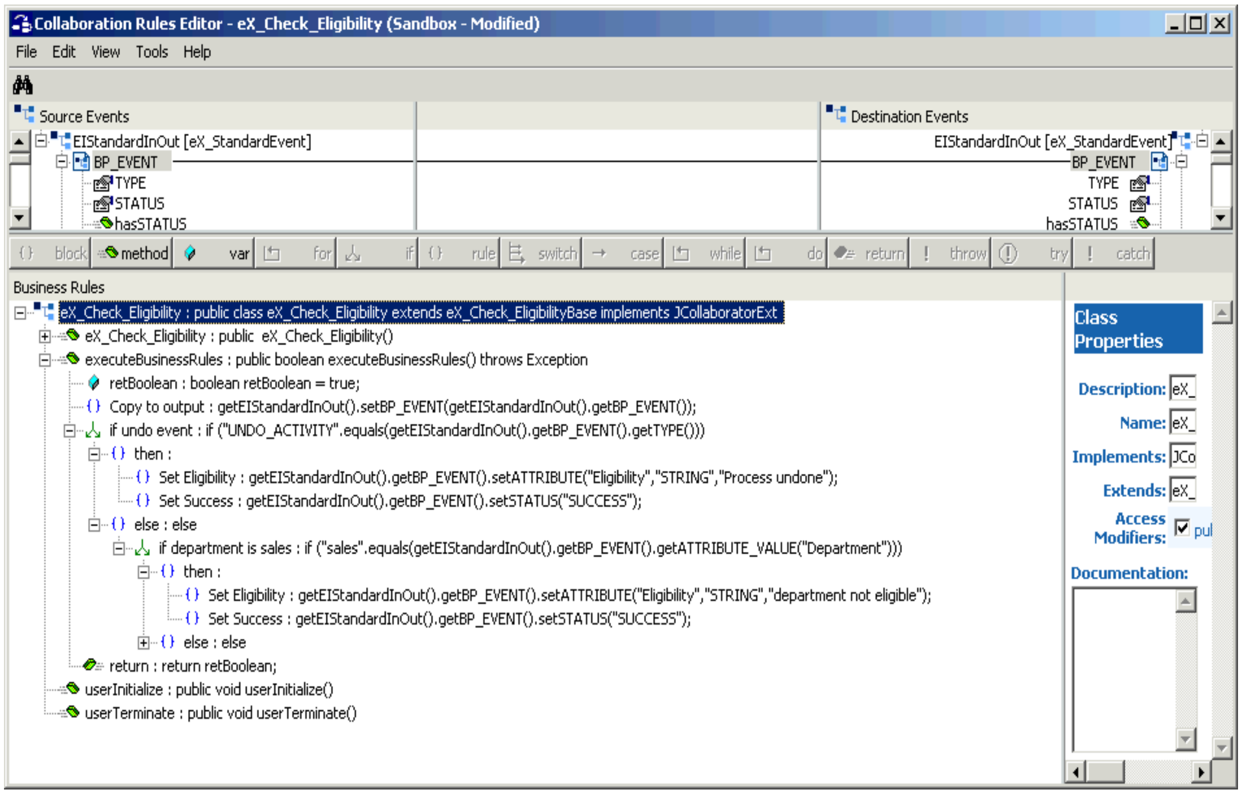
1. In the e*Insight GUI, open the **Check_Eligibility** activity properties.
2. On the **General** tab, **e*Gate Module** section, select a **Module Type** of **Multi-Mode e*Way**.
3. Click **New**.

   You may be required to log into e*Gate.

   The Define Collaboration dialog appears.
4. Click **OK**.
5. Create eX_Check_Eligibility.xpr.

   Figure 43, on the following page, shows the **eX_Check_Eligibility.xpr** CRS used in this example.

Figure 43 eX_Check_Eligibility.xpr CRS (Java)



```
EIStandardInOut [eX_StandardEvent]
  BP_EVENT
    TYPE
    STATUS
    hasSTATUS
```

```
eX_Check_Eligibility : public class eX_Check_Eligibility extends eX_Check_EligibilityBase implements JCollaboratorExt
  eX_Check_Eligibility : public  eX_Check_Eligibility()
  executeBusinessRules : public boolean executeBusinessRules() throws Exception
    retBoolean : boolean retBoolean = true;
    Copy to output : getEIStandardInOut().setBP_EVENT(getEIStandardInOut().getBP_EVENT());
    if undo event : if ("UNDO_ACTIVITY".equals(getEIStandardInOut().getBP_EVENT().getTYPE()))
      then :
        Set Eligibility : getEIStandardInOut().getBP_EVENT().setATTRIBUTE("Eligibility","STRING","Process undone");
        Set Success : getEIStandardInOut().getBP_EVENT().setSTATUS("SUCCESS");
      else : else
        if department is sales : if ("sales".equals(getEIStandardInOut().getBP_EVENT().getATTRIBUTE_VALUE("Department")))
          then :
            Set Eligibility : getEIStandardInOut().getBP_EVENT().setATTRIBUTE("Eligibility","STRING","department not eligible");
            Set Success : getEIStandardInOut().getBP_EVENT().setSTATUS("SUCCESS");
          else : else
    return : return retBoolean;
  userInitialize : public void userInitialize()
  userTerminate : public void userTerminate()
```

6 Compile and save the CRS.

7 Close the editor.

8 In the **Check_Eligibility** Activity properties, click **Configure e*Gate Schema**.

9 Click **OK** to close the information dialog.

10 Close the **Check_Eligibility** Activity properties.

Creating the eX_Calculate_Bonus BOB

The Calculate_Bonus translation implements the logic associated with calculating a bonus. The Grade is used to calculate the bonus.

In addition, this translation demonstrates how e*Insight handles "undoing" a partially completed business process. If no Grade is defined, then "FAILURE" is returned to the e*Insight engine which in turn issues "undo" Events for any activities upstream from the failed activity. In this example there is only one, Check_Eligibility, and the Check_Eligibility CRS handles reversing that already completed activity.

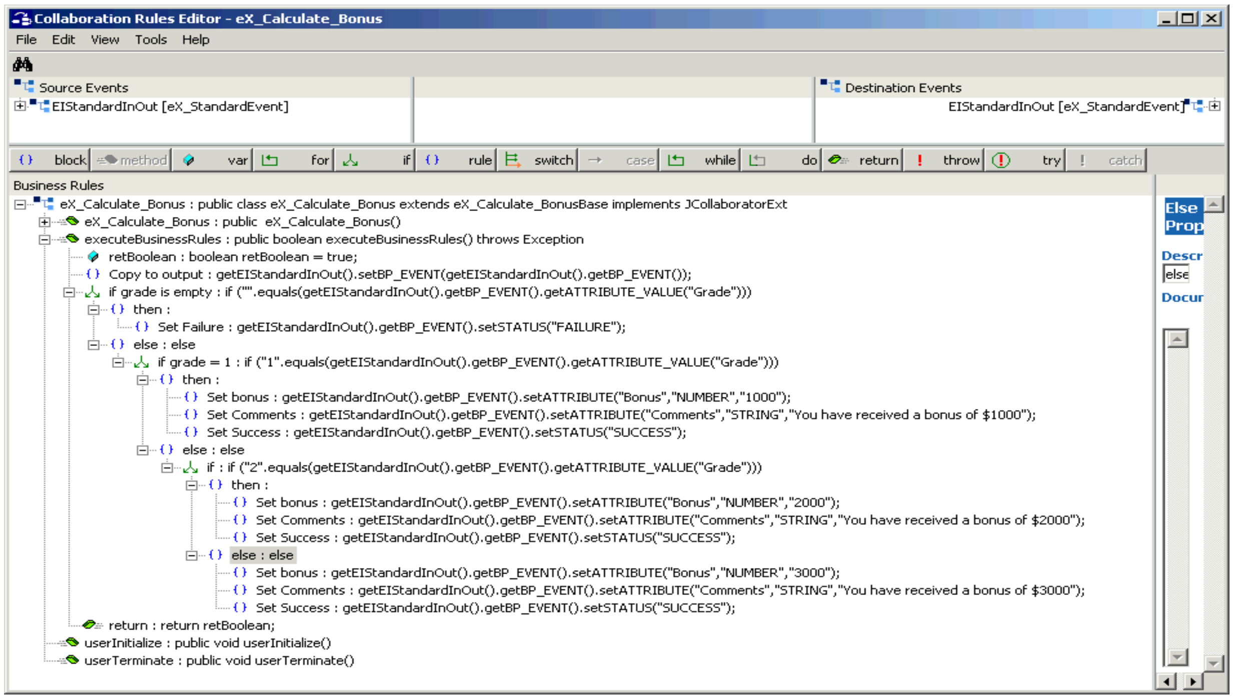
To configure the Calculate_Bonus activity

1. In the e*Insight GUI, open the **Calculate_Bonus** activity properties.
2. On the **General** tab, **e*Gate Module** section, select a **Module Type** of **BOB**.
3. Click **New**.

   The **Define Collaboration** dialog appears.
4. Click **OK**.
5. Create eX_Calculate_Bonus.xpr.

   Figure 44 shows the eX_Calculate_Bonus CRS used in this example.

Figure 44 eX_Calculate_Bonus.xpr CRS


Collaboration Rules Editor - eX_Calculate_Bonus
File Edit View Tools Help
Source Events
EIStandardInOut [eX_StandardEvent]
Destination Events
EIStandardInOut [eX_StandardEvent]
block method var for if rule switch case while do return throw try catch
Business Rules
Else Prop
Descr
else
Docur

```
eX_Calculate_Bonus : public class eX_Calculate_Bonus extends eX_Calculate_BonusBase implements JCollaboratorExt
  eX_Calculate_Bonus : public  eX_Calculate_Bonus()
  executeBusinessRules : public boolean executeBusinessRules() throws Exception
    retBoolean : boolean retBoolean = true;
    Copy to output : getEIStandardInOut().setBP_EVENT(getEIStandardInOut().getBP_EVENT());
    if grade is empty : if ("".equals(getEIStandardInOut().getBP_EVENT().getATTRIBUTE_VALUE("Grade")))
      then :
        Set Failure : getEIStandardInOut().getBP_EVENT().setSTATUS("FAILURE");
      else : else
        if grade = 1 : if ("1".equals(getEIStandardInOut().getBP_EVENT().getATTRIBUTE_VALUE("Grade")))
          then :
            Set bonus : getEIStandardInOut().getBP_EVENT().setATTRIBUTE("Bonus","NUMBER","1000");
            Set Comments : getEIStandardInOut().getBP_EVENT().setATTRIBUTE("Comments","STRING","You have received a bonus of $1000");
            Set Success : getEIStandardInOut().getBP_EVENT().setSTATUS("SUCCESS");
          else : else
            if : if ("2".equals(getEIStandardInOut().getBP_EVENT().getATTRIBUTE_VALUE("Grade")))
              then :
                Set bonus : getEIStandardInOut().getBP_EVENT().setATTRIBUTE("Bonus","NUMBER","2000");
                Set Comments : getEIStandardInOut().getBP_EVENT().setATTRIBUTE("Comments","STRING","You have received a bonus of $2000");
                Set Success : getEIStandardInOut().getBP_EVENT().setSTATUS("SUCCESS");
              else : else
                Set bonus : getEIStandardInOut().getBP_EVENT().setATTRIBUTE("Bonus","NUMBER","3000");
                Set Comments : getEIStandardInOut().getBP_EVENT().setATTRIBUTE("Comments","STRING","You have received a bonus of $3000");
                Set Success : getEIStandardInOut().getBP_EVENT().setSTATUS("SUCCESS");
    return : return retBoolean;
  userInitialize : public void userInitialize()
  userTerminate : public void userTerminate()
```

6. Compile and save the CRS.
7. Close the editor.
8. In the Calculate_Bonus Activity properties, click **Configure e*Gate Schema**.

9. Click **OK** to close the information dialog.
10. Close the Calculate_Bonus Activity properties.

Process_Payroll e*Way Configuration

The Process_Payroll Collaboration is configured using e*Gate, see **"Configure the Process_Payroll e*Way" on page 113**. The e*Way and basic components should be created from within the e*Insight GUI.

To create the Process_Payroll Activity e*Way

1. In the e*Insight GUI, open the **Process_Payroll** activity properties.
2. On the **General** tab, **e*Gate Module** section, from the **Module Type** list select the **Single-Mode e*Way**.
3. Click **Configure e*Gate Schema**.
4. Click **OK**, to close the information dialog.
5. Close the **Process_Payroll** Activity properties.

8.4 Configure the Integration Schema (e*Gate)

The remaining components used in this implementation are configured from the e*Gate Enterprise Manager. You must start the e*Gate Enterprise Manager and open the **Payroll** schema that you created in **"To create a copy of eIJSchema" on page 103**.

8.4.1 Configure the e*Insight Engine

The e*Insight engine runs in a specially configured Multi-Mode e*Way. You must make changes to the e*Insight engine connection configuration file for this e*Way to conform to the requirements of your system. For example, you must specify the name of the e*Insight database to which the e*Way connects.

Note: *This example uses only one e*Insight engine. In an actual implementation, more than one e*Insight engine can be configured to handle the required workload. In such a case, you must make changes to each of the e*Insight engines.*

Edit the eIcp_eInsightEngine Connection Configuration File

Most of the parameter settings in the eIcp_eInsightEngine connection's configuration file should not be changed. **"Configuring the e*Insight Engine Connection" on page 34** discusses the parameters that may need to be changed depending on the implementation. Use the e*Way Editor and the information in **"Configuring the e*Insight Engine Connection" on page 34** to make the required changes for the Payroll example.

8.4.2 Configure the JMS Connection

The JMS connection for e*Insight must be configured for your system. The minimal configuration required for this implementation is described in this section. For more information on JMS IQ Services, see *SeeBeyond JMS Intelligent Queue User's Guide*.

To configure the JMS connection

1. From the e*Gate Enterprise Manager GUI components view, select the **e*Way Connections** folder.
2. Select the **eI_cpeInsightJMS** connection, and click the **Properties** tool.
3. In the **e*Way Connection Configuration File** section, click **Edit**.
4. From the **Goto Section** list, select **Message Service**.
5. Enter a **Server Name** and **Host Name** where your JMS server resides.

8.5 Configure User-defined e*Gate Components

The user-defined components in an e*Insight implementation consist of two types: the first type *starts* the business process, and second type runs *as part of* the business process. The activity components are of the second type.

The Payroll example uses a file e*Way to start the business process and BOBs to run all the activities except the last. The last activity is represented by an additional file e*Way.

Configuration Order for the User-defined Components

Table 18 shows the configuration order for the user-defined components.

Table 18 Configuration Order for User-defined Components

	Task	Section
1	Add and configure the START_BP e*Way	**"Configure the START_BP e*Way" on page 108**
2	Configure the Process_Payroll e*Way	**"Configure the Process_Payroll e*Way" on page 113**

Important: *All the integration schema associations are displayed in table format at the end of this section. The sections dealing with e*Way configuration include tables detailing the non-default e*Way parameter settings. The sections dealing with the Monk and Java Collaboration Rules Scripts show screen shots of these scripts as they appear in the e*Gate Collaboration Editor.*

8.5.1 Configure the START_BP e*Way

The e*Way that sends the Event that starts the business process, named **START_BP** in this example, must convert the incoming data into e*Insight Event format, as well as send the appropriate acknowledgment to the e*Insight engine to create the Business Process Instance (BPI).

The **START_BP** e*Way is completely user defined and must be added to the **eIJSchema** in the e*Gate Enterprise Manager. In an actual implementation, the choice of e*Way (or BOB) would depend on the requirements of the situation. For example, if the data were coming from an SAP system, you might select an SAP ALE e*Way; or if the data were already in the e*Gate system, you could use a BOB to start the BPI. In the present case, a text file on the local system provides the input data, therefore this example uses a file e*Way to send the “Start” Event to the e*Insight engine.

Table 19 shows the steps to configure the **START_BP** e*Way.

Table 19 Configuration steps for the START_BP e*Way

	Step	Section
1	Add the e*Way and create the e*Way configuration file	**“Step 1: Create the START_BP e*Way” on page 109**
2	Create the Input ETD	**“Step 2: Create the Input ETD” on page 110**
3	Create the START_BP Collaboration Rules script (CRS)	**“Step 3: Create the START_BP Collaboration” on page 110**
4	Configure the Collaboration in the GUI	**“Step 4: Configure the Collaboration in the GUI” on page 112**

Step 1: Create the START_BP e*Way

The e*Way for the Payroll example is a simple file e*Way (executable: **stcewfile.exe**) that polls a directory (**<eGate>\client\data\Payroll**) for any file with the extension “.fin” and moves it into the e*Insight system.

Use the Enterprise Manager and the following table to add the **START_BP** e*Way and create its configuration file.

Table 20 Start e*Way Parameters

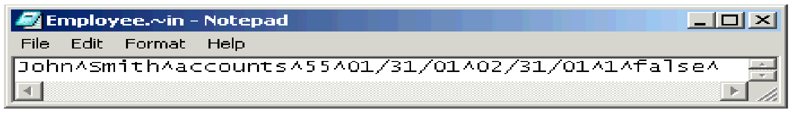
Screen	Parameter	Setting
General Settings	(All)	(Default)
Outbound (send) settings	(All)	(Default)
Poller (inbound) settings	PollDirectory	<eGate>\client\data\Payroll
	(All others)	(Default)
Performance Testing	(All)	(Default)

Step 2: Create the Input ETD

The input ETD is based on the format of the input data. The Payroll example uses a delimited text file (**Employee.fin**) that contains the data needed to process the order.

The input data file used in this example is shown in Figure 45. Place this data file at the directory location **c:\eGate\client\data\Payroll**.

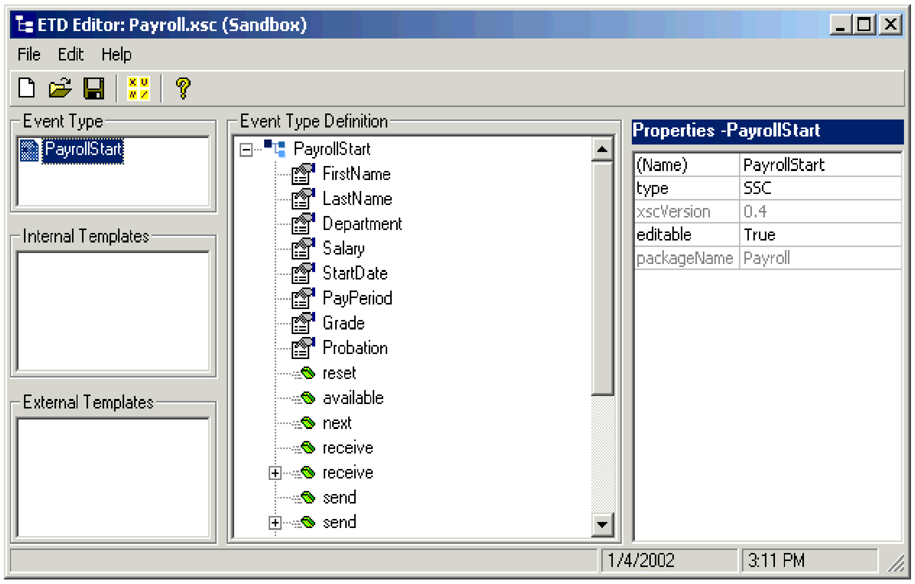
Figure 45 Input Text File (Employee.fin)



```
John^Smith^accounts^55^01/31/01^02/31/01^1^false^
```

Using the ETD Editor and the input data as a guide, create an ETD like the one shown in Figure 46. Set the global delimiter to a ^ character.For more information on using the ETD Editor see the ETD Editor’s online help.

Figure 46 Input ETD: PayrollStart.xsc (Java)



(Name)	PayrollStart
type	SSC
xscVersion	0.4
editable	True
packageName	Payroll

Step 3: Create the START_BP Collaboration

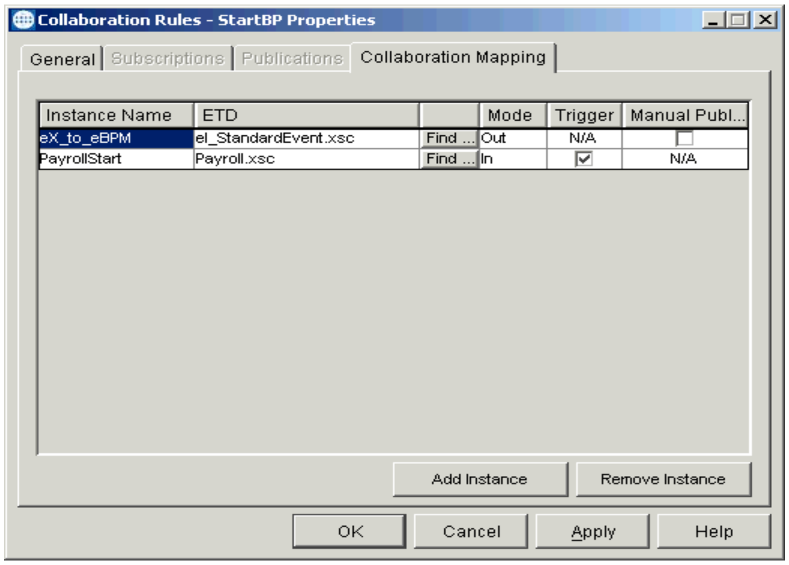
The Collaboration that sends the Event that starts the BPI must do two things:

- Put the data into e*Insight ETD (**eI_StandardEvent.xsc)** format.
- Populate the Event with the information the e*Insight engine needs to start a BPI.

In addition to these two tasks, the **START_BP** Collaboration also provides the recommended location for setting any global attributes that are required in your business process.

1. Create a Collaboration Rule, **START_BP**, that uses the Java service.
2. Configure the Collaboration Mapping tab, as shown in Figure 47.

Figure 47 Start_BP Properties, Collaboration Mapping Tab

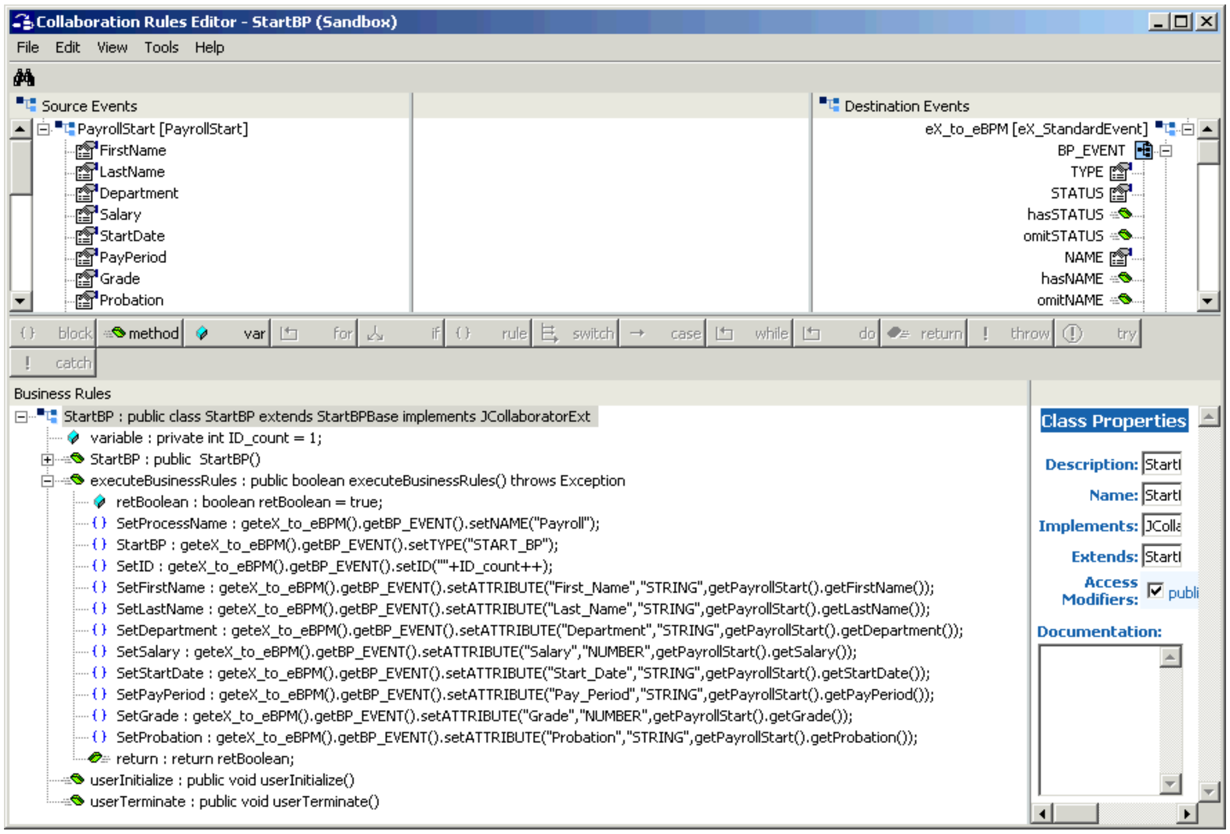


Instance Name	ETD		Mode	Trigger	Manual Publ...
eX_to_eBPM	eI_StandardEvent.xsc	Find ...	Out	N/A	☐
PayrollStart	Payroll.xsc	Find ...	In	✓	N/A

3 Click **Apply**, and click the **General Tab**.

4 Click **New** to create a new CRS, as show in Figure 48.

Figure 48 START_BP CRS



Source Events:

```
PayrollStart [PayrollStart]
    FirstName
    LastName
    Department
    Salary
    StartDate
    PayPeriod
    Grade
    Probation
```

Destination Events:

```
eX_to_eBPM [eX_StandardEvent]
    BP_EVENT
        TYPE
        STATUS
        hasSTATUS
        omitSTATUS
        NAME
        hasNAME
        omitNAME
```

Business Rules:

```
StartBP : public class StartBP extends StartBPBase implements JCollaboratorExt
    variable : private int ID_count = 1;
    StartBP : public StartBP()
    executeBusinessRules : public boolean executeBusinessRules() throws Exception
        retBoolean : boolean retBoolean = true;
        SetProcessName : geteX_to_eBPM().getBP_EVENT().setNAME("Payroll");
        StartBP : geteX_to_eBPM().getBP_EVENT().setTYPE("START_BP");
        SetID : geteX_to_eBPM().getBP_EVENT().setID(""+ID_count++);
        SetFirstName : geteX_to_eBPM().getBP_EVENT().setATTRIBUTE("First_Name","STRING",getPayrollStart().getFirstName());
        SetLastName : geteX_to_eBPM().getBP_EVENT().setATTRIBUTE("Last_Name","STRING",getPayrollStart().getLastName());
        SetDepartment : geteX_to_eBPM().getBP_EVENT().setATTRIBUTE("Department","STRING",getPayrollStart().getDepartment());
        SetSalary : geteX_to_eBPM().getBP_EVENT().setATTRIBUTE("Salary","NUMBER",getPayrollStart().getSalary());
        SetStartDate : geteX_to_eBPM().getBP_EVENT().setATTRIBUTE("Start_Date","STRING",getPayrollStart().getStartDate());
        SetPayPeriod : geteX_to_eBPM().getBP_EVENT().setATTRIBUTE("Pay_Period","STRING",getPayrollStart().getPayPeriod());
        SetGrade : geteX_to_eBPM().getBP_EVENT().setATTRIBUTE("Grade","NUMBER",getPayrollStart().getGrade());
        SetProbation : geteX_to_eBPM().getBP_EVENT().setATTRIBUTE("Probation","STRING",getPayrollStart().getProbation());
        return : return retBoolean;
    userInitialize : public void userInitialize()
    userTerminate : public void userTerminate()
```

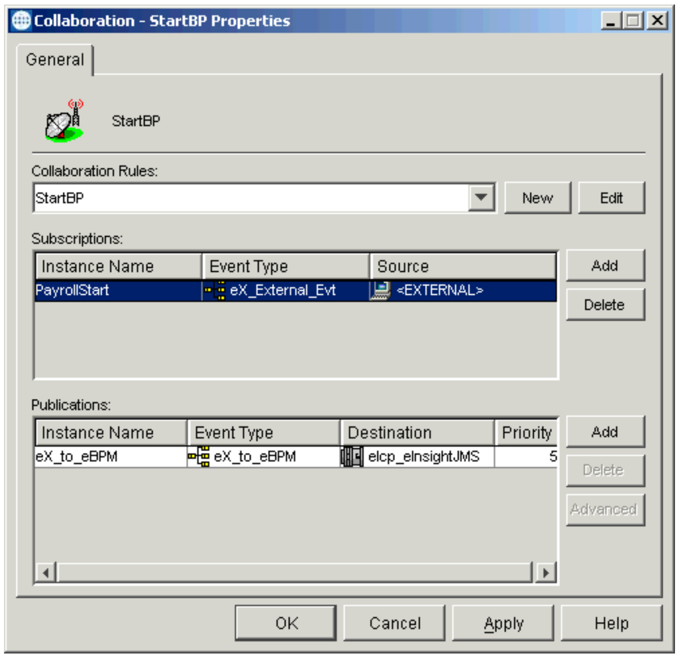
Class Properties: Description: StartI; Name: StartI; Implements: JColla; Extends: StartI; Access Modifiers: ✓ publi

Step 4: Configure the Collaboration in the GUI

In addition to creating the configuration file for the e*Way and the CRS used by the Collaboration, you must also configure the **Start_BP** e*Way's Collaboration in the Enterprise Manager GUI.

1 Create a Collaboration for the **Start_BP** e*Way configured as shown in Figure 49.

Figure 49 Start_BP Collaboration



StartBP

Collaboration Rules: StartBP [New] [Edit]

Subscriptions: [Add] [Delete]

Instance Name	Event Type	Source
PayrollStart	eX_External_Evt	<EXTERNAL>

Publications: [Add] [Delete] [Advanced]

Instance Name	Event Type	Destination	Priority
eX_to_eBPM	eX_to_eBPM	elcp_eInsightJMS	5

8.5.2 Configure the Process_Payroll e*Way

The last component that must be configured in the Payroll example is the **Process_Payroll** e*Way.

This e*Way must accomplish two tasks:

- Create a file containing the text of an update for the payroll system
- Return "SUCCESS" to the e*Insight engine

This e*Way simulates updating the payroll system by writing a short status message to a text file. When this is successful, an Event is returned to the e*Insight engine with the status node set to "SUCCESS".

Table 21 shows the steps to configure the Process_Payroll e*Way.

Table 21 Process_Payroll e*Way configuration steps

	Step	Section
1	Find the executable and create the e*Way configuration file	**"Step 1: Configure the e*Way" on page 114**
2	Create the Output ETD	**"Step 2: Create the Output ETD:PayrollProcess.xsc using Java" on page 114**
3	Create the eX_Process_Payroll.tsc CRS	**"Step 3: Create the Process_Payroll Collaboration Rule" on page 114**
4	Configure the Collaboration in the GUI	**"Step 4: Configure the Collaboration" on page 116**

Step 1: Configure the e*Way

First find the executable, then create the configuration file.

The **eX_Process_Payroll** e*Way is a simple file e*Way (**stcewfile.exe**) that writes a text file (**Payroll_output%d.dat**) to the directory **<egate>\client\data\Payroll_Out**. The file created contains the e-mail address of the person who placed the order, along with the status of the order. Use the following table to set the e*Way parameters in the configuration file:

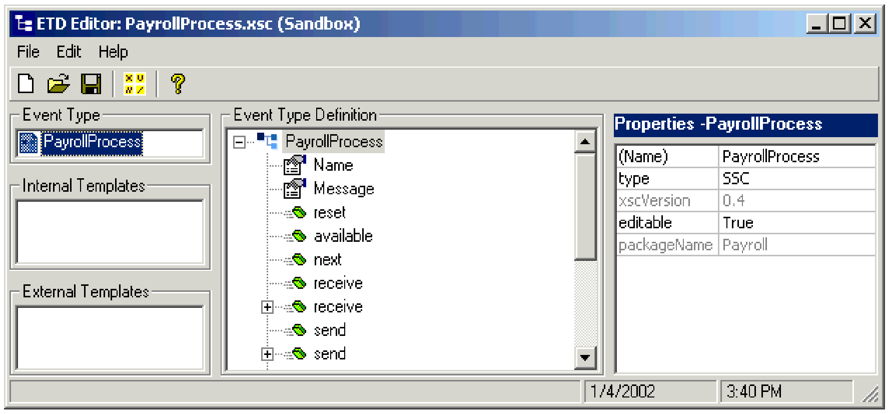
Table 22 Process_Payroll e*Way Parameters

Screen	Parameter	Setting
General Settings	AllowIncoming	No
	AllowOutgoing	Yes
	PerformanceTesting	No (Default)
Outbound (send) settings	OutputDirectory	<eGate>\client\data\Payroll_Out
	OutputFileName	Payroll_output%d.dat
	(All others)	(Default)
Poller (inbound) settings	(All)	(Default)
Performance Testing	(All)	(Default)

Step 2: Create the Output ETD:PayrollProcess.xsc using Java

Use the e*Gate ETD Editor to create an ETD like that shown in Figure 50. Set the global delimiter to a **,** (comma) character.For more information on using the ETD Editor see the ETD Editor's online help.

Figure 50 PayrollProcess.xsc ETD



(Name)	PayrollProcess
type	SSC
xscVersion	0.4
editable	True
packageName	Payroll

Step 3: Create the Process_Payroll Collaboration Rule

This CRS must accomplish three things:

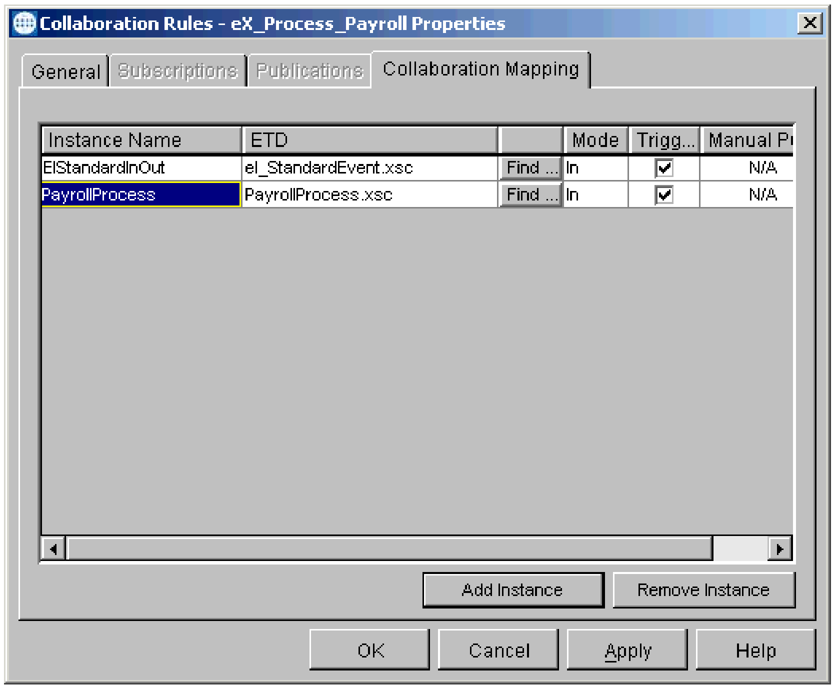
- put the output data into a readable format that can be written to a file

- set the BP status node to "SUCCESS"
- send the "Done" Event back to the e*Insight engine

To Configure the eX_Process_Payroll Collaboration Rule

1. From the eX_to_Process_Payroll Collaboration Rule **General** tab, clear the defined Collaboration Rule and Initialization File.
2. From the eX_to_Process_Payroll Collaboration Rule **Collaboration Mapping** tab, add the new ProcessPayroll instance as shown in Figure 51.

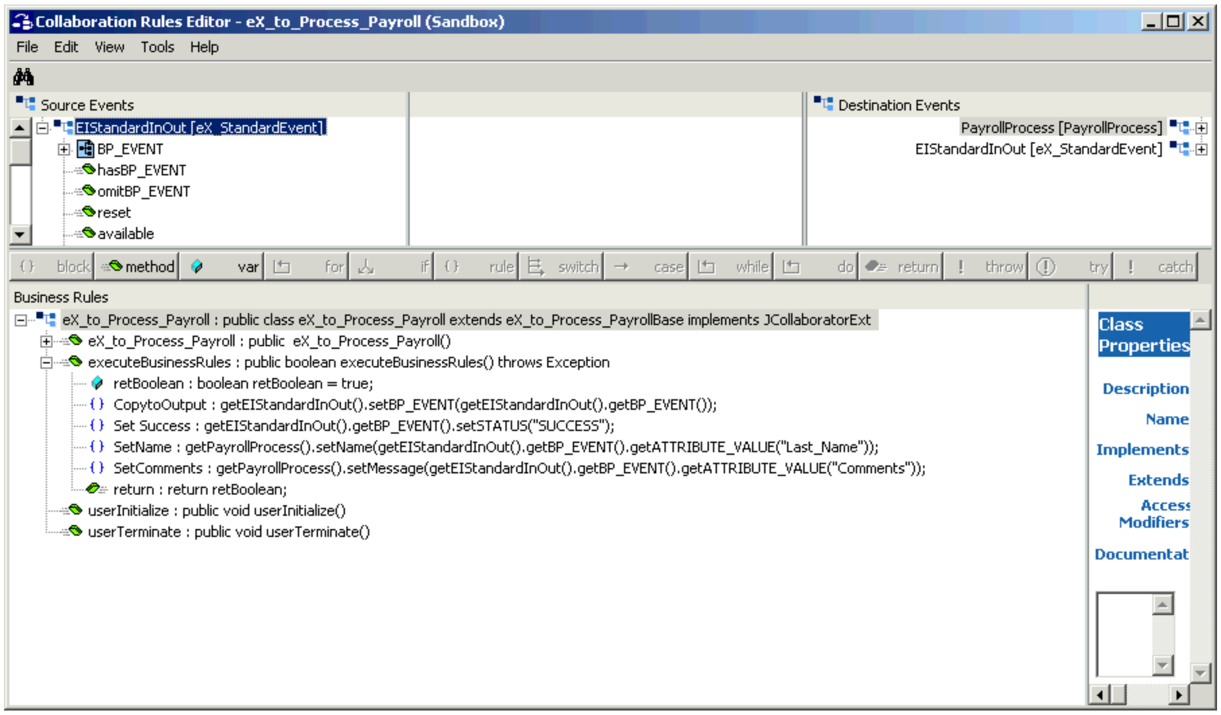
Figure 51 eX_to_Process_Payroll CR Properties, Collaboration Mapping tab



Instance Name	ETD		Mode	Trigg...	Manual Pı
EIStandardInOut	eI_StandardEvent.xsc	Find ...	In	✓	N/A
PayrollProcess	PayrollProcess.xsc	Find ...	In	✓	N/A

3. Click **Apply**, and click the **General Tab**.
4. Click **New** to create a new CRS, as show in Figure 52.

Figure 52 eX_to_Process_Payroll.xpr CRS


Collaboration Rules Editor - eX_to_Process_Payroll (Sandbox)
File Edit View Tools Help
block method var for if rule switch case while do return throw try catch
Class Properties
Description
Name
Implements
Extends
Access Modifiers
Documentat

Source Events
- EIStandardInOut [eX_StandardEvent]
  - BP_EVENT
  - hasBP_EVENT
  - omitBP_EVENT
  - reset
  - available

Destination Events
- PayrollProcess [PayrollProcess]
- EIStandardInOut [eX_StandardEvent]

Business Rules

```
eX_to_Process_Payroll : public class eX_to_Process_Payroll extends eX_to_Process_PayrollBase implements JCollaboratorExt
    eX_to_Process_Payroll : public  eX_to_Process_Payroll()
    executeBusinessRules : public boolean executeBusinessRules() throws Exception
        retBoolean : boolean retBoolean = true;
        CopytoOutput : getEIStandardInOut().setBP_EVENT(getEIStandardInOut().getBP_EVENT());
        Set Success : getEIStandardInOut().getBP_EVENT().setSTATUS("SUCCESS");
        SetName : getPayrollProcess().setName(getEIStandardInOut().getBP_EVENT().getATTRIBUTE_VALUE("Last_Name"));
        SetComments : getPayrollProcess().setMessage(getEIStandardInOut().getBP_EVENT().getATTRIBUTE_VALUE("Comments"));
        return : return retBoolean;
    userInitialize : public void userInitialize()
    userTerminate : public void userTerminate()
```

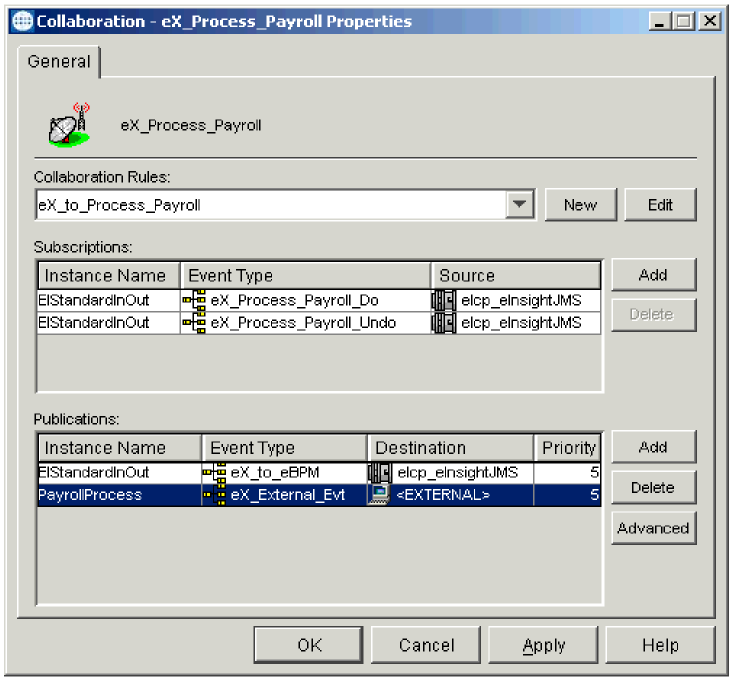
Step 4: Configure the Collaboration

The **eX_Process_Payroll** e*Way in Payroll example does not receive data back from and external system. Consequently, it requires only a single Collaboration. Use the following procedure to edit the two default Collaborations created by the e*Insight GUI during the configuration of the integration schema.

In the Enterprise Manager:

1. Highlight the **eX_Process_Payroll** e*Way.
2. Delete the two Collaborations **eX_to_Process_Payroll** and **eX_from_Process_Payroll**.
3. Add a Collaboration named **eX_Process_Payroll**.
4. Configure the Collaboration, as shown in Figure 53.

Figure 53 eX_Process_Payroll Collaboration


Collaboration - eX_Process_Payroll Properties

General

eX_Process_Payroll

Collaboration Rules:

eX_to_Process_Payroll

Subscriptions:

Instance Name	Event Type	Source
EIStandardInOut	eX_Process_Payroll_Do	elcp_eInsightJMS
EIStandardInOut	eX_Process_Payroll_Undo	elcp_eInsightJMS

Publications:

Instance Name	Event Type	Destination	Priority
EIStandardInOut	eX_to_eBPM	elcp_eInsightJMS	5
PayrollProcess	eX_External_Evt	<EXTERNAL>	5

8.6 Run and Test the e*Insight scenario

Once the schema has been set up in e*Gate you can run the scenario.

8.6.1 Testing the Standard Business Logic

The following procedure tests the standard business logic of the e*Insight Payroll case study example. That logic is as follows: a check is made to see whether or not the employee is eligible for a bonus. If they are the Calculate_Bonus activity is invoked and a message is generated that can be sent to the payroll system indicating that his bonus has been paid to him. If the employee is not eligible, then either the Update_Status activity or Process_Payroll is invoked which creates a message informing the customer that the bonus is unavailable.

The test is made by sending in data with different departments and probations, and verifying the correct processing. Input data with a department of accounts and a probation value of false is interpreted as being eligible. A department of sales or a probation value of true are interpreted as being ineligible.

Payroll Processing

Use the following procedure to test the functionality of the example for an employee that is eligible for a bonus.

1 Start the e*Insight GUI and select the Payroll business process. Switch to monitor mode.

Note: *Make sure that the business process has been enabled in the e*Insight GUI before attempting to run it.*

2 Make a final check of the e*Gate schema, using the tables to confirm all of the GUI associations. Make sure that all of the e*Insight components, including the user-defined components, are set to start automatically.

3 At the command line, type the following to start the schema. You must type the command on a single line.

```
stccb.exe -rh localhost -rs Payroll -ln localhost_cb
     -un username -up password
```

Substitute the appropriate username and password for your installation.

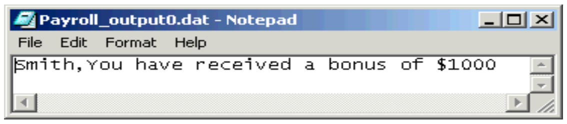
4 Start the e*Gate Monitor, and check the status of all the components. Any components displayed in red are not running. Any e*Insight components that are not running should be investigated before feeding data into the system.

5 Navigate to the location for the input data file, **Employee.~in**, shown in **Figure 45 on page 110** (**c:\eGate\client\data\Payroll**) and change the extension to "**.fin**".

Note: *The change of the extension to "**.~in**" indicates that the data file has been picked up by the **START_BP** e*Way.*

If everything is working correctly, an output file (**Payroll_output#.dat**) as shown in Figure 54 appears in the directory indicating successful completion of the BPI.

Figure 54 In Stock Output File



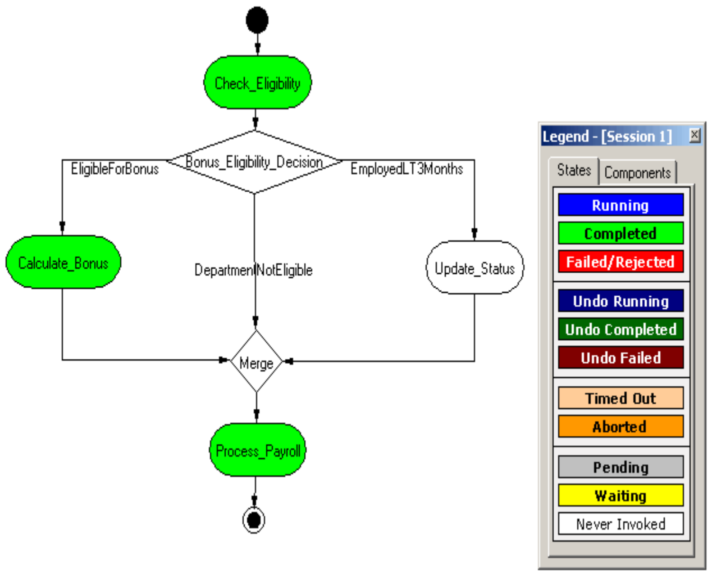
```
Smith,You have received a bonus of $1000
```

6 Switch to the e*Insight GUI and, while in monitor mode, select the most recent BPI from the **List** tab, and then select the **Diagram** tab to observe the path that the data has taken.

The activities that have completed successfully appear green. Any activities that are still running appear blue.

In the Payroll example, an activity that stays blue for more than couple minutes indicates a problem, and the e*Gate component associated with that activity should be investigated for the cause of the problem. Figure 55 illustrates how the successfully completed BPI appears in the e*Insight GUI.

Figure 55 Eligible for Bonus Completed BPI Diagram

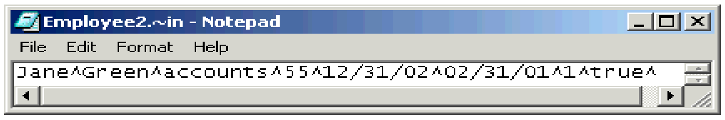


Not Eligible Processing

Testing the functionality for employees that are not eligible for a bonus uses exactly the same procedure as that for eligible processing except that different input data is submitted.

- Verify that sending in the data shown in Figure 56 with a department of sales causes the business process to take the "EmployedLT3Months" branch of the decision gate and create the diagram shown in Figure 57 and the output file shown in Figure 58.

Figure 56 EmployedLT3Months Input File



```
Jane^Green^accounts^55^12/31/02^02/31/01^1^true^
```

Figure 57 Employed Less Than 3 months Completed BPI Diagram

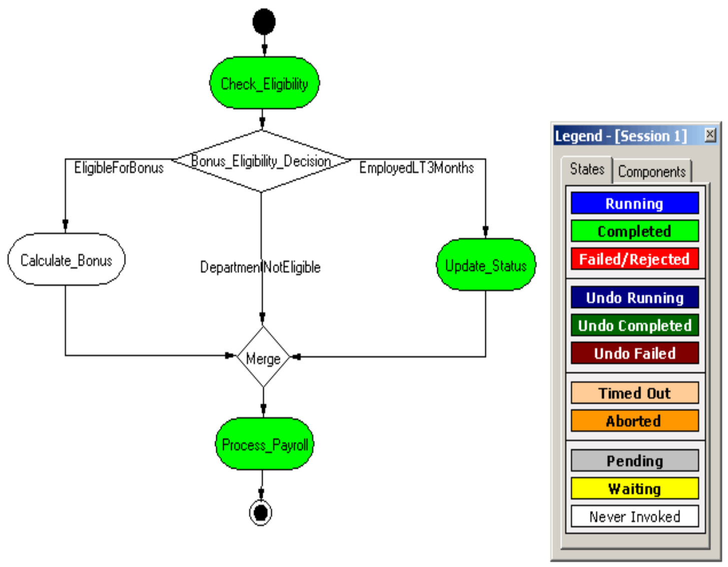
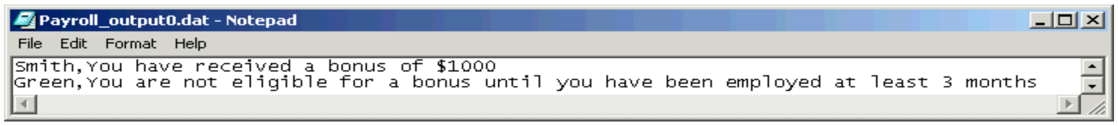


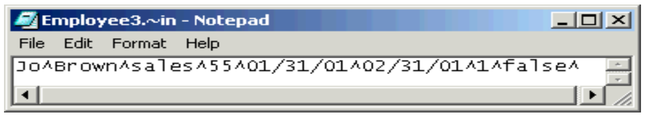
Figure 58 Out of Stock Output File



```
Smith,You have received a bonus of $1000
Green,You are not eligible for a bonus until you have been employed at least 3 months
```

- Verify that sending in the data shown in Figure 59 with a department of sales causes the business process to take the "DepartmentNotEligible" branch of the decision gate and create the diagram shown in Figure 60 and the output file shown in Figure 61.

Figure 59 DepartmentNotEligible Input File



```
Jo^Brown^sales^55^01/31/01^02/31/01^1^false^
```

Figure 60 Department Not Eligible Completed BPI Diagram

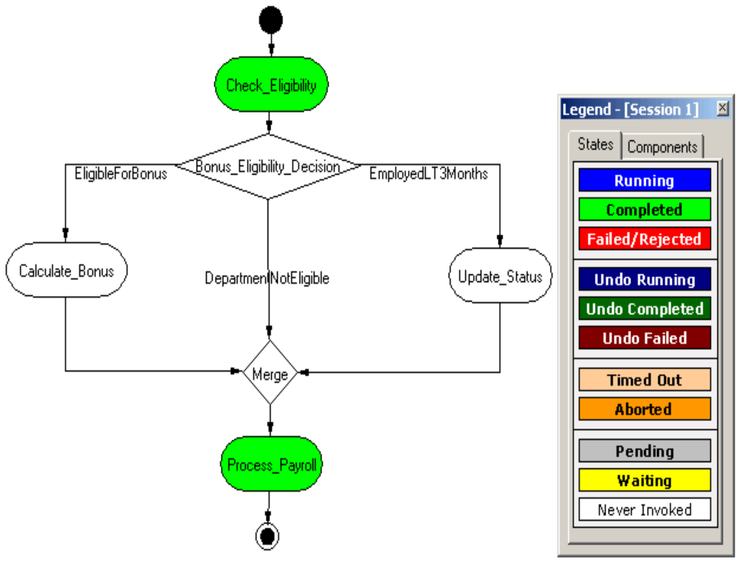
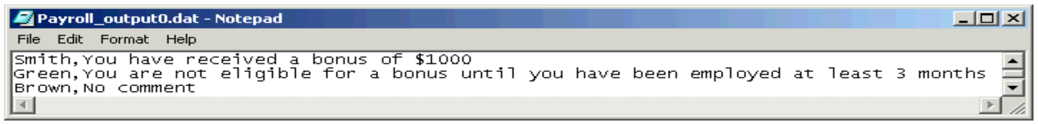


Figure 61 Department Not Eligible Output File



```
Smith,You have received a bonus of $1000
Green,You are not eligible for a bonus until you have been employed at least 3 months
Brown,No comment
```

8.6.2 Demonstrating Business Process Undo Functionality

e*Insight has two methods for undoing a failed business process instance (BPI): automatic and manual. Whether the failure of a particular activity generates an automatic undo of the entire BPI or whether the e*Insight engine waits for user intervention, is set on the **General** tab of the **Activity Properties** dialog box for that activity. The default setting is automatic undo.

When an activity is set to automatic undo and the activity “fails,” then e*Insight marks the activity as “Failed” in the GUI and publishes an “undo” Event (**eX_*Activity*_Undo**) for the last completed activity in the BPI. In this context, fails means that the e*Insight engine receives a “Done” Event where the status node is set to “FAILURE” rather than “SUCCESS”. If the last completed activity is undone successfully, then an “undo” Event is generated for the next activity upstream, and so on, until all the previously completed activities in that BPI have been undone.

If an activity fails and the **Manual Restart** check box is marked on the **General** tab of the **Activity Properties** dialog box for that activity, then e*Insight marks the activity as “Failed” in the GUI and then waits for the user to initiate the next course of action; skip, restart, or undo. If the user selects undo, then the BPI is undone as described in the paragraph above.

Manual Undo

Use the following procedure to test the functionality of manual undo in the e*Insight scenario.

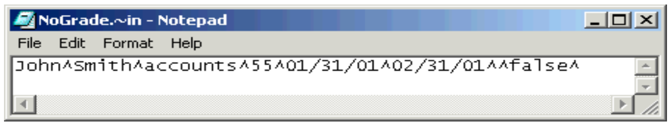
1 Perform steps 1 through 4 outlined in **"Payroll Processing" on page 118**.

2 Verify that **Manual Restart** has been marked for the activities in the business process.

If **Manual Restart** has not been marked and the check box itself is grayed out, you must delete the BPIs for the business process, or save the business process as a new version, before you can mark it. Refer to the *e*Insight Business Process Manager User's Guide* for information on how to do this.

3 Navigate to the location (**c:\eGate\client\data\Payroll\NoGrade.~in**) for the input data file with no grade defined as shown in Figure 62 and change the extension to "**.fin**".

Figure 62 Manual Undo Input File

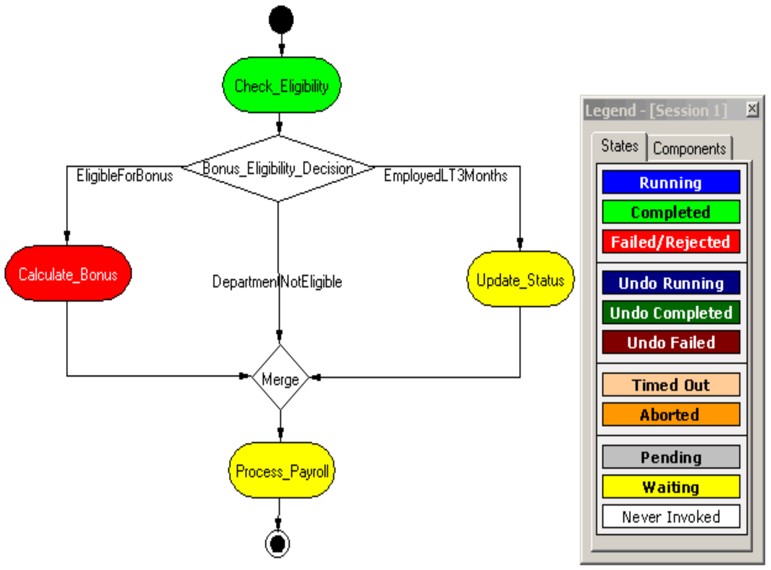


```
John^Smith^accounts^55^01/31/01^02/31/01^^false^
```

Note: *The change of the extension to "**.~in**" indicates that the data file has been picked up by the START_BP e*Way.*

4 Switch to the e*Insight GUI and, while in monitor mode, select the most recent business process instance. Observe the path that the data has taken, as shown in Figure 63 on the next page.

Figure 63 Manual Undo—Failed BPI Diagram



The **Check_Eligibility** activity should be green, indicating that it completed successfully, but the **Calculate_Bonus** activity should appear red, indicating that it has failed.

5 Right-click the **Calculate_Bonus** activity from the tree view, then select **Properties** from the popup menu.

The **Activity Properties - Monitor Mode: (Calculate_Bonus)** is displayed.

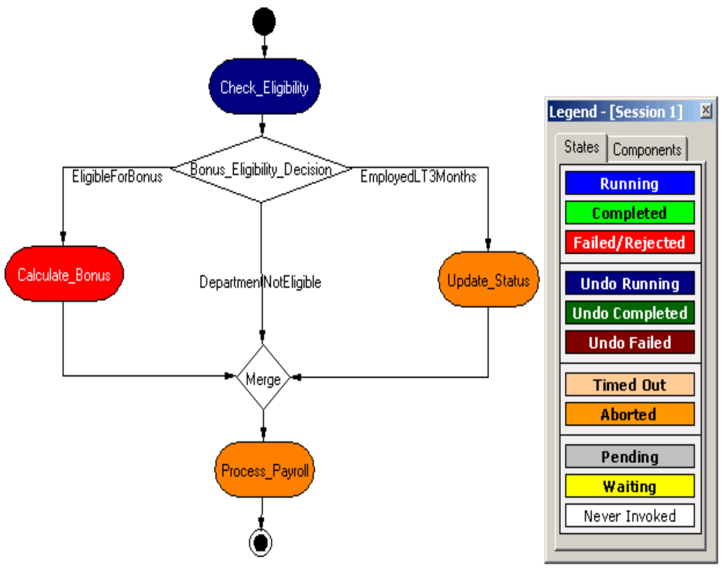
6 Select the **Business Process Attributes** tab.

7 Click **Undo Business Process**, and then click **OK** to close the **Activity Properties** dialog box.

8 Highlight the enabled business process version in the tree view.

The **Check_Eligibility** activity should now appear dark blue indicating that the activity has been successfully undone.

Figure 64 Manual Undo Completed BPI Diagram



8.6.3 Demonstrating Business Process Restart Functionality

An important feature of e*Insight is its ability to allow the operator to fix and restart a business process instance. If the data in one of the business process attributes used by an activity causes the business process to fail, the value can be corrected by the operator and the BPI restarted from the point of failure.

Repairing a String Attribute

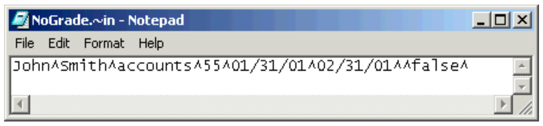
Attributes can be of various types; Boolean, number, string, and XML. The following example shows the procedure to repair an attribute of type string. For information on repairing an attribute with type XML, see the *e*Insight Business Process Manager User's Guide*.

1 Perform steps 1 through 4 outlined in **"Payroll Processing" on page 118**.

2 Verify that **Manual Restart** has been marked for the activities in the business process.

3 Navigate to the location (**c:\eGate\client\data\Payroll\NoGrade.~in**) for the input data file with the grade not defined as shown in Figure 65, and change the extension to "**.fin**".

Figure 65 Attribute Repair Input File

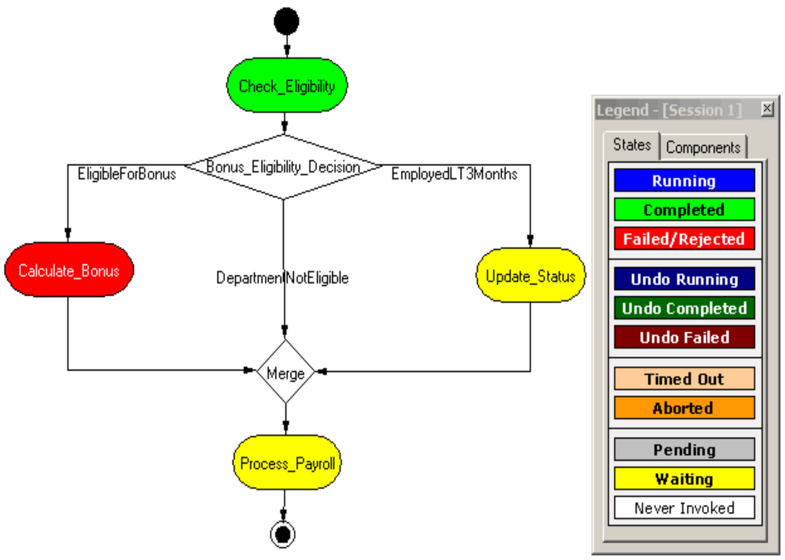


```
John^Smith^accounts^55^01/31/01^02/31/01^^false^
```

Note: *The change of the extension to "**.~in**" indicates that the data file has been picked up by the START_BP e*Way.*

4 Switch to the e*Insight GUI and, while in monitor mode, select the most recent business process instance. Observe the path that the data has taken.

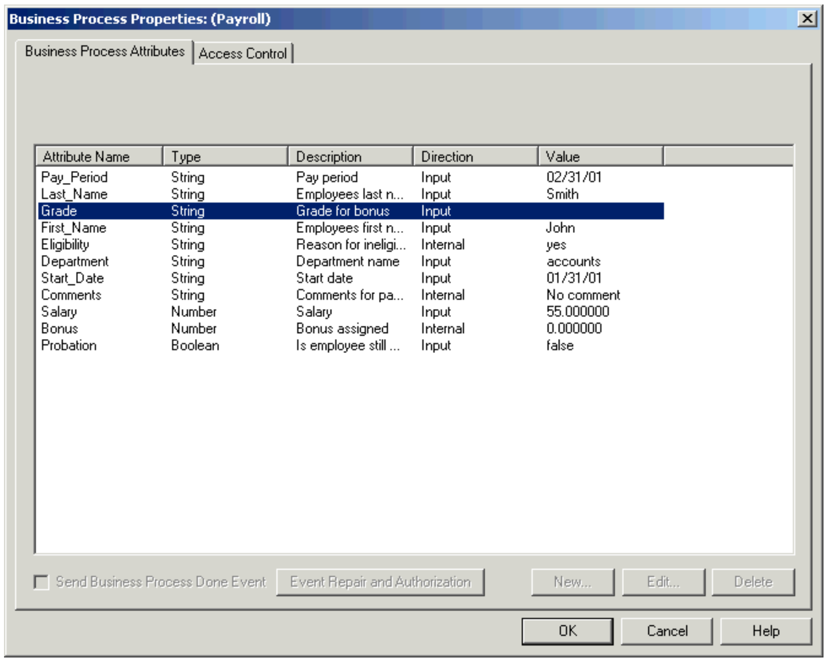
Figure 66 Attribute Repair—Failed BPI Diagram



The **Check_Eligibility** activity should be red, indicating that it failed, and the other activities should appear yellow, indicating that they are waiting.

5 Right-click the enabled **Payroll** business process version from the tree view, then select **Properties** from the popup menu.

Figure 67 Attribute Repair—Business Process Properties


Business Process Properties: (Payroll)
Business Process Attributes
Access Control
Send Business Process Done Event
Event Repair and Authorization
New...
Edit...
Delete
OK
Cancel
Help

Attribute Name	Type	Description	Direction	Value
Pay_Period	String	Pay period	Input	02/31/01
Last_Name	String	Employees last n...	Input	Smith
Grade	String	Grade for bonus	Input	
First_Name	String	Employees first n...	Input	John
Eligibility	String	Reason for ineligi...	Internal	yes
Department	String	Department name	Input	accounts
Start_Date	String	Start date	Input	01/31/01
Comments	String	Comments for pa...	Internal	No comment
Salary	Number	Salary	Input	55.000000
Bonus	Number	Bonus assigned	Internal	0.000000
Probation	Boolean	Is employee still ...	Input	false

6 On the **Business Process Attributes** tab in the **Business Process Properties** dialog

7 In the **Edit Business Process Attribute** dialog box, change the value of the attribute in the **Current Value:** box to **3**, and then click **OK**.

8 Click **OK** to close the **Business Process Properties** dialog box.

9 Right-click the **Check_Eligibility** activity from the tree view, then select **Properties** from the popup menu.

The **Activity Properties - Monitor Mode: (Check_Eligibility)** dialog box displays.

10 Select the **Result/Recourse** tab.

11 Click **Restart Activity**, and then click **OK**.

The BPI now completes successfully.

Chapter 9

e*Insight Authorization Activity Implementation (eIJSchema)

This chapter discusses the steps involved to enhance the previous case study to include the Authorization Activity.

You can use the Authorization Activity to stop the Business Process and wait for authorization. The decision to authorize or not authorize is entered via the e*Insight GUI.

This case study is a continuation of the previous example. See **"e*Insight Implementation (eIJSchema)" on page 96** for the initial configuration instructions.

9.1 Overview

The major steps in the implementation are:

1. Create and configure the Authorization Activity and Automated Activity in the e*Insight GUI.
2. Run and test the scenario.

The chapter begins with a description of the scenario and then shows how to use these steps to set it up.

9.1.1 Case Study: Payroll Processing

The bonus must be manually authorized by HR. After the bonus is calculated, the next activity is the Authorization Activity. The business process waits for a user to authorize or reject the activity in the e*Insight GUI. If the bonus is authorized, the process continues to the merge; otherwise the Bonus_Refused Activity runs. This activity uses an e*Insight script to update the Comments attribute. Then the process continues to the merge.

Important Considerations

The Authorization Activity has two fixed Local Attributes—**assignedTo** (the user to whom the Authorization process is assigned) and **performedBy** (a security measure to ensure the correct user is performing the Authorization). It is important to note the following:

- The user name of the assignedTo attribute must exactly match the name of the user logged into the e*Insight GUI or the name of the user group to which the name of the logged in user belongs
- The assignedTo attribute must have a value to complete the Authorization process.
- Any user assigned the role of Instance Manager can authorize, reject, or undo an Authorization Activity within a business process instance.

9.2 Step 1: Update the Payroll BP in e*Insight

The following is a summary of the procedure for creating a BP in the e*Insight GUI.

1. Add the Authorization Activity.
2. Edit the **assignedTo** Local Attribute to contain the correct user name.

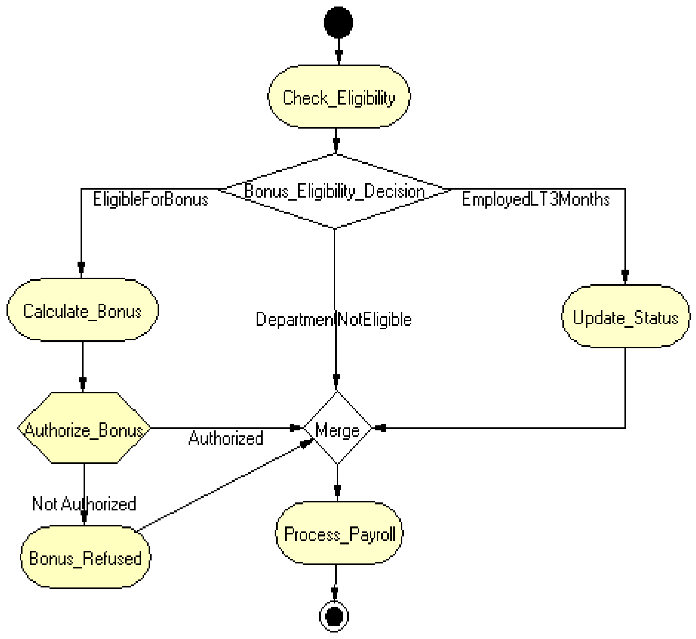
Note: *The Authorization Activity has two fixed Local Attributes—assignedTo (the user to whom the Authorization process is assigned) and performedBy (automatically assigned at run time as the user logged in to the e*Insight GUI). These two values must match in order for the user to Authorize, Reject, or Undo the business process instance.*

3. Add the additional Automated Activity.
4. Make the connections between the activities and merge.
5. Add the e*Insight script to the Authorization Activity.

For more information on creating a business process and using the e*Insight GUI, see the *e*Insight Business Process Manager User's Guide*.

Use the diagram shown in Figure 68 and the following tables to create the BP in e*Insight.

Figure 68 Payroll Business Process Model



9.2.1 Creating the processes performing the Activities

The Authorization Activity (**Authorize_Bonus**) can be configured to send notification to e*Gate. This means that a "DO" event is sent to e*Gate. However, unlike an Automated Activity, a "DONE" event is not expected to be returned to the engine. The "DONE" event is effectively created when the user clicks a button in the e*Insight GUI.

Sending a notification to e*Gate might be used to send an e-mail to the person who needs to authorize the activity. An e*Way would be configured that subscribes to the "DO" event and would perform the required processing to send an e-mail. In our example, we assume that no notification is required.

The **Bonus_Refused** activity is performed by an e*Insight Script. This is described below.

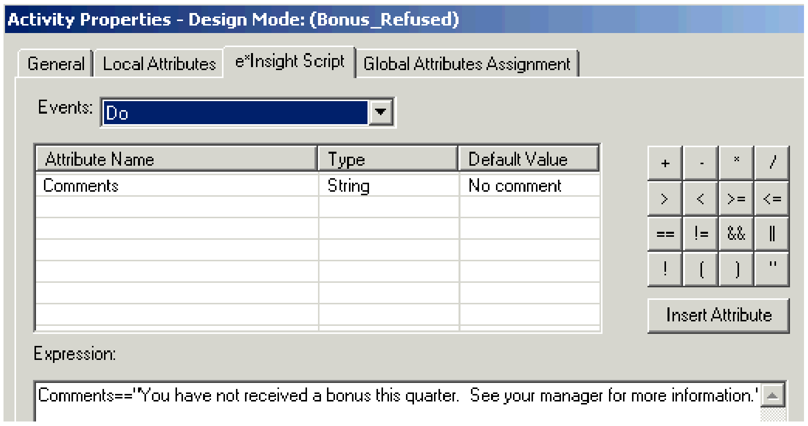
Configuring the e*Insight Script for Bonus_Refused

This script defines a message that appears on the pay slip. It sets the value of the **Comments** attribute to a short message indicating that the employee has not been paid a bonus.

To configure the e*Insight Script for Update_Status

1. From the **Bonus_Refused** properties, **Activity Performed by** area, select **e*Insight Script**.
2. Select the **e*Insight Script** tab.
3. Configure the script as shown in Figure 69.

Figure 69 Update_Status e*Insight Script Tab



Events: Do

Attribute Name	Type	Default Value
Comments	String	No comment

Expression:

```
Comments=='You have not received a bonus this quarter.  See your manager for more information.'
```

9.3 Step 2: Run and Test the e*Insight scenario

Once the schema has been set up in e*Gate you can run the scenario.

9.3.1 Testing the Standard Business Logic

The following procedure tests the additional logic provided by the Authorization Activity. The test is made by sending data that requires authorization and selecting both responses of authorized and not authorized.

Authorized Processing

Use the following procedure to test the functionality of the example.

1. Start the e*Insight GUI and select the Payroll business process. Switch to monitor mode.

Note: *Make sure that the business process has been enabled in the e*Insight GUI before attempting to run it.*

2. Make a final check of the e*Gate schema, to confirm all of the GUI associations. Make sure that all of the e*Insight components, including the user-defined components, are set to start automatically.
3. At the command line, type the following to start the schema. You must type the command on a single line.

```
stccb.exe -rh localhost -rs Payroll -ln localhost_cb
     -un username -up password
```

   Substitute the appropriate username and password for your installation.

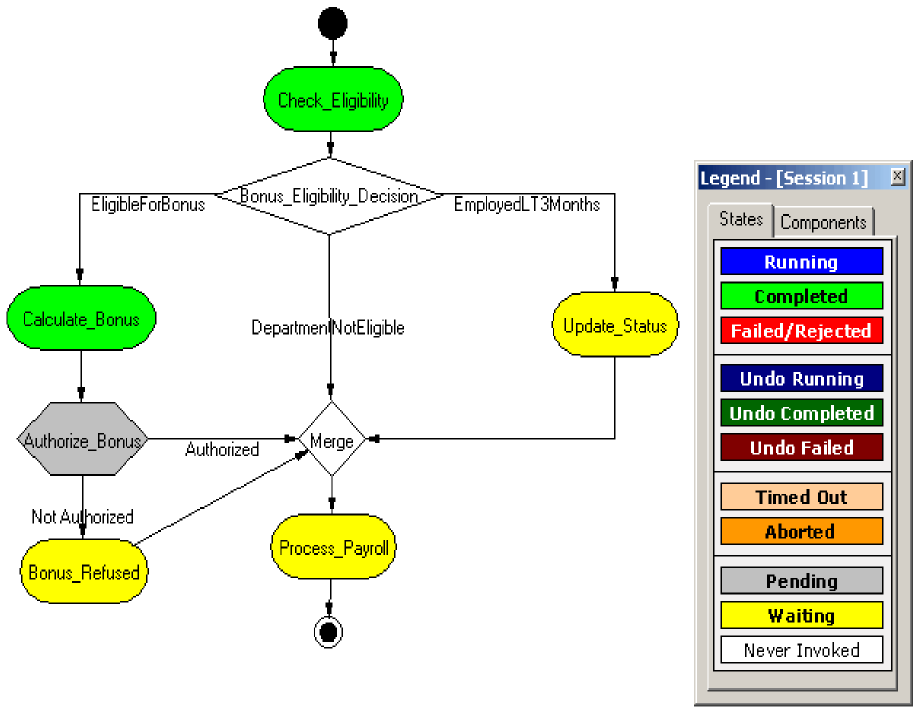
4. Start the e*Gate Monitor, and check the status of all the components. Any components displayed in red are not running. Any e*Insight components that are not running should be investigated before feeding data into the system.
5. Navigate to the location for the input data file, **Employee.~in**, (**c:\eGate\client\data\Payroll**) and change the extension to "**.fin**".

Note: *The change of the extension to "**.~in**" indicates that the data file has been picked up by the **START_BP** e*Way.*

6. Switch to the e*Insight GUI and, while in monitor mode, select the most recent BPI, and then observe the path that the data has taken.

   The **Authorize_Bonus** Authorization Activity should appear gray. This shows that the activity is pending.

Figure 70 Authorize_Bonus Pending BPI Diagram



7 Right-click the **Authorize_Bonus** activity from the tree view, then select **Properties** from the popup menu.

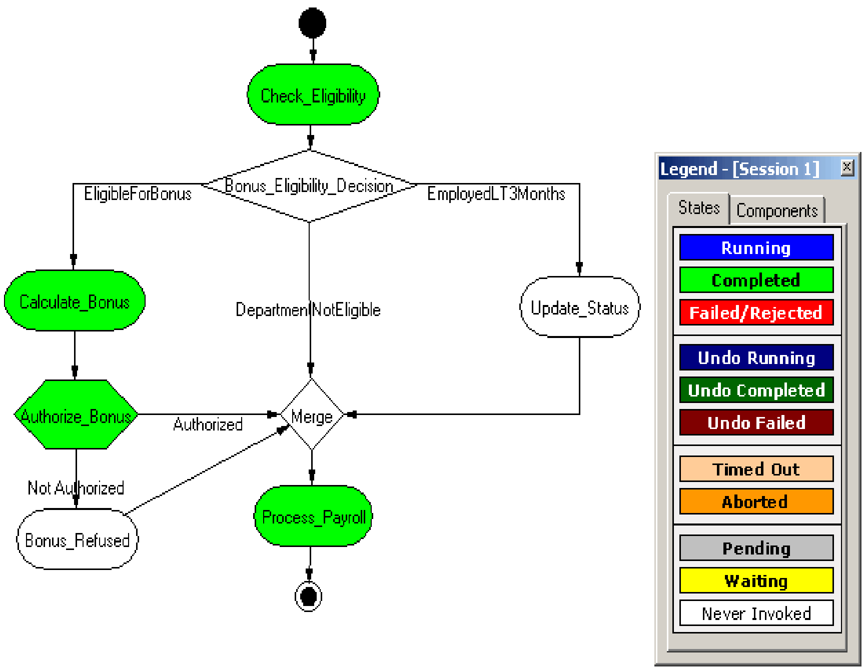
The **Authorization Activity Properties - Monitor Mode: (Authorize_Bonus)** is displayed.

8 Select the **Business Process Attributes** tab.

9 Click **Authorize**, and then click **OK** to close the **Activity Properties** dialog box.

10 The Business Process then completes using the route shown in Figure 71.

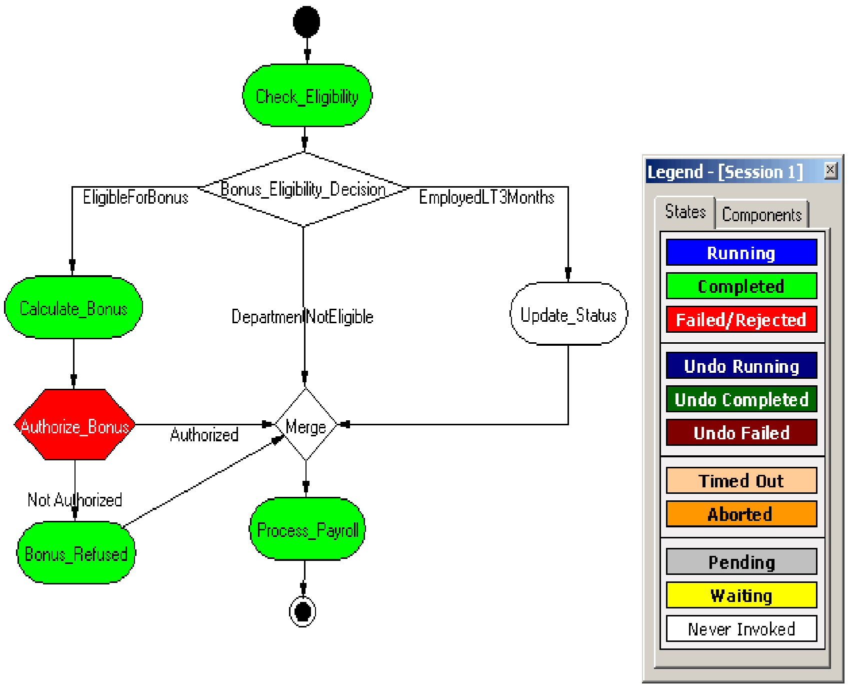
Figure 71 Authorization_Quantity - Authorized



Not Authorized Processing

Repeat the above procedure, but this time do not authorize the order.

Figure 72 Authorize_Bonus - Not Authorized



Chapter 10

e*Insight Implementation (eISchema)

This chapter discusses the steps involved to create an e*Insight Business Process Manager implementation based on eISchema (Classic).

The case study in this chapter was designed primarily to illustrate the functionality of e*Insight. In addition to showing a working example of a business process implementation, the following e*Insight features are demonstrated:

- Attribute value correction and business process restart
- Undoing a partially completed business process

This case study is extended to include authorization and user activities, and local, dynamic, and remote sub-processes.

Important: *The implementation contains instruction for using both Java and Monk for the Collaboration Rules scripts. You can use either the monk or java CRS for any BOB or e*Way, but do not configure both for the same module.*

10.1 Overview

The major tasks in the implementation are shown in Table 23.

Table 23 Overview of implementation tasks

	Task	Section
1	Create the business process (BP) in the e*Insight GUI	**"Create the ProcessOrder BP in e*Insight" on page 137**
2	Use the e*Insight GUI to configure the e*Gate schema that supports e*Insight	**"Configure the Integration Schema" on page 140**
3	Configure the e*Insight Engine	**"Configure the e*Insight Engine" on page 148**
4	Add and configure the user-defined e*Gate components	**"Configure User-defined e*Gate Components" on page 148**
5	Run and test the scenario	**"Run and Test the e*Insight scenario" on page 162**

The chapter begins with a description of the scenario and then shows how to set it up.

10.1.1 Case Study: Order Processing

The case study discussed in this chapter illustrates a simplified implementation of order processing. In this case, e*Insight receives an incoming order as a delimited text file. Once e*Insight has received the order an inventory check is made to see if the items ordered are in stock, if they are the order is shipped. An order status report is sent to the customer indicating whether or not the order was shipped.

Figure 74 shows the components involved in the business process implementation. The diagram is then separated into two sections and there is a description of how the data flows between these components.

Figure 73 e*Insight Data Flow Diagram

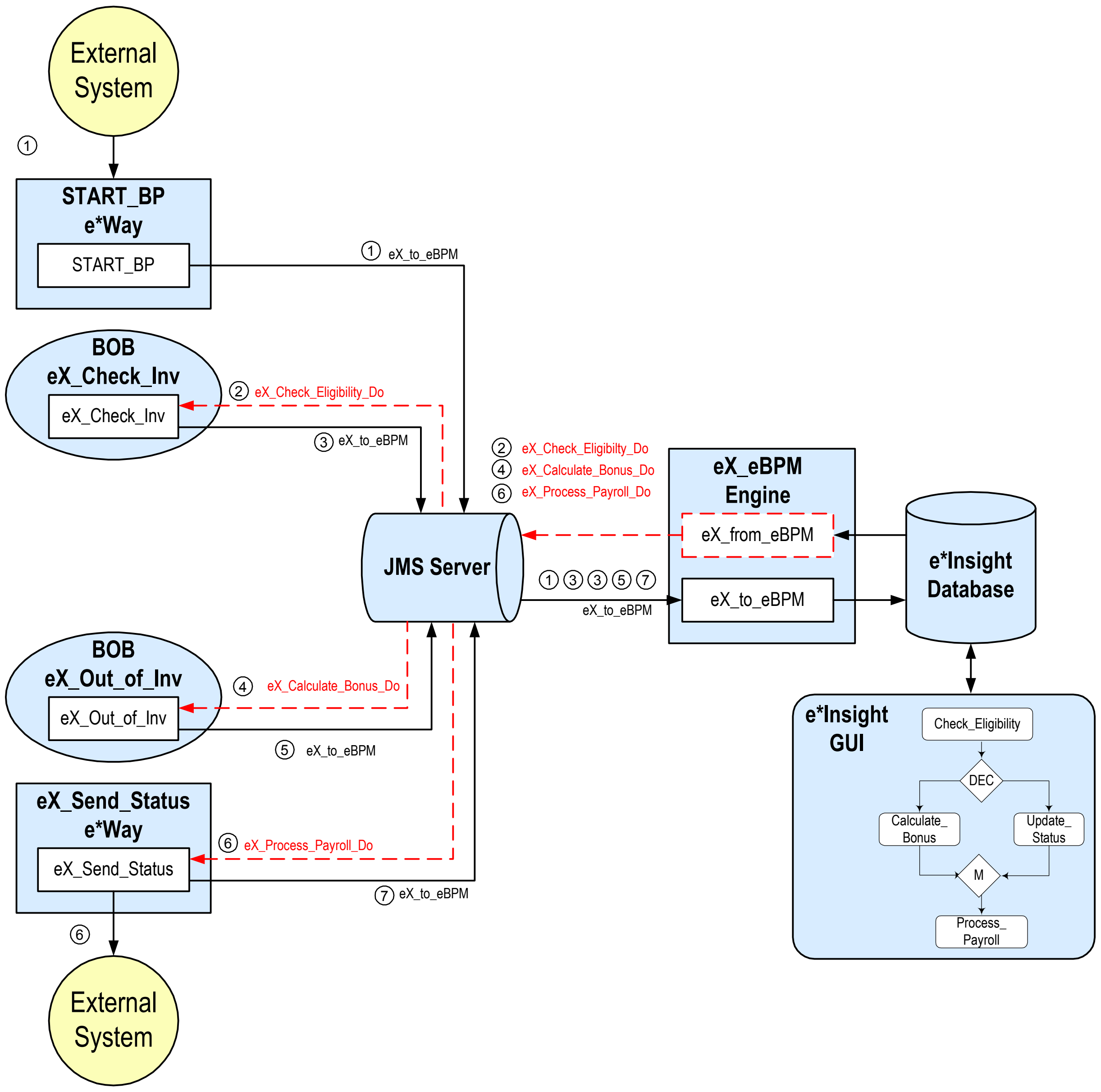
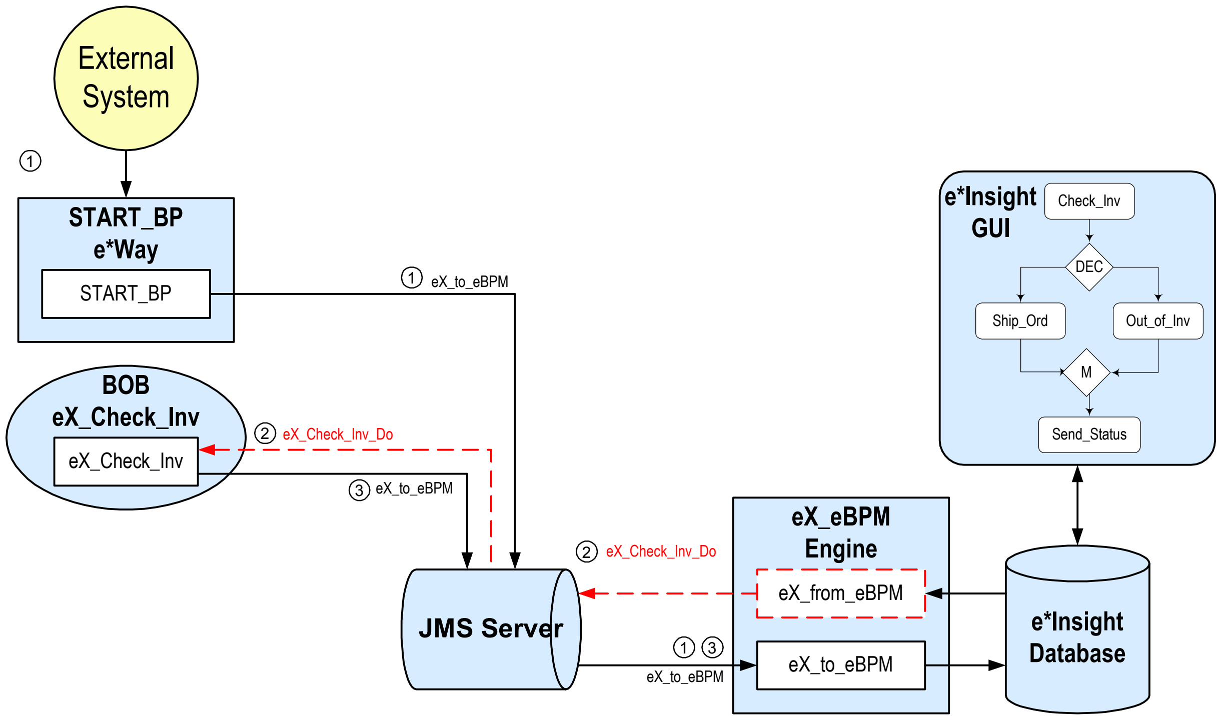


Figure 74 e*Insight Data Flow Diagram (Part 1)



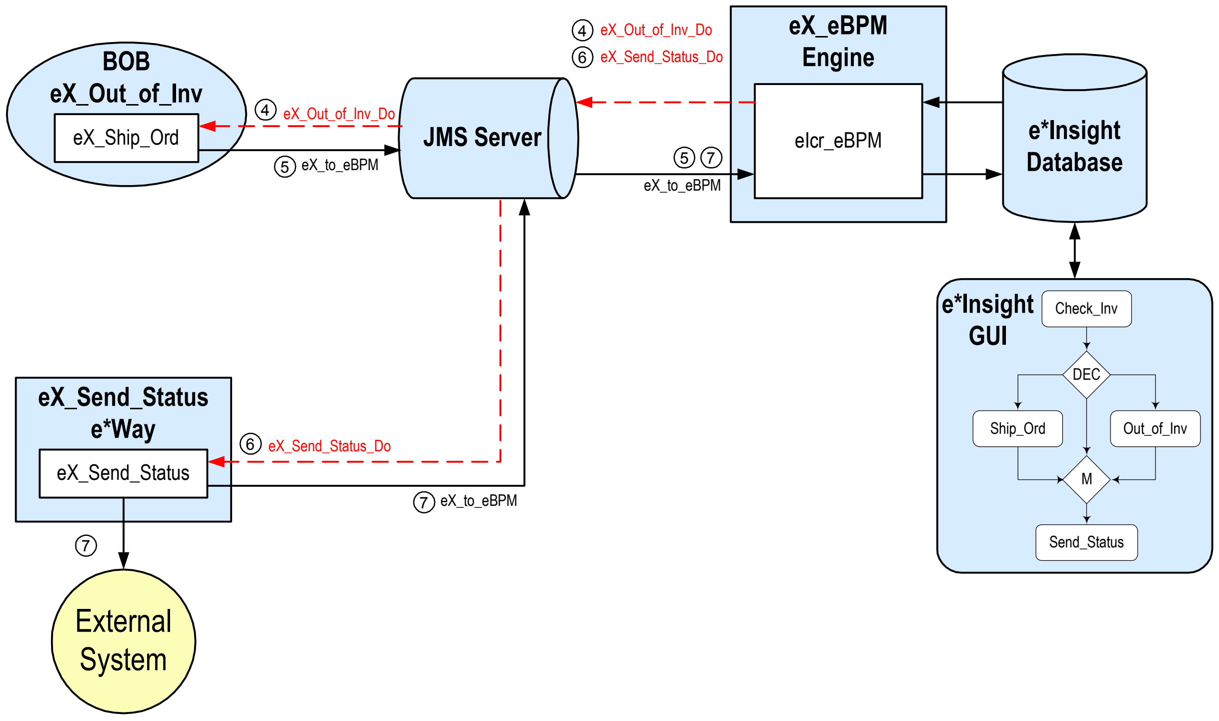
① The user-defined **START_BP** e*Way picks up the text file containing the order information from a shared location on the network, uses the order information to create the e*Insight Event that causes the e*Insight engine to start a business process instance, and publishes it using the **eX_to_eBPM** Event Type to the **eX_eBPM** IQ. The e*Insight engine retrieves the Event from the IQ and uses the information it contains to start the BPI.

② The e*Insight engine publishes a "Do" Event (**eX_Check_Inv_Do**) for first activity in the business process (**Check_Inv**). **eX_Check_Inv** BOB, the e*Gate component that corresponds to this activity in the business process, retrieves this Event from the **eX_eBPM** IQ and uses the information it contains to check the availability of the items ordered.

③ When the **Check_Inv** activity is finished, the **eX_Check_Inv** BOB publishes a "Done" Event using the **eX_to_eBPM** Event Type to the **eX_eBPM** IQ. The e*Insight engine retrieves the "Done" Event from the IQ, updates the BPI to reflect whether the items ordered are in stock, and then moves forward to the next activity in the business process based on the result of a decision gate. If the items are in stock, the next activity is **Ship_Ord**; otherwise the next activity is **Out_of_Inv**.

Let's assume the items are in stock. The e*Insight engine processes the e*Insight script corresponding to the **Ship_Ord** activity in the business process, and then moves forward to the next activity in the business process— **Send_Status**.

Figure 75 e*Insight Data Flow Diagram (Part 2)



④ Let's assume the items are not in stock. The e*Insight engine publishes a "Do" Event (**eX_Out_of_Inv_Do**) corresponding to the **Out_of_Inv** activity in the business process. The **eX_Out_of_Inv** BOB retrieves this Event from the **eX_eBPM** IQ and uses the information it contains to ship the order to the customer.

⑤ When the **Ship_Ord** activity is finished, the **eX_Out_of_Inv** BOB publishes a "Done" Event to the **eX_eBPM** IQ indicating that the order has been shipped. The e*Insight engine retrieves this Event, updates the BPI, and then moves forward to the next activity in the business process—**Send_Status**.

⑥ The e*Insight engine publishes a "Do" Event (**eX_Send_Status_Do**) corresponding to the **Send_Status** activity in the business process. The **eX_Send_Status** e*Way retrieves this Event from the **eX_eBPM** IQ and uses the information it contains to send a order status report to the customer.

⑦ The **eX_Send_Status** e*Way publishes two Events: one containing the status report to be sent to the customer to the external system responsible for sending out the report, and also the "Done" Event to the **eX_eBPM** IQ. The e*Insight engine retrieves the "Done" Event from the **eX_eBPM** IQ and uses the information it contains to update the BPI to indicate that the final activity in the business process has completed successfully.

10.2 Create the ProcessOrder BP in e*Insight

The following is a summary of the procedure for creating a BP in the e*Insight GUI.

1 Create a Business Process named ProcessOrder.

1 Add the activities.

2 Add the decision gates.

3 Make the connections between the activities and gates.

4 Add all the global attributes.

5 Assign global attributes to activities.

6 Add the logic to the decision gates.

7 Configure the properties for the activities.

For more information on creating this business process, see the *e*Insight Business Process Manager User's Guide*.

Use the diagram shown in Figure 76 and the following tables to create the BP in e*Insight.

Important: *Mark the check box for* ***Manual Restart*** *on the General tab of the Properties dialog box for each activity.*

Figure 76 ProcessOrder Business Process Model

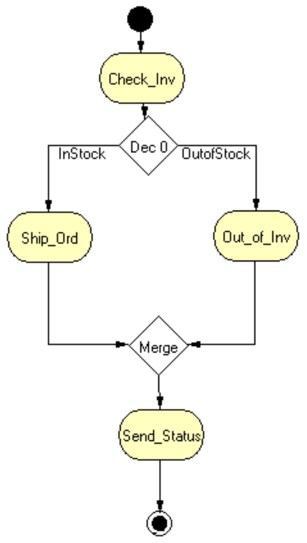


Table 24 BP Global Attributes

Attribute	Type	Data Direction
Address_Street	String	Input
Address_City	String	Input
Address_State	String	Input
Address_Zip	String	Input
Cust_Name	String	Input
Cust_email	String	Input
Item_Description	String	Input
Item_Number	String	Input
Order_Quantity	Number	Input
Order_Status	String	Internal
In_Stock	Boolean	Internal

Table 25 Activity Attributes

Activity	Attribute(s)	Input/Output
Check_Inv	Item_Number	Input/Output
	Order_Quantity	Input
	In_Stock	Output
	Order_Status	Output
Ship_Ord	Address_State	Input
	Order_Status	Output
Out_of_Inv	Item_Number	Input
	Item_Description	Input
	Order_Quantity	Input
	Order_Status	Output
Send_Status	Cust_email	Input
	Order_Status	Input
	Address_Street	Input
	Address_City	Input
	Address_State	Input
	Address_Zip	Input
	Cust_Name	Input
	Item_Description	Input
	Item_Number	Input
	Order_Quantity	Input

Table 26 Decision Gates

Feeding Activity	Expression
Check_Inv	In_Stock==true

10.2.1 Creating the processes performing the Activities

The activities in our scenario can either be performed by e*Gate or an e*Insight Script. Three of the activities (Check_Inv, Out_of_Inv, and Send_Status) use e*Gate. This is described in **“Configure the Integration Schema” on page 140**.

The Ship_Ord activity is performed by an e*Insight Script. This is described below.

Configuring the e*Insight Script for Ship_Ord

This script simulates the activity of sending out an item that is in stock. It sets the value of the Order_Status attribute to a short message indicating that the order has been sent from either California or Oregon depending on the customer’s zip code.

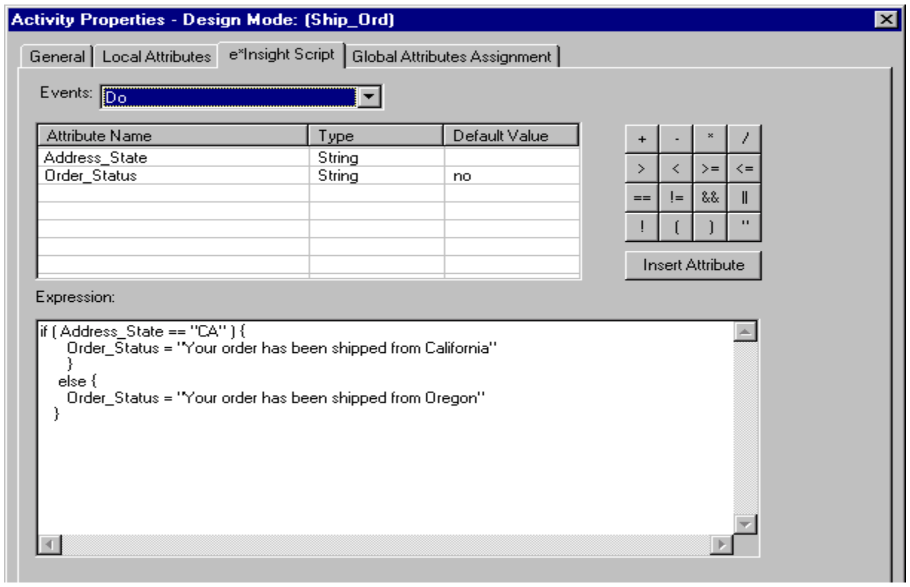
To configure the e*Insight Script for Ship_Ord

1 From the **Ship_Ord** properties, **Activity Performed by** area, select **e*Insight Script**.

2 Select the **e*Insight Script** tab.

3 Configure the script as shown in Figure 77.

Figure 77 Ship_Order e*Insight Script Tab



Events: Do

Attribute Name	Type	Default Value
Address_State	String	
Order_Status	String	no

Expression:

```
if ( Address_State == "CA" ) {
     Order_Status = "Your order has been shipped from California"
     }
   else {
     Order_Status = "Your order has been shipped from Oregon"
  }
```

10.3 Configure the Integration Schema

All the activities in this example, except Ship_Order are carried out using e*Gate components. You must first create a Schema (a copy of eISchema) with the basic components required for e*Insight. You then configure these components for your environment and create additional components for the activities.

To create a copy of eIJSchema

1. From the e*Insight GUI **File** menu, select **New e*Gate Schema**.
2. Enter or select a **Registry Host** on which to create the schema.
3. Enter a **Username** and **Password** that is valid on the Registry Host.
4. From the **Based on** list, select **eISchema (Classic)**.
5. In the **Name** box, enter **ProcessOrder**.
6. Click **OK**.

After creating the business process, you must configure the e*Gate Registry schema that supports the e*Insight system.

e*Insight allows you to specify the type of component (e*Way or BOB) associated with a particular activity and where it runs.

In the ProcessOrder example, all the components are BOBs except one: **Send_Status**. The **Send_Status** activity must be associated with an e*Way because it interfaces with an external component.

Integration Schema Activity Components Summary

The information in Table 27 shows a summary of the e*Gate components that support this example.

Table 27 Integration Schema Activity Components

Name	Type	Participating Host	Configuration Instructions
eX_Check_Inv	BOB	localhost	**“Creating the eX_Check_Inv BOB” on page 141**
eX_Out_of_Inv	BOB	localhost	**“Creating the eX_Out_of_Inv BOB” on page 144**
eX_Send_Status	e*Way	localhost	**“Send_Status e*Way Configuration” on page 147**

For information on how to use the e*Insight GUI to configure the e*Gate Registry see the *e*Insight Business Process Manger User’s Guide*.

Note: *This example runs all software components on a single machine (named "localhost"). In an actual implementation, these components could be distributed throughout a network, depending on the requirements of the system.*

Creating the eX_Check_Inv BOB

The eX_Check_Inv Collaboration runs the Check_Inv CRS. This checks to see what type of Event it has received, either a "Do" Event or an "Undo" Event. If it is an "Undo" Event the Order_Status attribute is populated with the string "The Check_Inv activity has been reversed" simulating the case of executing a compensating transaction for the activity.

If the Event is a "Do" Event, then the value for the Item_Number is checked. This simulates the checking of inventory by an inventory control system. Depending on what this number is the following happens.

99999 indicates that the Event is a failure. This simulates the case of an Event that contains bad data. The CRS sends a "FAILURE" Event back to the e*Insight engine indicating the activity could not be completed correctly. The e*Insight engine then implements the failure handling that is defined for this business process. In our example, the operator has the opportunity to make changes to the data an restart the business process.

33333 indicates that the item in question is in stock. The CRS sets the value of the In_Stock attribute to "yes" and sends a "SUCCESS" Event back to e*Insight, indicating that the activity has completed successfully.

Any other Item_Number is treated as being out of stock and the CRS set the value of In_Stock to "no" and sends "SUCCESS".

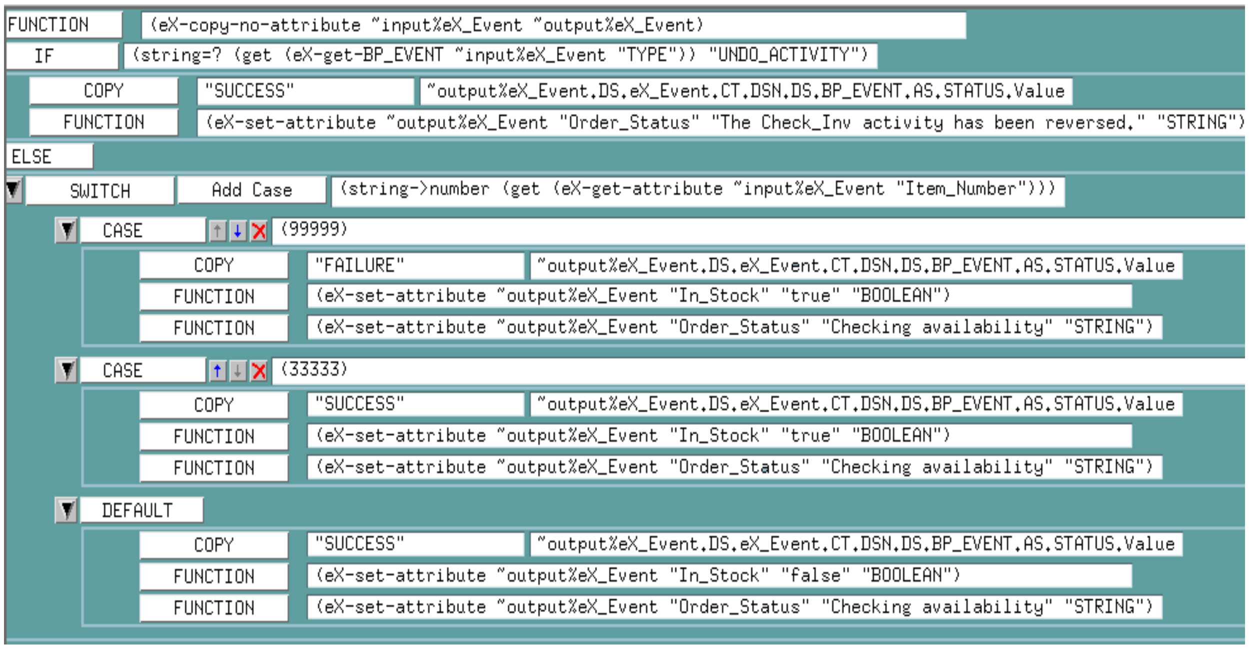
To configure the Check_Inv activity using Monk

1. In the e*Insight GUI, check that the Default Editor is Monk.
2. Open the Check_Inv activity properties.
3. On the **General** tab, select the **BOB** e*Gate module.
4. Click **New**.

   The **Define Collaboration** dialog appears.
5. Click **OK**.
6. Create eX_Check_Inv.tsc. The source and destination Event Type Definitions are eX_Standard_Event.

   Figure 78, on the following page, shows the **eX_Check_Inv.tsc** CRS used in this example.

Figure 78 eX_Check_Inv.tsc CRS (Monk)



```
FUNCTION   (eX-copy-no-attribute ~input%eX_Event ~output%eX_Event)
  IF       (string=? (get (eX-get-BP_EVENT ~input%eX_Event "TYPE")) "UNDO_ACTIVITY")
    COPY      "SUCCESS"   ~output%eX_Event.DS.eX_Event.CT.DSN.DS.BP_EVENT.AS.STATUS.Value
    FUNCTION  (eX-set-attribute ~output%eX_Event "Order_Status" "The Check_Inv activity has been reversed." "STRING")
ELSE
  SWITCH   Add Case   (string->number (get (eX-get-attribute ~input%eX_Event "Item_Number")))
    CASE   (99999)
      COPY      "FAILURE"   ~output%eX_Event.DS.eX_Event.CT.DSN.DS.BP_EVENT.AS.STATUS.Value
      FUNCTION  (eX-set-attribute ~output%eX_Event "In_Stock" "true" "BOOLEAN")
      FUNCTION  (eX-set-attribute ~output%eX_Event "Order_Status" "Checking availability" "STRING")
    CASE   (33333)
      COPY      "SUCCESS"   ~output%eX_Event.DS.eX_Event.CT.DSN.DS.BP_EVENT.AS.STATUS.Value
      FUNCTION  (eX-set-attribute ~output%eX_Event "In_Stock" "true" "BOOLEAN")
      FUNCTION  (eX-set-attribute ~output%eX_Event "Order_Status" "Checking availability" "STRING")
    DEFAULT
      COPY      "SUCCESS"   ~output%eX_Event.DS.eX_Event.CT.DSN.DS.BP_EVENT.AS.STATUS.Value
      FUNCTION  (eX-set-attribute ~output%eX_Event "In_Stock" "false" "BOOLEAN")
      FUNCTION  (eX-set-attribute ~output%eX_Event "Order_Status" "Checking availability" "STRING")
```

7. Validate and save the CRS.
8. Close the editor.
9. In the **Check_Inv** activity properties, click **Configure e*Gate Schema**.

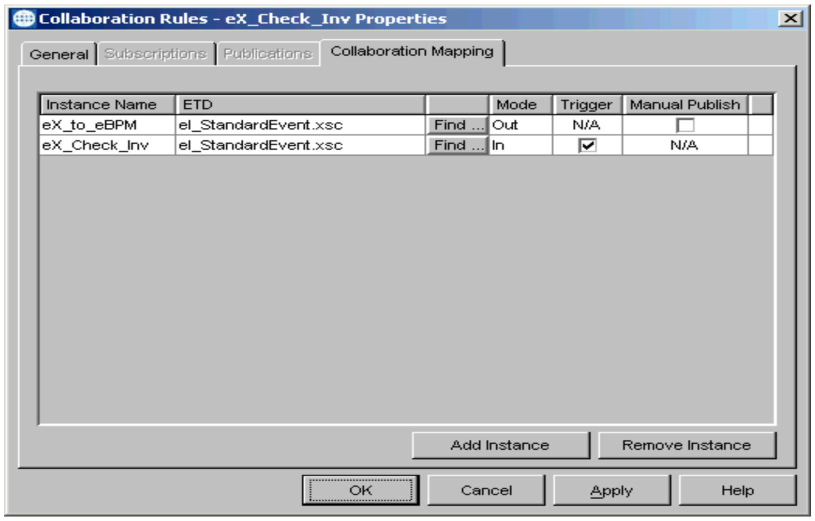
   You may be required to log into e*Gate.
10. Click **OK**, to close the information dialog.
11. Close the **Check_Inv** Activity properties.

To configure the Check_Inv activity using Java

1. In the e*Insight GUI, check that the Default Editor is Java.
2. Open the **Check_Inv** activity properties.
3. On the **General** tab, select the **BOB** e*Gate module.
4. Click **New**.

   The Define Collaboration dialog appears.
5. Select the **Define Mapping** tab.
6. Configure the instances as shown in Figure 79.

Figure 79 Define Mapping for eX_Check_Inv

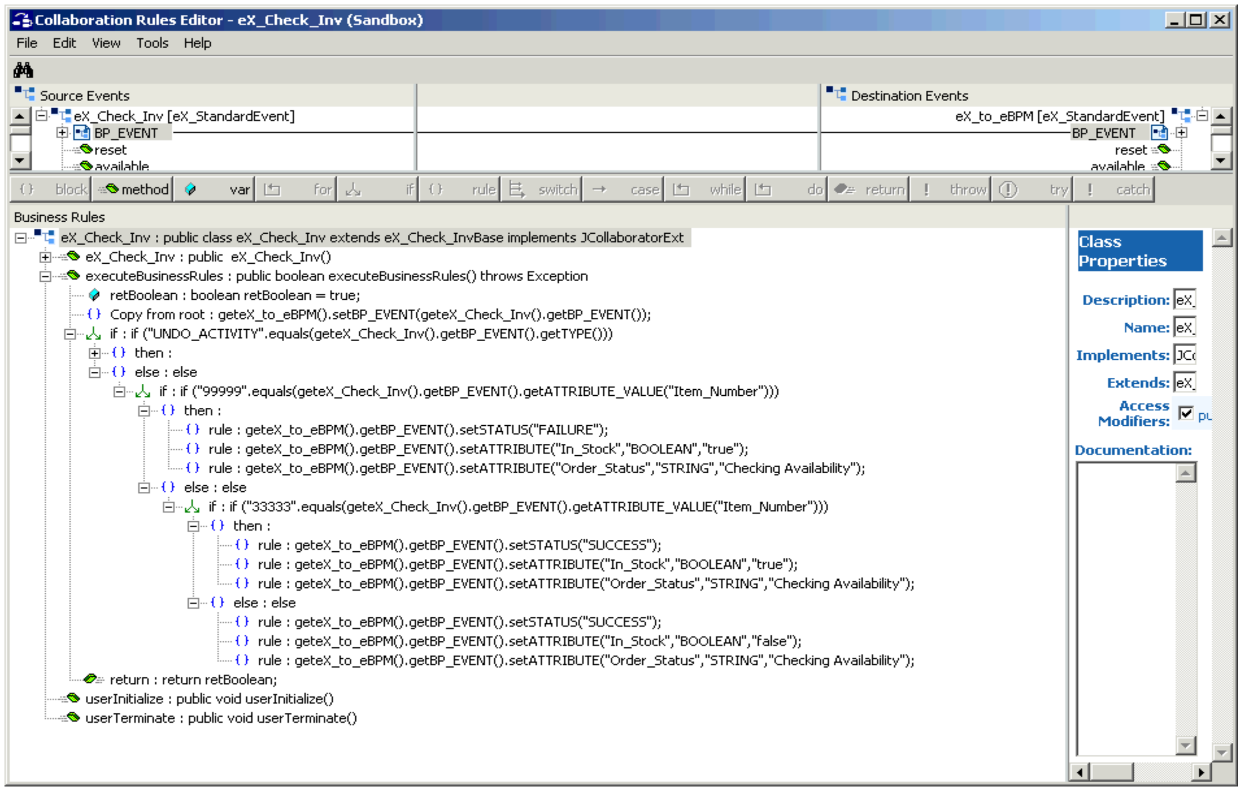


Instance Name	ETD		Mode	Trigger	Manual Publish
eX_to_eBPM	el_StandardEvent.xsc	Find ...	Out	N/A	☐
eX_Check_Inv	el_StandardEvent.xsc	Find ...	In	☑	N/A

7. Click **OK**.
8. Create eX_Check_Inv.xsc.

   Figure 80, on the following page, shows the **eX_Check_Inv.xsc** CRS used in this example.

Figure 80 eX_Check_Inv.xsc CRS (Java)


Collaboration Rules Editor - eX_Check_Inv (Sandbox)
File Edit View Tools Help
Source Events
eX_Check_Inv [eX_StandardEvent]
BP_EVENT
reset
available
Destination Events
eX_to_eBPM [eX_StandardEvent]
BP_EVENT
reset
available
block method var for if rule switch case while do return throw try catch
Business Rules
Class Properties
Description: eX
Name: eX
Implements: JC
Extends: eX
Access Modifiers: ✓ pu
Documentation:

```
eX_Check_Inv : public class eX_Check_Inv extends eX_Check_InvBase implements JCollaboratorExt
    eX_Check_Inv : public  eX_Check_Inv()
    executeBusinessRules : public boolean executeBusinessRules() throws Exception
        retBoolean : boolean retBoolean = true;
        Copy from root : geteX_to_eBPM().setBP_EVENT(geteX_Check_Inv().getBP_EVENT());
        if : if ("UNDO_ACTIVITY".equals(geteX_Check_Inv().getBP_EVENT().getTYPE()))
            then :
            else : else
                if : if ("99999".equals(geteX_Check_Inv().getBP_EVENT().getATTRIBUTE_VALUE("Item_Number")))
                    then :
                        rule : geteX_to_eBPM().getBP_EVENT().setSTATUS("FAILURE");
                        rule : geteX_to_eBPM().getBP_EVENT().setATTRIBUTE("In_Stock","BOOLEAN","true");
                        rule : geteX_to_eBPM().getBP_EVENT().setATTRIBUTE("Order_Status","STRING","Checking Availability");
                    else : else
                        if : if ("33333".equals(geteX_Check_Inv().getBP_EVENT().getATTRIBUTE_VALUE("Item_Number")))
                            then :
                                rule : geteX_to_eBPM().getBP_EVENT().setSTATUS("SUCCESS");
                                rule : geteX_to_eBPM().getBP_EVENT().setATTRIBUTE("In_Stock","BOOLEAN","true");
                                rule : geteX_to_eBPM().getBP_EVENT().setATTRIBUTE("Order_Status","STRING","Checking Availability");
                            else : else
                                rule : geteX_to_eBPM().getBP_EVENT().setSTATUS("SUCCESS");
                                rule : geteX_to_eBPM().getBP_EVENT().setATTRIBUTE("In_Stock","BOOLEAN","false");
                                rule : geteX_to_eBPM().getBP_EVENT().setATTRIBUTE("Order_Status","STRING","Checking Availability");
        return : return retBoolean;
    userInitialize : public void userInitialize()
    userTerminate : public void userTerminate()
```

9 Compile and save the CRS.

10 Close the editor.

11 In the **Check_Inv** Activity properties, click **Configure e*Gate Schema**.

You may be required to log into e*Gate.

12 Click **OK**, to close the information dialog.

13 Close the **Check_Inv** Activity properties.

Creating the eX_Out_of_Inv BOB

The Out_of_Inv translation implements the logic associated with processing an order for an item that is not in stock. The Item_Number is checked and a determination is made as to whether the item can be special ordered or a message must be created telling the customer that the item is unavailable.

In addition, this translation demonstrates how e*Insight handles “undoing” a partially completed business process. If Item_Number 11111 is encountered, then “FAILURE” is returned to the e*Insight engine which in turn issues “undo” Events for any activities upstream from the failed activity. In this example there is only one, Check_Inv, and the Check_Inv CRS handles reversing that already completed activity.

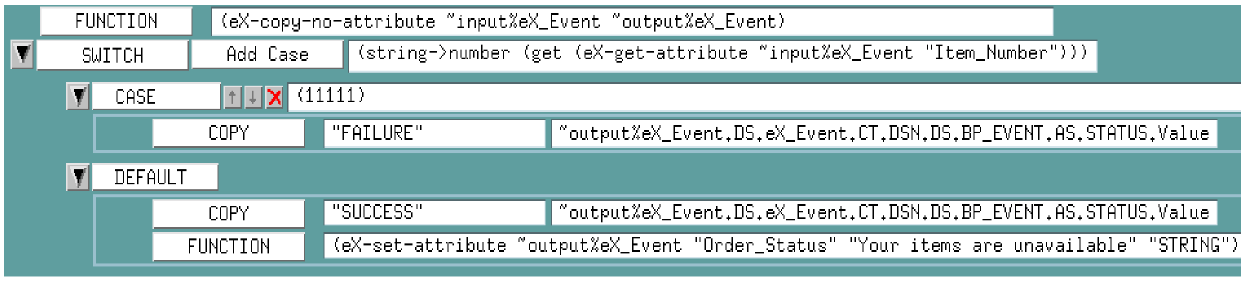
To configure the Out_of_Inv activity using Monk

1. In the e*Insight GUI, check that the Default Editor is Monk.
2. Open the **Out_of_Inv** activity properties.
3. On the **General** tab, select the **BOB** e*Gate module.
4. Click **New**.

   The **Define Collaboration** dialog appears.
5. Click **OK**.
6. Create eX_Out_of_Inv.tsc. The source and destination Event Type Definitions are eX_Standard_Event.ssc.

   Figure 81 shows the eX_Out_of_Inv CRS used in this example.

Figure 81 eX_Out_of_Inv.tsc CRS (Monk)



```
FUNCTION    (eX-copy-no-attribute ~input%eX_Event ~output%eX_Event)
SWITCH      Add Case    (string->number (get (eX-get-attribute ~input%eX_Event "Item_Number")))
    CASE    (11111)
        COPY        "FAILURE"     ~output%eX_Event.DS.eX_Event.CT.DSN.DS.BP_EVENT.AS.STATUS.Value
    DEFAULT
        COPY        "SUCCESS"     ~output%eX_Event.DS.eX_Event.CT.DSN.DS.BP_EVENT.AS.STATUS.Value
        FUNCTION    (eX-set-attribute ~output%eX_Event "Order_Status" "Your items are unavailable" "STRING")
```

7. Validate and save the CRS.
8. Close the editor.
9. In the **Out_of_Inv** Activity properties, click **Configure e*Gate Schema**.

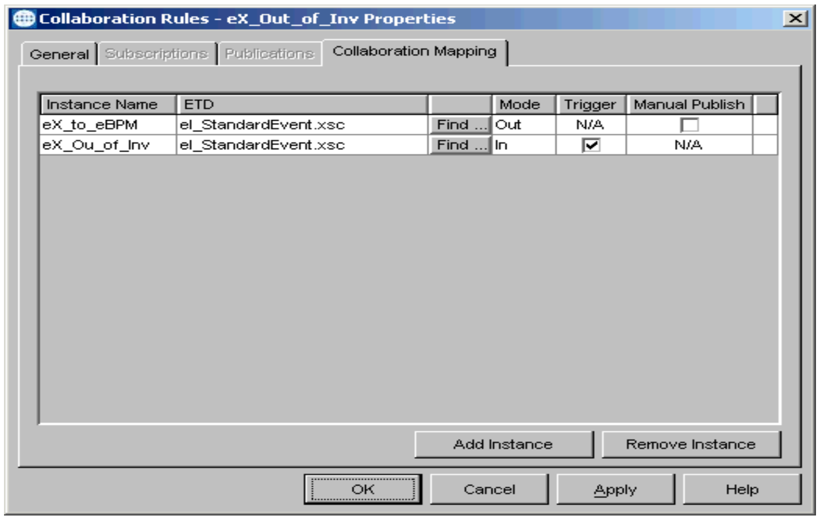
   You may be required to log into e*Gate.
10. Click **OK**, to close the information dialog.
11. Close the **Out_of_Inv** Activity properties.

To configure the Out_of_Inv activity using Java

1. In the e*Insight GUI, check that the Default Editor is Java.
2. Open the **Out_of_Inv** activity properties.
3. On the **General** tab, select the **BOB** e*Gate module.
4. Click **New**.

   The **Define Collaboration** dialog appears.
5. Select the **Define Mapping** tab.
6. Configure instances as shown in Figure 82.

Figure 82 Define Mapping for eX_Out_of_Inv (Java)

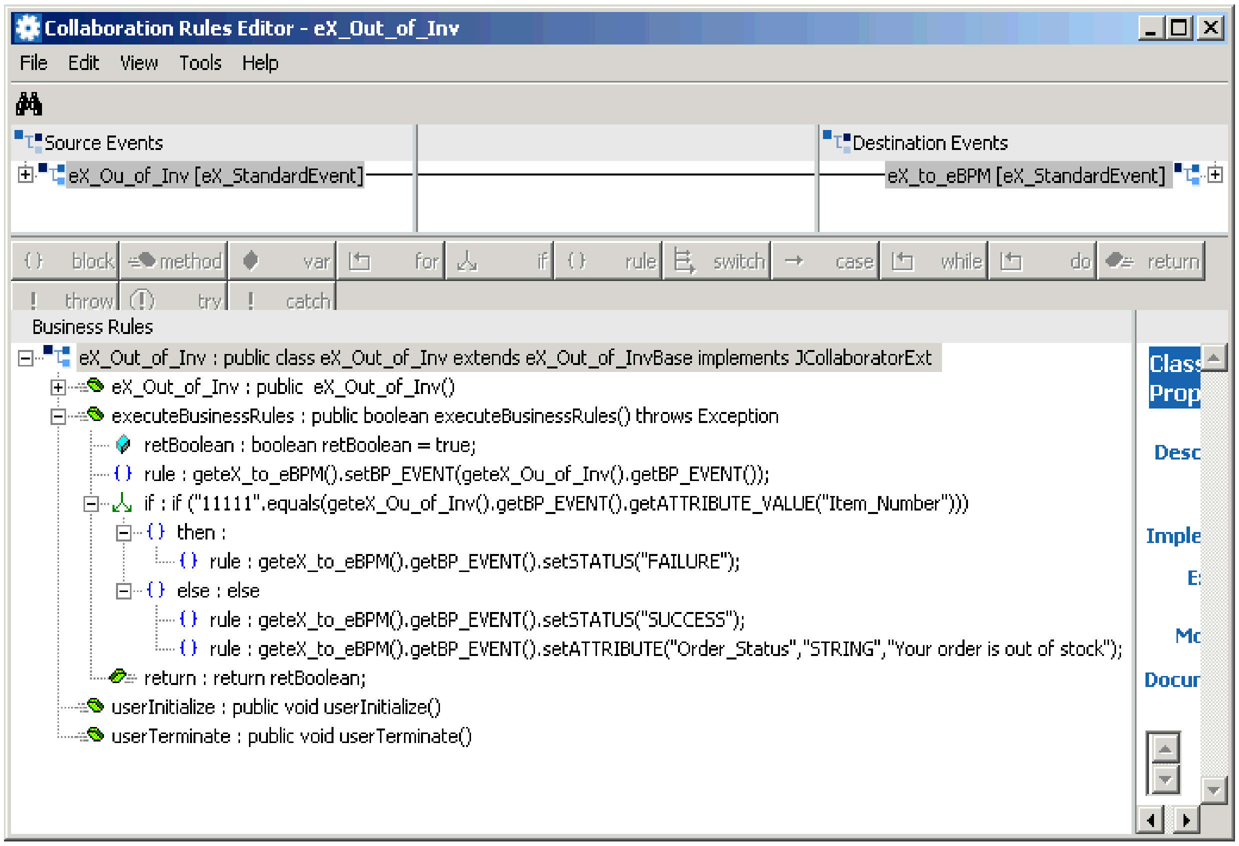


Instance Name	ETD		Mode	Trigger	Manual Publish
eX_to_eBPM	el_StandardEvent.xsc	Find ...	Out	N/A	☐
eX_Ou_of_Inv	el_StandardEvent.xsc	Find ...	In	☑	N/A

7. Click **OK**.
8. Create eX_Out_of_Inv.tsc.

   Figure 83 shows the eX_Out_of_Inv CRS used in this example.

Figure 83 eX_Out_of_Inv.xsc CRS (Java)



```
eX_Out_of_Inv : public class eX_Out_of_Inv extends eX_Out_of_InvBase implements JCollaboratorExt
    eX_Out_of_Inv : public  eX_Out_of_Inv()
    executeBusinessRules : public boolean executeBusinessRules() throws Exception
        retBoolean : boolean retBoolean = true;
        rule : geteX_to_eBPM().setBP_EVENT(geteX_Ou_of_Inv().getBP_EVENT());
        if : if ("11111".equals(geteX_Ou_of_Inv().getBP_EVENT().getATTRIBUTE_VALUE("Item_Number")))
            then :
                rule : geteX_to_eBPM().getBP_EVENT().setSTATUS("FAILURE");
            else : else
                rule : geteX_to_eBPM().getBP_EVENT().setSTATUS("SUCCESS");
                rule : geteX_to_eBPM().getBP_EVENT().setATTRIBUTE("Order_Status","STRING","Your order is out of stock");
        return : return retBoolean;
    userInitialize : public void userInitialize()
    userTerminate : public void userTerminate()
```

9 Compile and save the CRS.

10 Close the editor.

11 In the Out_of_Inv Activity properties, click **Configure e*Gate Schema**.

You may be required to log into e*Gate.

12 Click OK, to close the information dialog.

13 Close the Out_of_Inv Activity properties.

Send_Status e*Way Configuration

The Send_Status Collaboration is configured using e*Gate, see **"Configure the Send_Status e*Way" on page 156**. The e*Way and basic components should be created from within the e*Insight GUI.

To create the Send_Status Activity e*Way

1 In the e*Insight GUI, open the **Send_Status** activity properties.

2 On the **General** tab, select the **e*Way** e*Gate module.

3 Click **Configure e*Gate Schema**.

You may be required to log into e*Gate.

4 Click **OK**, to close the information dialog.

5 Close the **Send_Status** Activity properties.

10.4 Configure the e*Insight Engine

The e*Insight engine runs in a specially configured Java e*Way. You must make changes to the configuration file for this e*Way to conform to the requirements of your system. For example, you must specify the name of the e*Insight database to which the e*Way connects.

Note: *This example uses only one e*Insight engine. In an actual implementation, more than one e*Insight engine can be configured to handle the required workload. In such a case, you must make changes to each of the e*Insight engines.*

Edit the eX_eBPM Configuration File

Most of the parameter settings in the eX_eBPM engine's configuration file should not be changed. Table 28 discusses the parameters that may need to be changed depending on the implementation. Use the e*Way Editor and the information in **"Configuring the e*Insight Engine" on page 53** to make the required changes for the ProcessOrder example.

10.5 Configure User-defined e*Gate Components

The user-defined components in an e*Insight implementation consist of two types: the first type *starts* the business process, and second type runs *as part of* the business process. The activity components are of the second type.

The ProcessOrder example uses a file e*Way to start the business process and BOBs to run all the activities except the last. The last activity is represented by an additional file e*Way.

Configuration Order for the User-defined Components

Table 28 shows the configuration order for the user-defined components.

Table 28 Configuration Order for User-defined Components

	Task	Section
1	Add and configure the START_BP e*Way	**"Configure the START_BP e*Way" on page 149**
2	Configure the Send_Status e*Way	**"Configure the Send_Status e*Way" on page 156**

Important: *All the integration schema associations are displayed in table format at the end of this section. The sections dealing with e*Way configuration include tables detailing* *the non-default e*Way parameter settings. The sections dealing with the Monk and Java Collaboration Rules Scripts show screen shots of these scripts as they appear in the e*Gate Collaboration Editor.*

10.5.1 Configure the START_BP e*Way

The e*Way that sends the Event that starts the business process, named **START_BP** in this example, must convert the incoming data into e*Insight Event format, as well as send the appropriate acknowledgment to the e*Insight engine to create the Business Process Instance (BPI).

The **START_BP** e*Way is completely user defined and must be added to the **eISchema** in the e*Gate Enterprise Manager. In an actual implementation, the choice of e*Way (or BOB) would depend on the requirements of the situation. For example, if the data were coming from an SAP system, you might select an SAP ALE e*Way; or if the data were already in the e*Gate system, you could use a BOB to start the BPI. In the present case, a text file on the local system provides the input data, therefore this example uses a file e*Way to send the “Start” Event to the e*Insight engine.

Table 29 shows the steps to configure the **START_BP** e*Way.

Table 29 Configuration steps for the START_BP e*Way

	Step	Section	
		Monk	Java
1	Add the e*Way and create the e*Way configuration file	“Step 1: Create the START_BP e*Way using Monk” on page 149	“Step 1: Create the START_BP e*Way using Java” on page 152
2	Create the Input ETD	“Step 2: Create the Input ETD using Monk” on page 150	“Step 2: Create the Input ETD using Java” on page 152
3	Create the START_BP Collaboration Rules script (CRS)	“Step 3: Create the START_BP CRS using Monk” on page 150	“Step 3: Create the START_BP Collaboration using Java” on page 153
4	Configure the Collaboration in the GUI	“Step 4: Configure the START_BP Collaboration in the GUI using Monk” on page 151	“Step 4: Configure the Collaboration in the GUI using Java” on page 155

Step 1: Create the START_BP e*Way using Monk

The e*Way for the ProcessOrder example is a simple file e*Way (executable: **stcewfile.exe**) that polls a directory (**<eGate>\client\data\ProcessOrder**) for any file with the extension “.fin” and moves it into the e*Insight system.

Use the Enterprise Manager and the following table to add the **START_BP** e*Way and create its configuration file.

Table 30 Start e*Way Parameters

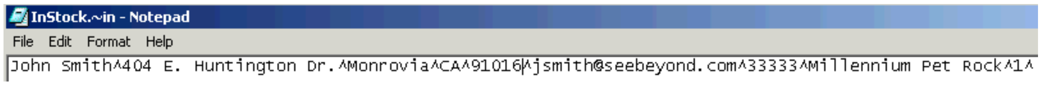
Screen	Parameter	Setting
General Settings	(All)	(Default)
Outbound (send) settings	(All)	(Default)
Poller (inbound) settings	PollDirectory	<eGate>\client\data\ProcessOrder
	(All others)	(Default)
Performance Testing	(All)	(Default)

Step 2: Create the Input ETD using Monk

The input ETD is based on the format of the input data. The ProcessOrder example uses a delimited text file (**InStock.~in**) that contains the data needed to process the order.

The input data file used in this example is shown in Figure 84. Place this data file at the directory location **<eGate>\client\data\ProcessOrder**.

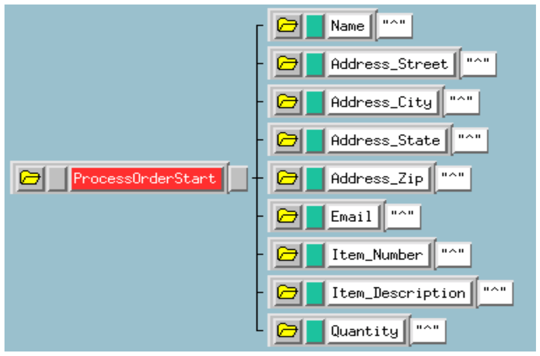
Figure 84 Input Text File (InStock.~in)



```
John Smith^404 E. Huntington Dr.^Monrovia^CA^91016^jsmith@seebeyond.com^33333^Millennium Pet Rock^1^
```

Using the ETD Editor and the input data as a guide, create an ETD like the one shown in Figure 85. For more information on using the ETD Editor see the ETD Editor's online help.

Figure 85 Input ETD: ProcessOrderStart.ssc (Monk)



Step 3: Create the START_BP CRS using Monk

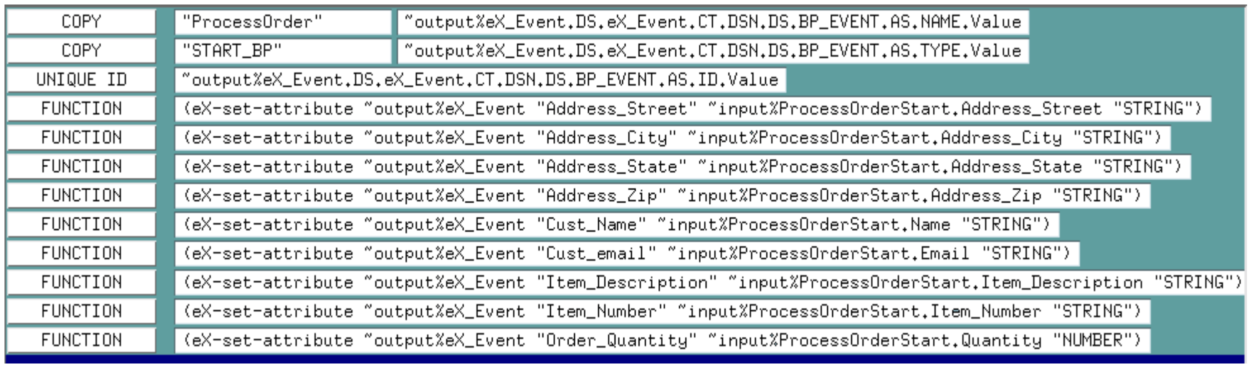
The Collaboration that sends the Event that starts the BPI must do two things:

- Put the data into e*Insight ETD (**eX_Standard_Event.ssc)** format.
- Populate the Event with the information the e*Insight engine needs to start a BPI.

In addition to these two tasks, the **START_BP** Collaboration also provides the recommended location for setting any global attributes that are required in your business process.

Figure 86, shows the **START_BP** CRS used in the ProcessOrder example:

Figure 86 START_BP CRS (Monk)



COPY	"ProcessOrder"	~output%eX_Event.DS.eX_Event.CT.DSN.DS.BP_EVENT.AS.NAME.Value
COPY	"START_BP"	~output%eX_Event.DS.eX_Event.CT.DSN.DS.BP_EVENT.AS.TYPE.Value
UNIQUE ID	~output%eX_Event.DS.eX_Event.CT.DSN.DS.BP_EVENT.AS.ID.Value	
FUNCTION	(eX-set-attribute ~output%eX_Event "Address_Street" ~input%ProcessOrderStart.Address_Street "STRING")	
FUNCTION	(eX-set-attribute ~output%eX_Event "Address_City" ~input%ProcessOrderStart.Address_City "STRING")	
FUNCTION	(eX-set-attribute ~output%eX_Event "Address_State" ~input%ProcessOrderStart.Address_State "STRING")	
FUNCTION	(eX-set-attribute ~output%eX_Event "Address_Zip" ~input%ProcessOrderStart.Address_Zip "STRING")	
FUNCTION	(eX-set-attribute ~output%eX_Event "Cust_Name" ~input%ProcessOrderStart.Name "STRING")	
FUNCTION	(eX-set-attribute ~output%eX_Event "Cust_email" ~input%ProcessOrderStart.Email "STRING")	
FUNCTION	(eX-set-attribute ~output%eX_Event "Item_Description" ~input%ProcessOrderStart.Item_Description "STRING")	
FUNCTION	(eX-set-attribute ~output%eX_Event "Item_Number" ~input%ProcessOrderStart.Item_Number "STRING")	
FUNCTION	(eX-set-attribute ~output%eX_Event "Order_Quantity" ~input%ProcessOrderStart.Quantity "NUMBER")	

Step 4: Configure the START_BP Collaboration in the GUI using Monk

In addition to creating the configuration file for the e*Way and the CRS used by the Collaboration, you must also configure the **START_BP** e*Way's Collaboration in the Enterprise Manager GUI.

To configure the Collaboration

1. Create a Collaboration Rule, **START_BP**, that uses the Monk service and the START_BP CRS created in step 2, subscribes to the **eX_External_Evt** Event Type, and publishes to the **eX_to_eBPM** Event Type.
2. Create a Collaboration for the **START_BP** e*Way that uses the **START_BP** Collaboration Rule, subscribes to the **eX_External_Evt** Event Type from **<EXTERNAL>**, and publishes the **eX_to_eBPM** Event Type to the **eX_eBPM** IQ.

Step 1: Create the START_BP e*Way using Java

The e*Way for the ProcessOrder example is a simple file e*Way (executable: **stcewfile.exe**) that polls a directory (**<eGate>\client\data\ProcessOrder**) for any file with the extension ".fin" and moves it into the e*Insight system.

Use the Enterprise Manager and the following table to add the **START_BP** e*Way and create its configuration file.

Table 31 Start e*Way Parameters

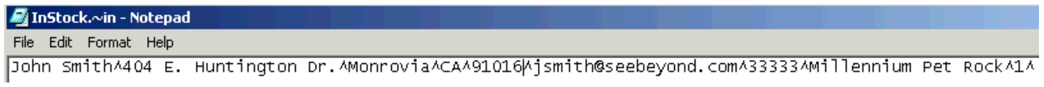
Screen	Parameter	Setting
General Settings	(All)	(Default)
Outbound (send) settings	(All)	(Default)
Poller (inbound) settings	PollDirectory	<eGate>\client\data\ProcessOrder
	(All others)	(Default)
Performance Testing	(All)	(Default)

Step 2: Create the Input ETD using Java

The input ETD is based on the format of the input data. The ProcessOrder example uses a delimited text file (**InStock.fin**) that contains the data needed to process the order.

The input data file used in this example is shown in Figure 87. Place this data file at the directory location **c:\eGate\client\data\ProcessOrder**.

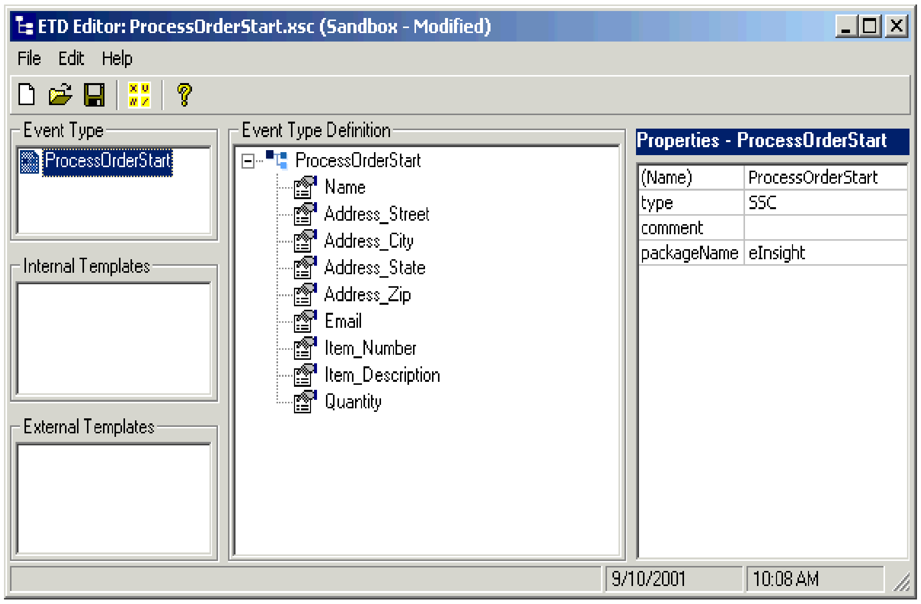
Figure 87 Input Text File (InStock.fin)



```
John Smith^404 E. Huntington Dr.^Monrovia^CA^91016^jsmith@seebeyond.com^33333^Millennium Pet Rock^1^
```

Using the ETD Editor and the input data as a guide, create an ETD like the one shown in Figure 88. Set the global delimiter to a **^** character.For more information on using the ETD Editor see the ETD Editor's online help.

Figure 88 Input ETD: ProcessOrderStart.xsc (Java)



- ProcessOrderStart
  - Name
  - Address_Street
  - Address_City
  - Address_State
  - Address_Zip
  - Email
  - Item_Number
  - Item_Description
  - Quantity

(Name)	ProcessOrderStart
type	SSC
comment	
packageName	eInsight

Step 3: Create the START_BP Collaboration using Java

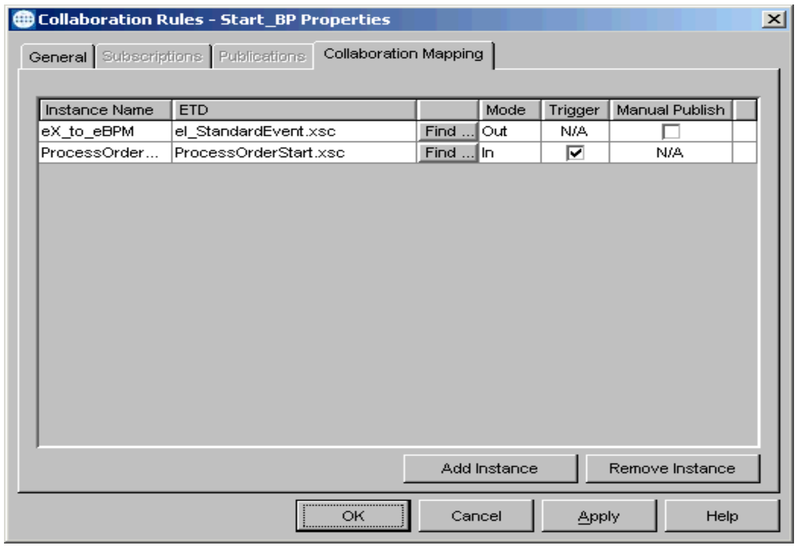
The Collaboration that sends the Event that starts the BPI must do two things:

- Put the data into e*Insight ETD (**eI_Standard_Event.xsc)** format.
- Populate the Event with the information the e*Insight engine needs to start a BPI.

In addition to these two tasks, the **START_BP** Collaboration also provides the recommended location for setting any global attributes that are required in your business process.

1. Create a Collaboration Rule, **START_BP**, that uses the Java service.
2. Configure the Collaboration Mapping tab, as shown in Figure 89.

Figure 89 Start_BP Properties, Collaboration Mapping Tab (Java)


Collaboration Rules - Start_BP Properties

General | Subscriptions | Publications | Collaboration Mapping

Instance Name	ETD		Mode	Trigger	Manual Publish
eX_to_eBPM	eI_StandardEvent.xsc	Find ...	Out	N/A	☐
ProcessOrder...	ProcessOrderStart.xsc	Find ...	In	☑	N/A

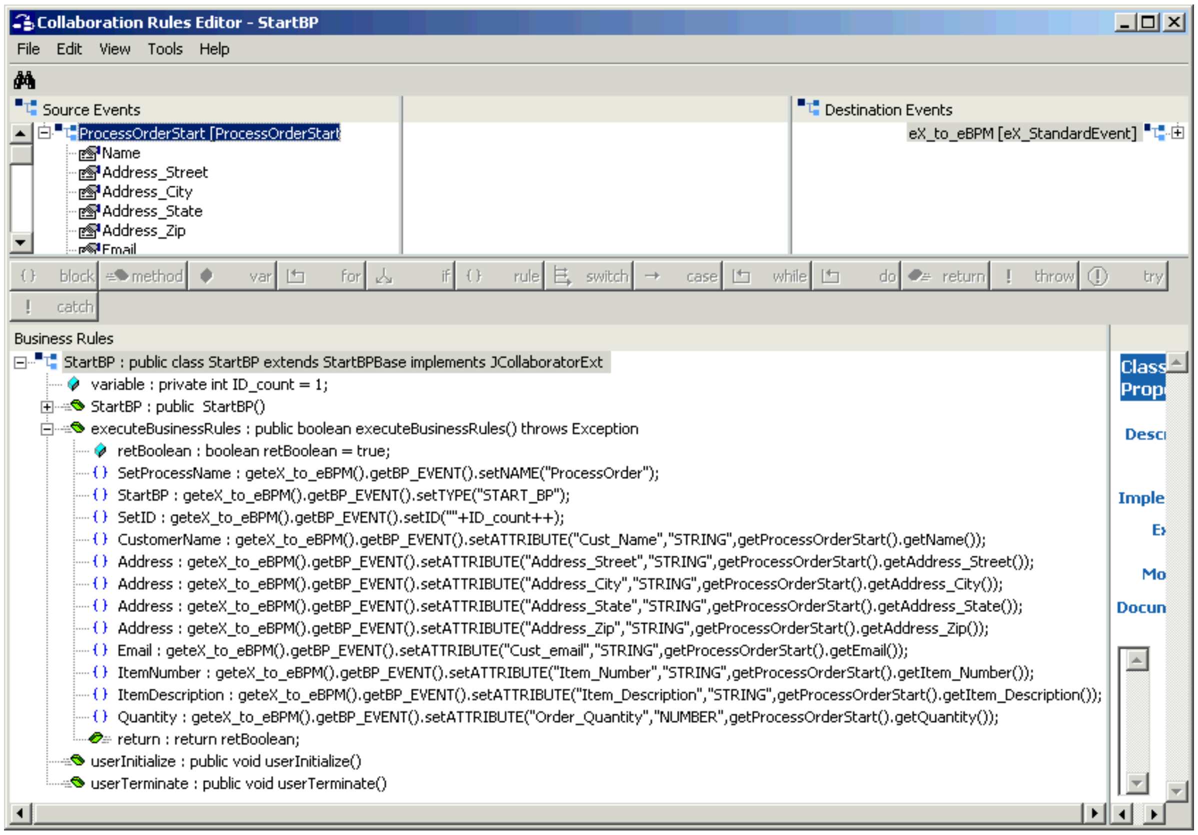
Add Instance | Remove Instance

OK | Cancel | Apply | Help

3 Click **Apply**, and click the **General Tab**.

4 Click **New** to create a new CRS, as show in Figure 90.

Figure 90 START_BP CRS (Java)


Collaboration Rules Editor - StartBP
File Edit View Tools Help
block method var for if rule switch case while do return throw try catch

Source Events
ProcessOrderStart [ProcessOrderStart]
Name
Address_Street
Address_City
Address_State
Address_Zip
Email

Destination Events
eX_to_eBPM [eX_StandardEvent]

Business Rules

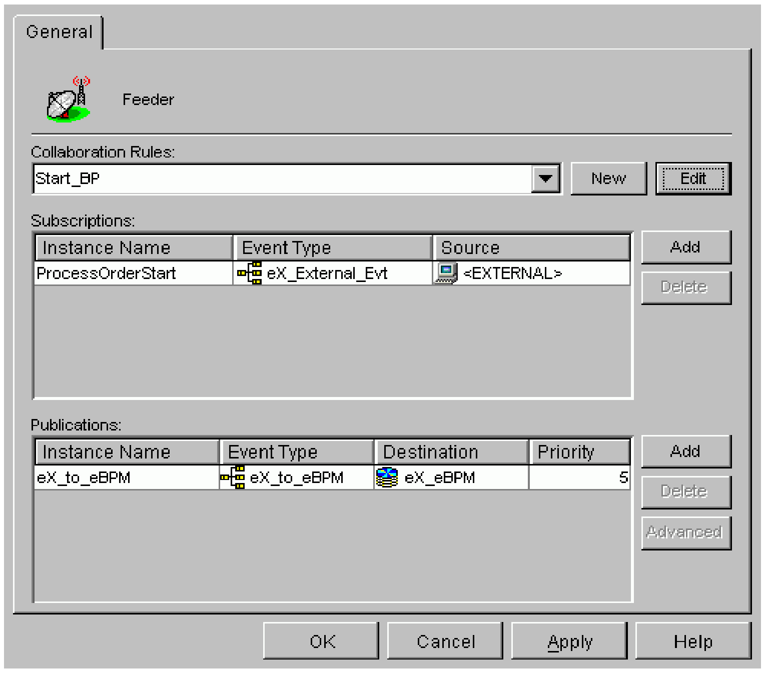
```
StartBP : public class StartBP extends StartBPBase implements JCollaboratorExt
    variable : private int ID_count = 1;
    StartBP : public  StartBP()
    executeBusinessRules : public boolean executeBusinessRules() throws Exception
        retBoolean : boolean retBoolean = true;
        SetProcessName : geteX_to_eBPM().getBP_EVENT().setNAME("ProcessOrder");
        StartBP : geteX_to_eBPM().getBP_EVENT().setTYPE("START_BP");
        SetID : geteX_to_eBPM().getBP_EVENT().setID(""+ID_count++);
        CustomerName : geteX_to_eBPM().getBP_EVENT().setATTRIBUTE("Cust_Name","STRING",getProcessOrderStart().getName());
        Address : geteX_to_eBPM().getBP_EVENT().setATTRIBUTE("Address_Street","STRING",getProcessOrderStart().getAddress_Street());
        Address : geteX_to_eBPM().getBP_EVENT().setATTRIBUTE("Address_City","STRING",getProcessOrderStart().getAddress_City());
        Address : geteX_to_eBPM().getBP_EVENT().setATTRIBUTE("Address_State","STRING",getProcessOrderStart().getAddress_State());
        Address : geteX_to_eBPM().getBP_EVENT().setATTRIBUTE("Address_Zip","STRING",getProcessOrderStart().getAddress_Zip());
        Email : geteX_to_eBPM().getBP_EVENT().setATTRIBUTE("Cust_email","STRING",getProcessOrderStart().getEmail());
        ItemNumber : geteX_to_eBPM().getBP_EVENT().setATTRIBUTE("Item_Number","STRING",getProcessOrderStart().getItem_Number());
        ItemDescription : geteX_to_eBPM().getBP_EVENT().setATTRIBUTE("Item_Description","STRING",getProcessOrderStart().getItem_Description());
        Quantity : geteX_to_eBPM().getBP_EVENT().setATTRIBUTE("Order_Quantity","NUMBER",getProcessOrderStart().getQuantity());
        return : return retBoolean;
    userInitialize : public void userInitialize()
    userTerminate : public void userTerminate()
```

Step 4: Configure the Collaboration in the GUI using Java

In addition to creating the configuration file for the e*Way and the CRS used by the Collaboration, you must also configure the **START_BP** e*Way's Collaboration in the Enterprise Manager GUI.

1 Create a Collaboration for the **START_BP** e*Way configured as shown in Figure 91.

Figure 91 START_BP Collaboration



Collaboration Rules: Start_BP

Subscriptions:

Instance Name	Event Type	Source
ProcessOrderStart	eX_External_Evt	<EXTERNAL>

Publications:

Instance Name	Event Type	Destination	Priority
eX_to_eBPM	eX_to_eBPM	eX_eBPM	5

10.5.2 Configure the Send_Status e*Way

The last component that must be configured in the ProcessOrder example is the **Send_Status** e*Way.

This e*Way must accomplish two tasks:

- Create a file containing the text of e-mail message that can be sent to an order-status message via e-mail (simulated; no actual mail is sent)
- Return "SUCCESS" to the e*Insight engine

This e*Way simulates sending an e-mail order status message by writing the customer's e-mail address and a short status message to a text file. When this is successful, an Event is returned to the e*Insight engine with the status node set to "SUCCESS".

Table 32 shows the steps to configure the Send_Status e*Way.

Table 32 Send_Status e*Way configuration steps

	Step	Section	
		Monk	**Java**
1	Find the executable and create the e*Way configuration file	**"Step 1: Configure the eX_Send_Status e*Way using Monk" on page 156**	**"Step 1: Configure the e*Way using Java" on page 159**
2	Create the Output ETD	**"Step 2: Create the Output ETD using Monk" on page 157**	**"Step 2: Create the Output ETD: SendStatus.xsc using Java" on page 159**
3	Create the eX_Send_Status.tsc CRS	**"Step 3: Create the eX_Send_Status.tsc CRS using Monk" on page 157**	**"Step 3: Create the Send_Status Collaboration Rule using Java" on page 160**
4	Configure the Collaboration in the GUI	**"Step 4: Configure the Collaboration using Monk" on page 158**	**"Step 4: Configure the Collaboration using Java" on page 161**

Step 1: Configure the eX_Send_Status e*Way using Monk

First find the executable, then create the configuration file.

The **eX_Send_Status** e*Way is a simple file e*Way (**stcewfile.exe**) that writes a text file (**ProcessOrder_output%d.dat**) to the directory **<egate>\client\data\ProcessOrder**. The file created contains the e-mail address of the person who placed the order, along with the status of the order. Use the following table to set the e*Way parameters in the configuration file:

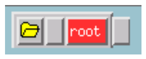
Table 33 Send_Status e*Way Parameters

Screen	Parameter	Setting
General Settings	AllowIncoming	No
	AllowOutgoing	Yes
	PerformanceTesting	No (Default)
Outbound (send) settings	OutputDirectory	<eGate>\client\data\ProcessOrder
	OutputFileName	ProcessOrder_output%d.dat
	(All others)	(Default)
Poller (inbound) settings	(All)	(Default)
Performance Testing	(All)	(Default)

Step 2: Create the Output ETD using Monk

Use the e*Gate ETD Editor to create a single node ETD like that shown in Figure 92.

Figure 92 root.ssc ETD



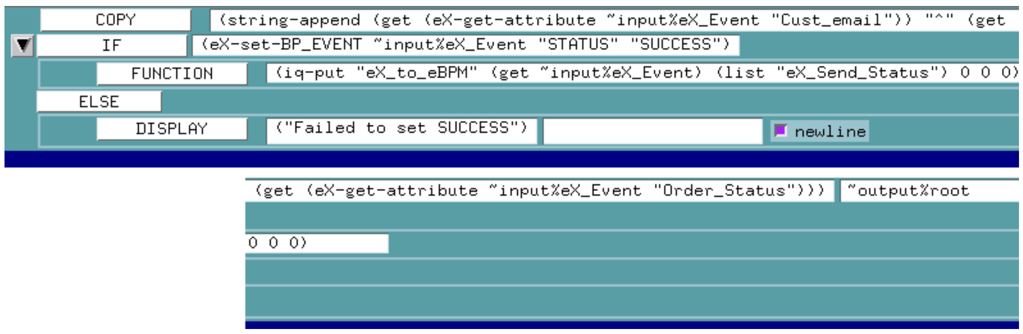
Step 3: Create the eX_Send_Status.tsc CRS using Monk

This CRS must accomplish three things:

- put the output data into a readable format that can be written to a file
- use the e*Insight helper function **eX-set-BP_EVENT** to set the BP status node to "SUCCESS"
- send the "Done" Event back to the e*Insight engine using the Monk function **iq-put**

The CRS shown in Figure 93 accomplishes these tasks. The source ETD is eX_Standard_Event.ssc and the destination ETD is root.ssc.

Figure 93 eX_Send_Status.tsc CRS (Monk)



```
COPY       (string-append (get (eX-get-attribute ~input%eX_Event "Cust_email")) "^" (get
IF         (eX-set-BP_EVENT ~input%eX_Event "STATUS" "SUCCESS")
  FUNCTION   (iq-put "eX_to_eBPM" (get ~input%eX_Event) (list "eX_Send_Status") 0 0 0)
ELSE
  DISPLAY    ("Failed to set SUCCESS")                    newline

(get (eX-get-attribute ~input%eX_Event "Order_Status")))   ~output%root

0 0 0)
```

Step 4: Configure the Collaboration using Monk

The **eX_Send_Status** e*Way in ProcessOrder example does not receive data back from and external system. Consequently, it requires only a single Collaboration. Use the following procedure to edit the two default Collaborations created by the e*Insight GUI during the configuration of the integration schema.

To configure the collaboration

1. Highlight the **eX_Send_Status** e*Way.
2. Delete the two Collaborations **eX_to_Send_Status** and **eX_from_Send_Status**.
3. Add a Collaboration named **eX_Send_Status**.
4. Highlight the **Collaboration Rules** folder.
5. Delete the two Collaboration Rules **eX_to_Send_Status** and **eX_from_Send_Status**.
6. Add a Collaboration Rule named **eX_Send_Status**.
7. Edit the Collaboration Rule.
8. In the **Collaboration Rules Properties** dialog box, select the **Monk** service.
9. Find the CRS **eX_Send_Status.tsc** and associate it with the Collaboration Rule.
10. On the **Subscriptions** tab, move the **eX_Send_Status_Do** and **eX_Send_Status_Undo** Event Types to the **Selected Input Event Types** box.
11. On the **Publications** tab, move the **eX_External_Evt** and the **eX_to_eBPM** Event Types to the **Selected Output Event Types** box.

    Verify that the **eX_External_Evt** Event Type is marked as the default.
12. Click **OK** to close the **Collaboration Rules Properties** dialog box.
13. Highlight the **eX_Send_Status** e*Way and edit the **eX_Send_Status** Collaboration you associated with it in step 3.
14. In the **Collaboration Properties** dialog box, select the **eX_Send_Status** Collaboration Rule.
15. Under **Subscriptions** add the **eX_Send_Status_Do** and **eX_Send_Status_Undo** Event Types from the **eX_from_eBPM** source.
16. Under **Publications** add the Event Type **eX_External_Evt** with destination **<EXTERNAL>** and Event Type **eX_to_eBPM** with destination **eX_eBPM** IQ.

Step 1: Configure the e*Way using Java

First find the executable, then create the configuration file.

The **eX_Send_Status** e*Way is a simple file e*Way (**stcewfile.exe**) that writes a text file (**ProcessOrder_output%d.dat**) to the directory **<egate>\client\data\ProcessOrder**. The file created contains the e-mail address of the person who placed the order, along with the status of the order. Use the following table to set the e*Way parameters in the configuration file:

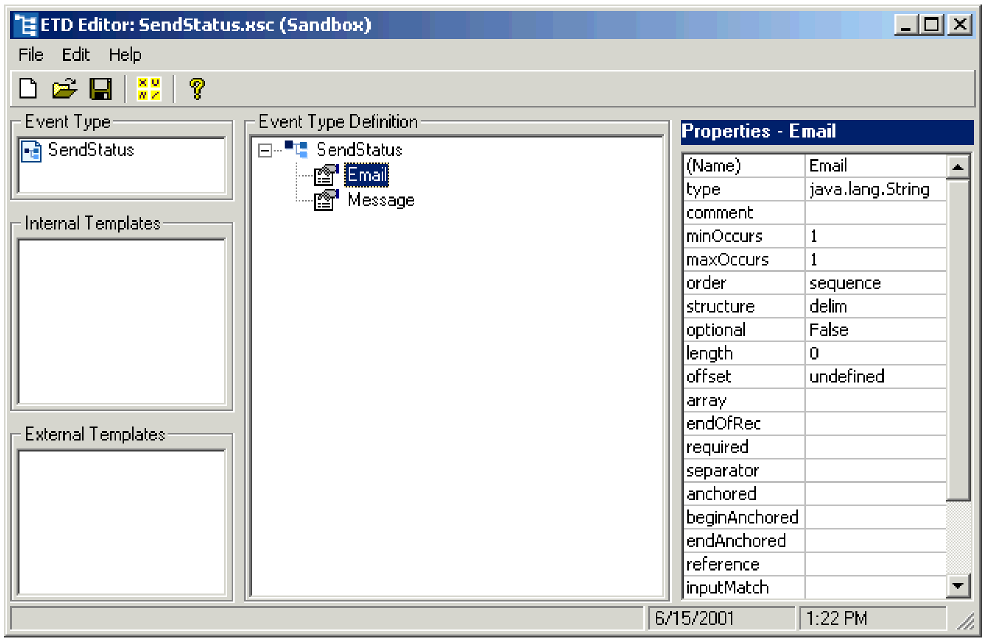
Table 34 Send_Status e*Way Parameters

Screen	Parameter	Setting
General Settings	AllowIncoming	No
	AllowOutgoing	Yes
	PerformanceTesting	No (Default)
Outbound (send) settings	OutputDirectory	<eGate>\client\data\ProcessOrder
	OutputFileName	ProcessOrder_output%d.dat
	(All others)	(Default)
Poller (inbound) settings	(All)	(Default)
Performance Testing	(All)	(Default)

Step 2: Create the Output ETD: SendStatus.xsc using Java

Use the e*Gate ETD Editor to create an ETD like that shown in Figure 94.

Figure 94 SendStatus.xsc ETD (Java)



(Name)	Email
type	java.lang.String
comment	
minOccurs	1
maxOccurs	1
order	sequence
structure	delim
optional	False
length	0
offset	undefined
array	
endOfRec	
required	
separator	
anchored	
beginAnchored	
endAnchored	
reference	
inputMatch	

Step 3: Create the Send_Status Collaboration Rule using Java

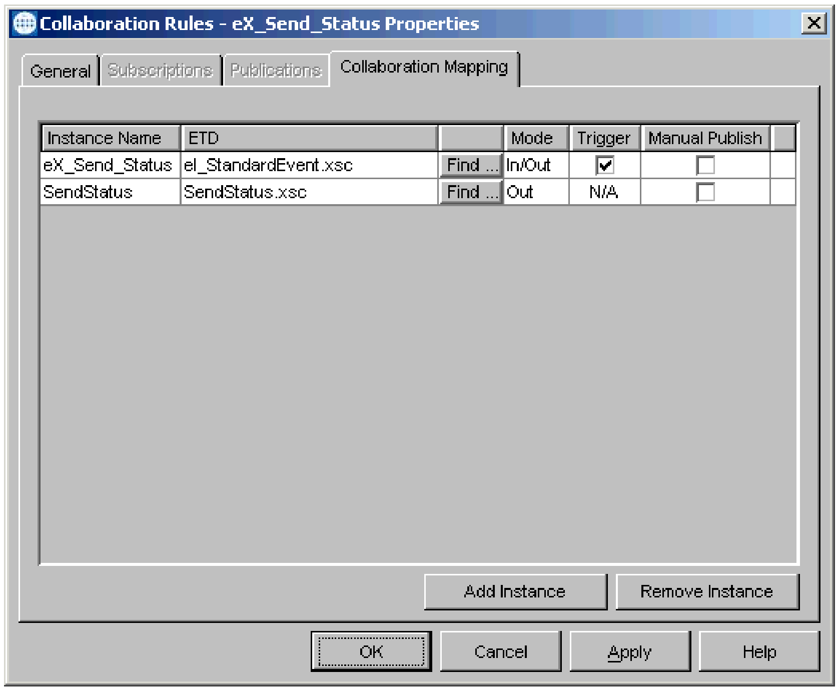
This CRS must accomplish three things:

- put the output data into a readable format that can be written to a file
- set the BP status node to “SUCCESS”
- send the “Done” Event back to the e*Insight engine

To Configure the eX_Send_Status Collaboration Rule

1. From the eX_Send_Status Collaboration Rule General tab, select the Java Collaboration Service.
2. From the eX_Send_Status Collaboration Rule Collaboration Mapping tab, create two new instances as shown in Figure 95.

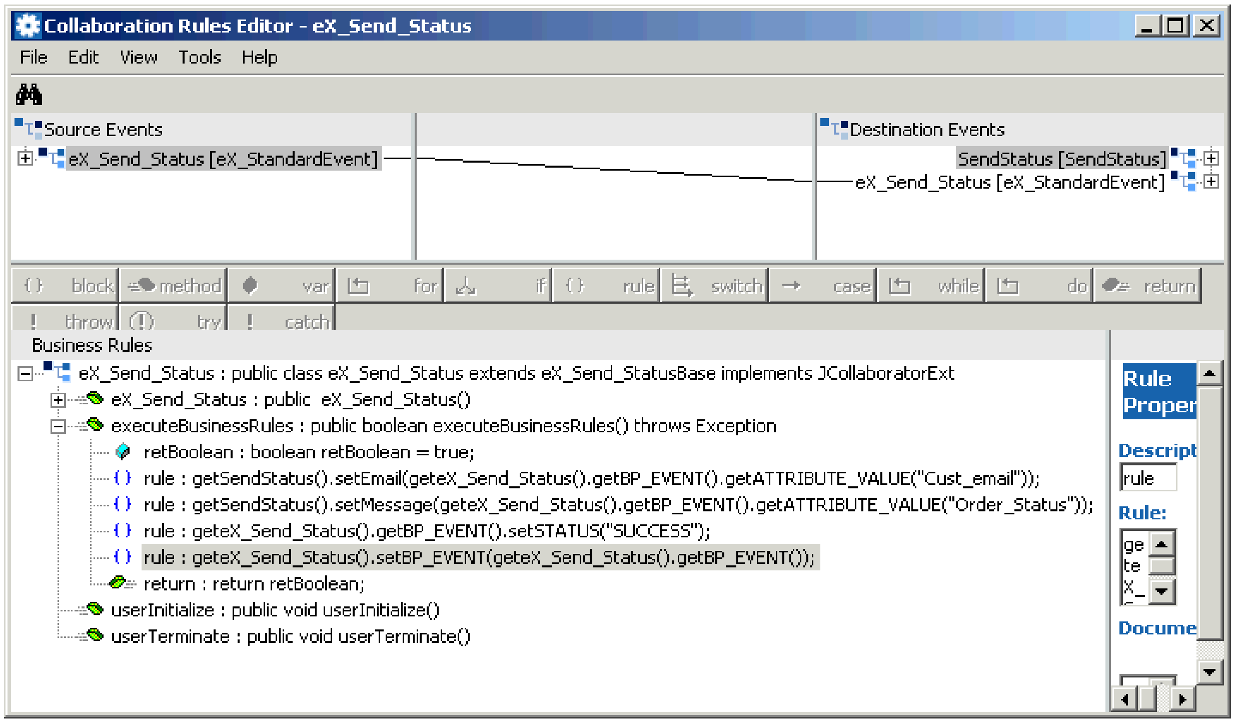
Figure 95 eX_Send_Status CR Properties, Collaboration Mapping tab (Java)



Instance Name	ETD		Mode	Trigger	Manual Publish
eX_Send_Status	eI_StandardEvent.xsc	Find ...	In/Out	☑	☐
SendStatus	SendStatus.xsc	Find ...	Out	N/A	☐

3. Click **Apply**, and click the **General Tab**.
4. Click **New** to create a new CRS, as show in Figure 96.

Figure 96 eX_Send_Status.xsc CRS (Java)



```
eX_Send_Status : public class eX_Send_Status extends eX_Send_StatusBase implements JCollaboratorExt
    eX_Send_Status : public  eX_Send_Status()
    executeBusinessRules : public boolean executeBusinessRules() throws Exception
        retBoolean : boolean retBoolean = true;
        rule : getSendStatus().setEmail(geteX_Send_Status().getBP_EVENT().getATTRIBUTE_VALUE("Cust_email"));
        rule : getSendStatus().setMessage(geteX_Send_Status().getBP_EVENT().getATTRIBUTE_VALUE("Order_Status"));
        rule : geteX_Send_Status().getBP_EVENT().setSTATUS("SUCCESS");
        rule : geteX_Send_Status().setBP_EVENT(geteX_Send_Status().getBP_EVENT());
        return : return retBoolean;
    userInitialize : public void userInitialize()
    userTerminate : public void userTerminate()
```

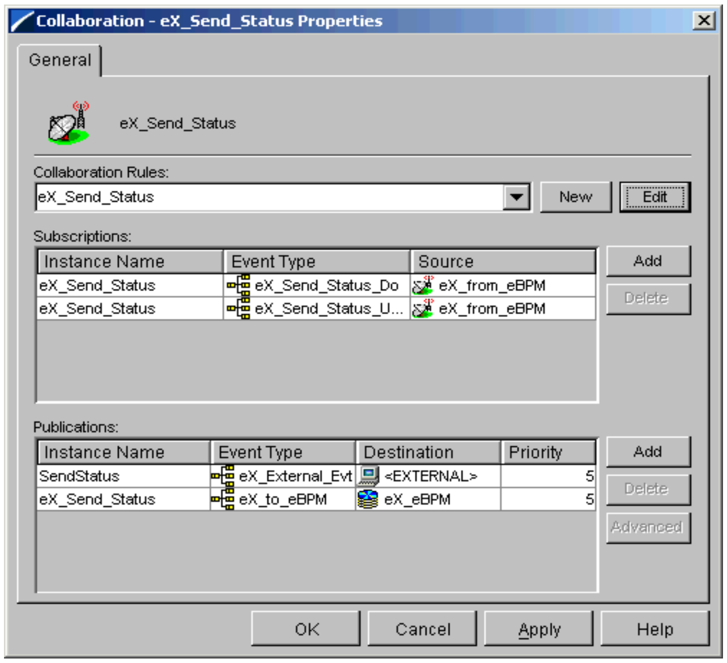
Step 4: Configure the Collaboration using Java

The **eX_Send_Status** e*Way in ProcessOrder example does not receive data back from and external system. Consequently, it requires only a single Collaboration. Use the following procedure to edit the two default Collaborations created by the e*Insight GUI during the configuration of the integration schema.

In the Enterprise Manager:

1. Highlight the **eX_Send_Status** e*Way.
2. Delete the two Collaborations **eX_to_Send_Status** and **eX_from_Send_Status**.
3. Add a Collaboration named **eX_Send_Status**.
4. Configure the Collaboration, as shown in Figure 97.

Figure 97 eX_Send_Status Collaboration (Java)


Collaboration - eX_Send_Status Properties

General

eX_Send_Status

Collaboration Rules: eX_Send_Status

Subscriptions:

Instance Name	Event Type	Source
eX_Send_Status	eX_Send_Status_Do	eX_from_eBPM
eX_Send_Status	eX_Send_Status_U...	eX_from_eBPM

Publications:

Instance Name	Event Type	Destination	Priority
SendStatus	eX_External_Evt	<EXTERNAL>	5
eX_Send_Status	eX_to_eBPM	eX_eBPM	5

10.6 Run and Test the e*Insight scenario

Once the schema has been set up in e*Gate you can run the scenario.

10.6.1 Testing the Standard Business Logic

The following procedure tests the standard business logic of the e*Insight ProcessOrder case study example. That logic is as follows: a check is made to see whether or not the item ordered is available. If it is in stock the Ship_Order activity is invoked and a message is generated that can be sent to the customer indicating that his order has been shipped to him. If the item is unavailable, then the Out_of_Inv activity is invoked which creates a message informing the customer that his item is unavailable.

The test is made by sending in data with different item numbers and verifying the correct processing. Input data with an item number of 33333 is interpreted as being in stock and any other number except for the three special numbers (11111, 22222, and 99999) is interpreted as being out of stock.

In-Stock Processing

Use the following procedure to test the functionality of the example for an item that is in stock.

1 Start the e*Insight GUI and select the ProcessOrder business process. Switch to monitor mode.

Note: *Make sure that the business process has been enabled in the e*Insight GUI before attempting to run it.*

2 Make a final check of the e*Gate schema, using the tables to confirm all of the GUI associations. Make sure that all of the e*Insight components, including the user-defined components, are set to start automatically.

3 At the command line, type the following to start the schema. You must type the command on a single line.

```
stccb.exe -rh localhost -rs ProcessOrder -ln localhost_cb
      -un username -up password
```

Substitute the appropriate username and password for your installation.

4 Start the e*Gate Monitor, and check the status of all the components. Any components displayed in red are not running. Any e*Insight components that are not running should be investigated before feeding data into the system.

5 Navigate to the location for the input data file, **InStock.~in**, shown in **Figure 84 on page 150** (**c:\eGate\client\data\ProcessOrder**) and change the extension to "**.fin**".

Note: *The change of the extension to "**.~in**" indicates that the data file has been picked up by the **START_BP** e*Way.*

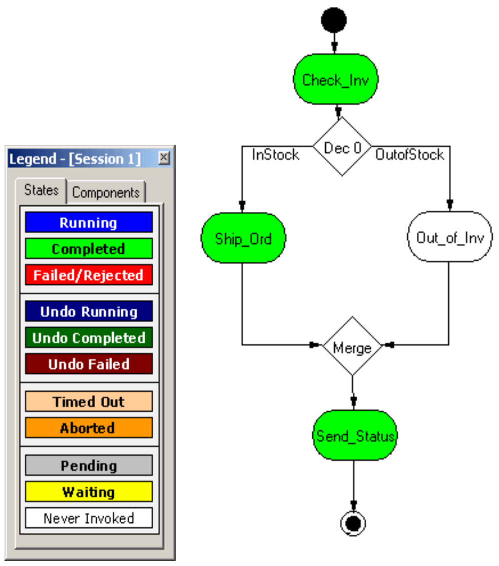
If everything is working correctly, an output file (**ProcessOrder_output#.dat**) as appears in the directory indicating successful completion of the BPI.

6 Switch to the e*Insight GUI and, while in monitor mode, select the most recent BPI from the **List** tab, and then select the **Diagram** tab to observe the path that the data has taken.

The activities that have completed successfully appear green. Any activities that are still running appear blue.

In the ProcessOrder example, an activity that stays blue for more than couple minutes indicates a problem, and the e*Gate component associated with that activity should be investigated for the cause of the problem. Figure 98 illustrates how the successfully completed BPI appears in the e*Insight GUI.

Figure 98 In Stock Completed BPI Diagram

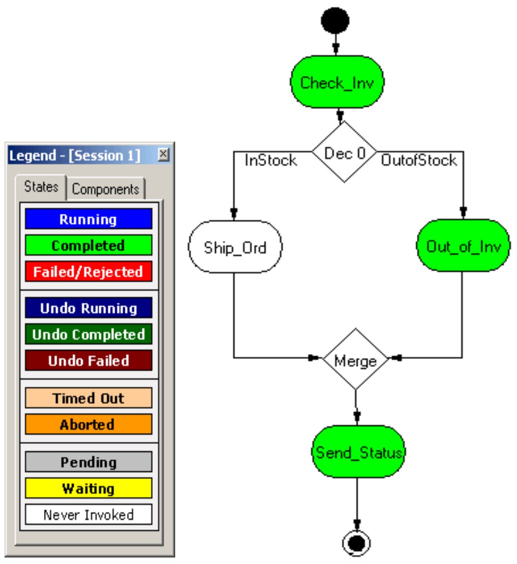


Out-of-Stock Processing

Testing the functionality for out of stock processing uses exactly the same procedure as that for in stock processing except that different input data is submitted.

- Verify that sending in the data with an item number of 44444 causes the business process to take the "FALSE" branch of the decision gate and create the diagram shown in Figure 99.

Figure 99 Out of Stock Completed BPI Diagram



10.6.2 Demonstrating Business Process Undo Functionality

e*Insight has two methods for undoing a failed business process instance (BPI): automatic and manual. Whether the failure of a particular activity generates an automatic undo of the entire BPI or whether the e*Insight engine waits for user intervention, is set on the **General** tab of the **Activity Properties** dialog box for that activity. The default setting is automatic undo.

When an activity is set to automatic undo and the activity "fails," then e*Insight marks the activity as "Failed" in the GUI and publishes an "undo" Event (**eX_*Activity*_Undo**) for the last completed activity in the BPI. In this context, fails means that the e*Insight engine receives a "Done" Event where the status node is set to "FAILURE" rather than "SUCCESS". If the last completed activity is undone successfully, then an "undo" Event is generated for the next activity upstream, and so on, until all the previously completed activities in that BPI have been undone.

If an activity fails and the **Manual Restart** check box is marked on the **General** tab of the **Activity Properties** dialog box for that activity, then e*Insight marks the activity as "Failed" in the GUI and then waits for the user to initiate the next course of action; skip, restart, or undo. If the user selects undo, then the BPI is undone as described in the paragraph above.

Manual Undo

Use the following procedure to test the functionality of manual undo in the e*Insight scenario.

1 Perform steps 1 through 4 outlined in **"In-Stock Processing" on page 162**.

2 Verify that **Manual Restart** has been marked for the activities in the business process.

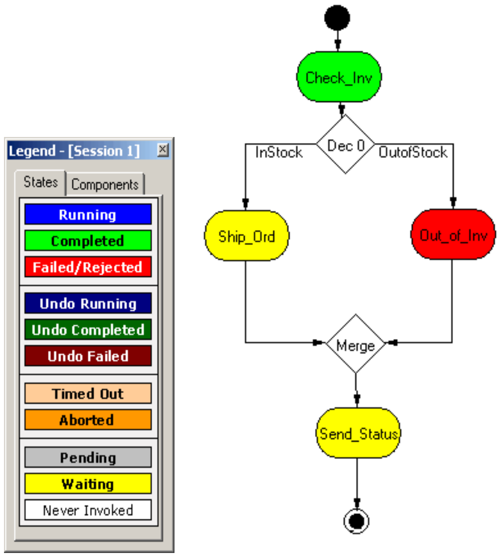
   If **Manual Restart** has not been marked and the check box itself is grayed out, you must delete the BPIs for the business process, or save the business process as a new version, before you can mark it. Refer to the *e*Insight Business Process Manager User's Guide* for information on how to do this.

3 Navigate to the location (**c:\eGate\client\data\ProcessOrder\ManualUndo.~in**) and create an input data file with an item number of 11111 and change the extension to "**.fin**".

Note: *The change of the extension to "**.~in**" indicates that the data file has been picked up by the START_BP e*Way.*

4 Switch to the e*Insight GUI and, while in monitor mode, select the most recent business process instance. Observe the path that the data has taken, as shown in Figure 100 on the next page.

Figure 100 Manual Undo—Failed BPI Diagram



The **Check_Inv** activity should be green, indicating that it completed successfully, but the **Out_of_Inv** activity should appear red, indicating that it has failed.

5 Right-click the **Out_of_Inv** activity from the tree view, then select **Properties** from the popup menu.

The **Activity Properties - Monitor Mode: (Out_of_Inv)** is displayed.

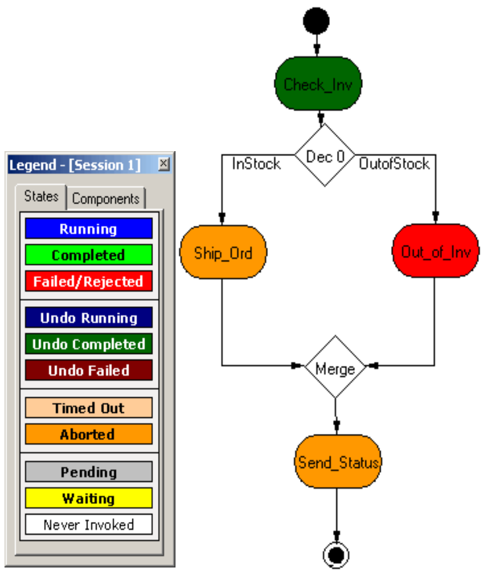
6 Select the **Business Process Attributes** tab.

7 Click **Undo Business Process**, and then click **OK** to close the **Activity Properties** dialog box.

8 Highlight the enabled business process version in the tree view.

The **Check_Inv** activity should now appear dark green indicating that the activity has been successfully undone.

Figure 101 Manual Undo Completed BPI Diagram



10.6.3 Demonstrating Business Process Restart Functionality

An important feature of e*Insight is its ability to allow the operator to fix and restart a business process instance. If the data in one of the business process attributes used by an activity causes the business process to fail, the value can be corrected by the operator and the BPI restarted from the point of failure.

Repairing a String Attribute

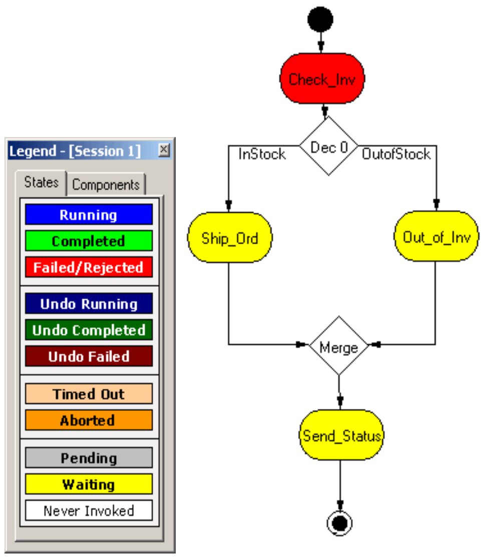
Attributes can be of various types; Boolean, number, string, and XML. The following example shows the procedure to repair an attribute of type string. For information on repairing an attribute with type XML, see the *e*Insight Business Process Manager User's Guide*.

1. Perform steps 1 through 4 outlined in **"In-Stock Processing" on page 162**.
2. Verify that **Manual Restart** has been marked for the activities in the business process.
3. Navigate to the location (**c:\eGate\client\data\ProcessOrder\AttributeRepair.~in**) and create an input data file with an item number of 99999, and change the extension to "**.fin**".

Note: *The change of the extension to "**.~in**" indicates that the data file has been picked up by the START_BP e*Way.*

4. Switch to the e*Insight GUI and, while in monitor mode, select the most recent business process instance from the **List** tab. Select the **Diagram** tab to observe the path that the data has taken.

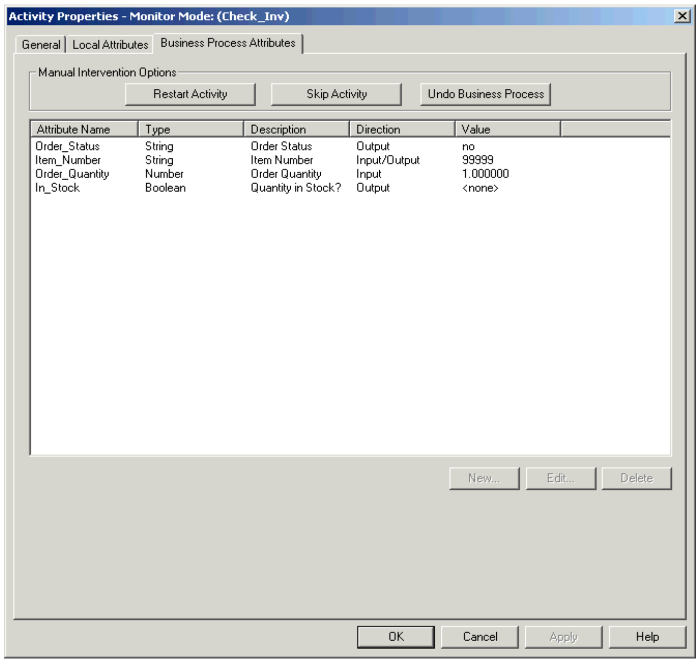
Figure 102 Attribute Repair—Failed BPI Diagram



The **Check_Inv** activity should be red, indicating that it failed, and the other activities should appear yellow, indicating that they are waiting.

5 Double-click the **Check_Inv** activity, then click the **Business Process Attributes** tab.

Figure 103 Attribute Repair—Business Process Attributes tab

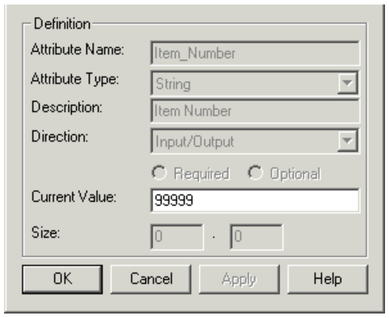


Attribute Name	Type	Description	Direction	Value
Order_Status	String	Order Status	Output	no
Item_Number	String	Item Number	Input/Output	99999
Order_Quantity	Number	Order Quantity	Input	1.000000
In_Stock	Boolean	Quantity in Stock?	Output	<none>

6 Highlight the **Item_Number** attribute line, and then click **Edit**.

The **Edit Business Process Attribute** dialog box is displayed, as shown in Figure 104.

Figure 104 Attribute Repair—Edit BP Attribute



Definition

Attribute Name:	Item_Number
Attribute Type:	String
Description:	Item Number
Direction:	Input/Output
	◯ Required ◯ Optional
Current Value:	99999
Size:	0 . 0

7 In the **Edit Business Process Attribute** dialog box, change the value of the attribute in the **Current Value:** box from 99999 to 44444, and then click **OK**.

The value 99999 was supplied to the e*Insight engine by the input file and it is this value that causes the **Check_Inv** Collaboration to return "FAILURE" to the e*Insight engine.

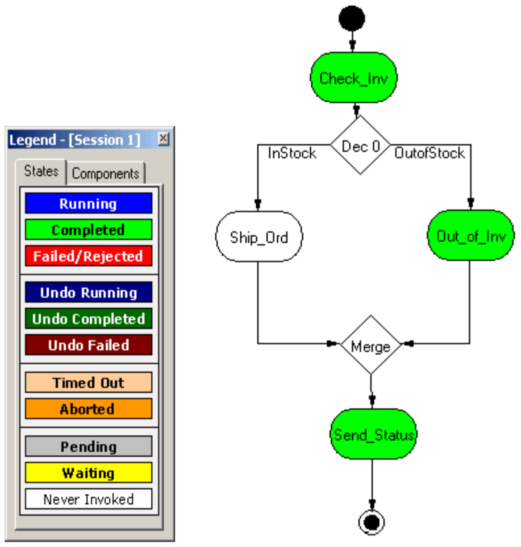
8 Click **OK** to close the **Edit Business Process Attribute** dialog box.

9 Click **Restart Activity**, and then click **OK**.

10 Highlight the enabled ProcessOrder business process version in the tree view.

The completed BPI diagram is displayed, as shown in Figure 105 on the next page.

Figure 105 Attribute Repair Completed BPI Diagram



The **Check_Inv**, **Out_of_Inv**, and **Send_Status** activities now appear green indicating that the BPI has been restarted and has now completed successfully.

11 Verify that a text file (**ProcessOrder_output#.dat**) to be sent as e-mail is created indicating that item number 44444 is unavailable.

Chapter 11

e*Insight Authorization Activity Implementation (eISchema)

This chapter discusses the steps involved to enhance the previous case study to include the Authorization Activity.

You can use the Authorization Activity to stop the Business Process and wait for authorization. The decision to authorize or not authorize is entered via the e*Insight GUI.

This case study is a continuation of the previous example. See **"e*Insight Implementation (eISchema)" on page 133** for the initial configuration instructions.

11.1 Overview

The major steps in the implementation are:

1. Create and configure the Decision gate and Authorization Activity in the e*Insight GUI.
2. Use the e*Insight GUI to configure the e*Gate schema that supports e*Insight.
3. Add and configure the user-defined e*Gate components.
4. Run and test the scenario.

The chapter begins with a description of the scenario and then shows how to use these steps to set it up.

11.1.1 Case Study: Order Processing

The Order Process only automatically processes orders where less than 100 items are ordered. If 100 or more items are ordered then the order should be manually authorized.

Figure 106, shows the additional components involved in the business process implementation. Below the diagram is a description of how the data flows between these components for an item that is shipped successfully.

For the original diagram and description, see **"Case Study: Order Processing" on page 134**.

Figure 106 e*Insight Data Flow Diagram

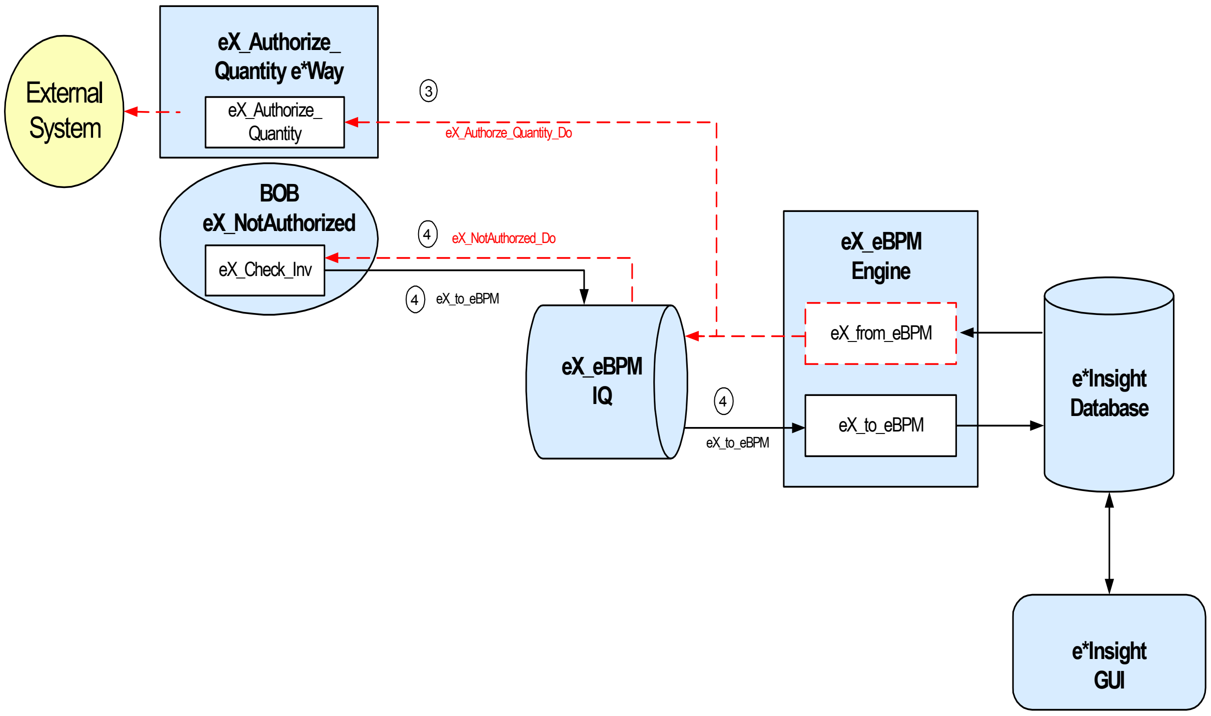


Figure 106 data flow description

③ When the **Check_Inv** activity is finished, the **eX_Check_Inv** BOB publishes a "Done" Event using the **eX_to_eBPM** Event Type to the **eX_eBPM** IQ. The e*Insight engine retrieves the "Done" Event from the IQ, updates the BPI to reflect whether the items ordered are in stock, and then moves forward to the next activity in the business process based on the result of a decision gate. If the items are in stock, the next activity is another decision gate; otherwise the next activity is **Out_of_Inv**.

Let's assume the items are in stock. The BPI moves to the next activity based on the result of the second decision gate. If the quantity is less than 100, the next activity is **Ship_Ord**; otherwise the next activity is **eX_Authorize_Quantity**.

Let's assume the quantity is greater than or equal to 100. The e*Insight engine publishes a "Do" Event (**eX_Authorize_Quantity_Do**) corresponding to the **Authorize_Quantity** authorization activity in the business process. The **eX_Authorize_Quantity** e*Way retrieves this Event from the **eX_eBPM** IQ and uses the information it contains to retrieve the BPI ID and alert the relevant person that this instance requires authorization.

When the **eX_Authorize_Quantity** activity has either been authorized or rejected, the e*Insight engine moves forward to the next activity in the business process. This is **Not_Authorized** if the quantity was not authorized; otherwise the next activity is **Ship_Order**.

④ Let's assume the quantity is not authorized. The e*Insight engine publishes a "Do" Event (**eX_Not_Authorized_Do**) corresponding to the **Not_Authorized** activity in the business process. The **eX_Not_Authorized** BOB retrieves this Event from the **eX_eBPM** IQ and updates the status. When the **Not_Authorized** activity is finished, the **eX_Not_Authorized** BOB publishes a "Done" Event using the **eX_to_eBPM** Event Type to the **eX_eBPM** IQ. The e*Insight engine retrieves the "Done" Event from the IQ, updates the BPI to reflect the status, and then moves forward to the **Send_Status** activity.

Let's assume that either the quantity is authorized, or the item was in stock, but the quantity was less than 100. The e*Insight engine processes the e*Insight script corresponding to the **Ship_Ord** activity in the business process, and then moves forward to the **Send_Status** activity.

Important Considerations

The Authorization Activity has two fixed Local Attributes—**assignedTo** (the user to whom the Authorization process is assigned) and **performedBy** (a security measure to ensure the correct user is performing the Authorization). It is important to note the following:

- The user name of the assignedTo attribute must exactly match the name of the user logged into the e*Insight GUI or the name of the user group to which the name of the logged in user belongs
- The assignedTo attribute must have a value to complete the Authorization process.
- Any user assigned the role of Instance Manager can authorize, reject, or undo an Authorization Activity within a business process instance.

11.2 Step 1: Create the ProcessOrder BP in e*Insight

The following is a summary of the procedure for creating a BP in the e*Insight GUI.

1. Add the Authorization Activity.
2. Edit the assignedTo Local Attribute to contain the correct user name.

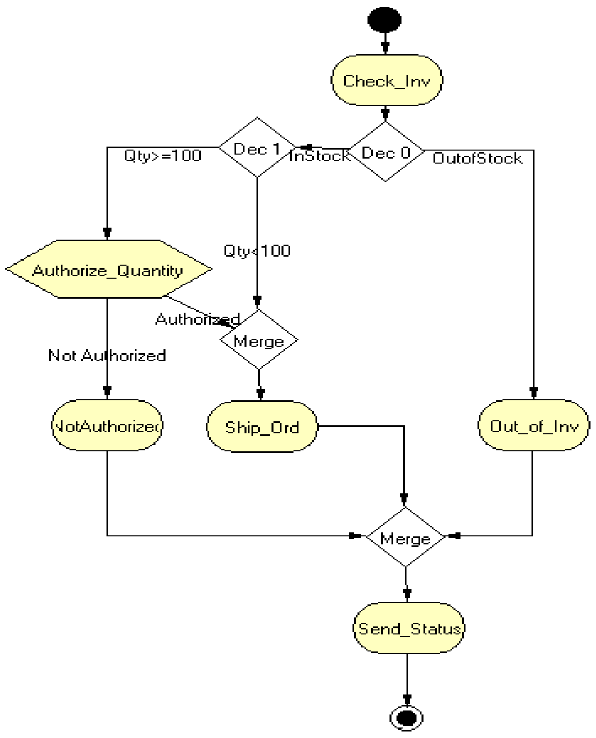
Note: *The Authorization Activity has two fixed Local Attributes—assignedTo (the user to whom the Authorization process is assigned) and performedBy (automatically assigned at run time as the user logged in to the e*Insight GUI). These two values must match in order for the user to Authorize, Reject, or Undo the business process instance.*

3. Add the additional Decision gate.
4. Make the connections between the activities and gates.
5. Add the logic to the Decision gate.

For more information on creating a business process and using the e*Insight GUI, see the *e*Insight Business Process Manager User's Guide*.

Use the diagram shown in Figure 107 and the following tables to create the BP in e*Insight.

Figure 107 ProcessOrder Business Process Model



11.3 Step 2: Configure the Integration Schema

After creating the additional components, you must configure the e*Gate Registry schema that supports the e*Insight system.

e*Insight allows you to specify the type of component (e*Way or BOB) associated with a particular activity and where it runs.

In this example, you use an additional BOB for the **NotAuthorized** Activity, and an additional e*Way for the **Authorize_Quantity** Authorization Activity. The **Send_Status** activity must be associated with an e*Way because it interfaces with an external component.

Integration Schema Activity Components Summary

Use the information in Table 35 to configure the e*Gate schema that supports this example.

Table 35 Integration Schema Activity Components

Name	Type	Participating Host	Active/ Passive	Manual Restart	TimeOut
eX_Authorize_Quantity	e*Way	localhost	Active	Yes	Not used
eX_NotAuthorized	BOB	localhost	Active	Yes	Not used

For information on how to use the e*Insight GUI to configure the e*Gate Registry see the *e*Insight Business Process Manger User's Guide*.

11.4 Step 3: Configure User-defined e*Gate Components

This example requires the configuration of an additional BOB and e*Way.

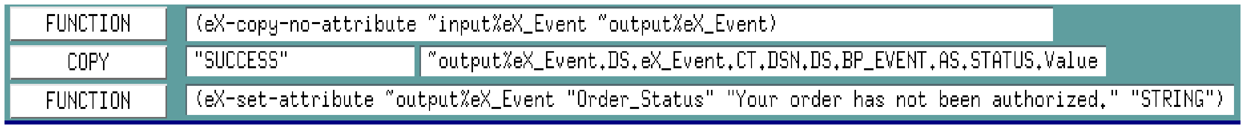
Configure the Activity BOB CRS in the Enterprise Manager GUI

Not_Authorized CRS

The Not_Authorized translation simulates the activity of sending out an item that is in stock. It sets the value of the Order_Status attribute to a short message indicating that the order has not been authorized, and returns "SUCCESS" to the e*Insight engine.

Figure 108, on the following page, shows the **Not_Authorized.tsc** CRS used in this example. The source and destination ETDs are eX_Standard_Event.ssc.

Figure 108 Not_Authorized.tsc CRS



```
FUNCTION   (eX-copy-no-attribute ~input%eX_Event ~output%eX_Event)
COPY       "SUCCESS"   ~output%eX_Event.DS.eX_Event.CT.DSN.DS.BP_EVENT.AS.STATUS.Value
FUNCTION   (eX-set-attribute ~output%eX_Event "Order_Status" "Your order has not been authorized." "STRING")
```

Configure the Activity BOB Collaborations in the Enterprise Manager GUI

Once you have created the CRS for the **eX_NotAuthorized** BOB, you must associate it with the corresponding Collaboration Rule in the e*Gate GUI. You must:

1 Highlight the BOB's Collaboration.

2 Open the **Collaboration Properties** dialog box for the Collaboration.

3 Edit the Collaboration Rules.

4 Change the Service to **Monk**.

5 Find the "Not_Authorized.tsc" file and associate it with the Collaboration Rule.

6 Click **OK** to continue.

11.4.1 Configure the Authorize_Quantity e*Way

The last component that must be configured in the ProcessOrder example is the **Authorize_Quantity** e*Way.

This e*Way must create a file containing the text of e-mail message that can be sent to advise that an order requires authorization that can be sent via e-mail (simulated; no actual mail is sent)

This e*Way simulates sending an e-mail order status message simply by writing a short status message to a text file giving the Business Process Instance ID. An Event is not returned to the e*Insight engine as it is expecting authorization via the e*Insight GUI in this example.

Follow these steps to configure the Send_Status e*Way:

1 Create the **Authorize_Quantity.tsc** CRS.

2 Find the executable and create the e*Way configuration file.

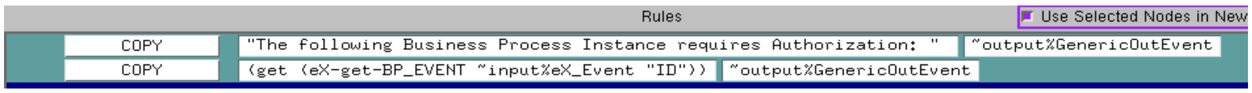
3 Configure the Collaboration in the GUI.

Step 2: Create the Authorize_Quantity.tsc CRS

The Authorize_Quantity translation copies a fixed text message and the Business Process Instance ID into the outgoing message. No message is sent back to the e*Insight engine.

The Authorize_Quantity.tsc CRS is shown in Figure 109. The source ETD is eX_Standard_Event.ssc and the destination ETD is GenericOutEvent.ssc

Figure 109 Authorize_Quantity.tsc CRS



```
COPY  "The following Business Process Instance requires Authorization: "  ~output%GenericOutEvent
COPY  (get (eX-get-BP_EVENT ~input%eX_Event "ID"))  ~output%GenericOutEvent
```

Step 3: Configure the e*Way

First find the executable, then create the configuration file.

The **Authorize_Quantity** e*Way is a simple file e*Way (**stcewfile.exe**) that writes a text file (**output%d.dat**) to the directory **c:\eGate\client\data\Authorize**. The file created contains the Business Process Instance ID of the order that requires authorization. Use the following table to set the e*Way parameters in the configuration file:

Table 36 Send_Status e*Way Parameters

Screen	Parameter	Setting
General Settings	AllowIncoming	No
	AllowOutgoing	Yes
	PerformanceTesting	No (Default)
Outbound (send) settings	OutputDirectory	c:\eGate\client\data\Authorize
	OutputFileName	output%d.dat
	(All others)	(Default)
Poller (inbound) settings	(All)	(Default)
Performance Testing	(All)	(Default)

Step 4: Configure the Collaboration

The **eX_Authorize_Quantity** e*Way in this example does not receive data back from an external system. Consequently, it requires only a single Collaboration. Use the following procedure to edit the two default Collaborations created by the e*Insight GUI during the configuration of the integration schema.

In the Enterprise Manager:

1 Highlight the **eX_Authorize_Quantity** e*Way.

2 Delete the two Collaborations **eX_to_Authorize_Quantity** and **eX_from_Authorize_Quantity**.

3 Add a Collaboration named **eX_Authorize_Quantity**.

4 Highlight the **Collaboration Rules** folder.

5 Delete the two Collaboration Rules **eX_to_Authorize_Quantity** and **eX_from_Authorize_Quantity**.

6 Add a Collaboration Rule named **eX_Authorize_Quantity**.

7 Edit the Collaboration Rule.

8 In the **Collaboration Rules Properties** dialog box, select the **Monk** service.

9. Find the CRS **Authorize_Quantity.tsc** and associate it with the Collaboration Rule.
10. On the **Subscriptions** tab, move the **eX_Authorize_Quantity_Do** and **eX_Authorize_Quantity_Undo** Event Types to the **Selected Input Event Types** box.
11. On the **Publications** tab, move the **eX_External_Evt** to the **Selected Output Event Types** box.
12. Click **OK** to close the **Collaboration Rules Properties** dialog box.
13. Highlight the **eX_Authorize_Quantity** e*Way and edit the **eX_Authorize_Quantity** Collaboration you associated with it in step 3.
14. In the **Collaboration Properties** dialog box, select the **eX_Authorize_Quantity** Collaboration Rule.
15. Under **Subscriptions** add the **eX_Authorize_Quantity_Do** and **eX_Authorize_Quantity_Undo** Event Types from the **eX_from_eBPM** source.
16. Under **Publications** add the Event Type **eX_External_Evt** with destination **<EXTERNAL>**.

11.5 Step 5: Run and Test the e*Insight scenario

Once the schema has been set up in e*Gate you can run the scenario.

11.5.1 Testing the Standard Business Logic

The following procedure tests the additional logic provided by the Authorization Activity. The test is made by sending data that requires authorization and selecting both responses of authorized and not authorized.

Authorized Processing

Use the following procedure to test the functionality of the example.

1. Start the e*Insight GUI and select the ProcessOrder business process. Switch to monitor mode.

Note: *Make sure that the business process has been enabled in the e*Insight GUI before attempting to run it.*

2. Make a final check of the e*Gate schema, using the tables to confirm all of the GUI associations. Make sure that all of the e*Insight components, including the user-defined components, are set to start automatically.
3. At the command line, type the following to start the schema. You must type the command on a single line.

```
stccb.exe -rh localhost -rs ProcessOrder -ln localhost_cb
     -un username -up password
```

Substitute the appropriate username and password for your installation.

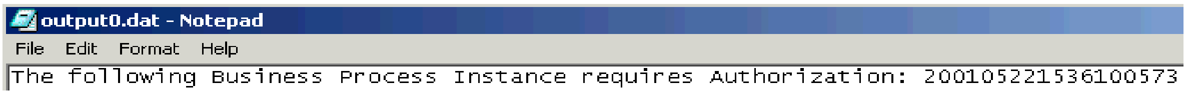
4 Start the e*Gate Monitor, and check the status of all the components. Any components displayed in red are not running. Any e*Insight components that are not running should be investigated before feeding data into the system.

5 Navigate to the location for the input data file, **InStock.~in**, (**c:\eGate\client\data\ProcessOrder**) and change the extension to "**.fin**".

Note: *The change of the extension to "**.~in**" indicates that the data file has been picked up by the **START_BP** e*Way.*

If everything is working correctly, an output file (**output#.dat**) as shown in Figure 110 appears in the directory indicating that an order requires authorization.

Figure 110 In Stock Output File

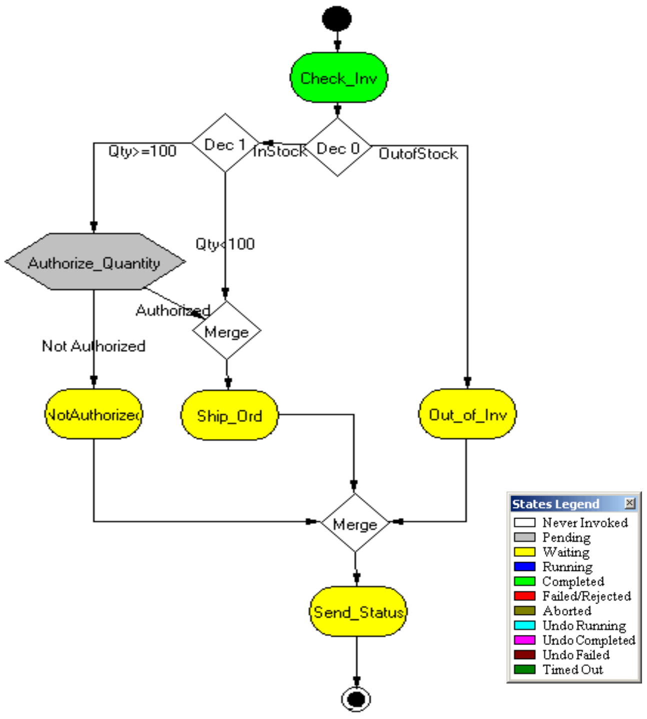


```
The following Business Process Instance requires Authorization: 20010522153610057З
```

6 Switch to the e*Insight GUI and, while in monitor mode, select the most recent BPI from the **List** tab, and then select the **Diagram** tab to observe the path that the data has taken.

The **Authorize_Quantity** Authorization Activity should appear gray. This shows that the activity is pending.

Figure 111 Authorize_Quantity Pending BPI Diagram



7 Right-click the **Authorize_Quantity** activity from the tree view, then select **Properties** from the popup menu.

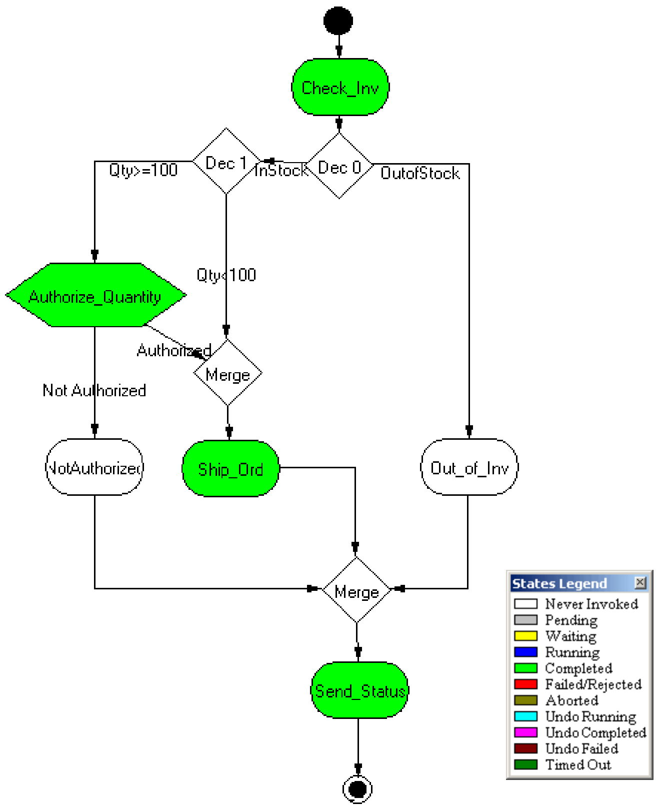
The **Authorization Activity Properties - Monitor Mode: (Authorize_Quantity)** is displayed.

8 Select the **Business Process Attributes** tab.

9 Click **Authorize**, and then click **OK** to close the **Activity Properties** dialog box.

10 The Business Process then completes using the route shown in Figure 112.

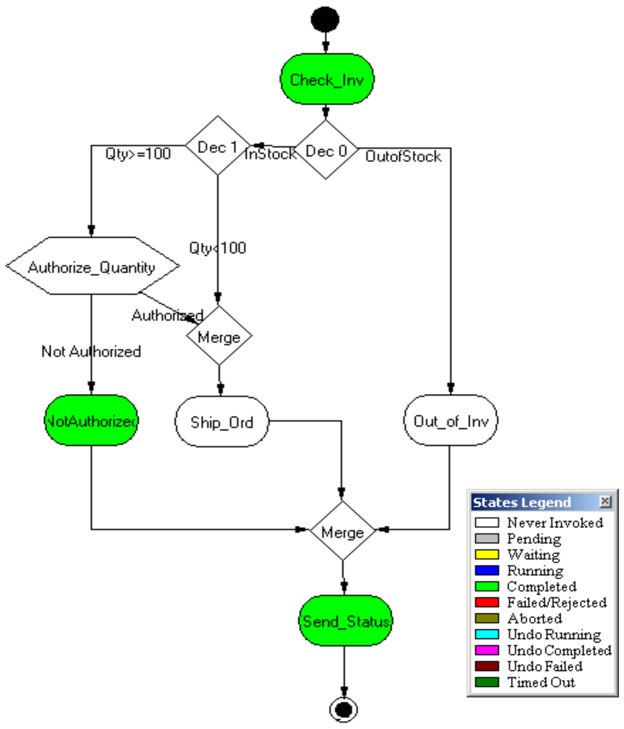
Figure 112 Authorization_Quantity - Authorized



Not Authorized Processing

Repeat the above procedure, but this time do not authorize the order.

Figure 113 Authorization_Quantity - Not Authorized



Chapter 12

e*Insight User Activity Implementation

This chapter discusses the implementation of the User Activity considering aspects of deployment and security.

This chapter also considers how the previous case studies could be enhanced to include the User Activity.

12.1 Overview of the User Activity

User Activities allow external applications to access attributes in the business process using an API. The API provides a set of functions that allow the external application to access attributes for the User Activity from the e*Insight database. The e*Insight engine uses the returned value of the attributes to continue the business process.

For more information on the API, see **"e*Insight User Activity API Methods" on page 393**.

The role of the APIs is to allow the external application to communicate with the e*Insight engine for the purpose of setting security and verifying users, retrieving business model related information, getting and returning attribute types and values, tracking instances, and many other functions related to the business process instances. However, design and development of the external application is left up to the developer.

12.1.1 User Activity Security

Three security checks are performed when connecting to the database using the User Activity methods. First, use the **initialize** method to connect to the database. You should use a user that has no authority to access any of the Business Processes.

Once that connection has been made, use the **authenticate** method to pass the user name and password for a user that has privileges for the Business Process. This user should have the necessary authority for the Business Processes that they are accessing. For subsequent messages sent during the session, use the **setUser** method to re-establish the user security, or **resetUser** to establish security for a new user.

To create a user for the initial connection

1. Use e*Xchange Administrator to create a user (for example, Connection_User), and assign a password.

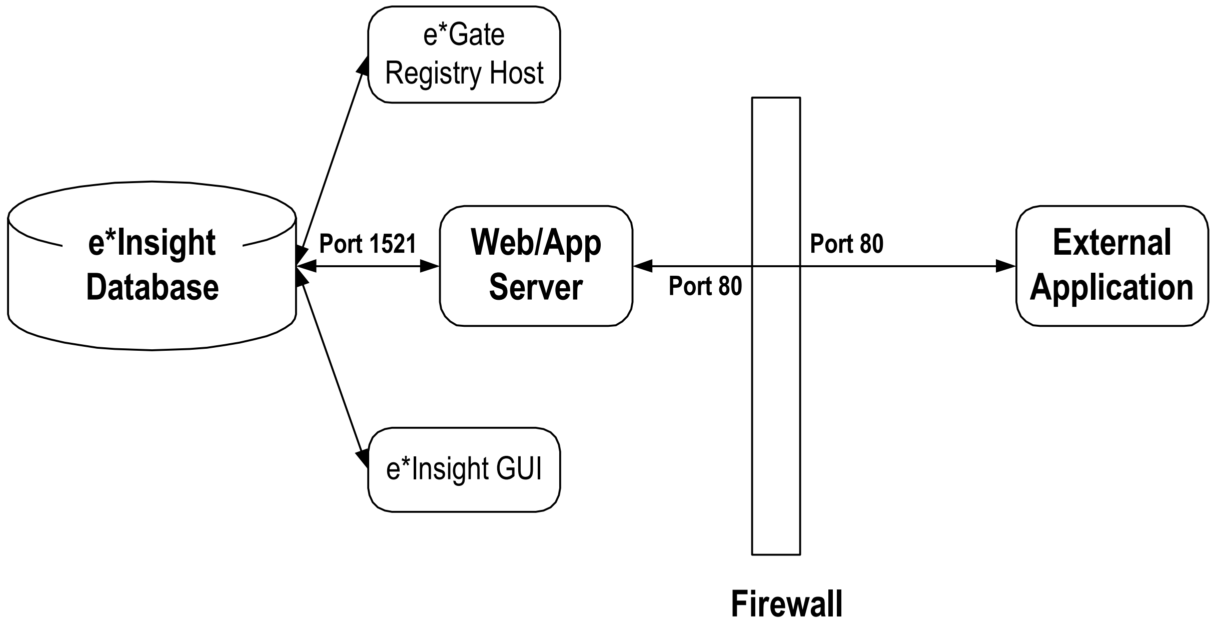
2 Do not give this user any authorization rights within e*Insight.

Note: *For additional security, create the connection user directly in the database rather than using e*Xchange Administrator.*

12.1.2 Deployment of the User Activity

The application used with the User Activity may connect directly to the database, or it may be a Web-based application that connects to the database via a Web/App Server. The security described in **"User Activity Security" on page 181** is still valid when connecting via a Web/App Server, however there are additional considerations. Figure 114 shows how e*Insight, the Web/App Server, and the external application may be deployed.

Figure 114 User Activity Deployment



The firewall is configured to use port 80 to communicate with the Web/App Server. The Web/App Server is configured to communicate with the database using a direct database connection.

An example User Activity scenario using a Web-based application is provided with e*Insight. For information on installing this, see the *e*Insight Business Process Manager Installation Guide*.

12.2 Overview of the Payroll BP

In this example, we assume we have a custom Web application developed for the purpose of allowing various users to set the bonus values. The current example uses an Automated Activity to do a similar function from within the e*Insight GUI. By replacing the Automated Activity with a User Activity, the business process can now integrate with an external application such as a Web-based application. This application would allows users to login, view lists of the business process instances, make appropriate changes, and have those changes re-enter the e*Insight business model for continued processing.

While this example specifically uses a custom Web application, design and development of the external application is left up to the developer. The external application uses the Java API's provided with the e*Insight engine (a complete description and list of these APIs is available in **"e*Insight User Activity API Methods" on page 393**). The role of the APIs is to allow the external application to communicate with the e*Insight engine for the purpose of setting security and verifying users, retrieving business model related information, getting and returning attribute types and values, tracking instances, and many other functions related to the business process instances.

12.3 Overview

The major steps in the implementation are:

1. Add the User Activity to the Business Process.
2. Use the e*Insight GUI to configure the e*Gate schema that supports e*Insight.
3. Add and configure the user-defined e*Gate components.
4. Run and test the scenario.

12.3.1 Case Study: Payroll Processing with User Activity

The Payroll example uses an external application to allow various users to determine the bonus for an employee. The external application retrieves new instances, and for those employees that are eligible for a bonus, the manager enters a bonus value. The user of the external application could also perform other changes to the order based on capabilities developed into the external application.

12.4 Step 1: Update the Payroll BP in e*Insight

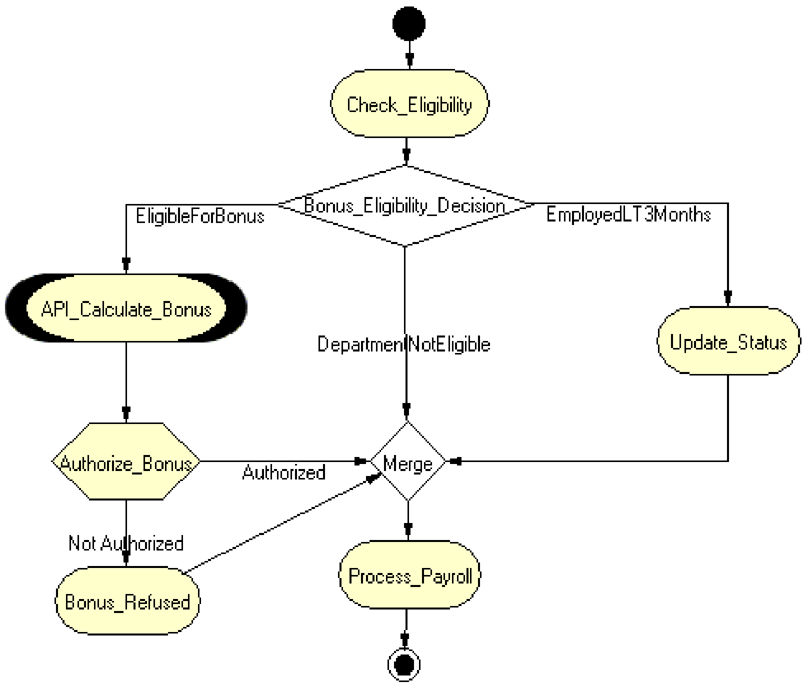
The following is a summary of the procedures for adding a User Activity in the e*Insight GUI.

1. Delete the **Calculate_Bonus** Activity.
2. Add a User Activity called **API_Calculate_Bonus**.
3. Make the connections between the Decision gates, Merge gate, and User Activity, as shown in Figure 115.

Note: ***assignedTo*** *and* ***performedBy*** *are default Local Attributes used to ensure the user performing the Authorize/Reject/Undo activity is sanctioned to do so.*

For more information on creating a business process and using the e*Insight GUI, see the *e*Insight Business Process Manager User's Guide*.

Figure 115 Payroll Business Process Model



12.5 Step 2: Configure the Integration Schema

After creating the additional component, you must configure the e*Gate Registry schema that supports the e*Insight system.

You can send notifications through e*Gate to the external application but in this example, the same result is achieved using only the e*Insight engine. To use only the e*Insight engine, double click the User Activity, and make sure the "Send Notifications through e*Gate check box is not marked.

Note: *To send notifications through e*Gate, make sure the check box is marked, and configure your e*Way or BOB as in all other cases.*

12.6 Step 3: Run and Test the e*Insight scenario

Once the schema has been set up in e*Gate you can run the scenario.

12.6.1 Testing the User Activity

Use the following procedure to test the functionality of the User Activity.

1 Start the e*Insight GUI and select the ProcessOrder business process. Switch to monitor mode.

Note: *Make sure that the business process has been enabled in the e*Insight GUI before attempting to run it.*

2 Make a final check of the e*Gate schema, using the tables to confirm all of the GUI associations. Make sure that all of the e*Insight components, including the user-defined components, are set to start automatically.

3 At the command line, type the following to start the schema. You must type the command on a single line.

```
stccb.exe -rh localhost -rs Payroll -ln localhost_cb
      -un username -up password
```

Substitute the appropriate username and password for your installation.

4 Start the e*Gate Monitor, and check the status of all the components. Any components displayed in red are not running. Any e*Insight components that are not running should be investigated before feeding data into the system.

5 Navigate to the location for the input data file, **Employee.~in**, (**c:\eGate\client\data\Payroll**) and change the extension to "**.fin**".

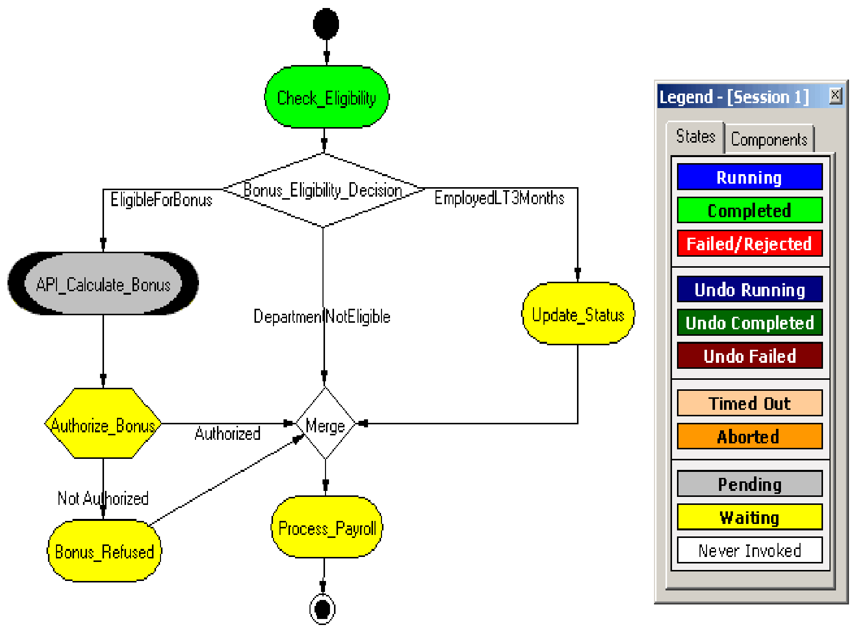
Note: *The change of the extension to "**.~in**" indicates that the data file has been picked up by the **START_BP** e*Way.*

6 Switch to the e*Insight GUI and, while in monitor mode, select the most recent BPI from the **List** tab, and then select the **Diagram** tab to observe the path that the data has taken.

The **API_Authorize_Quantity** should be in a pending state. This remains in a pending state until the e*Insight engine receives a message from an external application relating to this Business Process Instance.

Figure 116 shows how the BPI Diagram appears.

Figure 116 API_Calculate_Bonus Pending BPI Diagram



7 Start the external application. Return to the e*Insight GUI. The Business Process then completes using the route shown in **Figure 117 on page 187**.

The external application refers to any application developed by you to be inserted into the business process. A common example could be a method by which users can view lists of business instances assigned to them. These instances may require a simple review/authorize/reject process or, a more complex task using a custom-developed GUI interface to adjust or introduce attribute values. After the application completes its job, the updated business process instance information re-enters the User Activity and the business process continues.

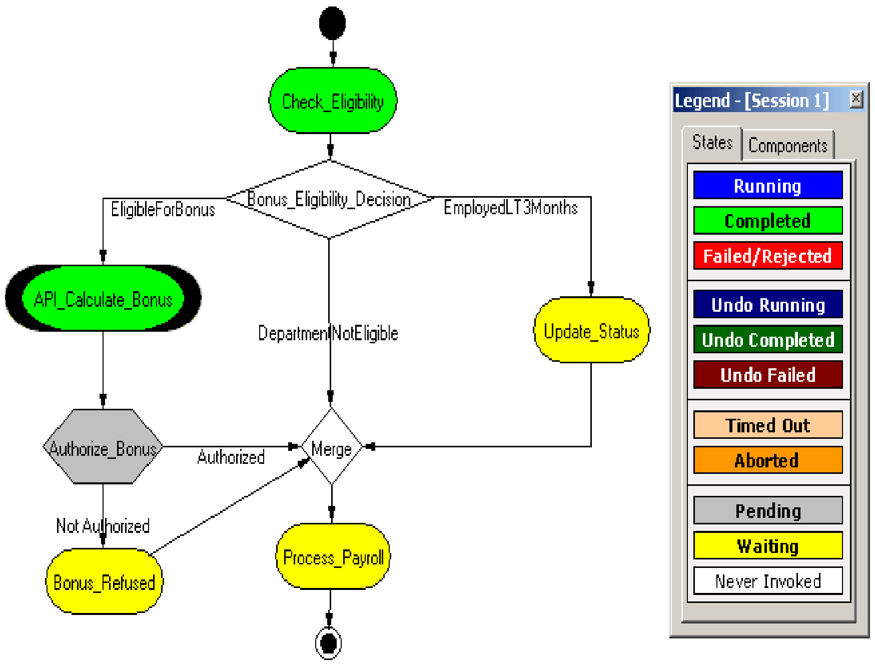
The external application uses an API to communicate with the e*Insight engine. Listed below are some examples of API functions to help illustrate what kind of information is available to the external application (a complete list of API functions can be found in Chapter 16).

- **setUser**—This function passes the name of the user to the e*Insight engine if the external application uses login/password.
- **checkUserPrivileges**—This function establishes the security rights for the session using security levels as set within e*Insight.
- **getGlobalAttributeValue** and **getLocalAttributeValue**—This function receives Attribute values for a business process instance for use within the external application.
- **getBusinessModelInstanceIds**—The external application uses this function to retrieve identification information for business model instances and then uses this information to appropriately track instances when sending information back into the business process.

A example of a business model including a User Activity and an external application are shipped with e*Insight 4.5.2. This example can be installed by you and set up and used to demonstrate a business process using a Web-based application to allow users to log in, and view lists of instances that need to be reviewed and adjusted. The external application then sends the information back to e*Insight, thus allowing the business process to continue.

Note: *For information on installing the sample business model see the e*Insight Business Process Manager Installation Guide.*

Figure 117 User Activity - Bonus set



12.7 Overview of the ProcessOrder BP

In this example, we assume we have a custom Web application developed for the purpose of allowing various users to set the quantities. The current example uses an Authorization Activity to do a similar function from within the e*Insight GUI. By replacing the Authorization Activity with a User Activity, the business process can now integrate with an external application such as a Web-based application. This application would allows users to login, view lists of the business process instances, make appropriate changes, and have those changes re-enter the e*Insight business model for continued processing.

While this example specifically uses a custom Web application, design and development of the external application is left up to the developer. The external application uses the Java API's provided with the e*Insight engine (a complete description and list of these APIs is available in **"e*Insight User Activity API**

Methods" on page 393). The role of the APIs is to allow the external application to communicate with the e*Insight engine for the purpose of setting security and verifying users, retrieving business model related information, getting and returning attribute types and values, tracking instances, and many other functions related to the business process instances.

12.8 Overview

The major steps in the implementation are:

1. Add the User Activity to the Business Process.
2. Use the e*Insight GUI to configure the e*Gate schema that supports e*Insight.
3. Add and configure the user-defined e*Gate components.
4. Run and test the scenario.

12.8.1 Case Study: Order Processing with User Activity

The Order Process example uses an external application to allow various users to authorize the quantity. The external application retrieves new orders, and for those orders that are in stock, the users evaluate all orders with quantity >100. Based on certain criteria, users either authorize the order for shipment or reject the order. The user of the external application could also perform other changes to the order based on capabilities developed into the external application.

Note: *The external application can also retrieve orders and deliver them to the correct user if the developer of the application uses specific API functions.*

12.9 Step 1: Update the ProcessOrder BP in e*Insight

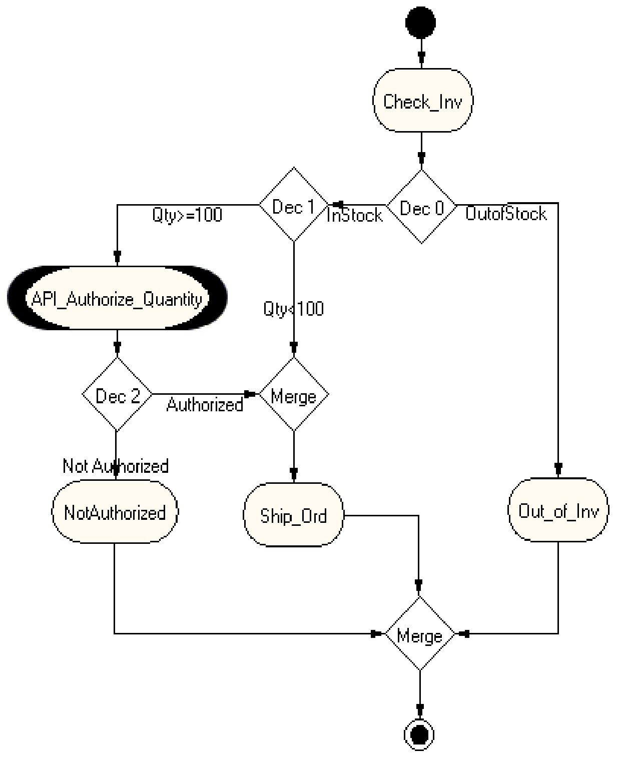
The following is a summary of the procedures for adding a User Activity in the e*Insight GUI.

1. Delete the **Authorize_Quantity** Activity.
2. Add a User Activity called **API_Authorize_Order**.
3. Add a Decision gate to receive the output from the User Activity.
4. Make the connections between the Decision gates, Merge gate, and Not Authorized Activity, as shown in **Figure 115 on page 184**.

Note: ***assignedTo*** *and* ***performedBy*** *are default Local Attributes used to ensure the user performing the Authorize/Reject/Undo activity is sanctioned to do so.*

For more information on creating a business process and using the e*Insight GUI, see the *e*Insight Business Process Manager User's Guide*.

Figure 118 ProcessOrder Business Process Model



12.10 Step 2: Configure the Integration Schema

After creating the additional component, you must configure the e*Gate Registry schema that supports the e*Insight system.

You can send notifications through e*Gate to the external application but in this example, the same result is achieved using only the e*Insight engine. To use only the e*Insight engine, double click the User Activity, and make sure the "Send Notifications through e*Gate check box is not marked.

Note: *To send notifications through e*Gate, make sure the check box is marked, and configure your e*Way or BOB as required.*

12.11 Step 3: Run and Test the e*Insight scenario

Once the schema has been set up in e*Gate you can run the scenario.

12.11.1 Testing the User Activity

Use the following procedure to test the functionality of the User Activity.

1. Start the e*Insight GUI and select the ProcessOrder business process. Switch to monitor mode.

Note: *Make sure that the business process has been enabled in the e*Insight GUI before attempting to run it.*

2. Make a final check of the e*Gate schema, using the tables to confirm all of the GUI associations. Make sure that all of the e*Insight components, including the user-defined components, are set to start automatically.

3. At the command line, type the following to start the schema. You must type the command on a single line.

```
stccb.exe -rh localhost -rs ProcessOrder -ln localhost_cb
     -un username -up password
```

   Substitute the appropriate username and password for your installation.

4. Start the e*Gate Monitor, and check the status of all the components. Any components displayed in red are not running. Any e*Insight components that are not running should be investigated before feeding data into the system.

5. Navigate to the location for the input data file, **InStock.~in**, (**c:\eGate\client\data\ProcessOrder**) and change the extension to "**.fin**".

The example instance should have the inventory quantity set to >100. This ensures that the business process instance follows the correct path to test the User Activity.

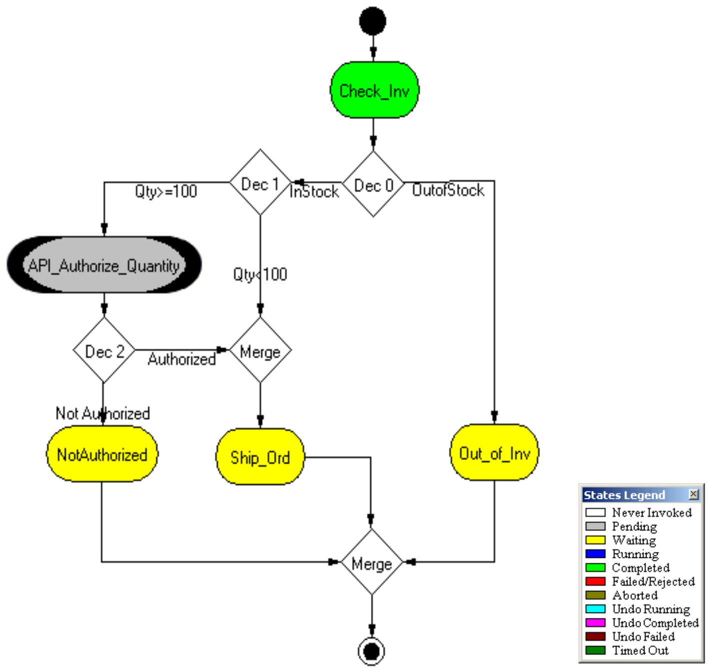
Note: *The change of the extension to "**.~in**" indicates that the data file has been picked up by the **START_BP** e*Way.*

6. Switch to the e*Insight GUI and, while in monitor mode, select the most recent BPI from the **List** tab, and then select the **Diagram** tab to observe the path that the data has taken.

   The **API_Authorize_Quantity** should be in a pending state. This remains in a pending state until the e*Insight engine receives a message from an external application relating to this Business Process Instance.

   Figure 119 shows how the BPI Diagram appears.

Figure 119 API_Authorize_Quantity Pending BPI Diagram



7 Start the external application. Return to the e*Insight GUI. The Business Process then completes using the route shown in **Figure 120 on page 192**.

The external application refers to any application developed by you to be inserted into the business process. A common example could be a method by which users can view lists of business instances assigned to them. These instances may require a simple review/authorize/reject process or, a more complex task using a custom-developed GUI interface to adjust or introduce attribute values. After the application completes its job, the updated business process instance information re-enters the User Activity and the business process continues.

The external application uses an API to communicate with the e*Insight engine. Listed below are some examples of API functions to help illustrate what kind of information is available to the external application (a complete list of API functions can be found in Chapter 16).

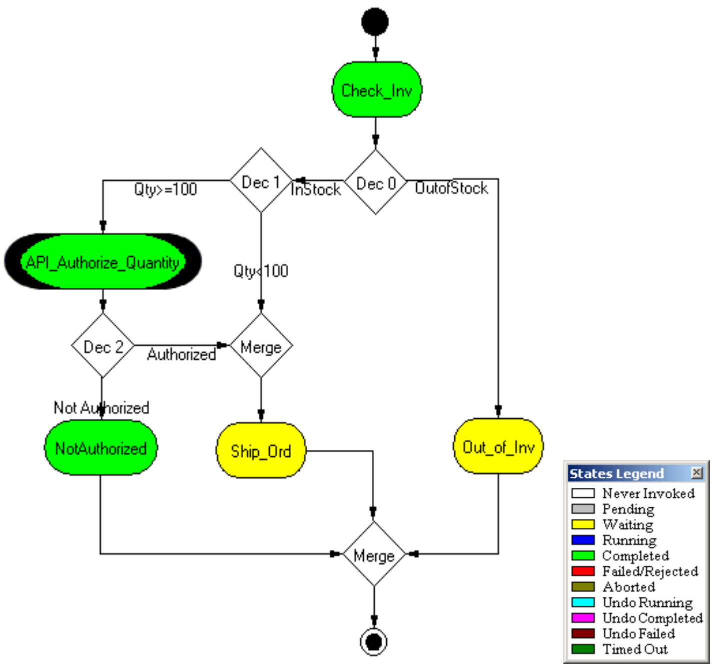
- **setUser**—This function passes the name of the user to the e*Insight engine if the external application uses login/password.
- **checkUserPrivileges**—This function establishes the security rights for the session using security levels as set within e*Insight.
- **getGlobalAttributeValue** and **getLocalAttributeValue**—This function receives attribute values for a business process instance for use within the external application.
- **getBusinessModelInstanceIds**—The external application uses this function to retrieve identification information for business model instances and then uses

this information to appropriately track instances when sending information back into the business process.

A example of a business model including a User Activity and an external application are shipped with e*Insight 4.5.2. This example can be installed by you and set up and used to demonstrate a business process using a Web-based application to allow users to log in, and view lists of instances that need to be reviewed and adjusted. The external application then sends the information back to e*Insight, thus allowing the business process to continue.

Note: *For information on installing the sample business model see the e*Insight Business Process Manager Installation Guide.*

Figure 120 User Activity - Item in stock



Chapter 13

e*Insight Sub-Process Implementation (eIJSchema)

This chapter discusses the steps involved to enhance the previous case study to include the Sub-Process.

The implementation starts with a local Sub-Process and then enhanced to demonstrate the use of the Remote Sub-Process and Dynamic Sub-Process.

This case study is a continuation of the previous example. See **"e*Insight Implementation (eIJSchema)" on page 96** for the initial configuration instructions.

13.1 Overview of the Sub-Process Example

The major steps in the implementation are:

1. Create and configure a new business process to perform the activities required for checking inventory.
2. Update the Payroll business process to replace the Calculate_Bonus activity with a Sub-Process.
3. Use the e*Insight GUI to configure the e*Gate schema that supports e*Insight.
4. Add and configure the user-defined e*Gate components.
5. Run and test the scenario.

The chapter begins with a description of the scenario and then shows how to use these steps to set it up.

13.2 Create the CalculateBonus BP in e*Insight

The following is a summary of the procedure for creating a BP in the e*Insight GUI.

1 Create a business process named CalculateBonus.

2 Add the activities.

3 Make the connections between the activities.

4 Enable the business process version.

For more information on creating a business process and using the e*Insight GUI, see the *e*Insight Business Process Manager User's Guide*.

Use the diagram shown in Figure 121 and the following tables to create the BP in e*Insight.

Figure 121 CalculateBonus Business Process Model

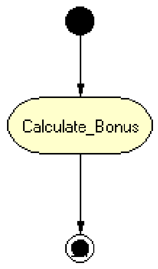


Table 37 BP Global Attributes

Attribute	Type	Data Direction	Default Value
Salary	Number	Input	
Grade	String	Input	
Bonus	Number	Output	0

Table 38 Activity Attributes

Activity	Attribute(s)	Input/Output
Calculate_Bonus	Salary	Input
	Grade	Input
	Bonus	Output

13.3 Configure the Integration Schema for CalculateBonus

The Calculate_Bonus activity uses the eX_Calculate_Bonus BOB that was created in **“Creating the eX_Calculate_Bonus BOB” on page 106**.

To configure the Calculate_Bonus Activity

1. In the e*Insight GUI, open the Calculate_Bonus activity properties.
2. On the **General** tab, select the **BOB** e*Gate module.
3. Check the module name is **eX_Calculate_Bonus**, and modify if necessary.
4. Check that the correct Participating Host is selected.
5. Close the **Calculate_Bonus** Activity properties.

13.4 Modify the Payroll BP in e*Insight

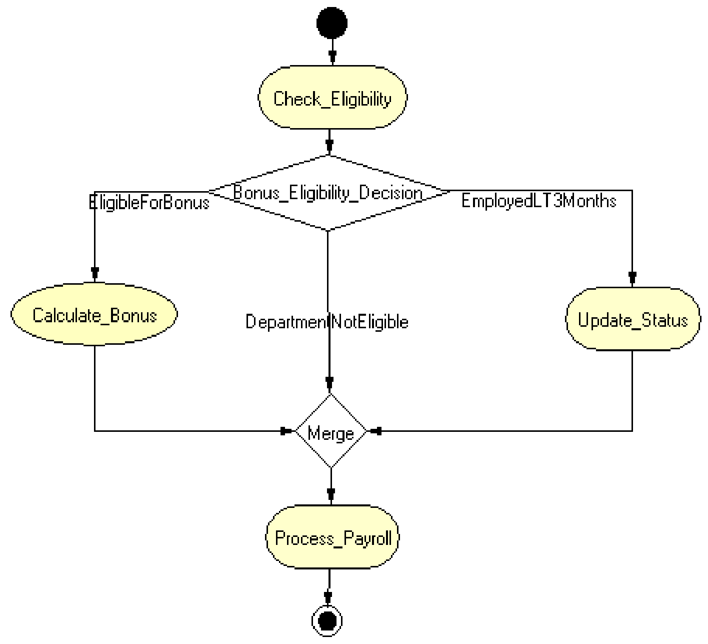
The following is a summary of the procedure for modifying the Payroll BP in the e*Insight GUI.

1 Delete the Calculate_Bonus activity.

2 Add the Calculate_Bonus Sub-Process.

3 Make the connections between the sub-process, decision gate and activity as shown in Figure 122.

Figure 122 Payroll BP with Sub-Process



4 Configure the Calculate_Bonus sub-process properties.

A Assign the CalculateBonus business process as the sub-process.

B Map the sub-process attributes to business process attributes as defined in Table 39.

Table 39 Sub-Process attribute mapping

Sub-Process Attributes	Business Process Attributes	Direction
Salary	Salary	Input
Grade	Grade	Input
Bonus	Bonus	Output

Note: *In this example, the sub-process attribute names and the business process attributes names are the same. This is not a requirement for the sub-process, however, it is a requirement for the dynamic sub-process.*

13.5 Configure the Integration Schema for Payroll

The schema created in **"e*Insight Implementation (eIJSchema)" on page 96** does not need to be modified for this example.

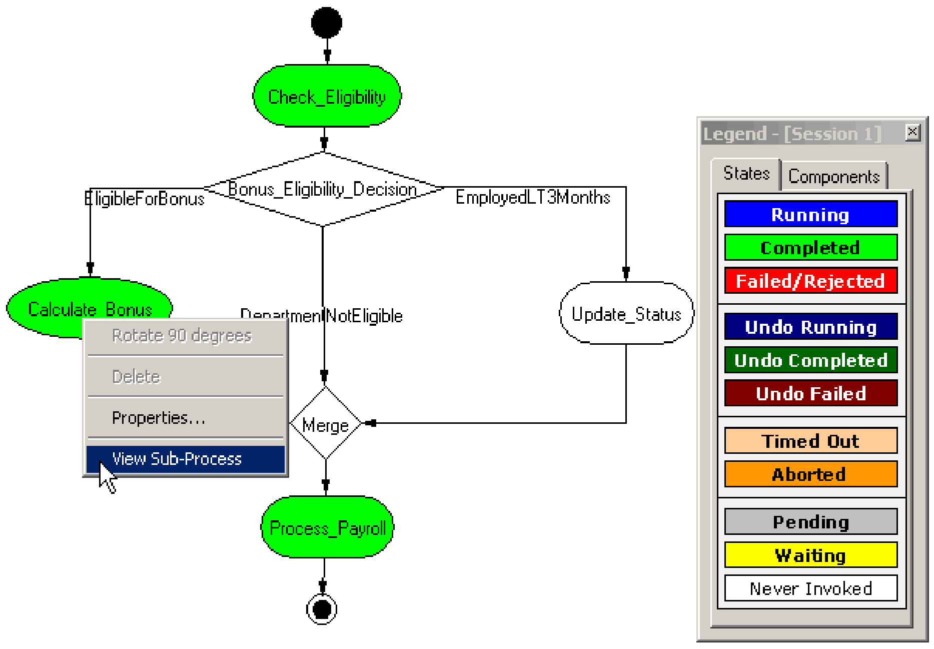
13.6 Run and Test the e*Insight scenario

Once the schema has been set up in e*Gate you can run the scenario.

Use the instructions in **"Run and Test the e*Insight scenario" on page 117** to test your sub-process.

Note: *You can access the CalculateBonus business process directly from the Payroll business process. Right-click on the Sub-Process in the parent business process as shown in Figure 123.*

Figure 123 Accessing the CalculateBonus business process



13.7 Overview of the Dynamic Sub-Process Example

The Dynamic Sub-Process Example is a continuation of the previous example.

In this example the Business Process that is called as the Sub-Process depends on where the department that the employee works for. If the department is accounts, then the **Calculate_Bonus_accounts** Business Process is called, and if the department is marketing, then the **Calculate_Bonus_marketing** Business Process is called.

The major steps in the implementation are:

1. Create and configure two new business processes (**Calculate_Bonus_accounts** for accounts and **Calculate_Bonus_marketing** for marketing) to perform the activities required for calculating the bonus.
2. Update the Payroll business process to replace the Calculate_Bonus sub-process with a dynamic sub-process.
3. Use the e*Insight GUI to configure the e*Gate schema that supports e*Insight.
4. Add and configure the user-defined e*Gate components.
5. Run and test the scenario.

The chapter begins with a description of the scenario and then shows how to use these steps to set it up.

13.8 Create the accounts BP in e*Insight

The following is a summary of the procedure for creating a Business Process for California orders.

1. Export business process named CalculateBonus.

Note: *The exported file is used in* **"Create the marketing BP in e*Insight" on page 201**.

2. Rename the CalculateBonus business process **accounts**.
3. Select the enabled version, and from the **File** menu, select **Save as New Version**.
4. Rename the activity **Calculate_Bonus_accounts**.
5. Enable the business process version.

13.9 Configure the Integration Schema for accounts

To configure the Calculate_Bonus_accounts activity

1. In the e*Insight GUI, open the **Calculate_Bonus_accounts** activity properties.
2. On the **General** tab, select the **BOB** e*Gate module.
3. Click **New**.

   The Define Collaboration dialog appears.
4. Click **OK**.
5. Create eX_Calculate_Bonus_marketing.xpr.

   The actual CRS is created later in **"To create eX_Calculate_Bonus_accounts.xpr" on page 200**.
6. Save the CRS.
7. Close the editor.
8. Click **Configure e*Gate**.

   You may be required to log into e*Gate.
9. Click **OK**, to close the information dialog.
10. Close the **Calculate_Bonus_accounts** Activity properties.

For more information on creating a business process and using the e*Insight GUI, see the *e*Insight Business Process Manager User's Guide*.

Creating the CRS in e*Gate

Since the CRS for eX_Calculate_Bonus_accounts BOB is very similar to the eX_Calculate_Bonus CRS previously created you can copy this instead of creating a new script from scratch.

To create eX_Calculate_Bonus_accounts.xpr

1. Open the Collaboration Editor.
2. Open eX_Calculate_Bonus.xpr.
3. From the **File** menu, select **Save As**, and enter the name **eX_Calculate_Bonus_accounts.xpr**.

Important: *You should replace the existing file in* ***Collaboration_Rules\Calculate_Bonus_accounts****.*

4. Compile and then close the editor.

13.10 Create the marketing BP in e*Insight

The following is a summary of the procedure for creating a Business Process for marketing.

1. Import the business process that was exported in **"Create the accounts BP in e*Insight" on page 199**.
2. Rename the imported CalculateBonus business process **marketing**.
3. Rename the activity Calculate_Bonus_marketing.
4. Enable the business process version.

13.11 Configure the Integration Schema for marketing

To configure the Calculate_Bonus_marketing activity using Monk

1. In the e*Insight GUI, open the **Calculate_Bonus_marketing** activity properties.
2. On the **General** tab, select the **BOB** e*Gate module.
3. Click **New**.

   The **Define Collaboration** dialog appears.
4. Click **OK**.
5. Create eX_Calculate_Bonus_marketing.xpr.

   The actual CRS script is created later in **"To create eX_Calculate_Bonus_marketing.xpr" on page 202**.
6. Save the CRS.
7. Close the editor.

Note: *You do not need to compile the script yet.*

8. In the **Calculate_Bonus_marketing** activity properties, check the module name is eX_Calculate_Bonus_marketing, and modify if necessary.
9. Click **Configure e*Gate**.

   You may be required to log into e*Gate.
10. Click **OK**, to close the information dialog.
11. Close the **Calculate_Bonus_marketing** Activity properties.

Creating the CRS for eX_Calculate_Bonus_marketing in e*Gate

Since the CRS for eX_Calculate_Bonus_marketing BOB is very similar to the eX_Calculate_Bonus CRS previously created you can copy this instead of creating a new script from scratch.

To create eX_Calculate_Bonus_marketing.xpr

1. Open the Collaboration Editor.
2. Open eX_Calculate_Bonus_marketing.xpr.
3. From the **File** menu, select **Save As**, and enter the name **eX_Calculate_Bonus_marketing.xpr**.

Important: *You should replace the existing file in* ***Collaboration_Rules\Calculate_Bonus_marketing****.*

4. Update the script to define grade 1 as 5000.
5. Compile and then close the editor.

13.12 Modify the Payroll BP in e*Insight

The following is a summary of the procedure for modifying the Payroll BP in the e*Insight GUI.

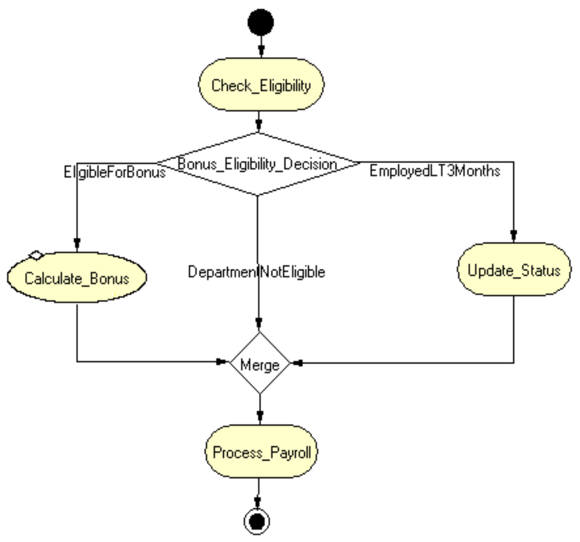
1 Save the enabled **Payroll** business process as a new version.

2 Delete the **Calculate_Bonus** activity.

3 Add the **Calculate_Bonus** Dynamic Sub-Process.

4 Make the connections between the dynamic sub-process, decision gate and activity as shown in Figure 124.

Figure 124 Payroll BP with Dynamic Sub-Process



5 Configure the **Calculate_Bonus** dynamic sub-process properties.

A Ensure that the dynamic sub-process uses the **Incoming Global Attribute Value**.

B Map the global attributes to the sub-process as defined in Table 39.

Table 40 Global attribute mapping

Global Attributes	Direction
Department	Input
Salary	Input
Grade	Input
Bonus	Output

C From the **Local Attributes** tab, edit the **subBPName**, and select **Department** from the Value drop down list.

13.13 Configure the Integration Schema for Payroll

The schema created in **"e*Insight Implementation (eIJSchema)" on page 96** does not need to be modified for this example.

13.14 Run and Test the e*Insight scenario

Once the schema has been set up in e*Gate you can run the scenario.

Use the instructions in **"Run and Test the e*Insight scenario" on page 117** to test your sub-process.

Chapter 14

e*Insight Sub-Process Implementation (eISchema)

This chapter discusses the steps involved to enhance the previous case study to include the Sub-Process.

The implementation starts with a local Sub-Process and then enhanced to demonstrate the use of the Remote Sub-Process and Dynamic Sub-Process.

This case study is a continuation of the previous example. See **“e*Insight Implementation (eISchema)” on page 133** for the initial configuration instructions.

14.1 Overview of the Sub-Process Example

The major steps in the implementation are:

1. Create and configure a new business process to perform the activities required for checking inventory.
2. Update the ProcessOrder business process to replace the Check_Inv activity with a Sub-Process.
3. Use the e*Insight GUI to configure the e*Gate schema that supports e*Insight.
4. Add and configure the user-defined e*Gate components.
5. Run and test the scenario.

The chapter begins with a description of the scenario and then shows how to use these steps to set it up.

14.2 Create the CheckInventory BP in e*Insight

The following is a summary of the procedure for creating a BP in the e*Insight GUI.

1. Create a business process named CheckInventory.
2. Add the activities.
3. Make the connections between the activities.
4. Enable the business process version.

For more information on creating a business process and using the e*Insight GUI, see the e*Insight Business Process Manager *User's Guide*.

Use the diagram shown in Figure 125 and the following tables to create the BP in the e*Insight.

Figure 125 CheckInventory Business Process Model

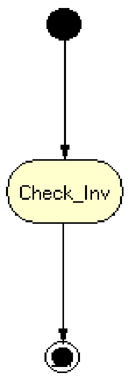


Table 41 BP Global Attributes

Attribute	Type	Data Direction
Item_Number	String	Input
Order_Quantity	Number	Input
Order_Status	String	Output
In_Stock	Boolean	Output

Table 42 Activity Attributes

Activity	Attribute(s)	Input/Output
Check_Inv	Item_Number	Input
	Order_Quantity	Input
	In_Stock	Output
	Order_Status	Output

14.3 Configure the Integration Schema for CheckInventory

The Check_Inv activity uses the eX_Check_Inv BOB that was created in **"Creating the eX_Check_Inv BOB" on page 141**.

To configure the Check_Inv Activity

1. In the e*Insight GUI, open the Check_Inv activity properties.
2. On the **General** tab, select the **BOB** e*Gate module.
3. Check the module name is **eX_Check_Inv**, and modify if necessary.
4. Check that the correct Participating Host is selected.
5. Close the **Check_Inv** Activity properties.

14.4 Modify the ProcessOrder BP in e*Insight

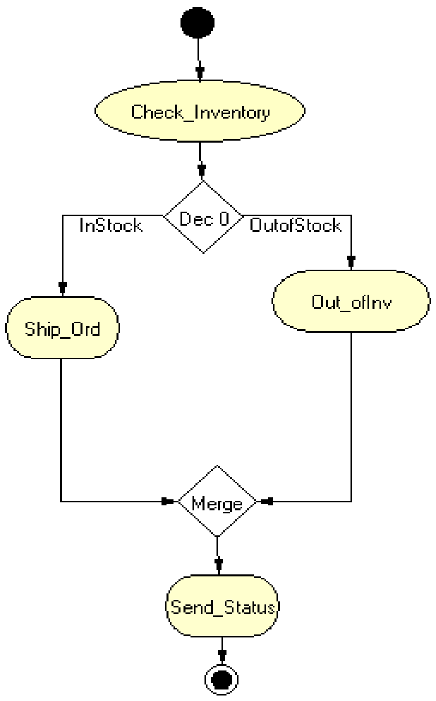
The following is a summary of the procedure for modifying the ProcessOrder BP in the e*Insight GUI.

1 Delete the Check_Inv activity.

2 Add the Check_Inventory Sub-Process.

3 Make the connections between the sub-process, decision gate and activity as shown in Figure 126.

Figure 126 ProcessOrder BP with Sub-Process



4 Configure the Check_Inventory sub-process properties.

A Assign the CheckInventory business process as the sub-process.

B Map the sub-process attributes to business process attributes as defined in Table 43.

Table 43 Sub-Process attribute mapping

Sub-Process Attributes	Business Process Attributes	Direction
Item_Number	Item_Number	Input
Order_Quantity	Order_Quantity	Input
In_Stock	In_Stock	Output
Order_Status	Order_Status	Output

Note: *In this example, the sub-process attribute names and the business process attributes names are the same. This is not a requirement for the sub-process, however, it is a requirement for the dynamic sub-process.*

For more information on creating a business process and using the e*Insight GUI, see the e*Insight Business Process Manager *User's Guide*.

14.5 Configure the Integration Schema for ProcessOrder

The schema created in **"e*Insight Implementation (eISchema)" on page 133** does not need to be modified for this example.

14.6 Run and Test the e*Insight scenario

Once the schema has been set up in e*Gate you can run the scenario.

Use the instructions in **"Run and Test the e*Insight scenario" on page 162** to test your sub-process.

14.7 Overview of the Dynamic Sub-Process Example

The Dynamic Sub-Process Example is a continuation of the previous example.

In this example the Business Process that is called as the Sub-Process depends on where the order is being delivered. If the order is for California, then the **Check_Inv_CA** Business Process is called, and if the order is for Oregon, then the **Check_Inv_OR** Business Process is called.

The major steps in the implementation are:

1. Create and configure two new business processes (**Check_Inv_CA** for California and **Check_Inv_OR** for Oregon) to perform the activities required for checking inventory.
2. Update the ProcessOrder business process to replace the Check_Inventory sub-process with a dynamic sub-process.
3. Use the e*Insight GUI to configure the e*Gate schema that supports e*Insight.
4. Add and configure the user-defined e*Gate components.
5. Run and test the scenario.

The chapter begins with a description of the scenario and then shows how to use these steps to set it up.

14.8 Create the CA BP in e*Insight

The following is a summary of the procedure for creating a Business Process for California orders.

1 Export business process named CheckInventory.

Note: *The exported file is used in* **"Create the OR BP in e*Insight" on page 214**.

2 Rename the CheckInventory business process **CA**.

3 Select the enabled version, and from the **File** menu, select **Save as New Version**.

4 Rename the activity **Check_Inv_CA**.

5 Enable the business process version.

14.9 Configure the Integration Schema for CA

To configure the Check_Inv_CA activity using Monk

1 In the e*Insight GUI, open the **Check_Inv_CA** activity properties.

2 On the **General** tab, select the **BOB** e*Gate module.

3 Click **New**.

The **Define Collaboration** dialog appears.

4 Click **OK**.

5 Create eX_Check_Inv_CA.tsc. The source and destination Event Type Definitions are eX_Standard_Event.

The actual CRS script is created later in **"To create eX_Check_Inv_CA.tsc using Monk" on page 213**.

6 Save the CRS.

7 Close the editor.

8 In the **Check_Inv_CA** activity properties, check the module name is eX_Check_Inv_CA, and modify if necessary.

9 Click **Configure e*Gate Schema**.

You may be required to log into e*Gate.

10 Click **OK**, to close the information dialog.

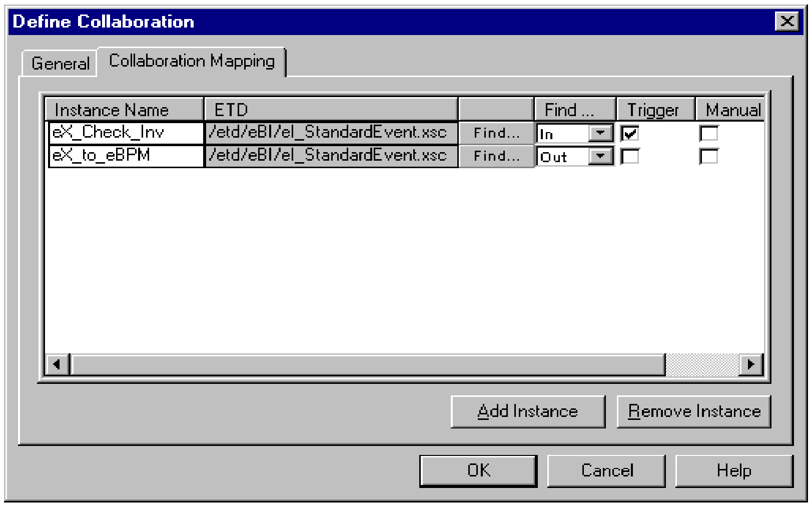
11 Close the **Check_Inv_CA** Activity properties.

To configure the Check_Inv_CA activity using Java

1. In the e*Insight GUI, open the **Check_Inv_CA** activity properties.
2. On the **General** tab, select the **BOB** e*Gate module.
3. Click **New**.

   The Define Collaboration dialog appears.
4. Select the **Define Mapping** tab.
5. Configure the instances as shown in Figure 127.

Figure 127 Define Mapping for eX_Check_Inv



Instance Name	ETD		Find ...	Trigger	Manual
eX_Check_Inv	/etd/eBI/eI_StandardEvent.xsc	Find...	In	✓	☐
eX_to_eBPM	/etd/eBI/eI_StandardEvent.xsc	Find...	Out	☐	☐

6. Click **OK**.
7. Create eX_Check_Inv_CA.xsc.

   The actual CRS is created later in **"To create eX_Check_Inv_CA.xsc using Java" on page 213**.
8. Save the CRS.
9. Close the editor.
10. In the **Check_Inv_CA** activity properties, check the module name is **eX_Check_Inv_CA**, and modify if necessary.
11. Click **Configure e*Gate**.

    You may be required to log into e*Gate.
12. Click **OK**, to close the information dialog.
13. Close the **Check_Inv_CA** Activity properties.

For more information on creating a business process and using the e*Insight GUI, see the e*Insight Business Process Manager *User's Guide*.

Creating the CRS in e*Gate

Since the CRS for eX_Check_Inv_CA BOB is very similar to the eX_Check_Inv CRS previously created you can copy this instead of creating a new script from scratch.

To create eX_Check_Inv_CA.tsc using Monk

1. Open the Collaboration Editor.
2. Open eX_Check_Inv.tsc.
3. From the **File** menu, select **Save As**, and enter the name **eX_Check_Inv_CA.tsc**. You should replace the existing file.
4. Close the editor.

To create eX_Check_Inv_CA.xsc using Java

1. Open the Collaboration Editor.
2. Open eX_Check_Inv.xsc.
3. From the **File** menu, select **Save As**, and enter the name **eX_Check_Inv_CA.xsc**. You should replace the existing file.
4. Compile and then close the editor.

14.10 Create the OR BP in e*Insight

The following is a summary of the procedure for creating a Business Process for Oregon.

1. Import the business process that was exported in **"Create the CA BP in e*Insight" on page 211**.
2. Rename the imported CheckInventory business process **OR**.
3. Rename the activity Check_Inv_OR.
4. Enable the business process version.

14.11 Configure the Integration Schema for OR

To configure the Check_Inv_OR activity using Monk

1. In the e*Insight GUI, open the **Check_Inv_OR** activity properties.
2. On the **General** tab, select the **BOB** e*Gate module.
3. Click **New**.

   The **Define Collaboration** dialog appears.
4. Click **OK**.
5. Create eX_Check_Inv_OR.tsc. The source and destination Event Type Definitions are eX_Standard_Event.

   The actual CRS script is created later in **"To create eX_Check_Inv_OR.tsc using Monk" on page 216**.
6. Save the CRS.
7. Close the editor.
8. In the **Check_Inv_OR** activity properties, check the module name is eX_Check_Inv_OR, and modify if necessary.
9. Click **Configure e*Gate**.

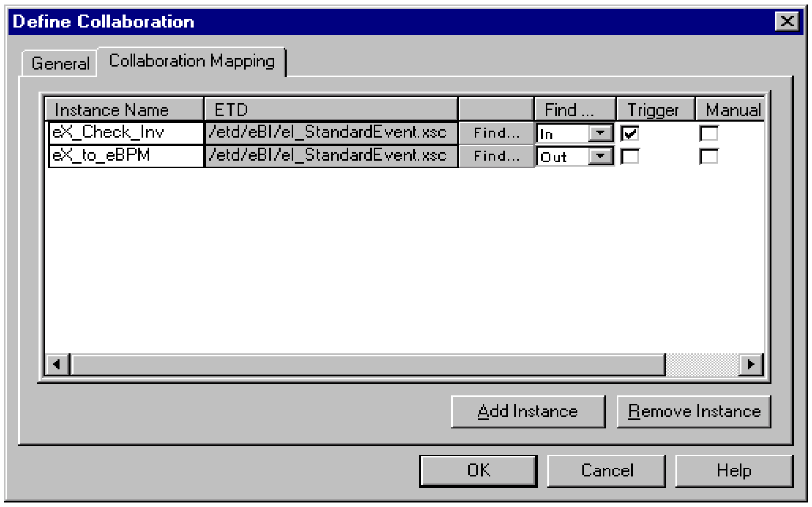
   You may be required to log into e*Gate.
10. Click **OK**, to close the information dialog.
11. Close the **Check_Inv_OR** Activity properties.

To configure the Check_Inv_OR activity using Java

1. In the e*Insight GUI, open the **Check_Inv_OR** activity properties.
2. On the **General** tab, select the **BOB** e*Gate module.
3. Click **New**.

   The Define Collaboration dialog appears.
4. Select the **Define Mapping** tab.
5. Configure the instances as shown in Figure 128.

Figure 128 Define Mapping for eX_Check_Inv



Instance Name	ETD		Find ...	Trigger	Manual
eX_Check_Inv	/etd/eBI/eI_StandardEvent.xsc	Find...	In	☑	☐
eX_to_eBPM	/etd/eBI/eI_StandardEvent.xsc	Find...	Out	☐	☐

6. Click **OK**.
7. Create eX_Check_Inv_OR.xsc.

   The actual CRS is created later in **"To create eX_Check_Inv_OR.xsc using Java" on page 216**.
8. Save the CRS.
9. Close the editor.
10. In the **Check_Inv_OR** activity properties, check the module name is **eX_Check_Inv_OR**, and modify if necessary.
11. Click **Configure e*Gate**.

    You may be required to log into e*Gate.
12. Click **OK**, to close the information dialog.
13. Close the **Check_Inv_OR** Activity properties.

For more information on creating a business process and using the e*Insight GUI, see the e*Insight Business Process Manager *User's Guide*.

Creating the CRS in e*Gate

Since the CRS for eX_Check_Inv_OR BOB is very similar to the eX_Check_Inv CRS previously created you can copy this instead of creating a new script from scratch.

To create eX_Check_Inv_OR.tsc using Monk

1. Open the Collaboration Editor.
2. Open eX_Check_Inv.tsc.
3. Update the script to define 11111 as in stock.
4. From the **File** menu, select **Save As**, and enter the name **eX_Check_Inv_OR.tsc**. You should replace the existing file.
5. Close the editor.

To create eX_Check_Inv_OR.xsc using Java

1. Open the Collaboration Editor.
2. Open eX_Check_Inv_OR.xsc.
3. From the **File** menu, select **Save As**, and enter the name **eX_Check_Inv_OR.xsc**. You should replace the existing file.
4. Update the script to define 11111 as out of stock.
5. Compile and then close the editor.

14.12 Modify the ProcessOrder BP in e*Insight

The following is a summary of the procedure for modifying the ProcessOrder BP in the e*Insight GUI.

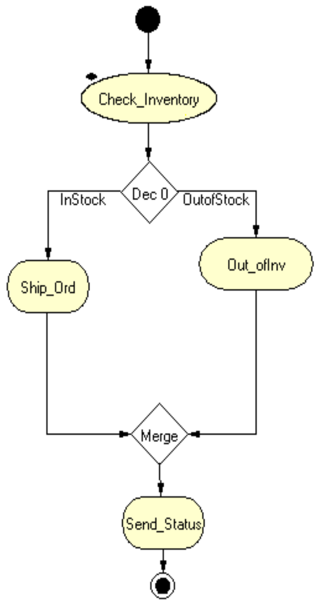
1 Save the enable ProcessOrder business process as a new version.

2 Delete the Check_Inv activity.

3 Add the Check_Inventory Dynamic Sub-Process.

4 Make the connections between the dynamic sub-process, decision gate and activity as shown in Figure 129.

Figure 129 ProcessOrder BP with Dynamic Sub-Process



5 Configure the Check_Inventory dynamic sub-process properties.

A Ensure that the dynamic sub-process uses the **Incoming Global Attribute Value**.

B Map the global attributes to the sub-process as defined in Table 43.

Table 44 Global attribute mapping

Global Attributes	Direction
Address_State	Input
Item_Number	Input
Order_Quantity	Input
In_Stock	Output
Order_Status	Output

C From the Local Attributes tab, edit **subBPName**, and select Address_State from the Value drop down list.

For more information on creating a business process and using the e*Insight GUI, see the e*Insight Business Process Manager *User's Guide*.

14.13 Configure the Integration Schema for ProcessOrder

The schema created in **"e*Insight Implementation (eISchema)" on page 133** does not need to be modified for this example.

14.14 Run and Test the e*Insight scenario

Once the schema has been set up in e*Gate you can run the scenario.

Use the instructions in **"Run and Test the e*Insight scenario" on page 162** to test your sub-process.

Chapter 15

e*Insight Remote Sub-Process Implementation

This chapter discusses the steps involved to configure the Remote Sub-Process. It covers the installation and configuration of Tomcat on single and multiple machines, and also works through a detailed implementation.

This case study is a continuation of a previous example. See either **"e*Insight Implementation (eIJSchema)" on page 96** or **"e*Insight Implementation (eISchema)" on page 133** for the initial configuration instructions.

15.1 Overview

The major topics covered in this chapter are:

- Installation and configuration of Tomcat
- Implementation steps

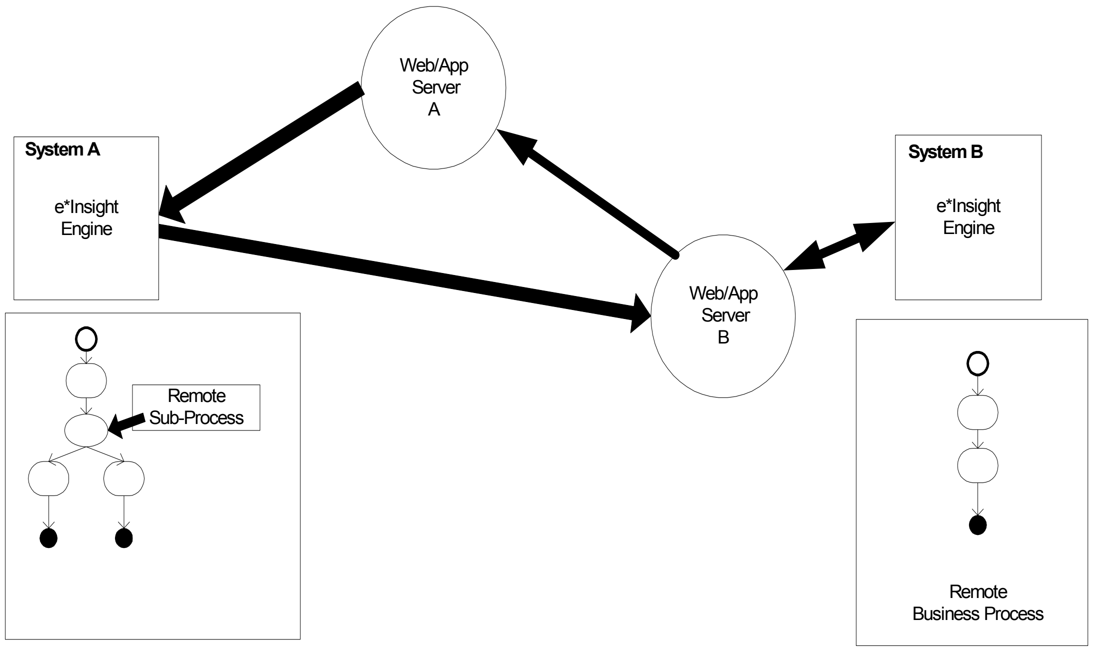
15.2 Overview of the Remote Sub-Process

The Remote Sub-Process allows you to access a Business Process defined on a different machine. Business Process messages are sent between the two e*Insight engines via a Web/Application Server. The Web/Application Server routes the Business Process message to the correct location.

Note: *e*Insight uses Apache/Tomcat Web/Application Server.*

The Remote Sub-Process can be used across machines with the LAN, WAN, or the Internet. The deployment of machines that are separated by a firewall is considered in **"Installation of Tomcat and e*Insight on Different Hosts" on page 222**.

Figure 130 Remote Sub-Process Overview



The Remote Sub-Process defined in this scenario uses Apache/Tomcat as the Web/ App Server. Tomcat is provided with e*Insight, and this needs to be installed and configured before using the Remote Sub-Process with SOAP.

15.3 Installation and Configuration of Tomcat

This section describes how to install and configure Tomcat. This procedure is divided into three sections:

Table 45 Steps for Installation and Configuration of Tomcat

	Step	Procedure
1	Install Tomcat	**"Installing Tomcat" on page 220**
2	Configure Tomcat	**"Configuring Tomcat" on page 221**
3	Deploy the SOAP service	**"Deploying the SOAP Service" on page 221**

Installing Tomcat

The e*Insight installation provides Tomcat and additional files required for the Soap implementation.

To install Tomcat

1 Install the SOAP add-on via the Installation Wizard. For more information, see the *e*Insight Business Process Manager Installation Guide*.

2 Unzip jakarta-tomcat-3.2.1.zip

Note: *When you install from the zip file (in the <eInsight>\SOAP folder), extract the files to the root folder on your C: drive. The zip file creates a folder named jakarta-tomcat-3.2.1 with accompanying files and subfolders.*

Configuring Tomcat

This section describes how to configure Tomcat once it has been installed. The procedure updates and copies files that are required for an e*Insight/Soap implementation.

To configure Tomcat

1 Open **<eInsight>\Soap\eInsight_tomcat.bat**. Locate the EINSIGHT variable and ensure that it is set to <eInsight>\Soap.

2 Copy the following files from <eInsight>\Soap.

- copy eInsight_startup.bat to <tomcat>\bin
- copy eInsight_tomcat.bat to <tomcat>\bin
- copy soap.war to <tomcat>\webapps

3 Edit EInsightBridge.properties to connect to correct database.

Note: *You should make sure that when you set parameter values, you do not add any trailing spaces, as these are interpreted as part of the parameter.*

4 Set system environment classpath c:\eInsight\soap.

A Go to Control Panel

B Select System

C Select Advanced tab

D Select Environment Variables

E Edit CLASSPATH on system variables

F Add "c:\eInsight\soap" to Variable Value to the front

5 From a command prompt, change directory to <tomcat> folder and run

```
bin\eInsight_startup.bat
```

Deploying the SOAP Service

The Tomcat Server has to be made aware of the SOAP service and the methods that it can use. This is achieved by deploying the SOAP service using the procedure described below.

To Deploy the SOAP Service

1 Open **<eInsight>\Soap\registerSoapService.cmd**.

2 Edit the following variables for your system:

```
SOAP_SERVICE
INTEGRATOR
TOMCAT_HOME
JAVA_HOME
```

3 Open **eInsightDeploymentDescripter.xml** in any text editor. Modify the following parameter:

```
id="<URN>"
```

where <URN> should be in the format **urn:<user defined name>**.

4 In a command prompt, change directory to <einsight>\soap.

5 Run **RegisterSoapService.cmd**.

Note: *Before you run registerSoapService.cmd, make sure your Apache Tomcat service is running.*

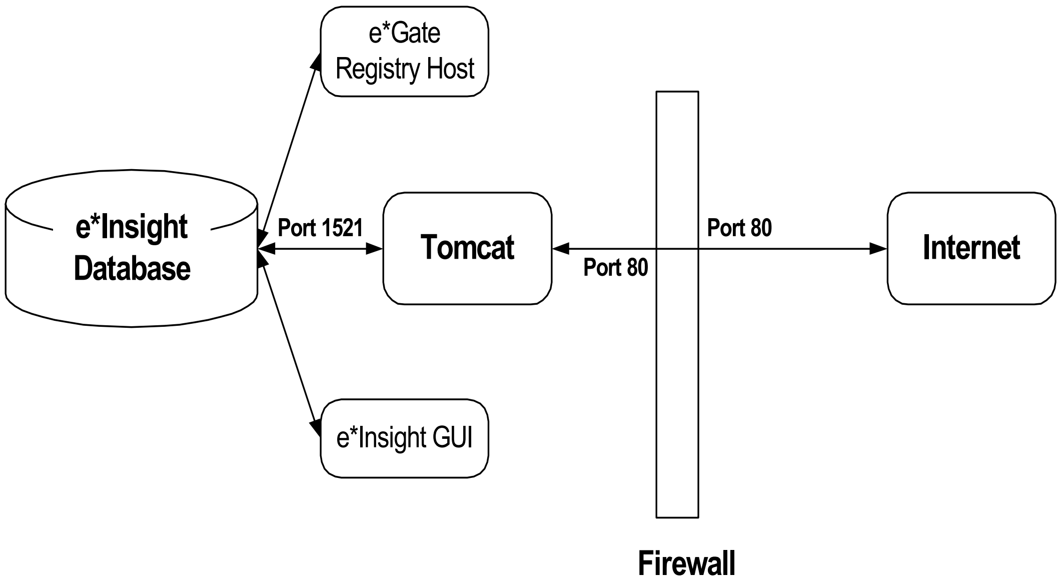
6 Start Internet Explorer and go to url: **http://<webhost>:8080/soap/admin/index.html**

7 Click List to check that the URN has been successfully deployed.

15.4 Installation of Tomcat and e*Insight on Different Hosts

There are a number of considerations when deploying Tomcat and e*Insight on different hosts. Figure 131 shows a possible scenario, where Tomcat and e*Insight are installed on separate machines and are separated from the external application by a firewall.

Figure 131 e*Insight and Tomcat Deployment



The firewall is configured to use port 80 to communicate with Tomcat. Tomcat is configured to communicate with the database using a direct database connection.

Installing Tomcat on a different machine to e*Insight

The installation of the SOAP add-on installs all the files required for the installation and configuration of Tomcat. To ensure that the correct files reside on the Tomcat machine either:

A From the Tomcat machine, run the installation wizard and install the SOAP add-on, or

B Install the SOAP add-on on another machine and then copy the <eInsight>\SOAP directory to the C drive on the Tomcat machine.

To configure Tomcat, see **"To configure Tomcat" on page 221**.

15.5 Overview of the Remote Sub-Process Example (eIJSchema)

This case study is a continuation of the Payroll example. The procedure that checked the inventory is going to be moved to a remote system and SOAP messages is used to send the data to and from the remote process.

The Payroll example must be completed before the remote sub-process example. See **"e*Insight Implementation (eIJSchema)" on page 96** for the initial configuration instructions.

For this example, SystemA is the machine where the Payroll example was created. SystemB is the machine where the business process that calculates the bonus runs.

The major steps in the implementation are:

1. Configure Tomcat on both machines.
2. Create the CalculateBonus business process.
3. Create the e*Gate schema that supports the CalculateBonus business process.
4. Update the Payroll business process.
5. Run and test the scenario.

15.6 Install and configure Tomcat

Tomcat needs to be installed on both SystemA and SystemB. In a real scenario, it may be necessary to run Tomcat on a different machine to e*Insight, but in this example we assume that Tomcat and e*Insight run on the same machine.

Follow the instructions in **"Installation and Configuration of Tomcat" on page 220** for both machines. Define the URN's as follows:

SystemA (Original system)	urn:Payroll
SystemB (Added for the Remote Sub-Process example)	urn:CalculateBonus

15.7 Create the CalculateBonus BP in e*Insight

The following is a summary of the procedure for creating a BP in the e*Insight GUI.

1. Create a business process named CalculateBonus.
2. Add the activities.
3. Make the connections between the activities.
4. Create and assign the global attributes.

Use the diagram shown in Figure 134 and the following tables to create the BP in e*Insight.

Figure 132 CalculateBonus Business Process Model

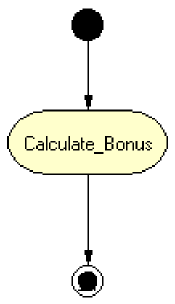


Table 46 BP Global Attributes

Attribute	Type	Data Direction	Default Value
Salary	Number	Input	
Grade	String	Input	
Bonus	Number	Output	0

Table 47 Activity Attributes

Activity	Attribute(s)	Input/Output
Calculate_Bonus	Salary	Input
	Grade	Input
	Bonus	Output

5 In the business process properties, select **Send Business Process Done Event**.

6 Enable the business process version.

To configure the Partner Information

1 From the **Options** menu, select **Define Information for Partners**.

2 Enter **UUID**.

This is a unique name that identifies your e*Insight database to other Trading Partners. For example, **SystemB**.

3 Enter the URN.

This corresponds to the URN setup in the SOAP client. For example, **urn:CalculateBonus**.

4 Enter the URL.

This is the location of the SOAP client. By default this is **http://localhost:8080/soap/servlet/rpcrouter**.

5 Click **OK**.

15.8 Configure the Integration Schema for CalculateBonus

All the activities in this example are carried out using e*Gate components.

15.8.1 Create the CalculateBonus Schema

Use the following procedure to create a copy of the eIJSchema:

1 From the e*Insight GUI **File** menu, select **New e*Gate Schema**.

2 Enter or select a **Registry Host** on which to create the schema.

3 Enter a **Username** and **Password** that is valid on the Registry Host.

4 From the **Based on** list, select **eIJSchema (Java)**.

5 In the **Name** box, enter **CalculateBonus**.

6 Click **OK**.

15.8.2 Configure the CalculateBonus Schema

Since the module for the Calculate_Bonus activity has been created in a previous case study, you only need to import the module into the CalculateBonus schema and configure the e*Insight engine and JMS server. These three procedures are described below.

Create the Calculate_Bonus activity BOB

1. In the **SystemA** e*Gate Enterprise Manager GUI, open the **Payroll** schema.
2. Select the eX_Calculate_Bonus **BOB**.
3. From the File menu select **Export Module Definitions to File**.

   The **Select Archive File** dialog appears.
4. Enter eX_Calculate_Bonus, and then click **OK**.

   The file **eX_Calculate_Bonus.zip** is created.
5. Log into the **SystemB** e*Gate Enterprise Manager GUI and open the **CalculateBonus** schema.
6. From the File menu select **Import Module Definitions from File**.
7. Locate the file created in step 4.
8. Click **OK**.

Edit the eIcp_eInsightEngine Connection Configuration File

Most of the parameter settings in the eIcp_eInsightEngine connection's configuration file should not be changed. **"Configuring the e*Insight Engine Connection" on page 34** discusses the parameters that may need to be changed depending on the implementation. Use the e*Way Editor and the information in **"Configuring the e*Insight Engine Connection" on page 34** to make the required changes for the CalculateBonus example.

Configure the JMS Connection

The JMS connection for e*Insight must be configured for your system. The minimal configuration required for this implementation is described in this section. For more information on JMS IQ Services, see *SeeBeyond JMS Intelligent Queue User's Guide*.

To configure the JMS connection

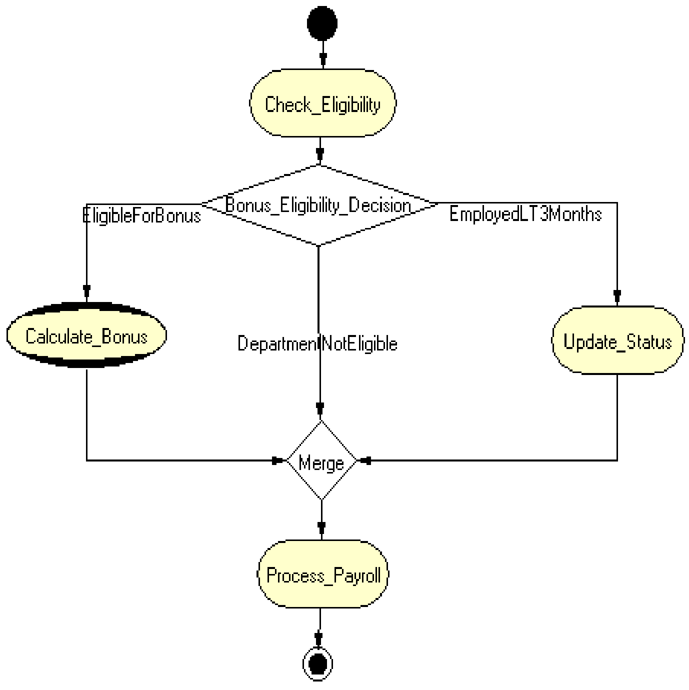
1. From the e*Gate Enterprise Manager GUI components view, select the **e*Way Connections** folder.
2. Select the **eX_cpeInsightJMS** connection, and click the **Properties** tool.
3. In the **e*Way Connection Configuration File** section, click **Edit**.
4. From the **Goto Section** list, select **Message Service**.
5. Enter a **Server Name** and **Host Name** where your JMS server resides.

15.9 Modify the Payroll BP in e*Insight

The following is a summary of the procedure for modifying the Payroll BP in the e*Insight GUI.

1. Delete the Calculate_Bonus activity.
2. Add the Calculate_Bonus Remote Sub-Process.
3. Make the connections between the remote sub-process, decision gate and activity as shown in Figure 135.

Figure 133 Payroll BP with Remote Sub-Process



4. Assign **EligibleForBonus** as the default link in **Bonus_Eligibility_Decision**.
5. Configure the Partner Information
   A. From the **Options** menu, select **Define Information for Partners**.
   B. Enter **UUID**.
   This is a unique name that identifies your e*Insight database to other Trading Partners. For example, **SystemA**.
   C. Enter the URN.
   This corresponds to the URN setup in the SOAP client. For example, **urn:Payroll**.
   D. Enter the URL.
   This is the location of the SOAP client. By default this is **http://localhost:8080/soap/servlet/rpcrouter**.

E Click **OK**.

6 Configure the **Calculate_Bonus** remote sub-process properties.

A Enter the relevant information for the remote e*Insight database including URL, URN, user name and password. Use the following values:

URL	http://<SystemB>:8080/soap/servlet/rpcrouter
URN	urn:CalculateBonus
User Name	Anonymous
Password	

Note: *To use an Anonymous login you must create a user called Anonymous on the remote system. Although e*Xchange Administrator requires that a password is defined to create the user, this does not need to be supplied when connecting via SOAP.*

Click **Connect**, and select the **CalculateBonus** business process from the Remote Sub-Process drop-down list

B Map Attributes as shown in Table 51.

Table 48 Sub-Process attribute mapping

Sub-Process Attributes	Business Process Attributes	Direction
Salary	Salary	Input
Grade	Grade	Input
Bonus	Bonus	Output

7 Enable the business process version.

For more information on creating a business process and using the e*Insight GUI, see the *e*Insight Business Process Manager User's Guide*.

15.10 Configure the Integration Schema for Payroll

The schema created in **"e*Insight Implementation (eISchema)" on page 133** does not need to be modified for this example.

15.11 Run and Test the e*Insight scenario

Once the schema has been set up in e*Gate you can run the scenario.

To test the Remote Sub-Process scenario

1. Start Tomcat on both machines.
2. Start the control broker for Payroll, and check all components are running.
3. Start the control broker for CalculateBonus, and check all components are running.
4. Rename the data file.

To monitor a remote business process

1. Select the remote business process, and then select the down arrow.
2. If prompted, enter a user name and password provided by the partner.

15.12 Overview of the Remote Sub-Process Example (eISchema)

This case study is a continuation of the Process Order example. The procedure that checked the inventory is going to be moved to a remote system and SOAP messages are used to send the data to and from the remote process.

The Process Order example must be completed before the remote sub-process example. See **"e*Insight Implementation (eISchema)" on page 133** for the initial configuration instructions.

For this example, SystemA is the machine where the Process Order example was created. SystemB is the machine where the business process that checks the inventory runs.

The major steps in the implementation are:

1. Configure Tomcat on both machines.
2. Create the CheckInventory business process.
3. Create the e*Gate schema that supports the CheckInventory business process.
4. Update the ProcessOrder business process.
5. Run and test the scenario.

15.13 Install and configure Tomcat

Tomcat needs to be installed on both SystemA and SystemB. In a real scenario, it may be necessary to run Tomcat on a different machine to e*Insight, but in this example we assume that Tomcat and e*Insight run on the same machine.

Follow the instructions in **"Installation and Configuration of Tomcat" on page 220** for both machines. Define the URN's as follows:

SystemA	urn:ProcessOrder
SystemB	urn:CheckInventory

15.14 Create the CheckInventory BP in e*Insight

The following is a summary of the procedure for creating a BP in the e*Insight GUI.

1. Create a business process named CheckInventory.
2. Add the activities.
3. Make the connections between the activities.
4. Create and assign the global attributes.

Use the diagram shown in Figure 134 and the following tables to create the BP in e*Insight.

Figure 134 CheckInventory Business Process Model

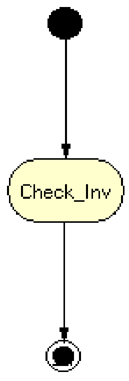


Table 49 BP Global Attributes

Attribute	Type	Data Direction
Item_Number	String	Input
Order_Quantity	Number	Input
Order_Status	String	Output
In_Stock	Boolean	Output

Table 50 Activity Attributes

Activity	Attribute(s)	Input/Output
Check_Inv	Item_Number	Input
	Order_Quantity	Input
	In_Stock	Output
	Order_Status	Output

5 In the business process properties, select **Send Business Process Done Event**.

6 Enable the business process version.

To configure the Partner Information

1 From the **Options** menu, select **Define Information for Partners**.

2 Enter **UUID**.

This is a unique name that identifies your e*Insight database to other Trading Partners. For example, **SystemB**.

3 Enter the URN.

This corresponds to the URN setup in the SOAP client. For example, **urn:CheckInventory**.

4 Enter the URL.

This is the location of the SOAP client. By default this is **http://localhost:8080/soap/servlet/rpcrouter**.

5 Click **OK**.

15.15 Configure the Integration Schema for CheckInventory

All the activities in this example are carried out using e*Gate components.

15.15.1 Create the CheckInventory Schema

Use the following procedure to create a copy of the eISchema:

1 Open the eISchema in the e*Gate Enterprise Manager GUI.

A Start the e*Gate Enterprise Manager.

B Log in to eISchema.

2 Export the eISchema to a file **<eGate>\client\eISchema.zip**.

A Select **Export Schema Definitions to File ...** from the **File** pull-down menu.

B In the **Select archive File** dialog box enter **eISchema.zip** in the **File name** text box, and then click **Save**.

3 Create a new schema using the eISchema export file as a template.

A Select **New Schema** from the **File** pull-down menu.

B Enter **CheckInventory** in the text box.

C Mark the **Create from export** check box.

D Click **Find** and browse for the eISchema.zip file created in step 2 above.

E Click **Open**.

15.15.2 Configure the e*Insight engine

Most of the parameter settings in the eX_eBPM engine's configuration file should not be changed. Use the e*Way Editor and the information in **"Configuring the e*Insight Engine" on page 53** to make the required changes for the CheckInventory example.

15.15.3 Create the Check_Inv activity BOB

1 In the **SystemA** e*Gate Enterprise Manager GUI, open the **ProcessOrder** schema.

2 Select the eX_Check_Inv **BOB**.

3 From the File menu select **Export Module Definitions to File**.

The **Select Archive File** dialog appears.

4 Enter eX_Check_Inv, and then click **OK**.

The file **eX_Check_Inv.zip** is created.

5 Log into the **SystemB** e*Gate Enterprise Manager GUI and open the CheckInventory schema.

6 From the File menu select **Import Module Definitions from File**.

7 Locate the file created in step 4.

8 Click **OK**.

15.16 Modify the ProcessOrder BP in e*Insight

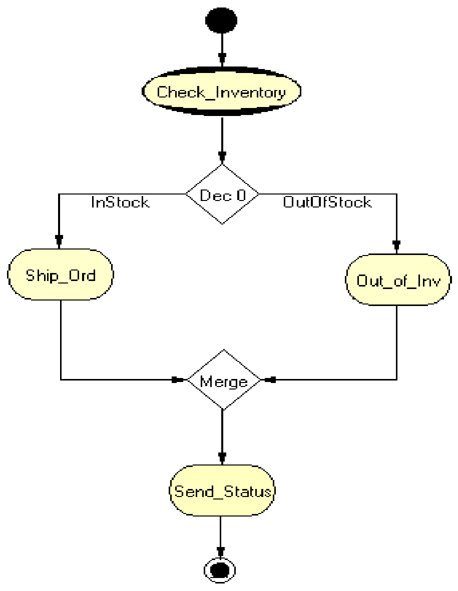
The following is a summary of the procedure for modifying the ProcessOrder BP in the e*Insight GUI.

1 Delete the Check_Inv activity.

2 Add the Check_Inventory Remote Sub-Process.

3 Make the connections between the remote sub-process, decision gate and activity as shown in Figure 135.

Figure 135 ProcessOrder BP with Remote Sub-Process



4 Configure the Partner Information.

A From the **Options** menu, select **Define Information for Partners**.

B Enter **UUID**.
This is a unique name that identifies your e*Insight database to other Trading Partners. For example, **SystemA**.

C Enter the URN.
This corresponds to the URN setup in the SOAP client. For example, **urn:ProcessOrder**.

D Enter the URL.
This is the location of the SOAP client. By default this is **http://localhost:8080/soap/servlet/rpcrouter**.

E Click **OK**.

5 Configure the **Check_Inventory** remote sub-process properties.

A Enter the relevant information for the remote e*Insight database including URL, URN, user name and password. Use the following values:

URL	http://SystemB:8080/soap/servlet/rpcrouter
URN	urn:CheckInventory
User Name	Anonymous
Password	

Note: *To use an Anonymous login you must create a user called Anonymous on the remote system. Although e*Xchange Administrator requires that a password is defined to create the user, this does not need to be supplied when connecting via SOAP.*

Click **Connect**, and select the **CheckInventory** business process from the Remote Sub-Process drop-down list

B Map Attributes as shown in Table 51.

Table 51 Sub-Process attribute mapping

Sub-Process Attributes	Business Process Attributes	Direction
Item_Number	Item_Number	Input
Order_Quantity	Order_Quantity	Input
In_Stock	In_Stock	Output
Order_Status	Order_Status	Output

6 Enable the business process version.

For more information on creating a business process and using the e*Insight GUI, see the *e*Insight Business Process Manager User's Guide*.

15.17 Configure the Integration Schema for ProcessOrder

The schema created in **"e*Insight Implementation (eISchema)" on page 133** does not need to be modified for this example.

15.18 Run and Test the e*Insight scenario

Once the schema has been set up in e*Gate you can run the scenario.

In-Stock Processing

Use the following procedure to test the functionality of the example for an item that is in stock.

To test the Remote Sub-Process scenario

1. Start Tomcat on both machines.
2. Start the control broker for ProcessOrder, and check all components are running.
3. Start the control broker for CheckInventory, and check all components are running.
4. Rename the data file.

To monitor a remote business process

1. Select the remote business process, and then select the down arrow.
2. If prompted, enter a user name and password provided by the partner.

Chapter 16

Active and Passive Modes

This chapter discusses the difference between implementing an actively controlled and passively controlled activity.

16.1 Overview

The difference between running an activity in active and passive mode is how the activity is started. In active mode e*Insight sends a message to e*Gate to start an activity. e*Insight requires a message from e*Gate to determine that the activity has completed processing. If you choose this option, you can manually repair and restart failed activities.

In passive mode, e*Insight does not send a message to start the activity, e*Gate must be configured to do this. e*Insight requires a message from e*Gate to determine that the activity has completed processing. If you choose this option, you cannot manually repair or restart failed activities.

This chapter describes a very simple business process that is first created with all activities using active mode. Then it is updated so one activity uses passive mode.

16.1.1 Case Study

The case study discussed in this chapter illustrates a simplified implementation of order processing. In this case, e*Insight receives an incoming order as a delimited text file. Once e*Insight has received the order the customer is billed and then the order is shipped. See Figure 138 for the Business Process diagram.

16.1.2 Case Study - Active Control Mode

Figure 136 shows the components involved in the business process implementation when both activities are running in an active mode. Below is a description of how the data flows between these components for an item that is shipped successfully.

Figure 136 e*Insight Data Flow Diagram

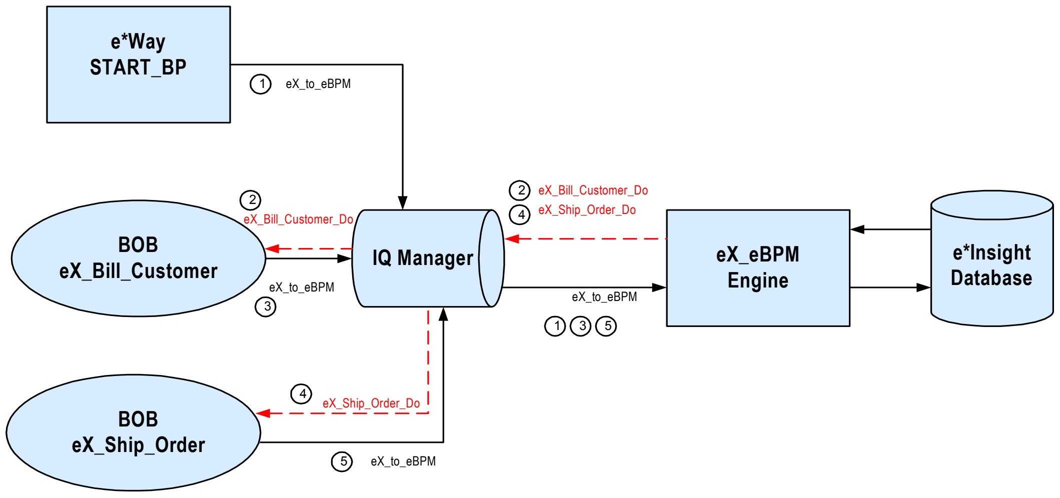


Figure 136 data flow description

① The user-defined **START_BP** e*Way picks up the text file containing the order information from a shared location on the network, uses the order information to create the event that causes the e*Insight engine to start a business process instance, and publishes it using the **eX_to_eBPM** Event Type. The e*Insight engine retrieves the Event and uses the information it contains to start the BPI.

② The e*Insight engine publishes a "Do" Event (**eX_Bill_Customer_Do**) for first activity in the business process (**Bill_Customer**). **eX_Bill_Customer** BOB, the e*Gate component that corresponds to this activity in the business process, retrieves this Event and uses the information it contains to check the availability of the items ordered.

③ When the **Bill_Customer** activity is finished, the **eX_Bill_Customer** BOB publishes a "Done" Event using the **eX_to_eBPM** Event Type.

④ The e*Insight engine publishes a "Do" Event (**eX_Ship_Order_Do**) corresponding to the **Ship_Order** activity in the business process. The **eX_Ship_Order** BOB retrieves this Event and uses the information it contains to ship the order to the customer.

⑤ When the **Ship_Order** activity is finished, the **eX_Ship_Order** BOB publishes a "Done" Event indicating that the order has been shipped.

16.1.3 Case Study - Passive Control Mode

We are now going to change the configuration of the Ship_Order activity to use passive control, that is e*Gate controls the Activity rather than e*Insight. The e*Gate configuration is modified so the Bill_Customer collaboration publishes eX_Ship_Order_Do in addition to returning a "Done" message to the e*Insight engine.

Figure 137 e*Insight Data Flow Diagram

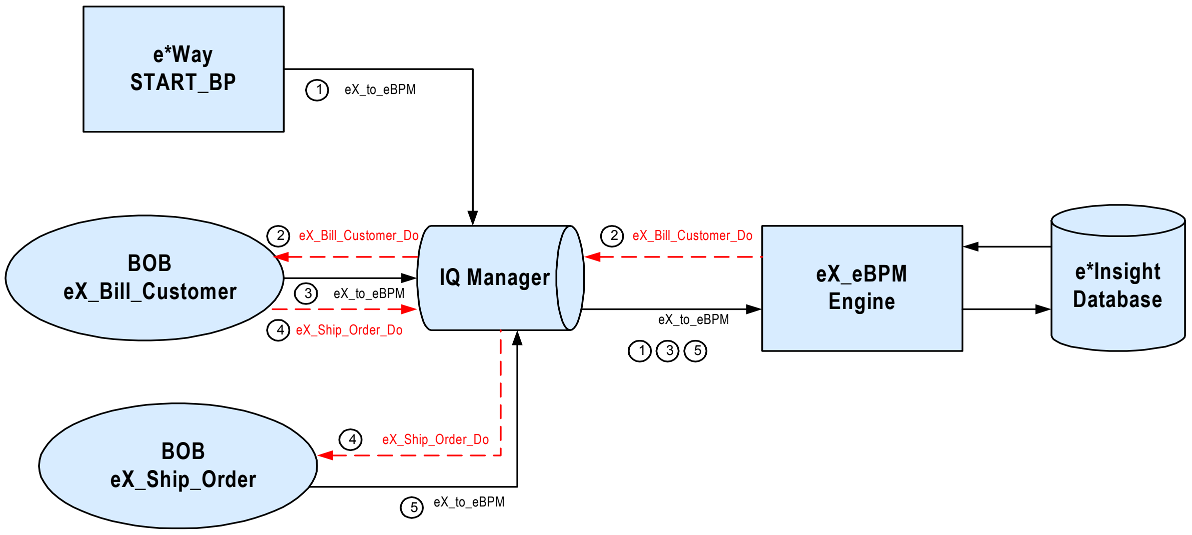


Figure 136 data flow description

① The user-defined **START_BP** e*Way picks up the text file containing the order information from a shared location on the network, uses the order information to create the event that causes the e*Insight engine to start a business process instance, and publishes it using the **eX_to_eBPM** Event Type. The e*Insight engine retrieves the Event from the IQ and uses the information it contains to start the BPI.

② The e*Insight engine publishes a "Do" Event (**eX_Bill_Customer**) for first activity in the business process (**eX_Bill_Customer_Do**). **eX_Bill_Customer** BOB, the e*Gate component that corresponds to this activity in the business process, retrieves this Event and uses the information it contains to check the availability of the items ordered.

③ When the **Bill_Customer** activity is finished, the **eX_Bill_Customer** BOB publishes a "Done" Event using the **eX_to_eBPM** Event Type.

④ The **Bill_Customer** activity also publishes a "Do" Event (**eX_Ship_Order_Do**) corresponding to the **Ship_Order** activity in the business process. The **eX_Ship_Order** BOB retrieves this Event and uses the information it contains to ship the order to the customer.

⑤ When the **Ship_Order** activity is finished, the **eX_Ship_Order** BOB publishes a "Done" Event indicating that the order has been shipped.

The major steps in the implementation are:

1. Create the business process (BP) in the e*Insight GUI.
2. Use the e*Insight GUI to configure the e*Gate schema that supports e*Insight.
3. Configure the e*Insight engine.
4. Add and configure the user-defined e*Gate components.
5. Run and test the scenario.

This chapter shows how to use these steps to set it up. Since the configuration for steps 2 to 4 is different for eIJSchema and eISchema, there is a separate section for each schema type.

Follow the steps below to configure the schema using the active mode for all activities:

- Step 1: **"Create the Order BP in e*Insight" on page 240**
- Step 2, 3, and 4:
  - **"Configure the Integration Schema (eIJSchema)" on page 241** or
  - **"Configure the Integration Schema (eISchema)" on page 250**
- Step 5: **"Run and Test the e*Insight scenario" on page 255**

Once the schema is working in active mode, then follow the steps below to modify one activity to use passive mode using eIJSchema:

A **"Modify the Order BP in e*Insight (eIJSchema)" on page 256**

B **"Modify User-defined e*Gate Components (eIJSchema)" on page 256**

C **"Run and Test the e*Insight scenario" on page 259**

Use the following steps if you are using eISchema:

A **"Modify the Order BP in e*Insight (eISchema)" on page 260**

B **"Modify User-defined e*Gate Components (eISchema)" on page 260**

C **"Run and Test the e*Insight scenario" on page 261**

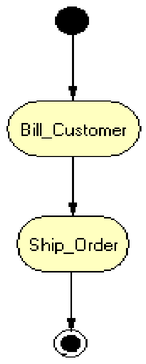
16.2 Create the Order BP in e*Insight

The following is a summary of the procedure for creating a BP in the e*Insight GUI.

1. Add the Activities.
2. Make the connections between the Activities.
3. Add all the global attributes.
4. Assign global attributes to Activities.
5. Configure the properties for the activities.

For more information on creating a business process and using the e*Insight GUI, see the *e*Insight Business Process Manager User's Guide*.

Figure 138 Order Process



The case study is implemented using both active and passive control for the Ship_Order Activity.

The following Attributes should be configured for both the active and passive examples.

Table 52 BP Global Attributes

Attribute	Type	Data Direction	Default Value
Customer_Name	String	Input	
Item_Number	String	Input	
Order_Quantity	String	Input	
Order_Status	String	Internal	Received

Table 53 Activity Attributes

Activity	Attribute(s)	Input/Output
Bill_Customer	Customer_Name	Input
	Order_Status	Output

Table 53 Activity Attributes

Activity	Attribute(s)	Input/Output
Ship_Order	Cust_Name	Input
	Item_Number	Input
	Order_Status	Output

16.3 Configure the Integration Schema (eIJSchema)

All the activities in this example are carried out using e*Gate components. You must first create a Schema (a copy of eIJSchema) with the basic components required for e*Insight. You then configure these components for your environment and create additional components for the activities.

To create a copy of eIJSchema

1. From the e*Insight GUI **File** menu, select **New e*Gate Schema**.
2. Enter or select a **Registry Host** on which to create the schema.
3. Enter a **Username** and **Password** that is valid on the Registry Host.
4. From the **Based on** list, select **eIJSchema (Java)**.
5. In the **Name** box, enter **Order**.
6. Click **OK**.

Integration Schema Activity Components Summary

Use the information in Table 57 to configure the e*Gate schema that supports the example.

Table 54 Integration Schema Activity Components

Name	Type	Participating Host	Active/Passive	Manual Restart	TimeOut
eX_Bill_Customer	BOB	localhost	Active	Yes	Not used
eX_Ship_Order	BOB	localhost	Active	Yes	Not used

For information on how to use the e*Insight GUI to configure the e*Gate Registry see the *e*Insight Business Process Manger User's Guide*.

Creating the eX_Bill_Customer BOB

The Bill_Customer translation implements the logic associated with billing the customer. In this simple example, this component sets the “Order_Status” attribute value.

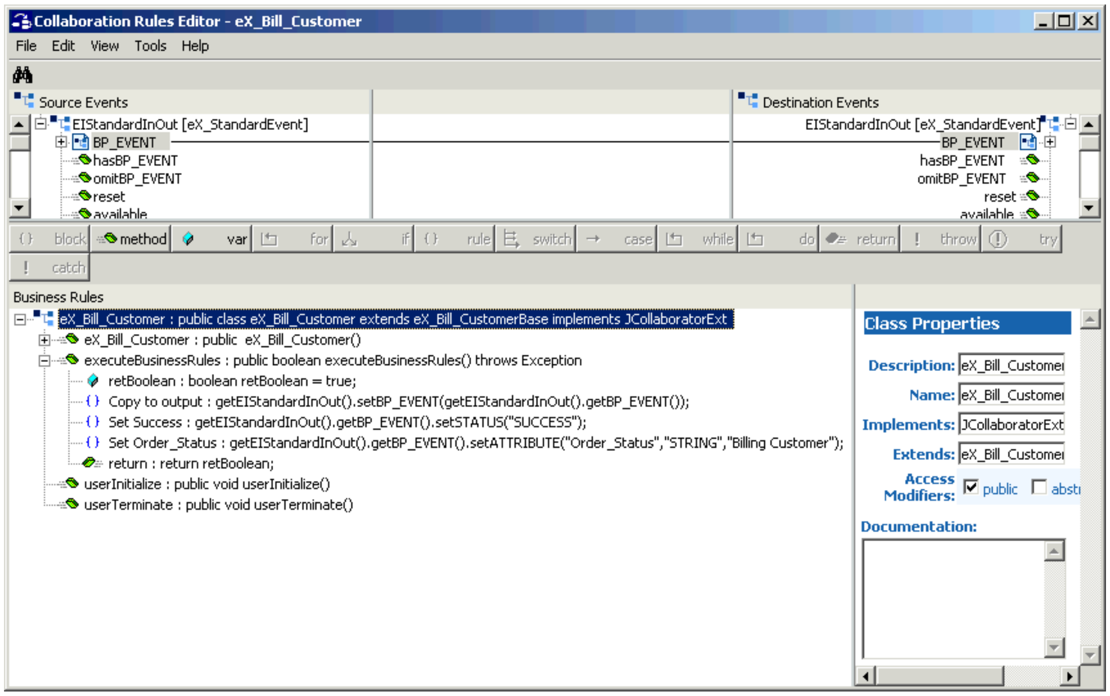
To configure the Bill_Customer activity

1. In the e*Insight GUI, open the **Bill_Customer** activity properties.
2. On the **General** tab, **e*Gate Module** section, select a **Module Type** of **BOB**.
3. Click **New**.

   The **Define Collaboration** dialog appears.
4. Click **OK**.
5. Create eX_Bill_Customer.xpr.

   Figure 139 shows the eX_Bill_Customer CRS used in the example.

Figure 139 eX_Bill_Customer.xpr CRS



```
eX_Bill_Customer : public class eX_Bill_Customer extends eX_Bill_CustomerBase implements JCollaboratorExt
    eX_Bill_Customer : public  eX_Bill_Customer()
    executeBusinessRules : public boolean executeBusinessRules() throws Exception
        retBoolean : boolean retBoolean = true;
        Copy to output : getEIStandardInOut().setBP_EVENT(getEIStandardInOut().getBP_EVENT());
        Set Success : getEIStandardInOut().getBP_EVENT().setSTATUS("SUCCESS");
        Set Order_Status : getEIStandardInOut().getBP_EVENT().setATTRIBUTE("Order_Status","STRING","Billing Customer");
        return : return retBoolean;
    userInitialize : public void userInitialize()
    userTerminate : public void userTerminate()
```

Description: eX_Bill_Customer
Name: eX_Bill_Customer
Implements: JCollaboratorExt
Extends: eX_Bill_Customer
Access Modifiers: [✓ public] abstr

6. Compile and save the CRS.
7. Close the editor.
8. In the Bill_Customer Activity properties, click **Configure e*Gate Schema**.

   You may be required to log into e*Gate.
9. Click **OK**, to close the information dialog.
10. Close the Bill_Customer Activity properties.

Creating the eX_Ship_Order BOB

The Ship_Order translation implements the logic associated with shipping to the customer. In this simple example, this component sets the “Order_Status” attribute value.

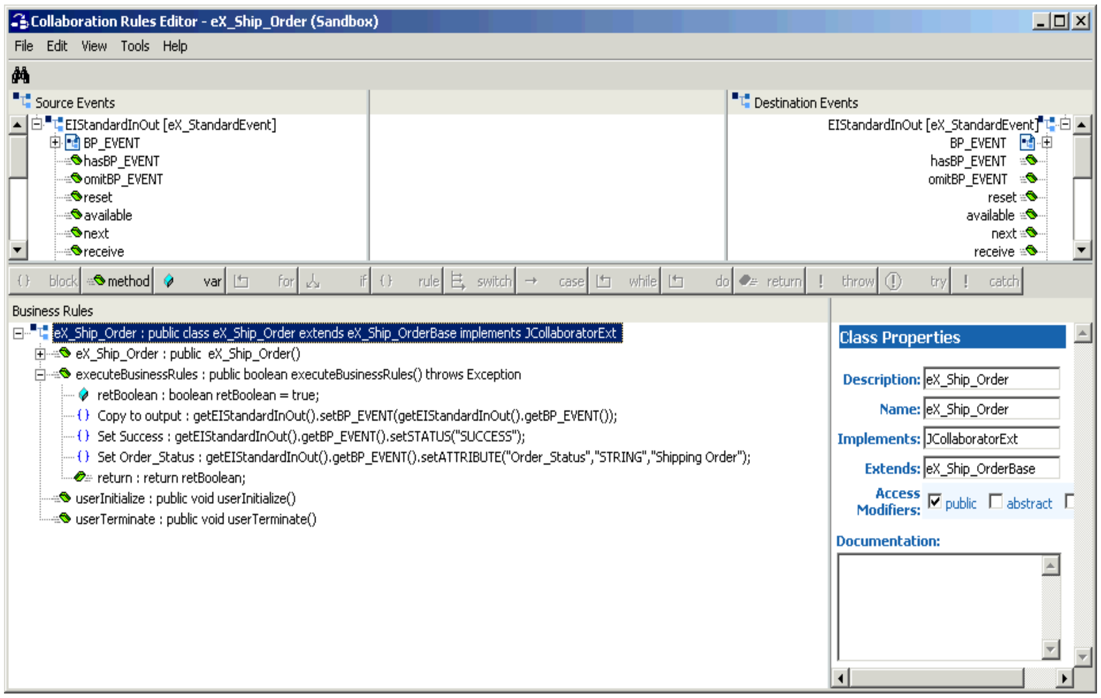
To configure the Ship_Order activity

1. In the e*Insight GUI, open the **Ship_Order** activity properties.
2. On the **General** tab, **e*Gate Module** section, select a **Module Type** of **BOB**.
3. Click **New**.

   The **Define Collaboration** dialog appears.
4. Click **OK**.
5. Create eX_Ship_Order.xpr.

   Figure 139 shows the eX_Ship_Order CRS used in the example.

Figure 140 eX_Ship_Order.xpr CRS



```
eX_Ship_Order : public class eX_Ship_Order extends eX_Ship_OrderBase implements JCollaboratorExt
    eX_Ship_Order : public  eX_Ship_Order()
    executeBusinessRules : public boolean executeBusinessRules() throws Exception
        retBoolean : boolean retBoolean = true;
        Copy to output : getEIStandardInOut().setBP_EVENT(getEIStandardInOut().getBP_EVENT());
        Set Success : getEIStandardInOut().getBP_EVENT().setSTATUS("SUCCESS");
        Set Order_Status : getEIStandardInOut().getBP_EVENT().setATTRIBUTE("Order_Status","STRING","Shipping Order");
        return : return retBoolean;
    userInitialize : public void userInitialize()
    userTerminate : public void userTerminate()
```

Description: eX_Ship_Order
Name: eX_Ship_Order
Implements: JCollaboratorExt
Extends: eX_Ship_OrderBase
Access Modifiers: [✓ public] abstract

6. Compile and save the CRS.
7. Close the editor.
8. In the Ship_Order Activity properties, click **Configure e*Gate Schema**.

   You may be required to log into e*Gate.
9. Click **OK**, to close the information dialog.
10. Close the Ship_Order Activity properties.

16.4 Configure the e*Insight Engine (eIJSchema)

The e*Insight engine runs in a specially configured Multi-Mode e*Way. You must make changes to the configuration file for this e*Way to conform to the requirements of your system. For example, you must specify the name of the e*Insight database to which the e*Way connects.

Note: *This example uses only one e*Insight engine. In an actual implementation, more than one e*Insight engine can be configured to handle the required workload. In such a case, you must make changes to each of the e*Insight engines.*

Edit the eIcp_eInsightEngine Connection Configuration File

Most of the parameter settings in the eIcp_eInsightEngine connection's configuration file should not be changed. **"Configuring the e*Insight Engine Connection" on page 34** discusses the parameters that may need to be changed depending on the implementation. Use the e*Way Editor and the information in **"Configuring the e*Insight Engine Connection" on page 34** to make the required changes for the example.

16.4.1 Configure the JMS Connection

The JMS connection for e*Insight must be configured for your system. The minimal configuration required for this implementation is described in this section. For more information on JMS IQ Services, see *SeeBeyond JMS Intelligent Queue User's Guide*.

To configure the JMS connection

1. From the e*Gate Enterprise Manager GUI components view, select the **e*Way Connections** folder.
2. Select the **eX_cpeInsightJMS** connection, and click the **Properties** tool.
3. In the **e*Way Connection Configuration File** section, click **Edit**.
4. From the **Goto Section** list, select **Message Service**.
5. Enter a **Server Name** and **Host Name** where your JMS server resides.

16.5 Configure User-defined e*Gate Components (eIJSchema)

The user-defined components in an e*Insight implementation consist of two types: the first type *starts* the business process, and second type runs *as part of* the business process. The activity components are of the second type.

The Order example uses a file e*Way to start the business process and BOBs to run all the other activities.

Configuration Order for the User-defined Components

1 Add and configure the **START_BP** e*Way.

2 Configure the Collaborations for the activity components running as BOBs.

Important: *All the integration schema associations are displayed in table format at the end of this section. The sections dealing with e*Way configuration include tables detailing the non-default e*Way parameter settings. The sections dealing with the Monk Collaboration Rules Scripts show screen shots of these scripts as they appear in the e*Gate Collaboration Editor.*

16.5.1 Configure the START_BP e*Way

The e*Way that sends the Event that starts the business process, named **START_BP** in this example, must convert the incoming data into e*Insight Event format, as well as send the appropriate acknowledgment to the e*Insight engine to create the Business Process Instance (BPI).

The **START_BP** e*Way is completely user defined and must be added to the **eIJSchema** in the e*Gate Enterprise Manager. In an actual implementation, the choice of e*Way (or BOB) would depend on the requirements of the situation. For example, if the data were coming from an SAP system, you might select an SAP ALE e*Way; or if the data were already in the e*Gate system, you could use a BOB to start the BPI. In the present case, a text file on the local system provides the input data, therefore the example uses a file e*Way to send the “Start” Event to the e*Insight engine.

Table 55 shows the steps to configure the **START_BP** e*Way.

Table 55 Configuration steps for the START_BP e*Way

	Step	Section
1	Add the e*Way and create the e*Way configuration file	**“Step 1: Create the START_BP e*Way” on page 245**
2	Create the Input ETD	**“Step 2: Create the Input ETD” on page 246**
3	Create the START_BP Collaboration Rules script (CRS)	**“Step 3: Create the START_BP Collaboration” on page 247**
4	Configure the Collaboration in the GUI	**“Step 4: Configure the Collaboration in the GUI” on page 249**

Step 1: Create the START_BP e*Way

The e*Way for the Payroll example is a simple file e*Way (executable: **stcewfile.exe**) that polls a directory (**<eGate>\client\data\Order**) for any file with the extension “.fin” and moves it into the e*Insight system.

Use the Enterprise Manager and the following table to add the **START_BP** e*Way and create its configuration file.

Table 56 Start e*Way Parameters

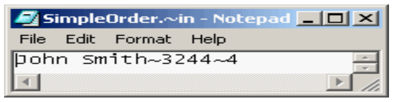
Screen	Parameter	Setting
General Settings	(All)	(Default)
Outbound (send) settings	(All)	(Default)
Poller (inbound) settings	PollDirectory	<eGate>\client\data\Order
	(All others)	(Default)
Performance Testing	(All)	(Default)

Step 2: Create the Input ETD

The input ETD is based on the format of the input data. The Order example uses a delimited text file (**SimpleOrder.~in**) that contains the data needed to process the order.

The input data file used in this example is shown in Figure 141. Place this data file at the directory location **c:\eGate\client\data\Order**.

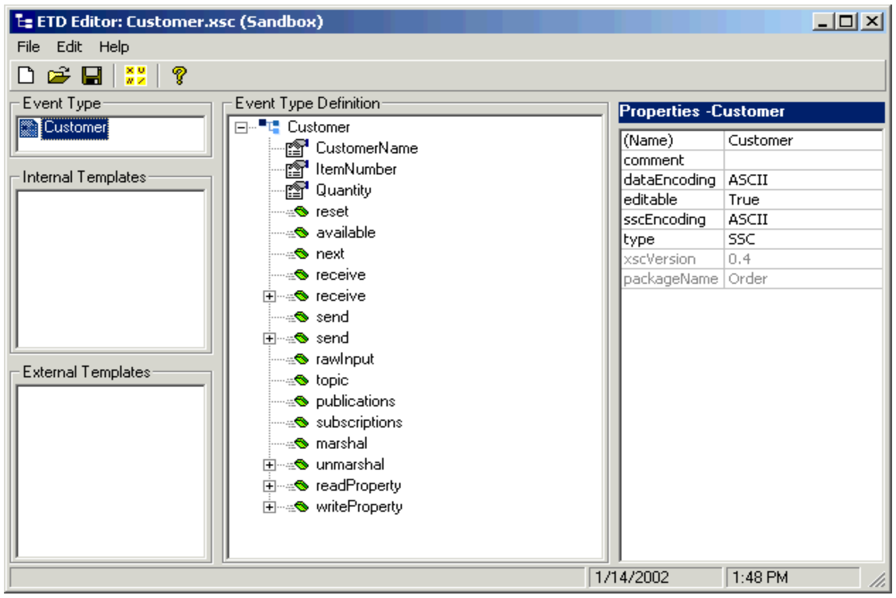
Figure 141 Input Text File (SimpleOrder.~in)



John Smith~3244~4

Using the ETD Editor and the input data as a guide, create an ETD like the one shown in Figure 147. Set the global delimiter to a ~ character. For more information on using the ETD Editor see the ETD Editor's online help.

Figure 142 Input ETD: Customer.xsc (Java)



(Name)	Customer
comment	
dataEncoding	ASCII
editable	True
sscEncoding	ASCII
type	SSC
xscVersion	0.4
packageName	Order

Step 3: Create the START_BP Collaboration

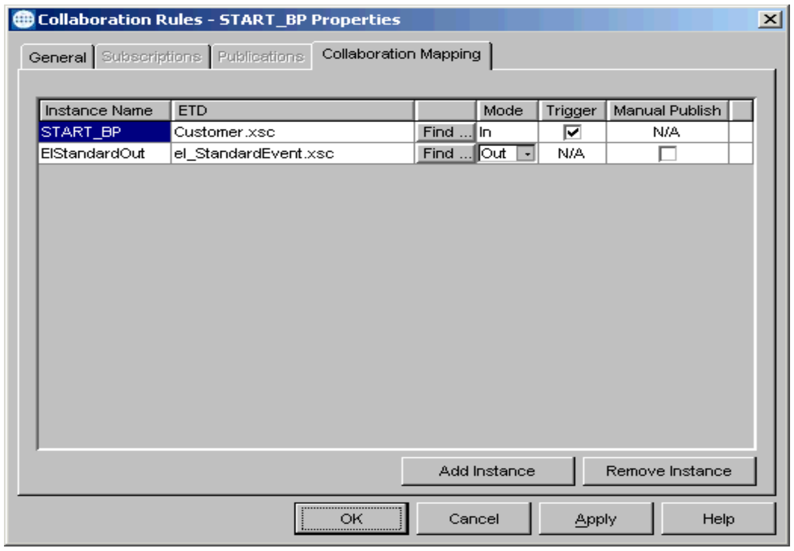
The Collaboration that sends the Event that starts the BPI must do two things:

- Put the data into e*Insight ETD (**eI_StandardEvent.xsc)** format.
- Populate the Event with the information the e*Insight engine needs to start a BPI.

In addition to these two tasks, the **START_BP** Collaboration also provides the recommended location for setting any global attributes that are required in your business process.

1. Create a Collaboration Rule, **START_BP**, that uses the Java service.
2. Configure the Collaboration Mapping tab, as shown in Figure 143.

Figure 143 Start_BP Properties, Collaboration Mapping Tab

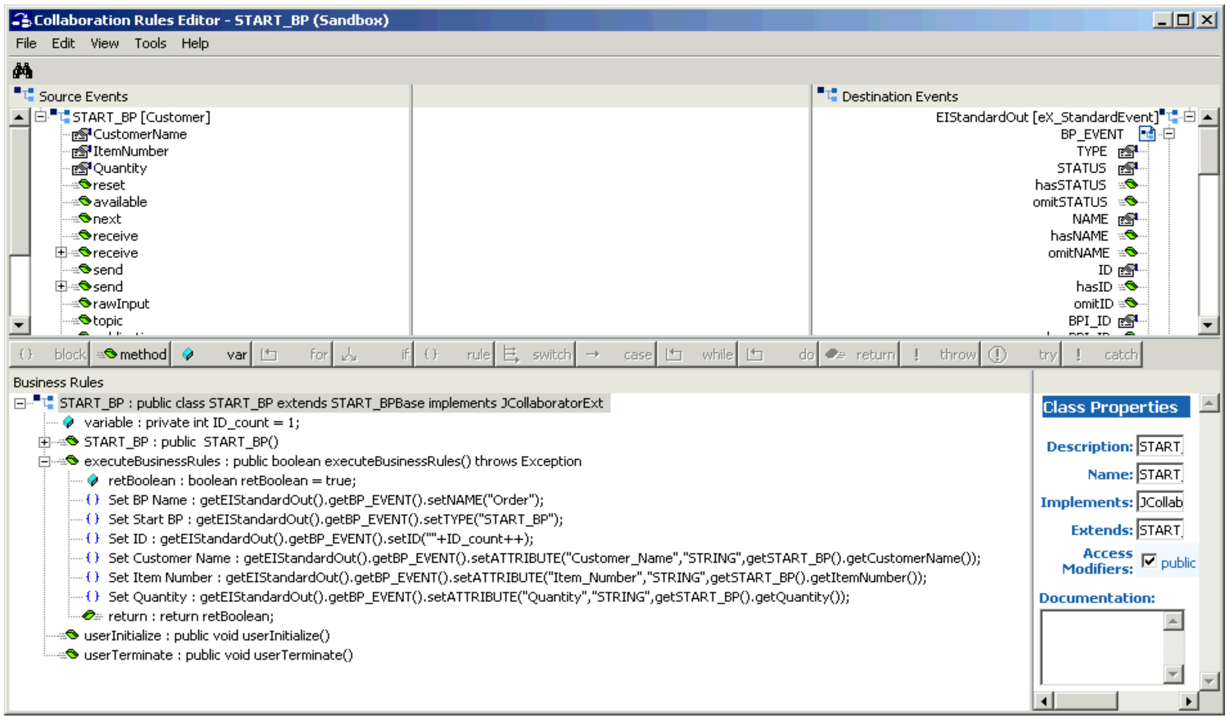


Instance Name	ETD		Mode	Trigger	Manual Publish
START_BP	Customer.xsc	Find ...	In	☑	N/A
EIStandardOut	el_StandardEvent.xsc	Find ...	Out	N/A	☐

3 Click **Apply**, and click the **General Tab**.

4 Click **New** to create a new CRS, as show in Figure 144.

Figure 144 START_BP CRS



Business Rules

```
START_BP : public class START_BP extends START_BPBase implements JCollaboratorExt
    variable : private int ID_count = 1;
    START_BP : public  START_BP()
    executeBusinessRules : public boolean executeBusinessRules() throws Exception
        retBoolean : boolean retBoolean = true;
        Set BP Name : getEIStandardOut().getBP_EVENT().setNAME("Order");
        Set Start BP : getEIStandardOut().getBP_EVENT().setTYPE("START_BP");
        Set ID : getEIStandardOut().getBP_EVENT().setID(""+ID_count++);
        Set Customer Name : getEIStandardOut().getBP_EVENT().setATTRIBUTE("Customer_Name","STRING",getSTART_BP().getCustomerName());
        Set Item Number : getEIStandardOut().getBP_EVENT().setATTRIBUTE("Item_Number","STRING",getSTART_BP().getItemNumber());
        Set Quantity : getEIStandardOut().getBP_EVENT().setATTRIBUTE("Quantity","STRING",getSTART_BP().getQuantity());
        return : return retBoolean;
    userInitialize : public void userInitialize()
    userTerminate : public void userTerminate()
```

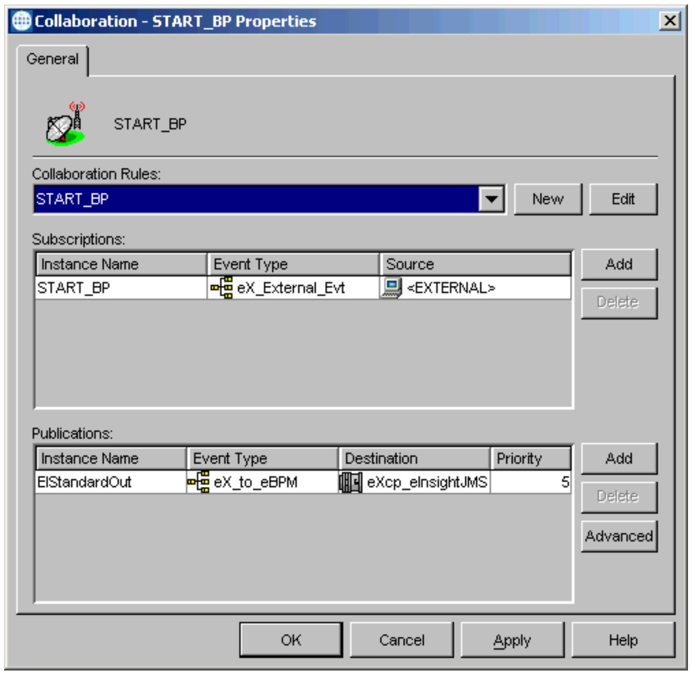
Description:	START
Name:	START
Implements:	JCollab
Extends:	START
Access Modifiers:	☑ public

Step 4: Configure the Collaboration in the GUI

In addition to creating the configuration file for the e*Way and the CRS used by the Collaboration, you must also configure the **Start_BP** e*Way's Collaboration in the Enterprise Manager GUI.

1 Create a Collaboration for the **Start_BP** e*Way configured as shown in Figure 145.

Figure 145 Start_BP Collaboration



Collaboration - START_BP Properties

General

START_BP

Collaboration Rules: START_BP

Subscriptions:

Instance Name	Event Type	Source
START_BP	eX_External_Evt	<EXTERNAL>

Publications:

Instance Name	Event Type	Destination	Priority
EIStandardOut	eX_to_eBPM	eXcp_eInsightJMS	5

16.6 Configure the Integration Schema (eISchema)

All the activities in this example are carried out using e*Gate components.

Important: *Before you begin to make changes to the e*Gate Registry, make a copy of the e*Insight schema. See* **"Copy the e*Insight Schema" on page 85** *for instructions on how to do this.*

After creating the business process, you must configure the e*Gate Registry schema that supports the e*Insight system.

e*Insight allows you to specify the type of component (e*Way or BOB) associated with a particular activity and where it runs.

Integration Schema Activity Components Summary

Use the information in Table 57 to configure the e*Gate schema that supports the example.

Table 57 Integration Schema Activity Components

Name	Type	Participating Host	Active/Passive	Manual Restart	TimeOut
eX_Bill_Customer	BOB	localhost	Active	Yes	Not used
eX_Ship_Order	BOB	localhost	Active	Yes	Not used

For information on how to use the e*Insight GUI to configure the e*Gate Registry see the *e*Insight Business Process Manger User's Guide*.

16.7 Configure the e*Insight Engine (eISchema)

The e*Insight engine runs in a specially configured Java e*Way. You must make changes to the configuration file for this e*Way to conform to the requirements of your system. For example, you must specify the name of the e*Insight database to which the e*Way connects.

Note: *This example uses only one e*Insight engine. In an actual implementation, more than one e*Insight engine can be configured to handle the required workload. In such a case, you must make changes to each of the e*Insight engines.*

Edit the eX_eBPM Engine's Configuration File

Most of the parameter settings in the eX_eBPM engine's configuration file should not be changed. Use the e*Way Editor and the information in **"Configuring the e*Insight Engine" on page 53** to make the required changes for the Order example.

16.8 Configure User-defined e*Gate Components (eISchema)

The user-defined components in an e*Insight implementation consist of two types: the first type *starts* the business process, and second type runs *as part of* the business process. The activity components are of the second type.

The Order example uses a file e*Way to start the business process and BOBs to run all the activities except the last. The last activity is represented by an additional file e*Way.

Configuration Order for the User-defined Components

1. Add and configure the **START_BP** e*Way.
2. Configure the Collaborations for the activity components running as BOBs.

Important: *All the integration schema associations are displayed in table format at the end of this section. The sections dealing with e*Way configuration include tables detailing the non-default e*Way parameter settings. The sections dealing with the Monk Collaboration Rules Scripts show screen shots of these scripts as they appear in the e*Gate Collaboration Editor.*

16.8.1 Configure the START_BP e*Way

The e*Way that sends the Event that starts the business process, named **START_BP** in this example, must convert the incoming data into e*Insight Event format, as well as send the appropriate acknowledgment to the e*Insight engine to create the Business Process Instance (BPI).

Follow these steps to configure the **START_BP** e*Way:

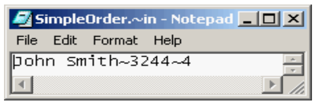
1. Create the Input ETD
2. Create the **START_BP** Collaboration Rules script (CRS)
3. Add the e*Way and create the e*Way configuration file
4. Configure the Collaboration in the GUI

Step 1: Create the Input ETD

The input ETD is based on the format of the input data. The Order example uses a delimited text file (**SimpleOrder.~in**) that contains the data needed to process the order.

The input data file used in this example is shown in Figure 146. Place this data file at the directory location **c:\eGate\client\data\Order**.

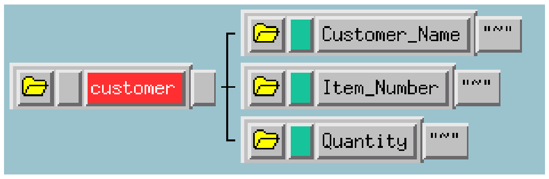
Figure 146 Input Text File (SimpleOrder.~in)



```
john smith~3244~4
```

Using the ETD Editor and the input data as a guide, create an ETD like the one shown in Figure 147. For more information on using the ETD Editor see the ETD Editor's online help.

Figure 147 Input ETD: Customer.ssc



Step 2: Create the START_BP Collaboration Rules Script (CRS)

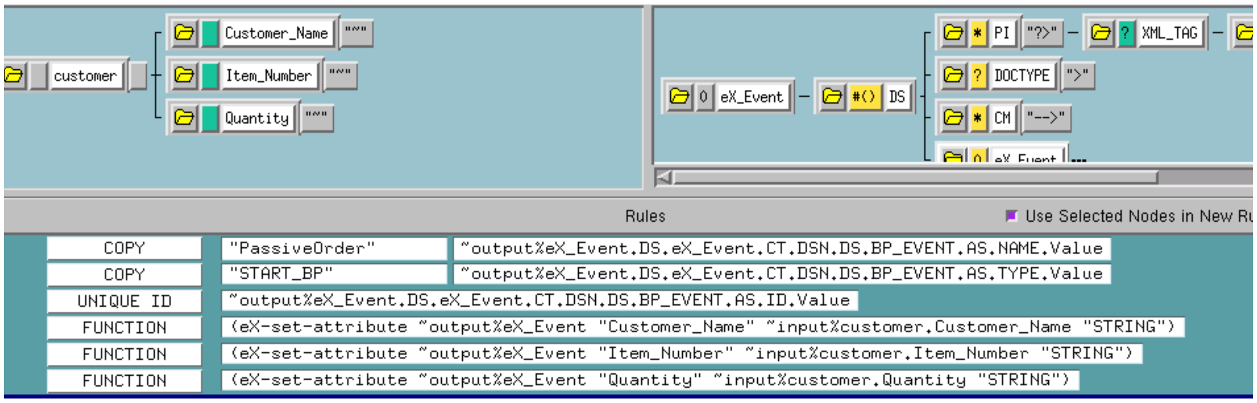
The Collaboration that sends the Event that starts the BPI must do two things:

- Put the data into e*Insight ETD (**eX_Standard_Event.ssc)** format.
- Populate the Event with the information the e*Insight engine needs to start a BPI.

In addition to these two tasks, the **START_BP** Collaboration also provides the recommended location for setting any global attributes that are required in your business process.

Figure 148 shows the **START_BP** CRS used in the Order example:

Figure 148 START_BP CRS



```
COPY        "PassiveOrder"  ~output%eX_Event.DS.eX_Event.CT.DSN.DS.BP_EVENT.AS.NAME.Value
COPY        "START_BP"      ~output%eX_Event.DS.eX_Event.CT.DSN.DS.BP_EVENT.AS.TYPE.Value
UNIQUE ID   ~output%eX_Event.DS.eX_Event.CT.DSN.DS.BP_EVENT.AS.ID.Value
FUNCTION    (eX-set-attribute ~output%eX_Event "Customer_Name" ~input%customer.Customer_Name "STRING")
FUNCTION    (eX-set-attribute ~output%eX_Event "Item_Number" ~input%customer.Item_Number "STRING")
FUNCTION    (eX-set-attribute ~output%eX_Event "Quantity" ~input%customer.Quantity "STRING")
```

Step 3: Add the e*Way and Create the e*Way Configuration File

The e*Way for the Order example is a simple file e*Way (executable: **stcewfile.exe**) that polls a directory (**c:\eGate\client\data\Order**) for any file with the extension ".fin" and moves it into the e*Insight system.

Use the Enterprise Manager and the following table to add the **START_BP** e*Way and create its configuration file.

Table 58 Start e*Way Parameters

Screen	Parameter	Setting
General Settings	(All)	(Default)
Outbound (send) settings	(All)	(Default)
Poller (inbound) settings	PollDirectory	c:\eGate\client\data\Order
	(All others)	(Default)
Performance Testing	(All)	(Default)

Step 4: Configure the Collaboration in the GUI

In addition to creating the configuration file for the e*Way and the CRS used by the Collaboration, you must also configure the **START_BP** e*Way's Collaboration in the Enterprise Manager GUI.

1 Create a Collaboration Rule, **START_BP**, that uses the Monk service and the START_BP CRS created in step 2, subscribes to the **eX_External_Evt** Event Type, and publishes to the **eX_to_eBPM** Event Type.

2 Create a Collaboration for the **START_BP** e*Way that uses the **START_BP** Collaboration Rule, subscribes to the **eX_External_Evt** Event Type from **<EXTERNAL>**, and publishes the **eX_to_eBPM** Event Type to the **eX_eBPM** IQ.

16.8.2 Configure the Activity BOBs

You must complete these two tasks in order to configure the Activity BOBs:

- Set up the activity Collaborations in the e*Gate GUI.
- Create the corresponding CRS used by the activity Collaborations.

Note: *Any time you create e*Ways, you must configure those e*Ways before e*Insight can communicate with external systems or components.*

Create the Activity BOB CRSs

Creating the CRSs associated with the activity components is the responsibility of the implementor. What these programs do varies depending on the type of component and the business logic they must implement. Nevertheless, they all must take the information provided to them by the e*Insight engine, process it, and return the appropriate response to the e*Insight engine. This includes setting the values of any output attributes and the BP_EVENT status node to "SUCCESS" or "FAILURE" depending on whether the activity completes successfully or not.

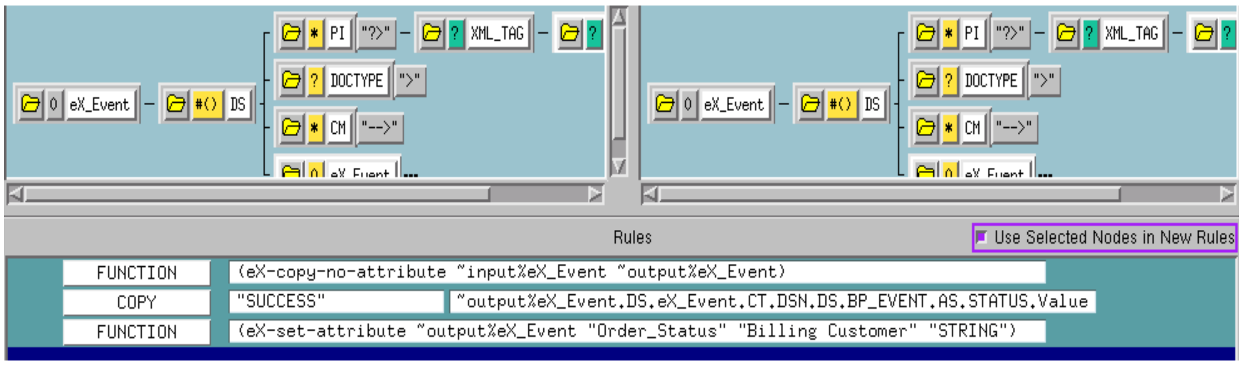
The CRSs associated with the BOBs used in this example are described in the following sections along with a screen capture showing the actual Monk Collaboration Rules Script used.

Bill Customer CRS

The CRS sets the value of the Order_Status attribute to "Billing Customer" and sends a "SUCCESS" Event back to e*Insight, indicating that the activity has completed successfully.

Figure 149, shows the **Billing_Customer.tsc** CRS used in the example.

Figure 149 Billing_Customer.tsc CRS



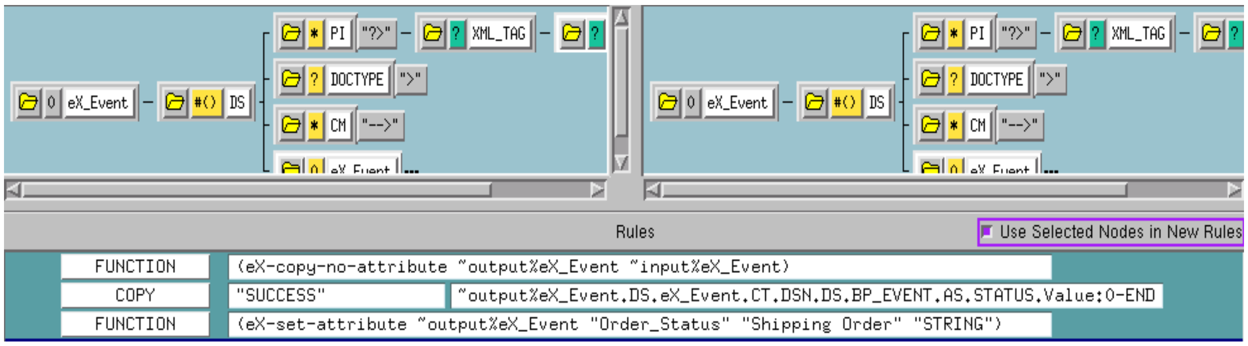
FUNCTION	(eX-copy-no-attribute ~input%eX_Event ~output%eX_Event)
COPY	"SUCCESS" → ~output%eX_Event.DS.eX_Event.CT.DSN.DS.BP_EVENT.AS.STATUS.Value
FUNCTION	(eX-set-attribute ~output%eX_Event "Order_Status" "Billing Customer" "STRING")

Ship_Order CRS

The Ship_Order translation simulates the activity of sending out an item that is in stock. It sets the value of the Order_Status attribute to a short message indicating that the order has been sent and returns "SUCCESS" to the e*Insight engine.

Figure 150 shows the Ship_Order CRS used in the example.

Figure 150 Ship_Order.tsc CRS



FUNCTION	(eX-copy-no-attribute ~output%eX_Event ~input%eX_Event)
COPY	"SUCCESS" → ~output%eX_Event.DS.eX_Event.CT.DSN.DS.BP_EVENT.AS.STATUS.Value:0-END
FUNCTION	(eX-set-attribute ~output%eX_Event "Order_Status" "Shipping Order" "STRING")

Configure the Activity BOB Collaborations in the Enterprise Manager GUI

Once you have created the CRS for a BOB, you must associate it with the corresponding Collaboration Rule in the e*Gate GUI. For each BOB you must:

1. Highlight the BOB's Collaboration.
2. Open the **Collaboration Properties** dialog box for the Collaboration.
3. Edit the Collaboration Rules.

4 Change the Service to **Monk**.

5 Find the corresponding ".tsc" file and associate it with the Collaboration Rule.

6 Click **OK** to continue.

16.9 Run and Test the e*Insight scenario

Once the schema has been set up in e*Gate you can run the scenario. Use the following procedure to test the functionality of the example.

1 Start the e*Insight GUI and select the Order business process. Switch to monitor mode.

Note: *Make sure that the business process has been enabled in the e*Insight GUI before attempting to run it.*

2 Make a final check of the e*Gate schema, using the tables to confirm all of the GUI associations. Make sure that all of the e*Insight components, including the user-defined components, are set to start automatically.

3 At the command line, type the following to start the schema. You must type the command on a single line.

```
stccb.exe -rh localhost -rs Order -ln localhost_cb
     -un username -up password
```

Substitute the appropriate username and password for your installation.

4 Start the e*Gate Monitor, and check the status of all the components. Any components displayed in red are not running. Any e*Insight components that are not running should be investigated before feeding data into the system.

5 Navigate to the location for the input data file, **SimpleOrder.~in**, shown in **Figure 146 on page 252** (**c:\eGate\client\data\Order**) and change the extension to "**.fin**".

Note: *The change of the extension to "**.~in**" indicates that the data file has been picked up by the **START_BP** e*Way.*

6 Switch to the e*Insight GUI and, while in monitor mode, select the most recent BPI from the **List** pane. The **Diagram** should show the activities as completed (green). If the activities are not green then the e*Gate component associated with that activity should be investigated for the cause of the problem.

16.10 Case Study - Passive Control Mode

We are now going to change the configuration of the Ship_Order activity to use passive control, that is e*Gate controls the Activity rather than e*Insight. The e*Gate configuration is modified so the Bill_Customer collaboration publishes eX_Ship_Order_Do in addition to returning a "Done" message to the e*Insight engine. The Ship_Order CRS is also modified to ensure that the "Done" message contains the correct information.

16.11 Passive Control Mode (eIJSchema)

16.11.1 Modify the Order BP in e*Insight (eIJSchema)

The following is a summary of the procedure for modifying the BP in the e*Insight GUI.

1. Save the business process as a new version.
2. Change the Ship_Order activity properties to use passive control.

16.11.2 Modify User-defined e*Gate Components (eIJSchema)

Configuration Order for the User-defined Components

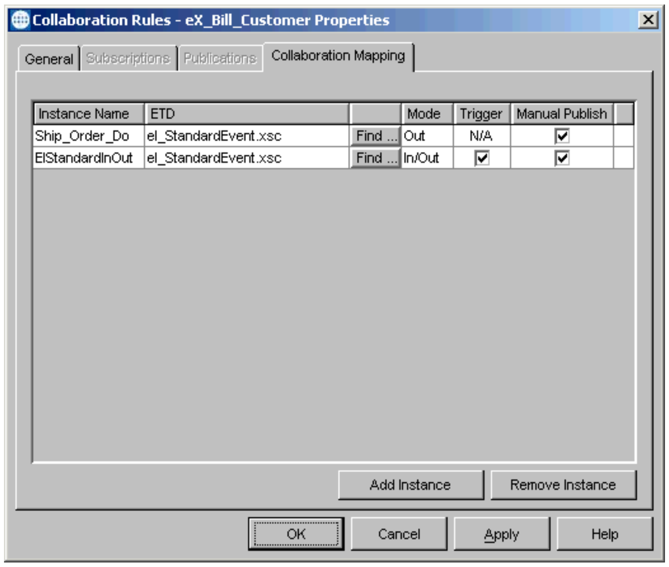
1. Modify the Bill_Customer CRS.
2. Modify the Ship_Order CRS.
3. Modify the Bill_Customer Collaboration Rule and Collaboration.

Configure the Bill_Customer Collaboration Rule and Collaboration

1. Update the Bill_Customer Collaboration Rule with the following Collaboration Mapping.

Note: *Both instances are now configured to publish the event manually.*

Figure 151 eX_Bill_Customer Collaboration Mapping



Instance Name	ETD		Mode	Trigger	Manual Publish
Ship_Order_Do	eI_StandardEvent.xsc	Find ...	Out	N/A	✓
EIStandardInOut	eI_StandardEvent.xsc	Find ...	In/Out	✓	✓

2 Update the Bill_Customer Collaboration to additionally publish a **Ship_Order_Do** instance, Event Type **eX_Ship_Order_Do**, to the destination **eIcr_eInsightJMS**.

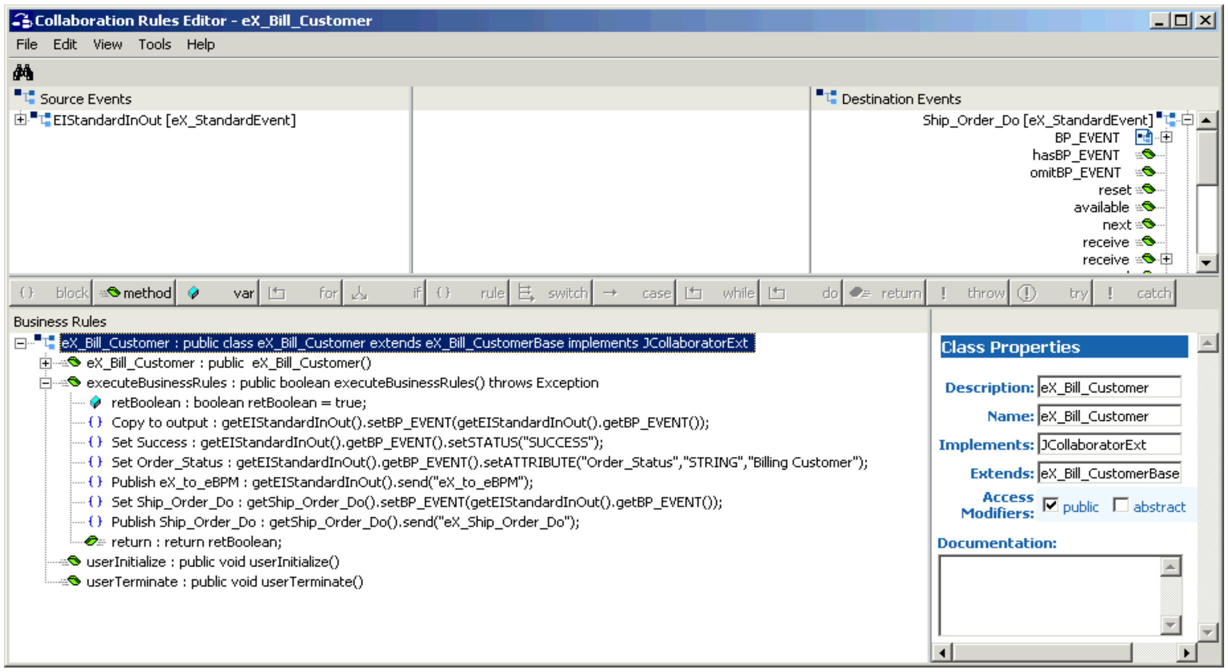
Bill Customer CRS (eIJSchema)

The CRS is already defined to set the value of the Order_Status attribute to “Billing Customer” and send a “SUCCESS” Event back to the e*Insight, indicating that the activity has completed successfully.

The script is modified to manually publish eX_to_eBPM, and to additionally publish **eX_Ship_Order_Do**.

Figure 152, shows the **Billing_Customer.xpr** CRS used in the example.

Figure 152 Bill_Customer.xpr CRS


Collaboration Rules Editor - eX_Bill_Customer
File Edit View Tools Help
Source Events
EIStandardInOut [eX_StandardEvent]
Destination Events
Ship_Order_Do [eX_StandardEvent]
BP_EVENT
hasBP_EVENT
omitBP_EVENT
reset
available
next
receive
receive
block method var for if rule switch case while do return throw try catch
Business Rules
Class Properties

```
eX_Bill_Customer : public class eX_Bill_Customer extends eX_Bill_CustomerBase implements JCollaboratorExt
    eX_Bill_Customer : public  eX_Bill_Customer()
    executeBusinessRules : public boolean executeBusinessRules() throws Exception
        retBoolean : boolean retBoolean = true;
        Copy to output : getEIStandardInOut().setBP_EVENT(getEIStandardInOut().getBP_EVENT());
        Set Success : getEIStandardInOut().getBP_EVENT().setSTATUS("SUCCESS");
        Set Order_Status : getEIStandardInOut().getBP_EVENT().setATTRIBUTE("Order_Status","STRING","Billing Customer");
        Publish eX_to_eBPM : getEIStandardInOut().send("eX_to_eBPM");
        Set Ship_Order_Do : getShip_Order_Do().setBP_EVENT(getEIStandardInOut().getBP_EVENT());
        Publish Ship_Order_Do : getShip_Order_Do().send("eX_Ship_Order_Do");
        return : return retBoolean;
    userInitialize : public void userInitialize()
    userTerminate : public void userTerminate()
```

Description: eX_Bill_Customer
Name: eX_Bill_Customer
Implements: JCollaboratorExt
Extends: eX_Bill_CustomerBase
Access Modifiers: [✓ public] abstract
Documentation:

Ship_Order CRS

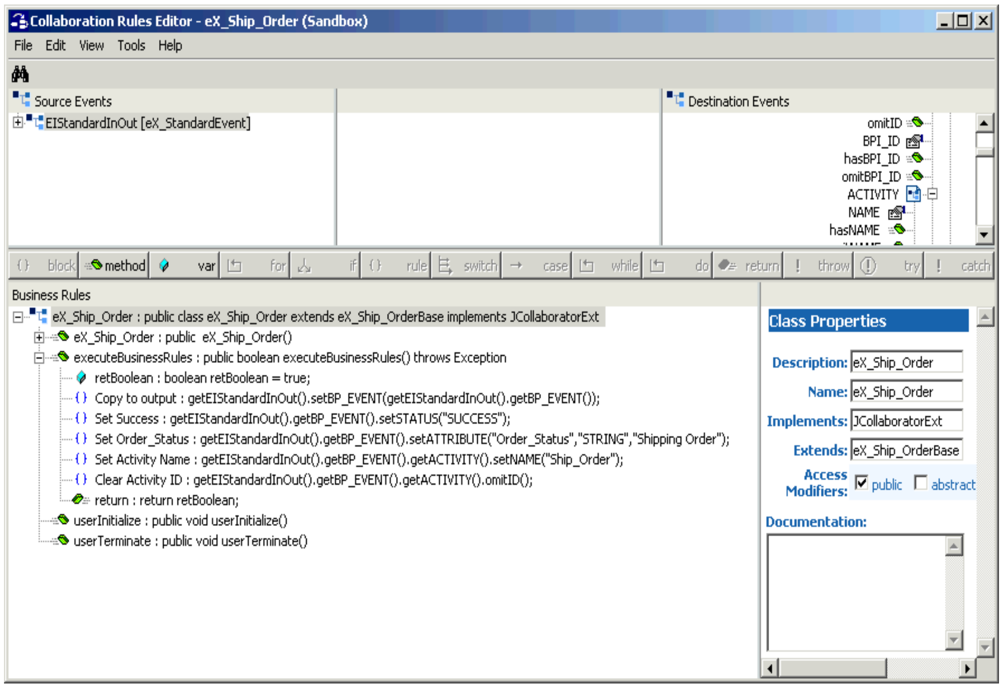
The Ship_Order translation simulates the activity of sending out an item that is in stock. It sets the value of the Order_Status attribute to a short message indicating that the order has been sent and returns “SUCCESS” to the e*Insight engine.

In the active example, the event is sent from the e*Insight engine with information pertaining to the Ship_Order activity. In the passive example, the event was originally created by the e*Insight engine for the Bill_Customer activity, and the eX_Bill_Customer Collaboration is configured to send a “Do” event. This “Do” event contains information pertaining to the Bill_Customer activity, rather than the Ship_Order activity, so the following values need to be updated:

- Activity Name — should be set to “Ship_Order”
- Activity ID — should be deleted

Figure 150 shows the Ship_Order CRS used in the example.

Figure 153 Ship_Order.xpr CRS


Collaboration Rules Editor - eX_Ship_Order (Sandbox)
File Edit View Tools Help
Source Events
EIStandardInOut [eX_StandardEvent]
Destination Events
omitID
BPI_ID
hasBPI_ID
omitBPI_ID
ACTIVITY
NAME
hasNAME
block method var for if rule switch case while do return throw try catch
Business Rules
Class Properties

```
eX_Ship_Order : public class eX_Ship_Order extends eX_Ship_OrderBase implements JCollaboratorExt
    eX_Ship_Order : public  eX_Ship_Order()
    executeBusinessRules : public boolean executeBusinessRules() throws Exception
        retBoolean : boolean retBoolean = true;
        Copy to output : getEIStandardInOut().setBP_EVENT(getEIStandardInOut().getBP_EVENT());
        Set Success : getEIStandardInOut().getBP_EVENT().setSTATUS("SUCCESS");
        Set Order_Status : getEIStandardInOut().getBP_EVENT().setATTRIBUTE("Order_Status","STRING","Shipping Order");
        Set Activity Name : getEIStandardInOut().getBP_EVENT().getACTIVITY().setNAME("Ship_Order");
        Clear Activity ID : getEIStandardInOut().getBP_EVENT().getACTIVITY().omitID();
        return : return retBoolean;
    userInitialize : public void userInitialize()
    userTerminate : public void userTerminate()
```

Description: eX_Ship_Order
Name: eX_Ship_Order
Implements: JCollaboratorExt
Extends: eX_Ship_OrderBase
Access Modifiers: [✓ public] abstract
Documentation:

16.11.3 Run and Test the e*Insight scenario

Run the Schema again as described in **"Run and Test the e*Insight scenario" on page 255**.

16.12 Passive Control Mode (eISchema)

This section describes the steps for updating a schema based on eISchema (Classic). For information on eIJSchema, see **"Passive Control Mode (eIJSchema)" on page 256**.

16.12.1 Modify the Order BP in e*Insight (eISchema)

The following is a summary of the procedure for modifying the BP in the e*Insight GUI.

1. Save the business process as a new version.
2. Change the Ship_Order activity properties to use passive control.

16.12.2 Modify User-defined e*Gate Components (eISchema)

Configuration Order for the User-defined Components

1. Modify the Bill_Customer CRS.
2. Modify the Ship_Order CRS.
3. Modify the Bill_Customer Collaboration Rule and Collaboration.

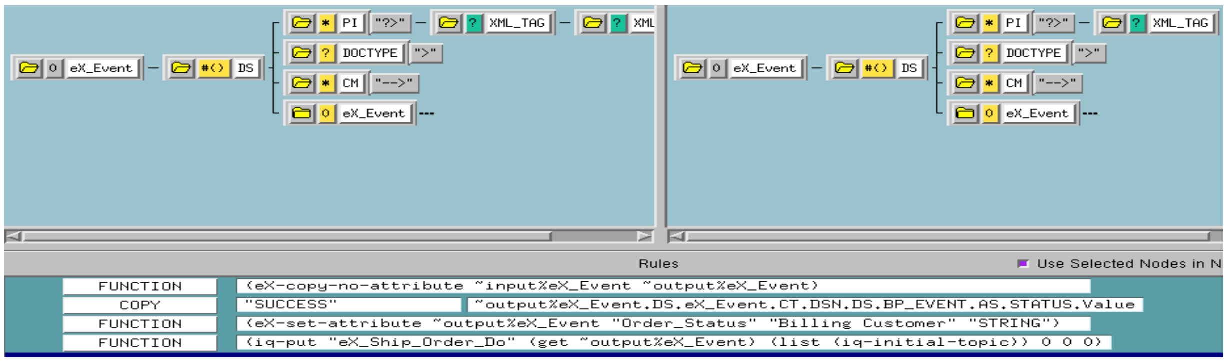
Bill Customer CRS

The CRS is already defined to set the value of the Order_Status attribute to "Billing Customer" and send a "SUCCESS" Event back to the e*Insight, indicating that the activity has completed successfully.

The script is modified to use an iq-put to publish **eX_Ship_Order_Do**.

Figure 152, shows the **Billing_Customer.tsc** CRS used in the example.

Figure 154 Billing_Customer.tsc CRS



```
FUNCTION   (eX-copy-no-attribute ~input%eX_Event ~output%eX_Event)
COPY       "SUCCESS"   ~output%eX_Event.DS.eX_Event.CT.DSN.DS.BP_EVENT.AS.STATUS.Value
FUNCTION   (eX-set-attribute ~output%eX_Event "Order_Status" "Billing Customer" "STRING")
FUNCTION   (iq-put "eX_Ship_Order_Do" (get ~output%eX_Event) (list (iq-initial-topic)) 0 0 0)
```

Ship_Order CRS

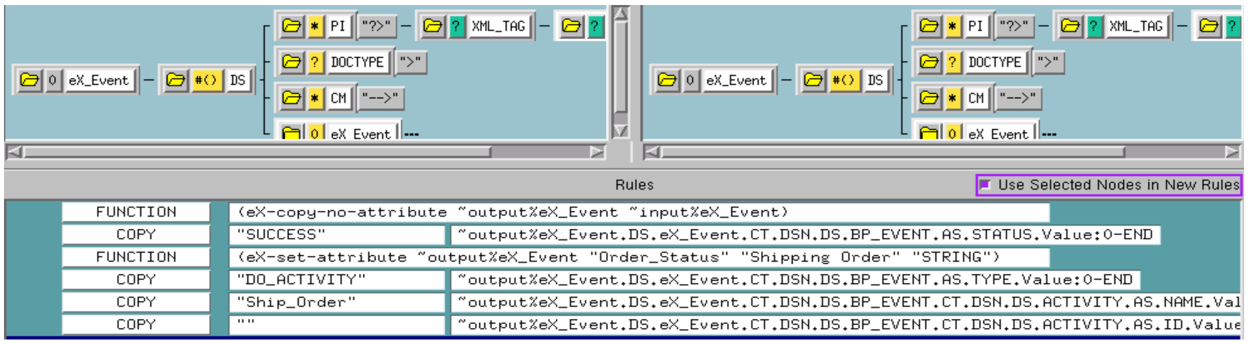
The Ship_Order translation simulates the activity of sending out an item that is in stock. It sets the value of the Order_Status attribute to a short message indicating that the order has been sent and returns "SUCCESS" to the e*Insight engine.

In the active example, the event is sent from the e*Insight engine with information pertaining to the Ship_Order activity. In the passive example, the event was originally created by the e*Insight engine for the Bill_Customer activity, and the eX_Bill_Customer Collaboration is configured to send a “Do” event. This “Do” event contains information pertaining to the Bill_Customer activity, rather than the Ship_Order activity, so the following values need to be updated:

- Activity Name — should be set to “Ship_Order”
- Activity ID — should be deleted

Figure 150 shows the Ship_Order CRS used in the example.

Figure 155 Ship_Order.tsc CRS



```
FUNCTION   (eX-copy-no-attribute ~output%eX_Event ~input%eX_Event)
COPY       "SUCCESS"          ~output%eX_Event.DS.eX_Event.CT.DSN.DS.BP_EVENT.AS.STATUS.Value:0-END
FUNCTION   (eX-set-attribute ~output%eX_Event "Order_Status" "Shipping Order" "STRING")
COPY       "DO_ACTIVITY"      ~output%eX_Event.DS.eX_Event.CT.DSN.DS.BP_EVENT.AS.TYPE.Value:0-END
COPY       "Ship_Order"       ~output%eX_Event.DS.eX_Event.CT.DSN.DS.BP_EVENT.CT.DSN.DS.ACTIVITY.AS.NAME.Val
COPY       ""                 ~output%eX_Event.DS.eX_Event.CT.DSN.DS.BP_EVENT.CT.DSN.DS.ACTIVITY.AS.ID.Value
```

Configure the Bill_Customer Collaboration Rule and Collaboration

1. Update the Bill_Customer Collaboration Rule to additionally publish **eX_Ship_Order_Do**. This should not be selected as the default publication.
2. Update the Bill_Customer Collaboration to additionally publish **eX_Ship_Order_Do** to the **eX_eBPM** IQ.

16.12.3 Run and Test the e*Insight scenario

Run the Schema again as described in **“Run and Test the e*Insight scenario” on page 255**.

Chapter 17

e*Insight Performance

The purpose of this chapter is to describe methods that can be used to improve performance. The chapter is divided into three sections; the first describes the performance enhancements that can be made to a schema based on eIJSchema (Java). The second describes the performance enhancements that can be made to a schema based on eISchema (Classic). The last section describes performance enhancements that are not schema specific.

17.1 Performance Improvements Using eIJSchema

The purpose of this section is to describe methods that can be used to improve performance of the eIJSchema.

- Instance caching
- Using multiple engines
- Using instance caching with multiple engines
- Using binary XML
- Configuring e*Insight to ignore e*Xchange ETD
- Changing the Event Type “get” Interval

17.1.1 Instance Caching

Instance Caching is the most efficient way to process Business Process Instances. Using instance caching keeps a cache of the instance information throughout the life span of the Business Process Instance. If instance caching is not used the instance information is retrieved from the database instead.

To configure the engine to use instance caching

1. In the **e*Way Connections** folder, open the engine’s e*Way connector properties.
2. Click **Edit** to open the engine’s configuration file.
3. In the **eBPM Setting** section, set **Instance Caching** to **YES**.

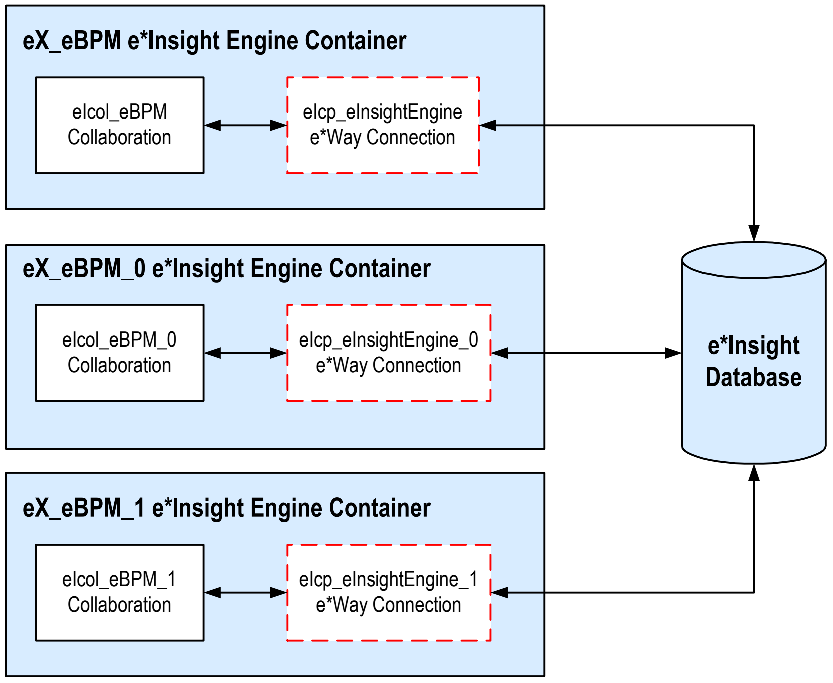
17.1.2 Using Multiple e*Insight Engines

An e*Insight engine is comprised of a specially configured Collaboration (eIcol_eBPM) and the e*Insight e*Way Connection (eIcp_eInsightEngine). The e*Insight engine runs within a Multi-Mode e*Way (eX_eBPM), which is referred to as the e*Insight Engine Container. You can use multiple e*Insight engines to increase performance. This section describes how to add and configure these. You can use instance caching to further improve performance. See **"e*Insight Engine Affinity (eIJSchema)" on page 266** for more information.

When you add e*Insight engines, you can either add them to an existing e*Insight engine container or create additional e*Insight engine containers. Figure 156 shows a scenario where three e*Insight engine containers are used, each with a single engine.

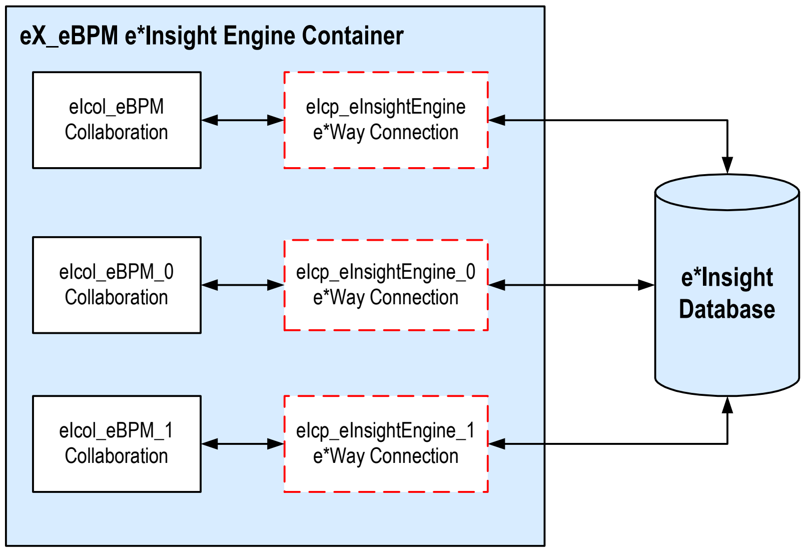
Important: *A single e*Insight e*Way connection cannot be used by multiple Collaborations.*

Figure 156 Multiple Engine Containers with a Single Engine


eX_eBPM e*Insight Engine Container
eIcol_eBPM Collaboration
eIcp_eInsightEngine e*Way Connection
eX_eBPM_0 e*Insight Engine Container
eIcol_eBPM_0 Collaboration
eIcp_eInsightEngine_0 e*Way Connection
e*Insight Database
eX_eBPM_1 e*Insight Engine Container
eIcol_eBPM_1 Collaboration
eIcp_eInsightEngine_1 e*Way Connection

Alternatively, you can have multiple e*Insight engines in a single e*Insight engine container. See Figure 157.

Figure 157 A Single Engine Container with Multiple Engines


eX_eBPM e*Insight Engine Container
eIcol_eBPM Collaboration
eIcp_eInsightEngine e*Way Connection
eIcol_eBPM_0 Collaboration
eIcp_eInsightEngine_0 e*Way Connection
eIcol_eBPM_1 Collaboration
eIcp_eInsightEngine_1 e*Way Connection
e*Insight Database

You can either create a new engine from scratch, or copy the engine container, Collaboration, Collaboration Rules, and engine connection.

To create a new e*Insight engine container

1. Add a new e*Insight engine in the participating host.
2. In the engine properties, **Executable file** section, click **Find**.
3. Browse for **stceway.exe** and click **Select**.
4. Clear the engine's configuration file.

Note: *You can create a new e*Insight engine container by copying the* ***eX_eBPM*** *e*Insight engine that is provided in the eIJSchema.*

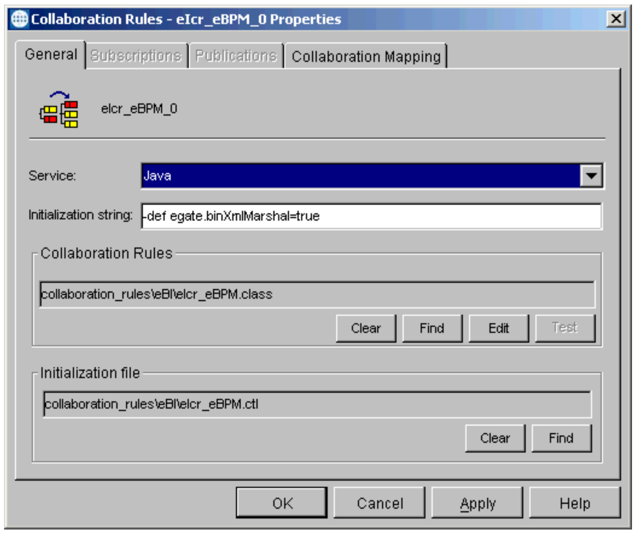
To add and configure an e*Insight e*Way Connection

1. Select the **e*Way Connections** folder and create a new e*Way Connection.
2. Open the Connection properties and set the e*Way Connection Type to e*Insight Engine.
3. Create and configure a new Configuration File. For information on the settings, see **"Configuring the e*Insight Engine Connection" on page 34**.

To configure an e*Insight engine container for a new e*Way Connection

1. Create a new e*Way Connection. See **"To add and configure an e*Insight e*Way Connection" on page 264**.
2. Add a Collaboration Rule and configure it as shown in Figure 158.

Figure 158 e*Insight Engine General Tab



eIcr_eBPM_0

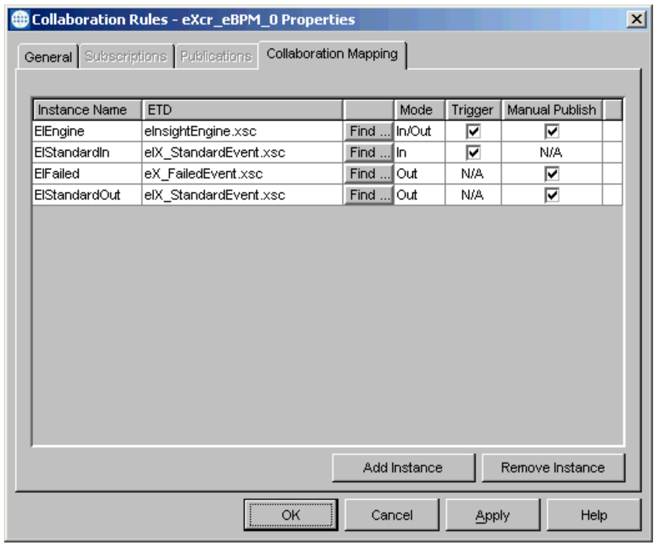
Service: Java

Initialization string: -def egate.binXmlMarshal=true

Collaboration Rules: collaboration_rules\eBI\eIcr_eBPM.class

Initialization file: collaboration_rules\eBI\eIcr_eBPM.ctl

Figure 159 e*Insight Engine Collaboration Mapping Tab

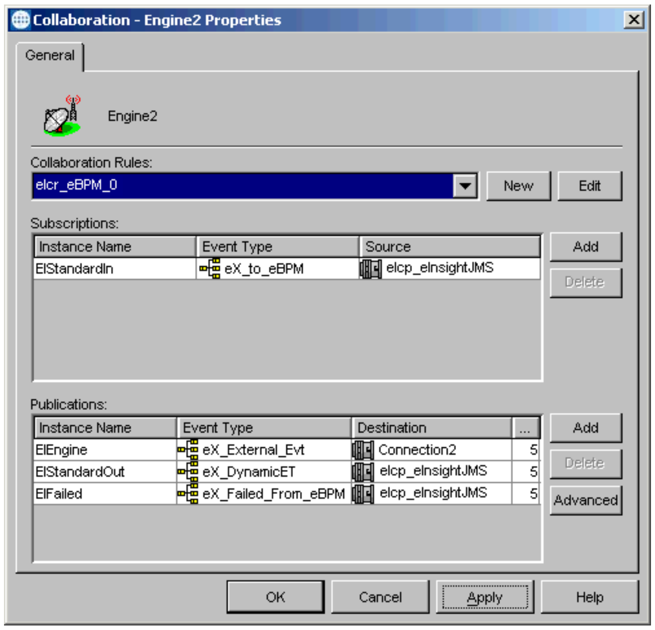


Instance Name	ETD		Mode	Trigger	Manual Publish
EIEngine	eInsightEngine.xsc	Find ...	In/Out	✓	✓
EIStandardIn	eIX_StandardEvent.xsc	Find ...	In	✓	N/A
EIFailed	eX_FailedEvent.xsc	Find ...	Out	N/A	✓
EIStandardOut	eIX_StandardEvent.xsc	Find ...	Out	N/A	✓

Note: *You can create this Collaboration Rule by copying the* ***eIcr_eBPM*** *Collaboration Rule that is provided in the eIJSchema.*

3 Add a Collaboration to the e*Insight engine container that uses the Collaboration Rule created in step 2. Configure the Collaboration as shown Figure 160. You should configure the Collaboration to publish to the e*Way Connection created in step 1.

Figure 160 e*Insight Engine Collaboration



Collaboration Rules: elcr_eBPM_0

Subscriptions:

Instance Name	Event Type	Source
EIStandardIn	eX_to_eBPM	elcp_eInsightJMS

Publications:

Instance Name	Event Type	Destination	...
EIEngine	eX_External_Evt	Connection2	5
EIStandardOut	eX_DynamicET	elcp_eInsightJMS	5
EIFailed	eX_Failed_From_eBPM	elcp_eInsightJMS	5

17.1.3 e*Insight Engine Affinity (eIJSchema)

e*Insight Engine Affinity allows e*Insight engines in a multi-engine environment to cache information about particular Business Process Instances as they flow through the e*Gate schema using Instance Caching. Since the engines hold the instance information in a cache, it is essential that an individual instance is always processed by the same engine or Collaboration. This is achieved by setting a JMS property in e*Gate 4.5.2, or using a different Event Type for each engine in e*Gate 4.5.1.

Using Engine Affinity can possibly improve the overall message throughput but if an engine is shut down for some reason, the instances associated with that engine do not finish being processed until the engine is manually restarted using the e*Gate Monitor.

17.1.4 Using Engine Affinity with e*Gate 4.5.2

The JMS Message Server uses message selectors to filter out certain messages from a specific queue before sending them to the JMS e*Way connection. The message selector ensures that an engine only receives Business Process Instances that it originally processed, or START_BP messages .

Note: *The message selector for the inbound eX_StandardEvent is only set if you are using e*Gate 4.5.2 and are subscribing to a JMS e*Way Connection Point. Otherwise, you must configure Engine Affinity using different Event Types, see* **"Using Engine Affinity with e*Gate 4.5.1" on page 267***.*

To configure multiple engines to use e*Insight Engine Affinity

1. Create multiple engines. See **"Using Multiple e*Insight Engines" on page 263**.
2. Edit the configuration file for each e*Insight engine connection. In the **eBPM Settings**, set the **Instance Caching** parameter to **Yes**.

Note: *The engines can refer to the same connection configuration file.*

3. Configure the Engine Affinity JMS properties in your Activity Collaborations, if you are using different source and destination instances for eI_StandardEvent. See **"Configuring the Engine Affinity JMS Properties"**.

Configuring the Engine Affinity JMS Properties

You need to set up the Engine Affinity JMS properties in your Activity Collaborations, if you are using different source and destination instances for eI_StandardEvent. You can set up the properties using the prepareReplyToSender method in the eI_StandardEvent ETD.

Create a rule before you return from executeBusinessRules() by dragging the root node of the inbound eI_StandardEvent into the parameter for the prepareReplyToSender() method of the outbound eI_StandardEvent. This generates code such as:

```
getEIStandardOut.prepareReplyToSender(getEIStandardIn());
```

Note: *If you are using one instance (such as EIStandardInOut), you do not need to use prepareReplyToSender() since the Engine Affinity JMS Properties already exist.*

For backwards compatibility, if the first Business Process Activity is passive, the e*Insight engine still echoes back the incoming START_BP message for synchronization. To disable this feature, add another System Property definition, **ei.pasvNoEcho=true**, in the Initialization string of the e*Insight Engine Collaboration Rules.

17.1.5 Using Engine Affinity with e*Gate 4.5.1

The procedures described in the next section use unique names for the Event Types based on the Collaboration name. You can alternatively use the e*Way name if the e*Way uses a single e*Way Connection. If multiple e*Way Connections are used then using the e*Way name does not provide a unique name for each engine. For information on configuring the e*Way Connection to use the e*Way name, see **"Configuring the e*Insight Engine Connection" on page 34**.

To configure multiple engines to use e*Insight Engine Affinity

1. Create multiple engines.
2. Edit the configuration file for each e*Insight engine. In the **eBPM Settings**, set the **Instance Caching** parameter to **Yes**.

Note: *The engines can refer to the same configuration file.*

3 Ensure that every Collaboration uses a unique Collaboration Rule.

4 Create an Event Type named **eX_to_<Collaboration Name>** for each e*Way connection.

Important: *An Event Type must be created for the default engine named **eX_to_eX_eBPM** in addition to the Event Types required for additional engines.*

5 For every e*Way connection, update the Collaboration that subscribes to eX_eBPM using the following procedure. Replace <Collaboration name> with the appropriate name.

- In the Collaboration properties Subscriptions box, change the **Event Type** to **eX_to_<Collaboration name>** with Source **eIcp_eInsightJMS**.

6 For every *Activity* e*Way or BOB, update the Collaboration that publishes eX_to_eBPM with the additional publications for the additional engines.

- In the Collaboration properties Subscriptions box, add an entry for every e*Way Connection Collaboration with the following properties: **EIStandardInOut** instance**, eX_to_<Collaboration name>** Event Type, with Destination **eIcp_eInsightJMS**.

Note: *The instance name may be different, depending on your implementation.*

17.1.6 Using e*Xchange with e*Insight (eIJSchema)

The Event Type Definition used in the engine's Collaboration Rules Script, eIX_StandardEvent.xsc contains two sections, BP_EVENT, containing information for e*Insight and TP_EVENT, containing information for e*Xchange. If you are not using the e*Xchange section of the ETD, then you can configure the engine ignore this section, reducing processing.

To configure the engine to ignore e*Xchange section of the ETD

1 Select the e*Way Connections folder.

2 Open the properties of the e*Insight engine e*Way Connector.

3 Click **Edit**.

4 Go to the **eBPM Settings** section, and find **Using e*Xchange with e*Insight**.

5 Click **No**.

6 Close the configuration editor and save the changes.

17.1.7 Using Binary XML (eIJSchema)

By default, the e*Insight engine generates a binary XML message. This data format reduces parsing and so can increase performance. The following initialization string in the Collaboration Rule properties determines that a binary XML message is created:

```
-def egate.binXmlMarshal=true
```

Important: *Monk is not able to interpret this data format so you should only configure Java Collaborations to subscribe to this Event.*

17.1.8 Subscribing to Event Types

The method of subscribing for multiple Event Types from a JMS e*Way Connection entails waiting 'n' milliseconds (configured in the Event Type "get" interval of the e*Way Connection Properties dialog, see **"Event Type "get" Interval" on page 270**) for the first Event Type and if none is found, waiting for 1 millisecond for the next Event Type and if none again, the JMS Server waits 'n' milliseconds for the second Event Type and so on. This method works well if the distribution of the different Event Types is relatively even, but there are likely to be more "Do" Events than there are "Undo" Events for e*Insight. Therefore, the engines spend a lot of time waiting for a Event Type that is not there.

You can avoid unnecessary wait times due to the "Do" and "Undo" Event Types, by:

- Subscribing to a single "Go" Event Type which retrieves both "Do" and "Undo" Events
- Having a separate Collaboration to subscribe to the "Do" and "Undo" Event Types
- Removing unnecessary subscriptions from Collaborations

The options are described in detail below.

Subscribing to a Single "Go" Event

The e*Insight engine can be configured to publish an Event Type **eI_<Activity Name>_Go**, rather than **eX_<Activity Name>_Do** or **eI_<Activity Name>_Undo**. This allows you to subscribe to a single Event Type and then check the type of Event within your Collaboration Rule script.

To configure the engine to publish the single Event Type, set the initialization string in the eI Engine Collaboration Rules dialog to:

```
-def egate.multiDef=;egate.binXmlMarshal=true;ei.oneETPerAct=true
```

This new Java System Property definition, **ei.oneETPerAct=true**, directs the engine to publish only one ET, **eI_<Activity Name>_Go**, for both "Do" and "Undo" Events.

Note: *You must use the egate.multiDef parameter to define more than one System Property definition in the initialization string. The first character after the equal (=) sign is the delimiter to use for separating different System Property key=value pairs (for example, the semi-colon (;) is used above). The Enterprise Manager does not allow the use of commas (,) since the e*Gate Registry uses this delimiter internally. Special characters can be specified by using standard escape codes (such as\t, \n) or Unicode escape codes (such as \u003d for '=').*

Configuring a Separate Collaboration for Do and Undo Events

To remove the requirement for a single Collaboration to subscribe to both "Do" and "Undo" Events, separate the "Do" and "Undo" logic in your Collaboration Rules script.

Then create a “Do” and an “Undo” Collaboration, each subscribing to a single Event Type.

Removing Unnecessary Subscriptions

Subscribing to multiple Event Types is less efficient than subscribing to a single Event Type. If you have not implemented “undo” logic in your business process, then remove the subscriptions to the eX_<Activity_Name>_Undo Event Types.

17.1.9 Event Type “get” Interval

The e*Insight engine uses the “get” interval to determine how long the JMS Server waits for an Event of a particular Event Type to arrive.

Ideally, a Collaboration should only subscribe to a single Event Type, but if that is not possible then the “get” interval can impact performance. The default interval is 10,000 milliseconds which should be suitable if the distribution of the different Event Types is even. If the distribution of the different Event Types is not even, the you might want to reduce the“get” interval. Consider an example where you are subscribing to Event Types A and B, and Event Type B rarely (or never) arrives. Using the default “get” interval, you wait 10,000 milliseconds after every Event Type A is processed. Reducing the “get” interval to a value less than 100 can dramatically increase performance.

The **Event Type “get” Interval** is set in the e*Way Connection properties.

17.1.10 Review JVM Settings

You may be able to improve performance by changing the memory allocated for the Java Virtual Machine. This is set in the e*Insight engine configuration file, **JVM Settings**, **Maximum Heap Size**. If this is left set to zero (0), the preferred value for the maximum heap size of the Java VM is used.

Note: *The e*Insight engine configuration file does not exist by default. You must create a new configuration file.*

17.2 Performance Improvements Using eISchema

The purpose of this section is to describe methods that can be used to improve performance of the eIJSchema.

- Instance caching
- Using multiple engines
- Using instance caching with multiple engines
- Setting the Exchange Data Interval

17.2.1 Instance Caching

Instance Caching is the most efficient way to process Business Process Instances. Using instance caching keeps a cache of the instance information throughout the life span of the Business Process Instance. If instance caching is not used the instance information is retrieved from the database instead. This allows more flexibility and fault tolerance at the cost of performance.

To configure the engine to use instance caching

1. Open the engine's properties.
2. Click **Edit** to open the engine's configuration file.
3. In the **eBPM Setting** section, set **Instance Caching** to **YES**.

17.2.2 Using Multiple e*Insight Engines (eISchema)

You can use multiple e*Insight engines to increase performance. This section describes how to add and configure additional engines. You can use instance caching to further improve performance. See **"e*Insight Engine Affinity (eISchema)" on page 272** for more information.

You can either create a new engine from scratch, or copy the engine and collaborations. Both procedures are described below.

To create a new e*Insight engine

1. Add a new e*Insight to the participating host.
2. Edit the engine's configuration file.
3. Add a Collaboration Rule that subscribes to **eX_External_Evt** and publishes **eX_Failed_From_eBPM**.
4. Add a Collaboration Rule that subscribes to **eX_to_eBPM** and publishes **eX_External_Evt**.
5. Add a Collaboration to the e*Insight engine that uses the Collaboration Rule created in step 3. Configure to subscribe to **External** and publish to **eX_Dead_Letter_Queue** IQ.
6. Add a Collaboration to the e*Insight engine that uses the Collaboration Rule created in step 3. Configure to subscribe to **eX_eBPM** and publish to **External** IQ.

To copy an existing engine

1. Copy the e*Insight engine.
2. Copy the eX_to_eBPM Collaboration.
3. Copy the eX_from_eBPM Collaboration.

Important: *When you copy the above components some elements are then used by both engines. These include the e*Insight engine configuration file and the Collaboration Rules. If you need to change these for one engine, but not the other, you must create a new version.*

17.2.3 e*Insight Engine Affinity (eISchema)

e*Insight Engine Affinity allows e*Insight engines in a multi-engine e*Gate schema to cache information about particular Business Process Instances as they flow through the e*Gate schema using Instance Caching. Using Engine Affinity can possibly improve the overall message throughput but if an engine is shutdown for some reason, the instances associated with that engine do not finish being processed until the engine is manually restarted using the e*Gate Monitor.

To configure multiple engines to use e*Insight Engine Affinity

1. Create multiple engines.
2. Edit the configuration file for each e*Insight engine. In the **eBPM Settings**, set the **Instance Caching** parameter to **Yes**.

Note: *The engines can refer to the same configuration file.*

3. Ensure that every Collaboration uses a unique Collaboration Rule.
4. Create an Event Type named **eX_to_<eInsight Engine Name>** for each e*Insight engine.

Important: *An Event Type must be created for the default engine named **eX_to_eX_eBPM** in addition to the Event Types required for additional engines.*

5. For every e*Insight engine, update the Collaboration that subscribes to eX_eBPM using the following procedure. Replace <eInsight engine name> with the appropriate name.
   - A In the Collaboration Rule properties, go to the Subscription tab and add **eX_to_<eInsight engine name>**.
   - B In the Collaboration properties Subscriptions box, add **eX_to_<eInsight engine name>** with Source **<ANY>**.
6. For every *Activity* e*Way or BOB, update the Collaboration that publishes eX_to_eBPM with the additional publications for the additional engines.

   To update a Monk Collaboration

   - A In the Collaboration Rule properties, go to the Publication tab and add **eX_to_<eInsight engine name>** for every e*Insight engine.
   - B In the Collaboration properties Publications box, add **eX_to_<eInsight engine name>** with Destination **eX_eBPM** for every e*Insight engine.
   - C Update the Monk Collaboration Rule Script to manually publish the event using eX-event-sendback-to-sender, and then suppress the default output. For details on these two lines of code see Figure 161.

   To update a Java Collaboration

   - A In the Collaboration Rule properties, go to the Collaboration Mapping tab and add **eX_to_<eInsight engine name>** instance for every e*Insight engine.

B In the Collaboration properties Subscriptions box, add an entry for every e*Insight engine with the following properties: **eX_to_<eInsight engine name>** instance**, eX_to_<eInsight engine name>** Event Type, with Destination **eX_eBPM**.

Manually Publishing Events using eX-event-sendback-to-sender

The eX-event-sendback-to-sender function performs an iq-put that dynamically assigns the destination Event Type. The destination Event Type is defined by appending the name of the e*Insight engine that sent the message to the string “eX_to”. The syntax is:

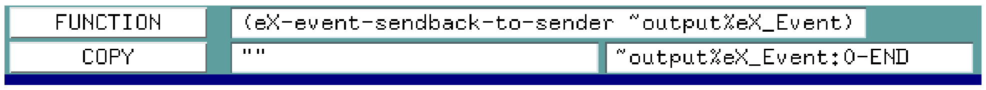
eX-event-sendback-to-sender <root-path>

You would usually replace <root-path> with **~output%eX_Event**. For an example of how this is used in a Collaboration Rule Script, see Figure 161.

Note: *Make sure that the Monk file **eX-event-sendback-to-sender.monk**, containing this function, is loaded before calling it in a Collaboration Rules Script. You can do this by putting it in the root of the **monk_library** directory, or loading it explicitly in your CRS.*

Finally, you must suppress the default output. This is achieved by overwriting the destination Event Type Definition with an empty string. Use the copy function as shown in Figure 161.

Figure 161 Monk Collaboration Rule Script for Engine Affinity



FUNCTION	(eX-event-sendback-to-sender ~output%eX_Event)	
COPY	""	~output%eX_Event:0-END

17.2.4 Exchange Data Interval (eISchema)

The e*Insight engine uses the interval to determine how often to poll the control table. The control table contains entries for batch tasks, such as manual restarts, and also contains entries for instances using the User Activity or Authorization Activity. Depending on your implementation of e*Insight, it may be desirable to reduce the polling frequency to increase performance.

The Exchange Data Interval is set in the e*Way Configuration file, **Communication Setup** section.

17.2.5 Review JVM Settings

You may be able to improve performance by changing the memory allocated for the Java Virtual Machine. This is set in the e*Insight engine configuration file, **JVM Settings**, **Maximum Heap Size**. If this is left set to zero (0), the preferred value for the maximum heap size of the Java VM is used.

17.3 General e*Insight Performance Tips

This section describes some ways of improving performance that are not specific to the type of e*Insight schema that you are running.

- Review the *SeeBeyond eBusiness Integration Suite Deployment Guide*.
- Use a third party tool to determine if hardware is the limitation.
- Enhance database access by one of the following:

  A Use RAID/stripe disks with a multi-controller.

  B Install the database and e*Insight engines on the same machine.

  C Tune the database.
- Use a Model Specific database — The Model Specific database uses a different structure for storing the attribute values, which increases performances. If you are using the Model Specific database, you can modify the database to your specific data requirements by controlling the size allocated to each attribute value. You need to create the necessary database tables before running your e*Insight schema. Every business process version uses its own set of tables.

  See the *e*Insight Business Process Manager User's Guide* for information on creating the Model Specific database.
- Do not automatically reload models — The Auto Model Reload engine configuration parameter determines if the engine dynamically loads an enabled Business Process Version if the enabled/disabled status of Business Process Version changes. If the value is set to YES then Business Process Versions that are enabled or disabled while the engine is running are immediately recognized. However, setting this value to YES may degrade performance.
- Do not preload unnecessary Business Processes — The Business Processes to Preload engine configuration parameter allows you to load all or a subset of all the business processes stored in the e*Insight database. The default setting is ALL. Naming specific business processes to preload may improve performance if you have a large number of business processes defined.
- Do not include input only attributes in the "Done" Event — To simplify your Collaboration Rules script, you may decide to copy everything from the source ETD to the destination ETD, especially when using Java. This copies all the input attributes to the destination ETD and, unless they are removed, they are included in the "Done" Event. This requires additional processing by the engine, as it tries to write the attribute to the database but is refused permission.

Chapter 18

Troubleshooting

One of the easiest ways to debug your e*Gate configuration is through the use of log files. All executable components—BOBs, e*Ways, IQ Managers, and Control Brokers—have the ability to create log files that contain whatever level of debugging information you select.

18.1 Log File Locations

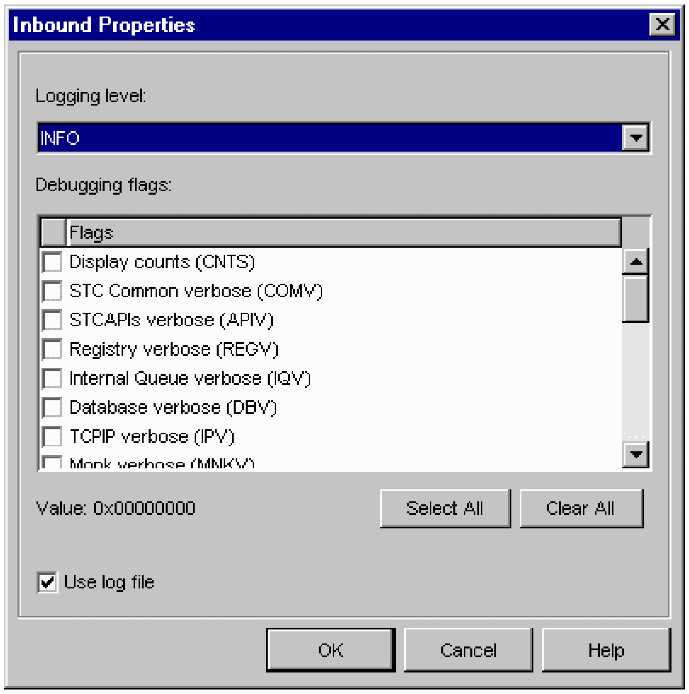
All log files are stored in the **\eGate\client\logs** directory on the Participating Host running the elements that generate the log entries. Logs are named after the component that creates them; for example, the **eX_eBPM** engine creates a log file called **eX_eBPM.log**.

18.2 Generating Log Files

To configure a component to generate a log file:

1. In the e*Gate Enterprise Manager window, select the component that you want to configure and display its properties.
2. Select the **Advanced** tab, and then click **Log**.
3. Select the desired logging options (see Figure 162).

Figure 162 Logging Options



You can view a component's log using any text editor, and you can view the log while the component is still running. However, depending on the editor, you may need to re-read the file to "refresh" your view of the log data. You cannot get log updates "on the fly."

The most common error most first-time e*Gate developers find in a log file is, "Unable to load module configuration." This message means that you have created an e*Gate component but not assigned both an executable file and a configuration file to it.

When you first start to debug your e*Insight schema, you should apply minimal flags so it is easier to find useful messages. You can start with the following flags with a DEBUG logging level selected:

- e*Way (EWY)
- Collaboration (COL) — for eIJSchema
- Monk (MNK) — for eISchema

Note: *A specific debugging flag does not appear in the flags list for the e*Insight engine, but if the logging level is set to anything other than None, then messages of type **EBPM** appear in the log.Setting the e*Insight engine logging level to **Trace** generates a very detailed log of the engine's activity.*

For more information about logging and debugging options, see the *e*Gate Integrator System Administration and Operations Guide.*

18.3 Common Problems

The common problems when you run an e*Insight schema initially broadly fall into two categories; either the e*Insight engine is incorrectly configured or the XML message that is sent to the e*Insight engine contains invalid information.

Table 59 and Table 60 show a number of common problems and suggested actions you should take. The tables also shows where to look for the error message and the logging required to display the error.

Table 59 Common errors (eIJSchema)

Error Message	Debug Flag	Problem	Resolution
JMSException: Could not connect to host: [hostname], port: 24053	COL	eIcp_eInsightJMS e*Way connection is configured incorrectly	In the eIcp_eInsightJMS connection configuration file, **Message Service** section, check the value for **Server Name** and **Host Name**.
Cannot connect to e*Insight Backend DataBase: Io exception: Invalid number format for port number	EWY	JDBC URL incorrect	In the eIcp_eInsightEngine configuration file, **eBPM Settings** section, check the value defined for **JDBC URL String**.
Cannot connect to e*Insight Backend DataBase: Invalid Oracle URL specified	EWY		
Cannot connect to e*Insight Backend DataBase: Io exception: Connection refused	EWY	Unable to connect to the database	In the eIcp_eInsightEngine configuration file, **eBPM Settings** section, check the value defined for **JDBC URL String**.
Cannot connect to e*Insight Backend DataBase: ORA-01017: invalid username/ password; logon denied	EWY	Incorrect user name or password	In the eIcp_eInsightEngine configuration file, **eBPM Settings** section, check the values defined for **Database User Name** and **Encrypted Password**.

Table 59 Common errors (eIJSchema)

Error Message	Debug Flag	Problem	Resolution
startActivity() failed: Definition not found for :*Payrolls*	EWY	Incorrect business process name	Check that the business process name defined in the CRS matches the business process name defined in e*Insight exactly. Also, check that a business process version has been enabled for this business process. Note: Although the error message appears in the eX_eBPM log, the problem is in the component that sent the message to the e*Insight engine.
startActivity() failed: Invalid BP_EVENT type:*STARTBP*	EWY	Incorrect BP_EVENT type	Check that the BP_EVENT type defined in the CRS is correct. Note: Although the error appears in the eX_eBPM log, the problem is in the component that sent the message to the e*Insight engine.
ewjx: Exception getMessage(): processOutgoing(): eBPM cannot process XML data: ORA-01407: cannot update ("EX_ADMIN"."BUSINESS_PROCESS_INSTANCE"."BPI_NM") to NULL	EWY	No ID set	Check that the ID is defined in the CRS. Note: Although the error appears in the eX_eBPM log, the problem is in the component that sent the message to the e*Insight engine.

Table 60 Common errors (eISchema)

Error Message	Debug Flag	Problem	Resolution
eX_eBPM (Fatal): ewjx: the "JNI DLL" (<jvm path>) specified is not a Java 2 version	EBPM	jvm configuration incorrect	In the eX_eBPM configuration file, Java VM Configuration section, check the value defined for JNI DLL.
eX_eBPM (Warning): Cannot connect to eBPM Backend DataBase: Io exception: Connection refused(DESCRIPTION=(TMP=)(VSNNUM=135290880)(ERR=12505)(ERROR_STACK=(ERROR=(CODE=12505)(EMFI=4))))	EBPM	jdbc url incorrect	In the eX_eBPM configuration file, eBPM Settings section, check the value defined for JDBC URL String.
eX_eBPM (Warning): Cannot connect to eBPM Backend DataBase: ORA-01017: invalid username/password; logon denied	EBPM	Incorrect user name or password	In the eX_eBPM configuration file, eBPM Settings section, check the values defined for Database User Name and Encrypted Password.
>>>>MONKEXCEPT:0036: RESOLVE_VARIABLE: v ariable <eX-set-attribute> has not been defined.	MNK	eX-eBPM-utils.monk has not been loaded	Copy eX-eBPM-utils.monk to <egate>\client\monk_library.
>>>>MONKEXCEPT:0069: throw: eBPM: Cannot process event	MNK	Incorrect business process name	Check that the business process name defined in the CRS matches the business process name defined in e*Insight exactly. Also, check that a business process version has been enabled for this business process. Note: Although the error message appears in the eX_eBPM log, the problem is in the component that sent the message to the e*Insight engine.
ewjx: Exception getMessage(): processOutgoing(): eBPM cannot process XML data: ERROR: Unable to load business process. bpoId not found for :fred	EWY		

Table 60 Common errors (eISchema)

Error Message	Debug Flag	Problem	Resolution
>>>>MONKEXCEPT:0069: throw: eBPM: Cannot process event	MNK	Incorrect BP_EVENT type	Check that the BP_EVENT type defined in the CRS is correct. Note: Although the error appears in the eX_eBPM log, the problem is in the component that sent the message to the e*Insight engine.
ewjx: Exception getMessage(): processOutgoing(): eBPM cannot process XML data: Invalid BP_EVENT type:START_BD	EWY		
>>>>MONKEXCEPT:0069: throw: eBPM: Cannot process event	MNK	No ID set	Check that the ID is defined in the CRS. Note: Although the error appears in the eX_eBPM log, the problem is in the component that sent the message to the e*Insight engine.
ewjx: Exception getMessage(): processOutgoing(): eBPM cannot process XML data: ORA-01407: cannot update ("EX_ADMIN"."BUSINESS_PROCESS_INSTANCE"."BPI_NM") to NULL	EWY		
ERROR - CONTINUING: BP attribute not found :Cust_Addres in createBPI() bpiId:1016	EBPM	Attribute does not exist in e*Insight	Check that the attribute name matches the name defined in the CRS. Note: Although the error appears in the eX_eBPM log, the problem is in the component that sent the message to the e*Insight engine.
(get (eX-get-attribute ~input%eX_Event "Item_Number")) >>>>MONKEXCEPT:0009: get: argument 1 must be a valid path.	MNK	Attribute does not exist in the message sent to an e*Way or BOB for an activity	Check that the attribute is defined as an input attribute for the activity. Also, check that the attribute name matches the name defined in the CRS. Note: This error appears in the log for the e*Way or BOB for the activity.

18.4 General Troubleshooting Tips

18.4.1 Locating the problem

Use the e*Insight GUI in monitor mode to determine whether a business process instance has been created. If the BPI has not been created then you should look at the eX_eBPM log and whatever component created the original message that was sent to the engine. If the BPI has been created then locate the activity that is having a problem and look in the appropriate log files.

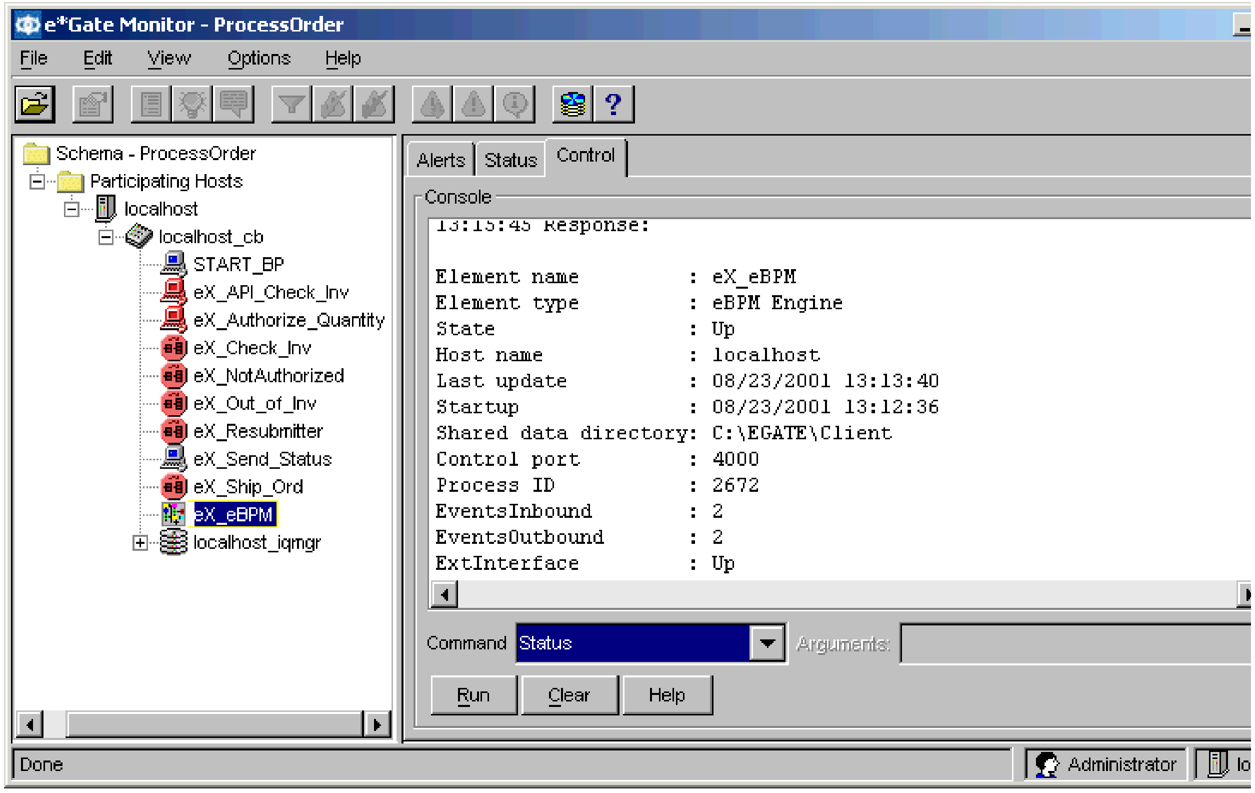
Use the e*Gate Enterprise Monitor GUI to check whether a message has been processed by a component. The number of inbound and outbound messages for a component is displayed by the status command.

To display the status of a component

1. In the navigator pane of the Enterprise Monitor, click the desired component.
2. Click the **Control** tab.
3. From the **Command** drop-down list box, select **Status**.
4. Click **Run**.

   Figure 163 shows an example status display. **EventsInbound** and **EventsOutbound** display the number of messages processed.

Figure 163 Example status display



```
13:15:45 Response:

Element name          : eX_eBPM
Element type          : eBPM Engine
State                 : Up
Host name             : localhost
Last update           : 08/23/2001 13:13:40
Startup               : 08/23/2001 13:12:36
Shared data directory: C:\EGATE\Client
Control port          : 4000
Process ID            : 2672
EventsInbound         : 2
EventsOutbound        : 2
ExtInterface          : Up
```

*Note: The **ExtInterface** value shows whether the e*Insight engine has successfully connected to the database. If the value is "Down" then the connection has not yet been successfully made, even though the e*Insight engine state is shown as "Up".*

18.4.2 Viewing the Message Content

The messages sent to and from the e*Insight engine can be viewed and interpreted. The sample below shows a "START_BP" message sent to the e*Insight engine. You should be able to determine from the message that the business process name is **ProcessOrder,** the ID is **200108231344480169**, and there are a number of global attributes defined (namely, **Cust_Address**, **Cust_Name**, **Cust_email**, **Item_Description**, **Item_Number**, and **Order_Quantity**).

```
13:44:48.219 IQV  D 2488 (iqput.cxx:114): sending to iq manager Data Follows
(bytes 581):
        3C 65 58 5F 45 76 65 6E 74 3E 0A 3C 42 50 5F 45   |   <eX_Event>.<BP_E
        56 45 4E 54 0A 54 59 50 45 3D 22 53 54 41 52 54   |   VENT.TYPE="START
        5F 42 50 22 0A 49 44 3D 22 32 30 30 31 30 38 32   |   _BP".ID="2001082
        33 31 33 34 34 34 38 30 31 36 39 22 0A 4E 41 4D   |   31344480169".NAM
        45 3D 22 50 72 6F 63 65 73 73 4F 72 64 65 72 22   |   E="ProcessOrder"
        3E 0A 3C 41 54 54 52 49 42 55 54 45 0A 56 41 4C   |   >.<ATTRIBUTE.VAL
        55 45 3D 22 34 30 34 20 45 2E 20 48 75 6E 74 69   |   UE="404 E. Hunti
        6E 67 74 6F 6E 20 44 72 2E 22 0A 54 59 50 45 3D   |   ngton Dr.".TYPE=
        22 53 54 52 49 4E 47 22 0A 4E 41 4D 45 3D 22 43   |   "STRING".NAME="C
        75 73 74 5F 41 64 64 72 65 73 73 22 3E 3C 2F 41   |   ust_Address"></A
        54 54 52 49 42 55 54 45 3E 0A 3C 41 54 54 52 49   |   TTRIBUTE>.<ATTRI
        42 55 54 45 0A 56 41 4C 55 45 3D 22 4A 6F 68 6E  |  BUTE.VALUE="John
        20 53 6D 69 74 68 22 0A 54 59 50 45 3D 22 53 54  |   Smith".TYPE="ST
        52 49 4E 47 22 0A 4E 41 4D 45 3D 22 43 75 73 74  |  RING".NAME="Cust
        5F 4E 61 6D 65 22 3E 3C 2F 41 54 54 52 49 42 55  |  _Name"></ATTRIBU
        54 45 3E 0A 3C 41 54 54 52 49 42 55 54 45 0A 56  |  TE>.<ATTRIBUTE.V
        41 4C 55 45 3D 22 6A 73 6D 69 74 68 40 73 65 65  |  ALUE="jsmith@see
        62 65 79 6F 6E 64 2E 63 6F 6D 22 0A 54 59 50 45  |  beyond.com".TYPE
        3D 22 53 54 52 49 4E 47 22 0A 4E 41 4D 45 3D 22  |  ="STRING".NAME="
        43 75 73 74 5F 65 6D 61 69 6C 22 3E 3C 2F 41 54  |  Cust_email"></AT
        54 52 49 42 55 54 45 3E 0A 3C 41 54 54 52 49 42  |  TRIBUTE>.<ATTRIB
        55 54 45 0A 56 41 4C 55 45 3D 22 4D 69 6C 6C 65  |  UTE.VALUE="Mille
        6E 6E 69 75 6D 20 50 65 74 20 52 6F 63 6B 22 0A  |  nnium Pet Rock".
        54 59 50 45 3D 22 53 54 52 49 4E 47 22 0A 4E 41  |  TYPE="STRING".NA
        4D 45 3D 22 49 74 65 6D 5F 44 65 73 63 72 69 70  |  ME="Item_Descrip
        74 69 6F 6E 22 3E 3C 2F 41 54 54 52 49 42 55 54  |  tion"></ATTRIBUT
        45 3E 0A 3C 41 54 54 52 49 42 55 54 45 0A 56 41  |  E>.<ATTRIBUTE.VA
        4C 55 45 3D 22 33 33 33 33 33 22 0A 54 59 50 45  |  LUE="33333".TYPE
        3D 22 53 54 52 49 4E 47 22 0A 4E 41 4D 45 3D 22  |  ="STRING".NAME="
        49 74 65 6D 5F 4E 75 6D 62 65 72 22 3E 3C 2F 41  |  Item_Number"></A
        54 54 52 49 42 55 54 45 3E 0A 3C 41 54 54 52 49  |  TTRIBUTE>.<ATTRI
        42 55 54 45 0A 56 41 4C 55 45 3D 22 31 22 0A 54  |  BUTE.VALUE="1".T
        59 50 45 3D 22 4E 55 4D 42 45 52 22 0A 4E 41 4D  |  YPE="NUMBER".NAM
        45 3D 22 4F 72 64 65 72 5F 51 75 61 6E 74 69 74  |  E="Order_Quantit
        79 22 3E 3C 2F 41 54 54 52 49 42 55 54 45 3E 3C  |  y"></ATTRIBUTE><
        2F 42 50 5F 45 56 45 4E 54 3E 3C 2F 65 58 5F 45  |  /BP_EVENT></eX_E
        76 65 6E 74 3E                                   |  vent>
```

The sample below shows an example of a “Do” message. You should be able to determine from the message that the business process name is **ProcessOrder,** the ID is **200108231344480169,** and the activity name is **Check_Inv**. There are two global attributes defined as input attributes for this activity, **Item_Number** and **Order_Quantity**.

```
14:42:27.716 MNKV D 2228 (monk_extension.cxx:745): Output topic:
eX_Check_Inv_Do
14:42:27.726 MNKV D 2228 (monk_extension.cxx:763): Msg body Data Follows
(bytes 389):
        3C 65 58 5F 45 76 65 6E 74 3E 0A 3C 42 50 5F 45  |  <eX_Event>.<BP_E
        56 45 4E 54 0A 42 50 49 5F 49 44 3D 22 31 36 31  |  VENT.BPI_ID="161
        36 2E 30 2E 32 30 32 3A 22 0A 54 59 50 45 3D 22  |  6.0.202:".TYPE="
        44 4F 5F 41 43 54 49 56 49 54 59 22 0A 49 44 3D  |  DO_ACTIVITY".ID=
        22 32 30 30 31 30 38 32 30 31 34 33 38 30 34 30  |  "200108231344480
        36 34 36 22 0A 4E 41 4D 45 3D 22 50 72 6F 63 65  |  169".NAME="Proce
        73 73 4F 72 64 65 72 22 3E 0A 3C 41 43 54 49 56  |  ssOrder">.<ACTIV
        49 54 59 20 4E 41 4D 45 3D 22 43 68 65 63 6B 5F  |  ITY NAME="Check_
        49 6E 76 22 20 49 44 3D 22 31 36 31 36 2E 35 36  |  Inv" ID="1616.56
        39 22 20 2F 3E 0A 3C 41 54 54 52 49 42 55 54 45  |  9" />.<ATTRIBUTE
        20 4E 41 4D 45 3D 22 49 74 65 6D 5F 4E 75 6D 62  |   NAME="Item_Numb
        65 72 22 20 54 59 50 45 3D 22 53 54 52 49 4E 47  |  er" TYPE="STRING
        22 20 56 41 4C 55 45 3D 22 39 39 39 39 39 22 20  |  " VALUE="99999"
        2F 3E 0A 3C 41 54 54 52 49 42 55 54 45 20 4E 41  |  />.<ATTRIBUTE NA
        4D 45 3D 22 4F 72 64 65 72 5F 51 75 61 6E 74 69  |  ME="Order_Quanti
        74 79 22 20 54 59 50 45 3D 22 4E 75 6D 62 65 72  |  ty" TYPE="Number
        22 20 56 41 4C 55 45 3D 22 31 2E 30 22 20 2F 3E  |  " VALUE="1.0" />
        0A 3C 41 54 54 52 49 42 55 54 45 0A 56 41 4C 55  |  .<ATTRIBUTE.VALU
        45 3D 22 65 42 50 4D 22 0A 4C 4F 43 41 54 49 4F  |  E="eBPM".LOCATIO
        4E 3D 22 45 4D 42 45 44 44 45 44 22 0A 54 59 50  |  N="EMBEDDED".TYP
        45 3D 22 54 52 41 4E 53 49 45 4E 54 22 0A 4E 41  |  E="TRANSIENT".NA
        4D 45 3D 22 65 58 5F 65 42 50 4D 53 65 72 76 65  |  ME="eX_eBPMServe
        72 22 3E 3C 2F 41 54 54 52 49 42 55 54 45 3E 3C  |  r"></ATTRIBUTE><
        2F 42 50 5F 45 56 45 4E 54 3E 3C 2F 65 58 5F 45  |  /BP_EVENT></eX_E
        76 65 6E 74 3E                                   |  vent>
```

Chapter 19

e*Insight Helper Monk Functions

This chapter provides information on the e*Insight Monk APIs. For e*Insight Monk helper functions (used when working with the e*Insight ETD) see **"e*Insight Helper Monk Functions" on page 285**.

19.1 e*Insight Helper Monk Functions

These functions allow you to set information in the e*Insight Event (eX_Standard_Event.ssc ETD) and to get information from it. These functions are contained in the following file:

- **eX-eBPM-utils.monk**

Important: *Make sure that the Monk file **eX-eBPM-utils.monk**, containing the e*Insight helper functions, are loaded before calling them in a Collaboration Rules Script. You can do this in several ways, by putting them in the root of the **monk_library** directory, loading them explicitly in your CRS, or using the **eX-init-eXchange** bootstrap file to load them via the Collaboration Rule.*

These functions are described in detail on the following pages:

eX-get-attribute

Syntax

```
(eX-get-attribute root-path attribute)
```

Description

eX-get-attribute finds the path to the value of the attribute specified in the e*Insight Event named in the root-path.

Parameters

Name	Type	Description
root-path	path	Either ~input%eX_Event or ~output%eX_Event
attribute	string	The name of the attribute as it appears in the e*Insight GUI.

Return Values

Returns one of the following values:

Boolean
Returns **#f** (false) if the attribute value is not found

path

Returns the path to the attribute in the e*Insight Event. Use **get** to return the actual value of the attribute.

Throws

None.

Example

For an Event where the value of Is_Valid_account? is "yes":

```
(get (eX-get-attribute ~input%eX_Event "Is_Valid_account?"))

=> yes
```

eX-count-attribute

Syntax

```
(eX-count-attribute root-path)
```

Description

eX-count-attribute searches the Event specified for attributes, and counts the number of attributes found.

Parameters

Name	Type	Description
root-path	path	Either ~input%eX_Event or ~output%eX_Event

Return Values

integer
Returns 0 to n depending on the number of attributes found.

Throws

None.

Example

For an Event containing three attributes:

```
(eX-count-attribute ~input%eX_Event)

=> 3
```

eX-set-attribute

Syntax

```
(eX-set-attribute root-path attribute value type)
```

Description

If the attribute exists in the Event specified in root-path, **eX-set-attribute** is reset to the new value, otherwise a new entry for the specified attribute is created at the appropriate location in the Event.

Parameters

Name	Type	Description
root-path	path	Either ~input%eX_Event or ~output%eX_Event
attribute	string	The name of the attribute as it appears in the e*Insight GUI.
value	string, XML or BIN	The value to which you want to set the attribute.
type	string	The type of Attribute. Examples: ▪ "BIN" - Interpreted as binary, however, must be suitably encoded for XML. ▪ "XML" - Interpreted as XML, however, must be Base64 encoded for XML. ▪ "STRING" - Interpreted as a string (default). ▪ "TRANSIENT" - Interpreted as a transient. The e*Insight engine does not process the value but simply return it as-is. ▪ "NUMBER" - Interpreted as a decimal number, however, must be given as a string. ▪ "BOOLEAN" - Interpreted as a boolean, such as "true" and "false".

Return Values

None.

Throws

None.

Example

```
(eX-set-attribute ~input%eX_Event "Is_Valid_account?" "no" "STRING")

=> sets the value of Is_Valid_account to "no".
```

eX-set-BP_EVENT

Syntax

```
(eX-set-BP_EVENT root-path event-type value)
```

Description

eX-set-BP_EVENT sets the value of the event type node in the e*Insight Event.

Parameters

Name	Type	Description
root-path	path	Either ~input%eX_Event or ~output%eX_Event
event-type	string	STATUS, ID, NAME, or TYPE
value	string	The value for the business process Event. For event-type "STATUS" value must be either "SUCCESS" or "FAILURE". For event-type "TYPE" value must be DO_ACTIVITY, START_BP, or UNDO_ACTIVITY.

Return Values

Boolean

Returns **#t** (true) except when an invalid parameter is passed, then **#f** (false) is returned.

Throws

None.

Example

```
(eX-set-BP_EVENT ~input%eX_Event "STATUS" "SUCCESS")

=> sets the status of the activity to SUCCESS
```

eX-get-BP_EVENT

Syntax

```
(eX-get-BP_EVENT root-path event-type)
```

Description

eX-get-BP_EVENT finds the path to the value for the event-type in the e*Insight Event named in the root-path.

Parameters

Name	Type	Description
root-path	path	Either ~input%eX_Event or ~output%eX_Event
event-type	string	STATUS, ID, NAME or TYPE

Return Values

Returns one of the following values:

Boolean
Returns **#f** (false) if no data is found.

path
Returns the path to the value in the e*Insight Event. Use **get** to return the actual value.

Throws

None.

Example

For an Event with an ID of 11111:

```
(get (eX-get-BP_EVENT ~input%eX_Event "ID"))
```

```
=> 11111
```

eX-get-Activity

Syntax

```
(eX-get-Activity root-path event-type)
```

Description

eX-get-Activity searches the e*Insight Event specified in for the name or ID of the current activity.

Parameters

Name	Type	Description
root-path	path	Either ~input%eX_Event or ~output%eX_Event
event-type	string	Either ID, NAME

Return Values

Returns one of the following values:

Boolean
Returns **#f** (false) if the requested value is not found.

path
Returns the path to the name of the current activity as found in the e*Insight Event. Use **get** to return the actual value.

Throws

None.

Example

For an Event with an activity name of "Check_Credit":

```
(get (eX-get-Activity ~input%eX_Event "NAME"))
```

```
=> Check_Credit
```

eX-set-Activity

Syntax

```
(eX-set-Activity root-path event-type data)
```

Description

eX-set-Activity sets the value of either the current activity name or ID.

Parameters

Name	Type	Description
root-path	path	Either ~input%eX_Event or ~output%eX_Event
event-type	string	Either ID, NAME
data	string	The value of the activity ID or NAME.

Return Values

None.

Throws

None.

Example

```
(eX-set-Activity ~input%eX_Event "ID" "12345")

=> sets the activity ID to "12345"
```

eX-string-set-attribute

Syntax

```
(eX-string-set-attribute root-path attribute value)
```

Description

eX-string-set-attribute automatically calls **eX-set-attribute** with the last argument as "STRING".

Parameters

Name	Type	Description
root-path	path	Either ~input%eX_Event or ~output%eX_Event
attribute	string	The name of the attribute as it appears in the e*Insight GUI.
value	string	The value to which you want to set the attribute.

Return Values

None.

Throws

None.

Example

```
(eX-sting-set-attribute ~input%eX_Event "Is_Valid_account?" "no")

=> sets the value of Is_Valid_account to "no".
```

eX-xml-set-attribute

Syntax

```
(eX-xml-set-attribute root-path attribute value)
```

Description

eX-xml-set-attribute automatically calls **eX-set-attribute** with last argument as "XML".

Parameters

Name	Type	Description
root-path	path	Either ~input%eX_Event or ~output%eX_Event
attribute	string	The name of the attribute as it appears in the e*Insight GUI.
value	XML	The base 64 encoded XML value to which you want to set the attribute.

Return Values

None.

Throws

None.

Example

```
(eX-xml-set-attribute ~input%eX_Event "Cust_name" (raw->base64
"<a>Bryce Ferney</a>)")

=> sets Cust_name to "PGE+QnJ5Y2UgRmVybmV5PC8+"
```

eX-bin-set-attribute

Syntax

```
(eX-bin-set-attribute root-path attribute value)
```

Description

eX-bin-set-attribute automatically calls **eX-set-attribute** with last argument as "BIN".

Parameters

Name	Type	Description
root-path	path	Either ~input%eX_Event or ~output%eX_Event
attribute	string	The name of the attribute as it appears in the e*Insight GUI.
value	string	The base 64 encoded binary value to which you want to set the attribute.

Return Values

None.

Throws

None.

Example

```
(eX-sting-set-attribute ~input%eX_Event "Cust_name" "<base64 encoded
binary data>")

=> sets Cust_name to the specified value
```

eX-count-local-attribute

Syntax

```
(eX-count-local-attribute root-path)
```

Description

eX-count-local-attribute counts the number of local attributes a specific e*Insight Event contains.

Parameters

Name	Type	Description
root-path	path	Either ~input%eX_Event or ~output%eX_Event

Return Values

integer
Returns 0 to n, depending on the number of attributes found.

Throws

None.

Example

For an Event containing three local attributes:

```
(eX-count-local-attribute ~input%eX_Event)
```

```
=> 3
```

eX-get-local-attribute

Syntax

```
(eX-get-local-attribute root-path attr-name)
```

Description

eX-get-local-attribute finds the path to the specified local attribute in the e*Insight Event named in the root-node.

Parameters

Name	Type	Description
root-path	path	Either ~input%eX_Event or ~output%eX_Event
attr-name	string	The name of the local attribute as it appears in the e*Insight GUI.

Return Values

Returns one of the following values:

Boolean
Returns **#f** (false) if the attribute value is not found.

path

Returns the path to the value of the local attribute. Use **get** to return the actual value.

Throws

None.

Example

For an Event where the value of the local attribute "Debit_Amount" is "500":

```
(get (eX-get-local-attribute ~input%eX_Event "Debit_Amount"))
```

```
=> 500
```

eX-set-local-attribute

Syntax

```
(eX-set-local-attribute root-path attr-name attr-value attr-type)
```

Description

If the local attribute exists in the Event specified in root-path, **eX-set-local-attribute** is reset to the new value; otherwise a new entry for the specified attribute is created at the appropriate location in the Event.

Parameters

Name	Type	Description
root-path	path	Either ~input%eX_Event or ~output%eX_Event
attr-name	string	The name of the attribute as it appears in the e*Insight GUI.
attr-value	string, XML or BIN	The value to which you want to set the attribute.
attr-type	string	The type of Attribute. Examples: ▪ "BIN" - Interpreted as binary, however, must be suitably encoded for XML. ▪ "XML" - Interpreted as XML, however, must be Base64 encoded for XML. ▪ "STRING" - Interpreted as a string (default). ▪ "TRANSIENT" - Interpreted as a transient. The e*Insight engine does not process the value but simply return it as-is. ▪ "NUMBER" - Interpreted as a decimal number, however, must be given as a string. ▪ "BOOLEAN" - Interpreted as a boolean, such as "true" and "false".

Return Values

None.

Throws

None.

Example

```
(eX-set-local-attribute ~input%eX_Event "Debit_Amount" "500"
"STRING")

=> sets the value of the local attribute "Debit_Amount" to 500.
```

eX-copy-no-attribute

Syntax

```
(eX-copy-no-attribute source-root-node dest-root-node)
```

Description

eX-copy-no-attribute copies all of the business process tracking information in the source e*Insight Event to the destination e*Insight Event. No attribute information is copied except for the machine-defined attribute **eX_eBPMServer**, which is used to return the e*Insight Event to the proper e*Insight engine.

The **eX_*Activity*_Do** (or Undo) Event published by the e*Insight engine contains tracking information (such as the business process instance ID, name of the activity, and so on) that must be included in the "Done" Event that is sent back to the e*Insight engine when the activity is finished.

eX-copy-no-attribute provides a convenient way for an activity Collaboration to copy the e*Insight tracking information from source to destination, without copying the input attribute information that does not belong in the "Done" Event. See **"Sending the "Done" Event Back to e*Insight (eIJSchema)" on page 88** for more information on how to use this function.

Parameters

Name	Type	Description
source-root-node	path	~input%eX_Event
dest-root-node	path	~output%eX_Event

Return Values

None.

Throws

None.

Example

```
(eX-copy-no-attribute ~input%eX_Event ~output%eX_Event)

=> copies all data except attribute data
```

eX-set-all-BP_EVENT

Syntax

```
(eX-set-all-BP_EVENT source-root-node type status id name [BPI_ID])
```

Description

eX-set-all-BP_EVENT is used to set all the information in the BP_EVENT node structure at one time.

Parameters

Name	Type	Description
source-root-node	path	Either ~input%eX_Event or ~output%eX_Event
type	string	The type of business process Event. Must be one of the following: "DO_ACTIVITY", "UNDO_ACTIVITY", or "START_BP".
status	string	For types "DO_ACTIVITY" or "UNDO_ACTIVITY" indicates whether the activity completed successfully or not. Must be either "SUCCESS" or "FAILURE". For type "START_BP" this parameter is ignored.
id	string	User-assigned unique identifier for the business process instance.
name	string	Name of the current business process. Must match the name in the e*Insight GUI.
BPI_ID	integer	Machine assigned ID used to speed up processing (optional).

Return Values

None.

Throws

None.

Example

```
(eX-set-all-BP_EVENT ~output%eX_Event "DO_ACTIVITY" "SUCCESS"
"UNIQUE_ID_1002345812" "WebOrder")

=> sets the value of all the BP_EVENT nodes in the output Event.
```

eX-get-all-attribute

Syntax

```
(eX-get-all-attribute source-root-node [attr1 attr2 ... attrN])
```

Description

eX-get-all-attribute provides a way to return a number of paths to attribute values at once, in a list format. The order of paths in the list is the same as the order in which you requested them, that is, path-to-attr1 first, path-to-attr2 second, and so on.

Parameters

Name	Type	Description
source-root-node	path	Either ~input%eX_Event or ~output%eX_Event
attr1	string	The name of the attribute whose path you want to be first in the list.
attr2	string	The name of the attribute whose path you want to be second in the list.
attrN	string	The name of the attribute whose path you want to be the last in the list.

Return Values

Returns one of the following values:

list
Returns a list composed of the paths to the values in the order you specified. If a specified attribute is not found, its value in the list is **#f** (false).

Boolean
Returns **#f** (false) if no attribute names are passed in as parameters.

Throws

None.

Example

```
(get (car (eX-get-all-attribute ~input%eX_Event "Cust_Name"
"Cust_Address" "Cust_e-mail")))

=> "Bryce Ferney"
```

eX-get-all-local-attribute

Syntax

```
(eX-get-all-local-attribute source-root-node [attr1 attr2 ... attrN])
```

Description

eX-get-all-local-attribute provides a way to return a number of paths to local attribute values at once, in a list format. The order of paths in the list is the same as the order in which you requested them, that is, path-to-attr1 first, path-to-attr2 second, and so on.

Parameters

Name	Type	Description
source-root-node	path	Either ~input%eX_Event or ~output%eX_Event
attr1	string	The name of the local attribute whose path you want to be first in the list.
attr2	string	The name of the local attribute whose path you want to be second in the list.
attrN	string	The name of the local attribute whose path you want to be the last in the list.

Return Values

Returns one of the following values:

list

Returns a list composed of the paths to the values in the order you specified. If a specified attribute is not found, its value in the list is **#f** (false).

Boolean

Returns **#f** (false) if no attribute names are passed in as parameters.

Throws

None.

Example

```
(get (car (eX-get-all-local-attribute ~input%eX_Event "Cust_Name"
"Cust_Address" "Cust_e-mail")))

=> "Bryce Ferney"
```

Chapter 20

Java Helper Methods

A number of Java methods have been added to make it easier to set information in the e*Insight Event (ETD) and to get information from it. These methods are contained in classes:

- **"ACTIVITY Class" on page 304**
- **"ATTRIBUTE Class" on page 324**
- **"BP_EVENT Class" on page 342**
- **"eX_StandardEvent Class" on page 378**

20.1 ACTIVITY Class

public class ACTIVITY

extends com.stc.jcsre.XMLETDImpl

implements com.stc.jcsre.ETD

A class to represent the ACTIVITY object of an e*Insight (Business Process Management) XML ETD. It is defined in the following DTD:

```
<!--eBusiness Process Manager Event section-->
 <!ELEMENT BP_EVENT (ACTIVITY?, ATTRIBUTE*)>
 <!ATTLIST BP_EVENT
    TYPE (START_BP | DO_ACTIVITY | UNDO_ACTIVITY | UNDO_BPI |
RESTART_ACTIVITY | SKIP_ACTIVITY | RELOAD_BP | AUTHORIZE |
DONT_AUTHORIZE) #REQUIRED
    STATUS (SUCCESS | FAILURE) #IMPLIED
    NAME CDATA #IMPLIED
    ID CDATA #IMPLIED
    BPI_ID CDATA #IMPLIED
 >
 <!ELEMENT ATTRIBUTE EMPTY>
 <!--ENCODING=base64 or whatever; eBPM only recognizes base64 for
TYPE=XML-->
 <!ATTLIST ATTRIBUTE
    TYPE (BIN | XML | STRING | TRANSIENT | NUMBER | BOOLEAN) #REQUIRED
    NAME CDATA #REQUIRED
    VALUE CDATA #REQUIRED
    ENCODING CDATA #IMPLIED
    LOCATION (FILE | DB | URL | EMBEDDED | AUTO) #IMPLIED
 >
 <!ELEMENT ACTIVITY (ATTRIBUTE*)>
 <!ATTLIST ACTIVITY
    NAME CDATA #IMPLIED
    ID CDATA #IMPLIED
 >
```

These methods are described in detail on the following pages:

addATTRIBUTE

Syntax

```
void addATTRIBUTE(ATTRIBUTE value)
void addATTRIBUTE(int index, ATTRIBUTE value)
```

Description

addATTRIBUTE inserts a new local Attribute into this ACTIVITY object.

Parameters

Name	Type	Description
index	integer	The offset to the list at which insertion occurs (zero-based).
value	ATTRIBUTE	The local Attribute.

Return Values

None.

Throws

None.

clearATTRIBUTE

Syntax

```
void clearATTRIBUTE()
```

Description

clearATTRIBUTE removes all the local Attributes from this ACTIVITY object.

Parameters

None.

Return Values

None.

Throws

None.

Example

```
clearATTRIBUTE();
```

countATTRIBUTE

Syntax

```
int countATTRIBUTE()
```

Description

countATTRIBUTE retrieves the number of local Attributes currently existing in this Activity of the Business Process object.

Parameters

None.

Return Values

integer
Returns the number of global Attributes.

Throws

None.

Example

```
countATTRIBUTE();
   => 5
```

getATTRIBUTE_VALUE

Syntax

```
java.lang.String getATTRIBUTE_VALUE(java.lang.String name)
```

Description

getATTRIBUTE_VALUE retrieves the value of a specific local Attribute by name.

Parameters

Name	Type	Description
name	java.lang.String	The name of the local Attribute.

Return Values

String
Returns the value of the local Attribute. Can be null if the Attribute of that name doesn't exist.

Throws

None.

Example

```
getInstance().getBP_EVENT().getACTIVITY.getATTRIBUTE_VALUE("In_Stock"
);
   => "yes"
```

getATTRIBUTE

Syntax

```
ATTRIBUTE[] getATTRIBUTE()
ATTRIBUTE getATTRIBUTE(int i)
ATTRIBUTE getATTRIBUTE(java.lang.String name)
```

Description

getATTRIBUTE retrieves local Attributes. A specific Attribute can be retrieved by name or by index. Alternatively, it can be used to retrieve all the local Attributes of an Activity of the Business Process as an array.

Parameters

Name	Type	Description
i	integer	The list index of the Attribute to be retrieved (zero-based).
name	string	The name of the local Attribute.

Return Values

Returns one of the following values:

ATTRIBUTE[]
Returns the array of local Attributes.

ATTRIBUTE
Returns the requested Attribute.

Throws

None.

getID

Syntax

```
java.lang.String getID()
```

Description

getID retrieves the internal unique identifier created by e*Insight for this Activity in the Business Process.

Parameters

None.

Return Values

java.lang.String
Returns the unique internal identifier created for this Activity.

Throws

None.

Example

```
getID();
   => "12345"
```

getNAME

Syntax

```
java.lang.String getNAME()
```

Description

getNAME retrieves the case-sensitive name of this Business Process Activity.

Parameters

None.

Return Values

java.lang.String
Returns the case-sensitive name of this ACTIVITY object.

Throws

None.

Example

```
getInstance().getBP_EVENT().getACTIVITY().getNAME();
   => "Ship_Order"
```

hasID

Syntax

```
boolean hasID()
```

Description

hasID checks if there is an unique identifier for this ACTIVITY object.

Parameters

None.

Return Values

boolean
Returns true if there exists an unique ID; otherwise returns false.

Throws

None.

Example

```
hasID();
   => true
```

hasNAME

Syntax

```
boolean hasNAME()
```

Description

hasNAME checks if there exists a name for this ACTIVITY object.

Parameters

None.

Return Values

boolean
Returns true if name exists for this ACTIVITY object; otherwise returns false.

Throws

None.

Example

```
hasNAME();
   => true
```

marshal

Syntax

```
void marshal(org.xml.sax.ContentHandler handler,
org.xml.sax.ErrorHandler errorHandler)
```

Description

marshal gathers the data contained within this ETD object and formulates it back into a serialized XML message.

Parameters

Name	Type	Description
handler	org.xml.sax.ContentHandler	The handler that converts content within to XML.
errorHandler	org.xml.sax.ErrorHandler	The handler to address errors during conversion.

Return Values

None.

Throws

None.

omitID

Syntax

```
void omitID()
```

Description

omitID removes the unique identifier definition for this ACTIVITY object.

Parameters

None.

Return Values

None.

Throws

None.

Example

```
omitID();
```

omitNAME

Syntax

```
void omitNAME()
```

Description

omitNAME removes the name definition from this ACTIVITY object.

Parameters

None

Return Values

None.

Throws

None.

Example

```
omitNAME();
```

removeATTRIBUTE

Syntax

```
void removeATTRIBUTE(java.lang.String name)
void removeATTRIBUTE(int index)
```

Description

removeATTRIBUTE removes a specific local attribute from this ACTIVITY object.

Parameters

Name	Type	Description
name	java.lang.String	The name of the local attribute.
index	int	The index to the list of local attributes (zero-based).

Return Values

None.

Throws

None.

Example

```
removeATTRIBUTE(1);
```

setATTRIBUTE

Syntax

```
void setATTRIBUTE(ATTRIBUTE[] val)
void setATTRIBUTE(int i, ATTRIBUTE val)
void setATTRIBUTE(java.lang.String name java.lang.String value)
void setATTRIBUTE(java.lang.String name, java.lang.String type,
java.lang.String value)
void setATTRIBUTE(java.lang.String name, java.lang.String type,
java.lang.String encoding,java.lang.String value)
void setATTRIBUTE(java.lang.String name, java.lang.String type,
java.lang.String value, java.lang.String encoding,java.lang.String
location)
```

Description

setATTRIBUTE can be used to set all the local Attributes of an Activity of the Business Process, set a local Attribute of an Activity of the Business Process, or set a specific local Attribute by name.

Parameters

Name	Type	Description
val	ATTRIBUTE	The Attribute object.
val	ATTRIBUTE[]	The array of local Attributes.
i	int	The list index of the Attribute to be retrieved (zero-based).
name	java.lang.String	The name of the global Attribute.
value	java.lang.String	The value of the global Attribute.
type	java.lang.String	The type of Attribute. Examples: ▪ "BIN" - Interpreted as binary, however, must be suitably encoded for XML. ▪ "XML" - Interpreted as XML, however, must be Base64 encoded for XML. ▪ "STRING" - Interpreted as a string (default). ▪ "TRANSIENT" - Interpreted as a transient. The e*Insight engine does not process the value but simply return it as-is. ▪ "NUMBER" - Interpreted as a decimal number, however, must be given as a string. ▪ "BOOLEAN" - Interpreted as a boolean, such as "true" and "false".

Name	Type	Description
encoding	java.lang.String	The encoding used for the value. Examples: ▪ "base64' - Standard MIME Base64 encoding. ▪ Null if plain text.
location	java.lang.String	The location of the actual data associated with the Attribute value. Examples: ▪ "FILE" - Attribute value specifies a file where actual data exists. ▪ "DB" - Attribute value is a reference to a row in a database table. ▪ "URL" - Attribute value specifies a URL of where actual data exists. ▪ "EMBEDDED" - Attribute value is the actual data (default). ▪ "AUTO" - Attribute value is actual data but storage in e*Insight is automatically determined.

Return Values

None.

Throws

None.

Example

```
getInstance().getBP_EVENT().getACTIVITY.setATTRIBUTE("In_Stock","STRI
NG","yes");
```

SeeBeyond Proprietary and Confidential

setID

Syntax

```
void setID(java.lang.String val)
```

Description

setID sets the internal unique identifier created by e*Insight for this Activity in the Business Process.

Parameters

Name	Type	Description
val	string	The unique internal identifier created for this Activity.

Return Values

None.

Throws

None.

Example

```
setID("12345");
```

setNAME

Syntax

```
void setNAME(java.lang.String val)
```

Description

setNAME sets the case-sensitive name of this Business Process Activity.

Parameters

Name	Type	Description
val	string	The case-sensitive name of this ACTIVITY object.

Return Values

None.

Throws

None.

Example

```
setNAME("Ship_Order");
```

toString

Syntax

```
java.lang.String toString()
```

Description

toString converts this ETD object to a printable String form.

Parameters

None.

Return Values

java.lang.String
Returns the XML message represent by this ETD object.

Throws

None.

Example

```
toSTRING();
```

unmarshal

Syntax

```
void unmarshal(org.xml.sax.InputSource inputSource,
com.stc.jcsre.sml.SAXLexer lexer)
```

Description

unmarshal takes a serialized (marshalled) form of the ACTIVITY XML Event and distributes (unmarshals) the data into this ETD object.

Parameters

Name	Type	Description
inputSource	org.xml.sax.InputSource	The input source for the serialized data.
lexer	com.stc.jcsre.sml.SAXLexer	The SAX lexer (parser) to distribute the data.

Return Values

None.

Throws

org.xml.sax.SAXException - thrown when the data cannot be parsed

com.stc.jcsre.UnmarshalException - throw when the data cannot be unmarshalled

20.2 ATTRIBUTE Class

public class ATTRIBUTE

extends com.stc.jcsre.XMLETDImpl

implements com.stc.jcsre.ETD

A class to represent the ATTRIBUTE object of an e*Insight (Business Process Management) XML ETD. It is defined in the following DTD:

```
<!--eBusiness Process Manager Event section-->
 <!ELEMENT BP_EVENT (ACTIVITY?, ATTRIBUTE*)>
 <!ATTLIST BP_EVENT
    TYPE (START_BP | DO_ACTIVITY | UNDO_ACTIVITY | UNDO_BPI |
RESTART_ACTIVITY | SKIP_ACTIVITY | RELOAD_BP | AUTHORIZE |
DONT_AUTHORIZE) #REQUIRED
    STATUS (SUCCESS | FAILURE) #IMPLIED
    NAME CDATA #IMPLIED
    ID CDATA #IMPLIED
    BPI_ID CDATA #IMPLIED
 >
 <!ELEMENT ATTRIBUTE EMPTY>
 <!--ENCODING=base64 or whatever; eBPM only recognizes base64 for
TYPE=XML-->
 <!ATTLIST ATTRIBUTE
    TYPE (BIN | XML | STRING | TRANSIENT | NUMBER | BOOLEAN) #REQUIRED
    NAME CDATA #REQUIRED
    VALUE CDATA #REQUIRED
    ENCODING CDATA #IMPLIED
    LOCATION (FILE | DB | URL | EMBEDDED | AUTO) #IMPLIED
 >
 <!ELEMENT ACTIVITY (ATTRIBUTE*)>
 <!ATTLIST ACTIVITY
    NAME CDATA #IMPLIED
    ID CDATA #IMPLIED
 >
```

These methods are described on the following pages:

getENCODING

Syntax

```
java.lang.String getENCODING()
```

Description

getENCODING retrieves the encoding algorithm for the data contained in the Business Process or Activity Attribute. Currently, only the **base64** algorithm is supported. If not defined, clear-text is assumed.

Parameters

None.

Return Values

java.lang.String
Returns the encoding algorithm used on the data.

Throws

None.

Example

```
getENCODING();
   => "base64"
```

getLOCATION

Syntax

```
java.lang.String getLOCATION()
```

Description

getLOCATION retrieves the location type of where the data for an Attribute is actually stored. In cases where the data is too long to be stored in standard database column, it can be stored in another table where the column can be defined as a **LONG RAW** for example, or it may be stored in a file on some file system. In such cases, a reference to the actual data location is stored as the data for the Attribute.

Parameters

None.

Return Values

java.lang.String

Returns the location type for the Attribute data. This is one of the following values:

"FILE"	Attribute data is the name of a file where actual data is stored.
"DB"	Attribute data is a reference such as **ROWID** to a row in a table.
"URL"	Attribute data is the URL to where the actual data is stored.
"EMBEDDED"	Attribute data is the actual data (this is the default).
"AUTO"	The actual data storage location is automatically determined by the e*Insight engine.

Throws

None.

Example

```
getLOCATION();
   => "EMBEDDED"
```

getNAME

Syntax

```
java.lang.String getNAME()
```

Description

getNAME retrieves the name of the Business Process or Activity Attribute.

Parameters

None.

Return Values

java.lang.String
Returns the name of the Attribute.

Throws

None.

Example

```
getInstance().getBP_EVENT().getNAME();
   => "ProcessOrder"
```

getTYPE

Syntax

```
java.lang.String getTYPE()
```

Description

getTYPE Retrieves the type of data stored in the Attribute.

Parameters

None.

Return Values

java.lang.String
Returns the type of data stored as one of the following values:

"BIN"	Attribute data is binary in nature. However, must be safely encoded for XML.
"XML"	Attribute data represents a XML message. However, must be base64 encoded.
"STRING"	Attribute data appears as clear-text string (this is the default).
"TRANSIENT"	Attribute data is not persisted in the e*Insight database; is passed-through.
"NUMBER"	Attribute data represents a decimal numeric string.
"BOOLEAN"	Attribute data represents a boolean string such as "true" or "false".

Throws

None.

Example

```
getInstance().getBP_EVENT().getTYPE();
   => "STRING"
```

getVALUE

Syntax

```
java.lang.String getVALUE()
```

Description

getVALUE retrieves the value of the Business Process or Activity Attribute.

Parameters

None.

Return Values

java.lang.String
Returns the value of the Attribute.

Throws

None.

Example

```
getVALUE();
   => "yes"
```

hasENCODING

Syntax

```
boolean hasENCODNG()
```

Description

hasENCODING checks if the encoding algorithm is defined for this ATTRIBUTE object.

Parameters

None.

Return Values

boolean
Returns true if the encoding algorithm exists; otherwise returns false.

Throws

None.

Example

```
hasENCODING();
   => true
```

hasLOCATION

Syntax

```
boolean hasLOCATION()
```

Description

hasLOCATION checks if the location is defined for this ATTRIBUTE object.

Parameters

None.

Return Values

boolean
Returns true if location exists, otherwise returns false.

Throws

None.

Example

```
hasLOCATION();
   => true
```

marshal

Syntax

```
void marshal(org.xml.sax.ContentHandler handler,
org.xml.sax.ErrorHandler errorHandler)
```

Description

marshal gathers the data contained within this ETD object and formulates it back into a serialized XML message.

Parameters

Name	Type	Description
handler	org.xml.sax.ContentHan dler	The handler that converts content within to XML.
errorHandler	or.xml.sax.ErrorHandler	The handler to address errors during conversion.

Return Values

None.

Throws

com.stc.jcsre.MarshalException

org.xml.sax.SAXException

omitENCODING

Syntax

```
void omitENCODING()
```

Description

omitENCODING removes the encoding algorithm definition for this ATTRIBUTE object.

Parameters

None.

Return Values

None.

Throws

None.

Example

```
omitENCODING();
```

omitLOCATION

Syntax

```
void omitLOCATION()
```

Description

omitLOCATION removes the location definition for this ATTRIBUTE object.

Parameters

None.

Return Values

None.

Throws

None.

Example

```
omitLOCATION();
```

setENCODING

Syntax

```
void setENCODING(java.lang.String val)
```

Description

setENCODING sets the encoding algorithm for the data contained in the Business Process or Activity Attribute. Currently, only the "base64" algorithm is supported. If not defined, clear-text is assumed.

Parameters

Name	Type	Description
val	string	The encoding algorithm used on the data.

Return Values

None.

Throws

None.

Example

```
setENCODING("base64");
```

setLOCATION

Syntax

```
void setLOCATION(java.lang.String val)
```

Description

setLOCATION sets the location type of where the data for an Attribute is actually stored. In cases where the data is too long to be stored in standard database column, it can be stored in another table where the column can be defined as a "LONG RAW" for example, or it may be stored in a file on some file system.

Parameters

Name	Type	Description
val	java.lang.String	The location type for the Attribute data. This can have one the following values: ▪ "FILE" - Attribute data is the name of a file where actual data is stored. ▪ "DB" - Attribute data is a reference such as "ROWID" to a row in a table. ▪ "URL" - Attribute data is the URL to where the actual data is stored. ▪ "EMBEDDED" - Attribute data is the actual data (this is the default). ▪ "AUTO" - The actual data storage location is automatically determined by the e*Insight engine.

Return Values

None.

Throws

None.

Example

```
setLOCATION("EMBEDDED");
```

setNAME

Syntax

```
void setNAME(java.lang.String val)
```

Description

setNAME sets the name of the Business Process or Activity Attribute.

Parameters

Name	Type	Description
val	java.lang.String	The name of the Attribute.

Return Values

None.

Throws

None.

Example

```
setNAME("In_Stock");
```

setTYPE

Syntax

```
void setTYPE(java.lang.String val)
```

Description

setTYPE sets the type of data stored in the Attribute.

Parameters

Name	Type	Description
val	java.lang.String	The type of data stored. This can take one of the following values: ▪ "BIN" - Attribute data is binary in nature. However, must be safely encoded for XML. ▪ "XML" - Attribute data represents a XML message. However, must be base64 encoded. ▪ "STRING"- Attribute data appears as clear-text string (this is the default). ▪ "TRANSIENT" - Attribute data is not persisted in the e*Insight database; is passed-thru. ▪ "NUMBER" - Attribute data represents a decimal numeric string. ▪ "BOOLEAN" - Attribute data represents a boolean string such as "true" or "false".

Return Values

None.

Throws

None.

Example

```
setTYPE("STRING");
```

setVALUE

Syntax

```
void setVALUE(java.lang.String val)
```

Description

setVALUE sets the value of the Business Process or Activity Attribute.

Parameters

Name	Type	Description
val	java.lang.String	The value of the Attribute.

Return Values

None.

Throws

None.

Example

```
setVALUE("yes");
```

toString

Syntax

```
java.lang.String toString()
```

Description

toString Converts this ETD object to a printable String form.

Parameters

None.

Return Values

java.lang.String
Returns the XML message represent by this ETD object.

Throws

None.

Example

```
toSTRING();
```

unmarshal

Syntax

```
void unmarshal(org.xml.sax.InputSource inputSource,
com.stc.jcsre.sml.SAXLexer lexer)
```

Description

unmarshal takes a serialized (marshalled) form of the e*Insight XML Event and distributes (unmarshals) the data into this ETD object.

Parameters

Name	Type	Description
inputSource	org.xml.sax.InputSource	The input source for the serialized data.
lexer	com.stc.jcsre.xml.SAXLexer	The SAX Lexer (parser) to distribute the data.

Return Values

None.

Throws

org.xml.sax.SAXException, when the data cannot be parsed.

com.stc.jcsre.UnmarshalException, when the data cannot be unmarshalled.

20.3 BP_EVENT Class

public class BP_EVENT

extends com.stc.jcsre.AMLETDImpl

implements com.stc.eBIpkg.BPEventETD

The BP_EVENT class represents the e*Insight Business Process Manager section of the SeeBeyond eBI Standard XML ETD which is used to communicate with the e*Insight engine. The DTD is:

```
<!--eBusiness Process Manager Event section-->
<!ELEMENT BP_EVENT (ACTIVITY?, ATTRIBUTE*)>
<!ATTLIST BP_EVENT
        TYPE (START_BP | DO_ACTIVITY | UNDO_ACTIVITY | UNDO_BPI
            |RESTART_ACTIVITY | SKIP_ACTIVITY)
        STATUS (SUCCESS | FAILURE) #IMPLIED
        NAME CDATA #IMPLIED
        ID CDATA #IMPLIED
        BPI_ID CDATA #IMPLIED
>
<!ELEMENT ATTRIBUTE EMPTY>
<!--ENCODING=base64 or whatever; eBPM only recognizes base64 for
        TYPE=XML-->
<!ATTLIST ATTRIBUTE
        TYPE (BIN | XML | STRING | TRANSIENT | NUMBER | BOOLEAN)
            #REQUIRED
        NAME CDATA #REQUIRED
        VALUE CDATA #REQUIRED
        ENCODING CDATA #IMPLIED
        LOCATION (FILE | DB | URL | EMBEDDED | AUTO) #IMPLIED
>
<!ELEMENT ACTIVITY (ATTRIBUTE*)>
<!ATTLIST ACTIVITY
        NAME CDATA #IMPLIED
        ID CDATA #IMPLIED
>
```

These methods are described in detail on the following pages:

hasBPI_ID on page 356
hasID on page 357
hasNAME on page 358
hasSTATUS on page 359
marshal on page 360
setSTATUS on page 374
setTYPE on page 375
toString on page 376
unmarshal on page 377

addATTRIBUTE

Syntax

```
void addATTRIBUTE(int index, ATTRIBUTE value)
```

Description

addATTRIBUTE adds a new global Attribute to this Business Process object.

Parameters

Name	Type	Description
index	integer	(Optional) The offset to the list at which insertion occurs (zero-based).
value	com.stc.eBIpkg.ATTRIBUTE	The global Attribute.

Return Values

None.

Throws

None.

clearATTRIBUTE

Syntax

```
void clearATTRIBUTE()
```

Description

clearATTRIBUTE removes all the global Attributes from this Business Process object.

Parameters

None.

Return Values

None.

Throws

None.

Example

```
clearATTRIBUTE();
```

countATTRIBUTE

Syntax

```
int countATTRIBUTE()
```

Description

countATTRIBUTE retrieves the number of global Attributes currently existing in this Business Process object.

Parameters

None.

Return Values

integer

Returns the number of global Attributes as an integer.

Throws

None

Example

```
countATTRIBUTE();
   => 5
```

getACTIVITY

Syntax

```
ACTIVITY getACTIVITY()
```

Description

getACTIVITY retrieves the current Activity for a Business Process.

Parameters

None.

Return Values

ACTIVITY
Returns the current Activity object.

Throws

None.

getATTRIBUTE_VALUE

Syntax

```
java.lang.String getATTRIBUTE_VALUE(java.lang.String name)
```

Description

getATTRIBUTE_VALUE retrieves the value of a specific global Attribute by name.

Parameters

Name	Type	Description
name	java.lang.String	The name of the global Attribute.

Return Values

java.lang.String

Returns the value of the global Attribute. Can be null if the Attribute of that name does not exist.

Throws

None.

Example

```
getATTRIBUTE_VALUE("In_Stock");
   => "yes"
```

getATTRIBUTE

Syntax

```
com.stc.eBIpkg.ATTRIBUTE[] getATTRIBUTE()
com.stc.eBIpkg.ATTRIBUTE getATTRIBUTE(int i)
com.stc.eBIpkg.ATTRIBUTE getATTRIBUTE(java.lang.String name)
```

Description

getATTRIBUTE retrieves a specific global Attribute by name.

Parameters

Name	Type	Description
i	integer	(Optional) The offset to the list where the Attribute appears.
name	java.lang.String	The global Attribute.

Return Values

Returns one of the following values:

ATTRIBUTE[]
Returns an array of global Attributes if no name or offset were specified.

ATTRIBUTE
Returns the global Attribute if the name or offset were specified.

Throws

None.

getBPI_ID

Syntax

```
java.lang.String getBPI_ID()
```

Description

getBPI_ID retrieves the internal Business Process ID used by the e*Insight engine. It must be returned as-is in a "Done" event back to the e*Insight engine when the active mode is enabled.

Parameters

None.

Return Values

java.lang.String
Returns the internal Business Process ID.

Throws

None.

Example

```
getBPI_ID();
   => "605.0.21"
```

getID

Syntax

```
java.lang.String getID()
```

Description

getID retrieves the Business Process user-assigned unique ID (relative to an ERP, for example).

Parameters

None.

Return Values

java.lang.String
Returns the Business Process user-assigned unique ID.

Throws

None.

Example

```
getID();
   => "12345"
```

getNAME

Syntax

```
java.lang.String getNAME()
```

Description

getNAME retrieves the case-sensitive name of the Business Process.

Parameters

None.

Return Values

java.lang.String
Returns the name of the Business Process.

Throws

None.

Example

```
getNAME();
   => "ProcessOrder"
```

getSTATUS

Syntax

```
java.lang.String getSTATUS()
```

Description

getSTATUS retrieves the status of the current Activity in a Business Process.

Parameters

None.

Return Values

java.lang.String
Returns "SUCCESS" if the current Activity has successfully completed; otherwise returns "FAILURE" if the current Activity has not successfully completed.

Throws

None.

Example

```
getSTATUS();
   => "SUCCESS"
```

getTYPE

Syntax

```
java.lang.String getTYPE()
```

Description

getTYPE retrieves the type of command represented by this Business Process object.

Parameters

None.

Return Values

java.lang.String
Returns one of the following:

"START_BP"	Instructs the e*Insight engine to start a Business Process Instance.
"DO_ACTIVITY"	Indicates a "Do" Event for the current Activity.
"UNDO_ACTIVITY"	Indicates an "Undo" Event for the current Activity.
"UNDO_BPI"	Indicates an "Undo" for the entire Business Process Instance.
"RESTART_ACTIVITY"	Indicates restarting the current Activity after it has paused.
"SKIP_ACTIVITY"	Indicates the current Activity should be skipped.
"RELOAD_BP"	Indicates the Business Process definition should be reloaded.
"AUTHORIZE"	Indicates the current Activity should be authorized.
"DONT_AUTHORIZE "	Indicates that the current Activity should not be authorized.

Throws

None.

Example

```
getTYPE();
   => "DO_ACTIVITY"
```

hasACTIVITY

Syntax

```
boolean hasACTIVITY()
```

Description

hasACTIVITY checks whether the Activity object exists in the Business Process object.

Parameters

None.

Return Values

boolean
Returns true if the Activity object is defined; otherwise returns false.

Throws

None.

Example

```
hasACTIVITY();
   => true
```

hasBPI_ID

Syntax

```
boolean hasBPI_ID()
```

Description

hasBPI_ID checks whether the internal Business Process ID exits in this Business Process object.

Parameters

None.

Return Values

boolean
Returns true if the internal Business Process ID is defined; otherwise returns false.

Throws

None.

Example

```
hasBPI_ID();
   => true
```

hasID

Syntax

```
boolean hasID()
```

Description

hasID checks whether there is a user-assigned unique ID for this Business Process.

Parameters

None.

Return Values

boolean
Returns true if a user-assigned unique ID exits; otherwise returns false.

Throws

None.

Example

```
hasID();
   => true
```

hasNAME

Syntax

```
boolean hasNAME()
```

Description

hasNAME checks whether the name exists in the Business Process object.

Parameters

None.

Return Values

boolean
Returns true if the name is defined; otherwise returns false.

Throws

None.

Example

```
hasNAME();
   => true
```

hasSTATUS

Syntax

```
boolean hasSTATUS()
```

Description

hasSTATUS checks whether there is a status defined.

Parameters

None.

Return Values

boolean
Returns true if there is a status defined; otherwise returns false.

Throws

None.

Example

```
hasSTATUS();
   => true
```

marshal

Syntax

```
void marshal(org.xml.sax.ContentHandler handler,
org.xml.sax.ErrorHandler errorHandler)
```

Description

marshal gathers the data contained within this ETD object and formulates it back into a serialized XML message.

Parameters

Name	Type	Description
handler	org.xml.sax.ContentHan dler	The handler that converts content within to XML.
errorHandler	or.xml.sax.ErrorHandler	The handler to address errors during conversion.

Return Values

None.

Throws

com.stc.jcsre.MarshalException

org.xml.sax.SAXException

omitACTIVITY

Syntax

```
void omitACTIVITY()
```

Description

omitACTIVITY removes the Activity object definition from this Business Process object.

Parameters

None.

Return Values

None.

Throws

None.

Example

```
omitACTIVITY();
```

omitBPI_ID

Syntax

```
void omitBPI_ID()
```

Description

omitBPI_ID removes the internal Business Process ID definition from the Business Process object.

Parameters

None.

Return Values

None.

Throws

None.

Example

```
omitBPI_ID();
```

omitID

Syntax

```
void omitID()
```

Description

omitID removes the user-assigned unique ID definition from the Business Process object.

Parameters

None.

Return Values

None.

Throws

None.

Example

```
omitID();
```

omitNAME

Syntax

```
void omitNAME()
```

Description

omitNAME removes the name definition from the Business Process object.

Parameters

None

Return Values

None.

Throws

None.

Example

```
omitNAME();
```

omitSTATUS

Syntax

```
void omitSTATUS()
```

Description

omitSTATUS removes the status definition from the Business Process object.

Parameters

None.

Return Values

None.

Throws

None.

Example

```
omitSTATUS();
```

removeATTRIBUTE

Syntax

```
void removeATTRIBUTE(java.lang.String name)
void removeATTRIBUTE(int index)
```

Description

removeATTRIBUTE removes a specific global attribute from the Business Process object.

Parameters

Name	Type	Description
name	java.lang.String	The name of the local attribute.
index	int	The index to the list of global attributes (zero-based).

Return Values

None.

Throws

None.

Example

```
removeATTRIBUTE(1);
```

setACTIVITY

Syntax

```
void setACTIVITY(ACTIVITY val)
```

Description

setACTIVITY sets the current Activity of a Business Process.

Parameters

Name	Type	Description
val	ACTIVITY	The current Activity object.

Return Values

None.

Throws

None.

setATTRIBUTE

Syntax

```
void setATTRIBUTE(ATTRIBUTE[] val)
void setATTRIBUTE(int i, ATTRIBUTE val)
void setATTRIBUTE(java.lang.String name java.lang.String value)
void setATTRIBUTE(java.lang.String name, java.lang.String type,
java.lang.String value)
void setATTRIBUTE(java.lang.String name, java.lang.String type,
java.lang.String encoding,java.lang.String value)
void setATTRIBUTE(java.lang.String name, java.lang.String type,
java.lang.String value, java.lang.String encoding,java.lang.String
location)
```

Description

setATTRIBUTE sets a global Attribute of the Business Process.

Parameters

Name	Type	Description
val	ATTRIBUTE[]	The Attribute object.
i	int	The list index of the Attribute to be retrieved (zero-based).
name	java.lang.String	The name of the global Attribute.
value	java.lang.String	The value of the global Attribute.
type	java.lang.String	The type of Attribute. Examples: ▪ "BIN" - Interpreted as binary, however, must be suitably encoded for XML. ▪ "XML" - Interpreted as XML, however, must be Base64 encoded for XML. ▪ "STRING" - Interpreted as a string (default). ▪ "TRANSIENT" - Interpreted as a transient. The e*Insight engine does not process the value but simply return it as-is. ▪ "NUMBER" - Interpreted as a decimal number, however, must be given as a string. ▪ "BOOLEAN" - Interpreted as a boolean, such as "true" and "false".
encoding	java.lang.String	The encoding used for the value. Examples: ▪ "base64' - Standard MIME Base64 encoding. ▪ Null if plain text.

Name	Type	Description
location	java.lang.String	The location of the actual data associated with the Attribute value. Examples: ▪ "FILE" - Attribute value specifies a file where actual data exists. ▪ "DB" - Attribute value is a reference to a row in a database table. ▪ "URL" - Attribute value specifies a URL of where actual data exists. ▪ "EMBEDDED" - Attribute value is the actual data (default). ▪ "AUTO" - Attribute value is actual data but storage in e*Insight is automatically determined.

Return Values

None.

Throws

None.

Example

```
setATTRIBUTE("In_Stock","STRING","yes");
```

setBPI_ID

Syntax

```
void setBPI_ID(java.lang.String val)
```

Description

setBPI_ID sets the internal Business Process ID used by the e*Insight engine. It must be returned as-is in a "Done" event back to the e*Insight engine when the active mode is enabled.

Parameters

Name	Type	Description
val	java.lang.String	The internal Business Process ID.

Return Values

None.

Throws

None.

Example

```
setBPI_ID("605.0.21");
```

setEventInfo

Syntax

```
void setEventInfo(java.lang.String type, java.lang.String status,
java.lang.String id, java.lang.String name)
void setEventInfo(java.lang.String type, java.lang.String status,
java.lang.String id, java.lang.String name, java.lang.String bpi_id)
```

Description

setEventInfo sets the Type, Status, ID and Name information for the Business Process object.

Parameters

Name	Type	Description
type	java.lang.String	The Type of command represented by this Business Process object.
status	java.lang.String	The Status of the current Activity.
id	java.lang.String	The user-assigned unique ID for this Business Process instance.
name	java.lang.String	The Name of the Business Process.
bpi_id	java.lang.String	The internal Business Process Instance ID.

Return Values

None.

Throws

None.

Example

```
setEventInfo("DO_ACTIVITY","SUCCESS","605","ProcessOrder","605.0.21")
;
```

setID

Syntax

```
void setID(java.lang.String val)
```

Description

setID sets the Business Process user-assigned unique ID (relative to an ERP, for example).

Parameters

Name	Type	Description
val	java.lang.String	The Business Process user-assigned unique ID.

Return Values

None.

Throws

None.

Example

```
setID("12345");
```

setNAME

Syntax

```
void setNAME(java.lang.String val)
```

Description

setNAME sets the case-sensitive name of the Business Process.

Parameters

Name	Type	Description
val	java.lang.String	The name of the Business Process.

Return Values

None.

Throws

None.

Example

```
setNAME("ProcessOrder");
```

setSTATUS

Syntax

```
void setSTATUS(java.lang.String val)
```

Description

setSTATUS sets the status of the current Activity in a Business Process.

Parameters

Name	Type	Description
val	java.lang.String	The status of the current Activity in a Business Process. Examples: ▪ "SUCCESS" - Indicates that the current Activity has successfully completed. ▪ "FAILURE" - Indicates that the current Activity has not successfully completed.

Return Values

None.

Throws

None.

Example

```
setSTATUS("SUCCESS");
```

setTYPE

Syntax

```
void setTYPE(java.lang.String val)
```

Description

setTYPE sets the type of command represented by the Business Process object.

Parameters

Name	Type	Description
val	java.lang.String	The command represented by the Business Process object. Examples: ▪ "START_BP" - Instructs the e*Insight engine to start a Business Process Instance. ▪ "DO_ACTIVITY" - Indicates a "Do" Event for the current Activity. ▪ "UNDO_ACTIVITY" - Indicates an "Undo" Event for the current Activity. ▪ "UNDO_BPI" - Indicates an "Undo" for the entire Business Process Instance. ▪ "RESTART_ACTIVITY" - Indicates restarting the current Activity after it has paused. ▪ "SKIP_ACTIVITY" - Indicates the current Activity should be skipped. ▪ "RELOAD_BP" - Indicates the Business Process definition should be reloaded. ▪ "AUTHORIZE" - Indicates that the current Activity should be authorized. ▪ "DONT_AUTHORIZE" - Indicates that the current Activity should not be authorized.

Return Values

None.

Throws

None.

Example

```
setTYPE("DO_ACTIVITY");
```

toString

Syntax

```
java.lang.String toString()
```

Description

toString converts this ETD object to a printable String form.

Parameters

None.

Return Values

java.lang.String
Returns the XML message to represent by this ETD object.

Throws

None.

Example

```
toSTRING();
```

unmarshal

Syntax

```
void unmarshal(org.xml.sax.InputSource inputSource,
com.stc.jcsre.sml.SAXLexer lexer)
```

Description

unmarshal takes a serialized (marshalled) form of the e*Insight XML Event and distributes (unmarshals) the data into this ETD object.

Parameters

Name	Type	Description
inputSource	org.xml.sax.InputSource	The input source for the serialized data.
lexer	com.stc.jcsre.xml.SAXLexer	The SAX Lexer (parser) to distribute the data.

Return Values

None.

Throws

org.xml.sax.SAXException, when the data cannot be parsed.

com.stc.jcsre.UnmarshalException, when the data cannot be unmarshalled.

20.4 eX_StandardEvent Class

public class eX_StandardEvent

extends com.stc.jcsre.SMLETDImpl

implements com.stc.jcsre.ETD

eX_StandardEvent class is an ETD class used to represent the standard XML message that is used to interchange information with the e*Insight Business Process Manager engine. The DTD is:

```
<!ELEMENT eX_Event (BP_EVENT?, TP_EVENT?)>
<!--eBusiness Process Manager Event section-->
<!ELEMENT BP_EVENT (ACTIVITY?, ATTRIBUTE*)>
<!ATTLIST BP_EVENT
        TYPE (START_BP | DO_ACTIVITY | UNDO_ACTIVITY | UNDO_BPI |
            RESTART_ACTIVITY | SKIP_ACTIVITY | RELOAD_BP | AUTHORIZE
            | DONT_AUTHORIZE) #REQUIRED
        STATUS (SUCCESS | FAILURE) #IMPLIED
        NAME CDATA #IMPLIED
        ID CDATA #IMPLIED
        BPI_ID CDATA #IMPLIED
>
<!ELEMENT ATTIBUTE EMPTY>
<!--ENCODING=base64 or whatever; eBPM only recognizes base64 for
    TYPE=XML-->
<!ATTLIST ATTRIBUTE
        TYPE (BIN | XML | STRING | TRANSIENT | NUMBER | BOOLEAN)
            #REQUIRED
        NAME CDATA #REQUIRED
        VALUE CDATA #REQUIRED
        ENCODING CDATA #REQUIRED
        LOCATION (FILE | DB | URL | EMBEDDED | AUTO) #IMPLIED
>
<!ELEMENT ACTIVITY (ATTRIBUTE*)>
<!ATTLIST ACTIVITY
        NAME CDATA #IMPLIED
        ID CDATA #IMPLIED
>
<!--ePartner Manager Input/Output Event section-->
<!ELEMENT TP_EVENT (PartnerName?, InternalName?, Direction?,
    MessageID?, OrigEventC
<!--External Partner Name-->
<!ELEMENT PartnerName (#PCDATA)>
<!--Internal Sending ERP (ex.SAP)-->
<!ELEMENT InternalName (#PCDATA)>
<!--Direction of Transaction to/from Trading Partner (ex.Outbound=O
    Inbound=I)-->
<!ELEMENT Direction (#PCDATA)>
<!--Original request ID from Internal Sending ERP-->
<!ELEMENT MessageID (#PCDATA)>
<!--Original Event Classification (ex.QAP for Query Price and
    Availability)-->
<!ELEMENT OrigEventClass (#PCDATA)>
<!--Usage Indicator of EDI message by Trading Partner (Production=P
    Test=T)-->
<!ELEMENT UsageIndicator (#PCDATA)>
<!--Payload to carry EDI message-->
<!ELEMENT Payload (#PCDATA)>
<!ATTLIST Payload
        TYPE (RAW | PROCESSED | ENCRYPTED) #REQUIRED
        LOCATION (FILE | DB | URL | EMBEDDED | AUTO) #IMPLIED
>
<!--RAW=Need translation PROCESSED=Already X12 or RN ENCRYPTED=from
    Trading Partner>
<!--Communication Protocol (ex. BATCH, HTTP) for sending to Trading
    Partner-->
<!ELEMENT CommProt (#PCDATA)>
<!--URL for EDI message to be exchanged with Trading Partner-->
<!ELEMENT Url (#PCDATA)>
<!--SSL information-->
<!ELEMENT SSLClientKeyFileName (#PCDATA)>
<!ELEMENT SSLClientKeyFileType (#PCDATA)>
<!ELEMENT SSLClientCertFileName (#PCDATA)>
<!ELEMENT SSLClientCertFileType (#PCDATA)>
<!--Message Index for Batched delivery, ex. 1|20 means 1 of 20-->
<!ELEMENT MessageIndex (#PCDATA)>
<!--TP Attribute will contain optional repeating name value pair for
storing of TP-->
<!ELEMENT TPAttribute (NameValuePair*)>
<!ELEMENT NameValuePair (Name, Value)>
<!ELEMENT Name (#PCDATA)>
<!ELEMENT Value (#PCDATA)>
```

These methods are described in detail on the following pages:

from_eBPMConvert

Syntax

```
void from_eBPMConvert(com.stc.eBIpkg.BP_EVENT bpevent
```

Description

from_eBPMConvert converts all pertinent global Attributes of an e*Insight (Business Process) Event back to an e*Xchange (Trading Partner) Event.

Parameters

Name	Type	Description
bpevent	com.stc.eBIpkg.BP_EVENT	The incoming e*Insight Event.

Return Values

None.

Throws

None.

getBP_EVENT

Syntax

```
com.stc.eBIpkg.BP_EVENT getBP_EVENT()
```

Description

getBP_EVENT retrieves the e*Insight (Business Process) portion of the e*Gate Standard XML Event.

Parameters

None.

Return Values

com.stc.eBIpkg.BP_EVENT
Returns the Business Process Event.

Throws

None.

getTP_EVENT

Syntax

```
com.stc.eBIpkg.TP_EVENT getTP_EVENT()
```

Description

getTP_EVENT retrieves the e*Xchange (Trading Partner) portion of the e*Gate Standard XML Event.

Parameters

None.

Return Values

com.stc.eBIpkg.TP_EVENT
Returns the Trading Partner Event.

Throws

None.

hasBP_EVENT

Syntax

```
boolean hasBP_EVENT()
```

Description

hasBP_EVENT tests whether the e*Gate Standard XML Event has an e*Insight (Business Process) portion.

Parameters

None.

Return Values

boolean
Returns true if the Business Process portion exists; otherwise returns false if the Business Process portion does not exist.

Throws

None.

Example

```
hasBP_EVENT();
   => true
```

hasTP_EVENT

Syntax

```
boolean hasTP_EVENT()
```

Description

hasTP_EVENT tests whether the e*Gate Standard XML Event has an e*Xchange (Trading Partner) portion.

Parameters

None.

Return Values

boolean
Returns true if the Trading Partner portion exists; otherwise returns false if the Trading Partner portion does not exist.

Throws

None.

Example

```
hasTP_EVENT();
   => true
```

marshal

Syntax

```
void marshal(org.xml.sax.ContentHandler handler,
org.xml.sax.ErrorHandler errorHandler)
```

Description

marshal gathers the data contained within this ETD object and formulates it back into a serialized XML message.

Parameters

Name	Type	Description
handler	org.xml.sax.ContentHan dler	The handler that converts content within to XML.
errorHandler	org.xml.sax.ErrorHandle r	The handler to address errors during conversion.

Return Values

None.

Throws

com.stc.jcsre.MarshalException

org.xml.sax.SAXException

omitBP_EVENT

Syntax

```
void omitBP_EVENT()
```

Description

omitBP_EVENT removes the e*Insight (Business Process) portion of the e*Gate Standard XML Event.

Parameters

None.

Return Values

None.

Throws

None.

Example

```
omitBP_EVENT();
```

omitTP_EVENT

Syntax

```
void omitTP_EVENT()
```

Description

omitTP_EVENT removes the e*Xchange (Trading Partner) portion of the e*Gate Standard XML Event.

Parameters

None.

Return Values

None.

Throws

None.

Example

```
omitTP_EVENT();
```

setBP_EVENT

Syntax

```
void setBP_EVENT(com.stc.eBIpkg.BP_EVENT val)
```

Description

setBP_EVENT sets the e*Insight (Business Process) portion of the e*Gate Standard XML Event.

Parameters

Name	Type	Description
val	com.stc.eBIpkg.BP_EVENT	The Business Process Event.

Return Values

None.

Throws

None.

setTP_EVENT

Syntax

```
void setTP_EVENT(com.stc.eBIpkg.TP_EVENT val)
```

Description

setTP_EVENT sets the e*Xchange (Partner Manager) portion of the e*Gate Standard XML Event.

Parameters

Name	Type	Description
val	com.stc.eBIpkg.TP_EVENT	The Trading Partner Event.

Return Values

None.

Throws

None.

to_eBPMConvert

Syntax

```
com.stc.eBIpkg.BP_EVENT to_eBPMConvert()
```

Description

to_eBPMConvert converts the e*Gate Standard XML Event entirely to e*Insight (Business Process) portion by saving all the e*Xchange (Trading Partner) information as global Attributes.

Parameters

None.

Return Values

com.stc.eBIpkg.BP_EVENT
Returns the Business Process portion of this ETD object.

Throws

None.

toString

Syntax

```
java.lang.String toString()
```

Description

toString converts this ETD object to a printable String form.

Parameters

None.

Return Values

java.lang.String
Returns the XML message to represent by this ETD object.

Throws

None.

Example

```
toSTRING();
```

unmarshal

Syntax

```
void unmarshal(org.xml.sax.InputSource inputSource,
com.stc.jcsre.sml.SAXLexer lexer)
```

Description

unmarshal takes a serialized (marshalled) form of the e*Insight XML Event and distributes (unmarshals) the data into this ETD object.

Parameters

Name	Type	Description
inputSource	org.xml.sax.InputSource	The input source for the serialized data.
lexer	com.stc.jcsre.xml.SAXLexer	The SAX Lexer (parser) to distribute the data.

Return Values

None.

Throws

org.xml.sax.SAXException, when the data cannot be parsed.

com.stc.jcsre.UnmarshalException, when the data cannot be unmarshalled.

Chapter 21

e*Insight User Activity API Methods

This chapter provides information on the e*Insight User Activity API methods. A number of Java API methods have been added to use with the User Activity object.

User Activities allow external applications to access attributes in the business process using User Activity methods, as described in this chapter. These methods allow the external application to access attributes for the User Activity from the e*Insight database. The e*Insight engine uses the returned value of the attributes to continue the business process.

21.0.1 User Activity Security

Three security checks are performed when connecting to the database using the User Activity methods. First, use the **initialize** method to connect to the database. You should use a user that has no authority to access any of the Business Processes.

Once that connection has been made, use the **authenticate** method to pass the user name and password for a user that has privileges for the Business Process. This user should have the necessary authority for the Business Processes that they are accessing. For subsequent messages sent during the sessionse, use the **setUser** method to re-establish the user security, or **resetUser** to establish security for a new user.

To create a user for the initial connection

1. Use e*Xchange Administrator to create a user (for example, Connection_User), and assign a password.
2. Do not give this user any authorization rights within e*Insight.

Note: *For additional security, create the connection user directly in the database rather than using e*Xchange Administrator.*

The User Activity methods are contained in:

- **"Imessage Interface" on page 395**
- **"UserActivityMessage Class" on page 419**
- **"IClient Interface" on page 420**
- **"EbpmMonitor Class" on page 458**

21.0.2 Defining the Classpath

In order to use the User Activity API methods, you must define the following files in your classpath:

activation.jar	DGutil.jar	stcjcs.jar
antlrall.jar	eIX_StandardEvent.jar	workflow.jar
classes12.zip	jconn2.jar	xerces.jar
DGbase.jar	js.jar	xml.jar
DGsqlserver.jar	mail.jar	
DGsybase.jar	soap.jar	

These files are all located in **<eInsight>\Integrator**.

21.1 Imessage Interface

These methods are described in detail on the following pages:

clearMessage

Syntax

```
void clearMessage()
```

Description

clearMessage clears the message.

Parameters

None.

Return Values

None.

Throws

None.

Example

```
clearMessage();
```

getActivityAttributesCount

Syntax

```
int getActivityAttributesCount()
```

Description

getActivityAttributesCount gets the activity attribute count.

Parameters

None.

Return Values

integer
Returns an integer in the range 0 to n depending on the number of activity attributes.

Throws

None.

Example

```
getActivityAttributesCount();
   => 3
```

getActivityAttributeValue

Syntax

```
java.lang.String getActivityAttributeValue(java.lang.String
attributeName)
```

Description

getActivityAttributeValue is used to determine the value of an activity attribute.

Parameters

Name	Type	Description
attributeName	java.lang.String	The attribute name.

Return Values

java.lang.String
Returns a string containing the activity attribute value.

Throws

None.

Example

```
getActivityAttributesValue(“In_Stock”);
   => “yes”
```

getActivityName

Syntax

```
java.lang.String getActivityName()
```

Description

getActivityName gets the activity name.

Parameters

None.

Return Values

java.lang.String

Returns an string containing the activity name.

Throws

None.

Example

```
getActivityName();
    => “API_Check_Inv”
```

getBusinessModelId

Syntax

```
java.lang.String getBusinessModelId()
```

Description

getBusinessModelId is used to retrieve the business model identifier.

Parameters

None.

Return Values

java.lang.String
Returns a string containing the business model id.

Throws

None.

Example

```
getBusinessModelId();
    => “12345”
```

getBusinessModelInstanceId

Syntax

```
java.lang.String getBusinessModelInstanceId()
```

Description

getBusinessModelInstanceId is used to retrieve the business model instance identifier.

Parameters

None.

Return Values

java.lang.String
Returns a string containing the business model instance id.

Throws

None.

Example

```
getBusinessModelId();
    => “123456789”
```

getBusinessModelName

Syntax

```
java.lang.String getBusinessModelName()
```

Description

getBusinessModelName is used to get the business model name.

Parameters

None.

Return Values

java.lang.String
Returns a string containing the business model name.

Throws

None.

Example

```
getBusinessModelName();
    => “ProcessOrder”
```

getGlobalAttributeCount

Syntax

```
int getGlobalAttributeCount()
```

Description

getGobalAttributeCount is used to get the global attribute count.

Parameters

None.

Return Values

integer
Returns an integer value in the range 0 to n depending on the number of global attributes.

Throws

None.

Example

```
getGlobalAttributeCount();
    => “12”
```

getGlobalAttributeType

Syntax

```
java.lang.String getGlobalAttributeType(java.lang.String
attributeName)
```

Description

getGlobalAttributeType is used to determine the type of an attribute passed in as a parameter.

Parameters

Name	Type	Description
attributeName	java.lang.String	The attribute name.

Return Values

java.lang.String
Returns a string containing the attribute type.

Throws

None.

Example

```
getGlobalAttributeType(“In_Stock”);
    => “String”
```

getGlobalAttributeValue

Syntax

```
java.lang.String getGlobalAttributeValue(java.lang.String
attributeName)
```

Description

getGlobalAttributeType is used to determine the value of an attribute passed in as a parameter.

Parameters

Name	Type	Description
attributeName	java.lang.String	The attribute name.

Return Values

java.lang.String
Returns a string containing the attribute value.

Throws

None.

Example

```
getGlobalAttributeValue("In_Stock");
    => "yes"
```

getMsgType

Syntax

```
java.lang.String getMsgType()
```

Description

getMsgType gets the message type.

Parameters

None.

Return Values

java.lang.String
Returns the message type.

Throws

None.

Example

```
getMsgType();
    => “DO_ACTIVITY”
```

removeActivity

Syntax

```
void removeActivity()
```

Description

removeActivity is used to remove the activity.

Parameters

None.

Return Values

None.

Throws

None.

Example

```
removeActivity();
```

removeGlobalAttribute

Syntax

```
void removeGlobalAttribute(java.lang.String attributeName)
```

Description

removeGlobalAttribute is used to remove the global attribute named as a parameter.

Parameters

Name	Type	Description
attributName	java.lang.String	The attribute name

Return Values

None.

Throws

None.

Example

```
removeGlobalAttribute(“In_Stock”);
```

setActivityAttributeValue

Syntax

```
void setActivityAttributeValue(java.lang.String attributeName,
java.lang.String attributeType, java.lang.String attributeValue)
```

Description

setActivityAttributeValue is used to set the activity attribute value.

Parameters

Name	Type	Description
attributeName	java.lang.String	The attribute name.
attributeType	java.lang.String	The attribute type.
attributeValue	java.lang.String	The attribute value.

Return Values

None.

Throws

None.

Example

```
setActivityAttributeValue(“In_Stock” , “String”, “no”);
```

setActivityName

Syntax

```
void setActivityName(java.lang.String activityName)
```

Description

setActivityName is used to set the activity name.

Parameters

Name	Type	Description
activityName	java.lang.String	The activity name.

Return Values

None.

Throws

None.

Example

```
setActivityName("API_Check_Inv");
```

setBPIStack

Syntax

```
void setBPIStack(java.lang.String bpiStack)
```

Description

setBPIStack sets the business model stack.

Parameters

Name	Type	Description
bpiStack	java.lang.String	The business model stack. For example “637.0.133”.

Return Values

None.

Throws

None.

Example

```
setBPIStack(“637.0.133”);
```

setBusinessModelInstanceId

Syntax

```
void setBusinessModelInstanceId(java.lang.String businessInstanceId)
```

Description

setBusinessModelInstanceId sets the business model instance id.

Parameters

Name	Type	Description
businessInstanceId	java.lang.String	The business model instance ID.

Return Values

None.

Throws

None.

Example

```
setBusinessModelInstanceId("602");
```

setBusinessModelId

Syntax

```
void setBusinessModelId(java.lang.String businessModelId)
```

Description

setBusinessModelId sets the business model id.

Parameters

Name	Type	Description
businessModelId	java.lang.String	The business model id.

Return Values

None.

Throws

None.

Example

```
setBusinessModelId(“12345”);
```

setBusinessModelName

Syntax

```
void setBusinessModelName(java.lang.String businessModelName)
```

Description

setBusinessModelName sets the business model name.

Parameters

Name	Type	Description
businessModelName	java.lang.String	The business model name.

Return Values

None.

Throws

None.

Example

```
setBusinessModelName(“ProcessOrder”);
```

setGlobalAttributeValue

Syntax

```
void setGlobalAttributeValue(java.lang.String attributeName,
java.lang.String attributeValue, java.lang.String attributeType)
```

Description

setGlobalAttributeValue sets the global attribute value.

Parameters

Name	Type	Description
attributeName	java.lang.String	The attribute name.
attributeValue	java.lang.String	The attribute value.
attributeType	java.lang.String	The attribute type.

Return Values

None.

Throws

None.

Example

```
setGlobalAttributeValue(“In_Stock” , “no” , “StringString”);
```

setMsgType

Syntax

```
void setMsgType(java.lang.String msgType)
```

Description

setMsgType sets the message type.

Parameters

Name	Type	Description
msgType	java.lang.String	The message type.

Return Values

None.

Throws

Example

```
setMsgType(“DO_ACTIVITY”);
```

setStatus

Syntax

```
void setStatus(java.lang.String status)
```

Description

setStatus is used to set the message status.

Parameters

Name	Type	Description
status	java.lang.String	The message status.

Return Values

None.

Throws

None.

Example

```
setStatus(“SUCCESS”);
```

toXML

Syntax

```
java.lang.String toXML()
```

Description

toXML converts the message to XML.

Parameters

None.

Return Values

java.lang.String

Returns the message in XML format.

Throws

None.

Example

```
toXML();
   =>  <BP_EVENT NAME="api" STATUS="SUCCESS" ID="231"
       BPI_ID="231.0.21:" TYPE="DO_ACTIVITY"><ATTRIBUTE
       NAME="In_Stock" TYPE="String" VALUE="no" /><ACTIVITY
       NAME="API_Check_Inv" /></BP_EVENT>
```

21.2 UserActivityMessage Class

Implements IMessage.

See **“Imessage Interface” on page 395**.

21.3 IClient Interface

These methods are described in detail on the following pages:

authenticate

Syntax

```
boolean authenticate(java.lang.String userId, java.lang.String
password)
```

Description

authenticate authenticates User Id and password with the e*Insight database.

Parameters

Name	Type	Description
userId	java.lang.String	The User id.
password	java.lang.String	The password.

Return Values

boolean

Returns true if the user Id and password are valid; otherwise returns false.

Throws

java.lang.Exception

Example

```
try {
     String userId = "joe_smith";
     String password = "xxxxx";
     boolean flag = client.authenticate(userId, password);
     System.out.println("flag=" + flag);
 } catch (Exception ex) {
     ex.printStackTrace();
 }
```

checkoutActivityInstance

Syntax

```
java.lang.String checkoutActivityInstance(java.lang.String
businessModelId, java.lang.String businessModelInstanceId,
java.lang.String activityName)
```

Description

checkoutActivityInstance puts a lock on the activity instance by this current userId.

Parameters

Name	Type	Description
businessModelId	java.lang.String	The business model id.
businessModelInstanceId	java.lang.String	The business model instance id.
activityName	java.lang.String	The activtity name.

Return Values

java.lang.String

Returns the userId who is using this actitivy instance.

Throws

java.lang.Exception

Example

```
checkoutActivityInstance(“12345” , “605” , “API_Check_Inv”);
    => “jo_smith”
```

getActivityGlobalAttributeNames

Syntax

```
java.lang.String[] getActivityGlobalAttributeNames(java.lang.String
businessModelId, java.lang.String activityName)
```

Description

getActivityGlobalAttributeNames retrieves a list of global attributes defined for an activity.

Parameters

Name	Type	Description
businessModelId	java.lang.String	The business model id.
activityName	java.lang.String	The activity name

Return Values

java.lang.String

Returns a string[] of activity global attribute names.

Throws

java.lang.Exception

Example

```
getActivityGlobalAttributeNames(“12345” , “API_Check_Inv”);
    => {“Customer_Name”, “In_Stock”}
```

getActivityInstanceEndTime

Syntax

```
java.lang.String getActivityInstanceEndTime(java.lang.String
businessModelInstanceId, java.lang.String businessModelId,
java.lang.String activityName)
```

Description

getActivityInstanceEndTime retrieves the end time of the activity instance on the e*Insight server.

Parameters

Name	Type	Description
businessModelInstanceId	java.lang.String	The business model instance id.
businessModelId	java.lang.String	The business model id.
activityName	java.lang.String	The activity name

Return Values

java.lang.String

Returns the end time of the activity instance.

Throws

java.lang.Exception

Example

```
getActivityInstanceEndTime(“602” , “12345” , “API_Check_Inv”);
    => “2001-05-23 14:31:56”
```

getActivityInstanceStartTime

Syntax

```
java.lang.String getActivityInstanceStartTime(java.lang.String
businessModelInstanceId, java.lang.String businessModelId,
java.lang.String activityName)
```

Description

getActivityInstanceStartTime retrieves the start time of the activity instance on the e*Insight server.

Parameters

Name	Type	Description
businessModelInstanceId	java.lang.String	The business model instance id.
businessModelId	java.lang.String	The business model id.
activityName	java.lang.String	The activity name

Return Values

java.lang.String

Returns the start time of the activity instance.

Throws

java.lang.Exception

Example

```
getActivityInstanceEndTime(“602” , “12345” , “API_Check_Inv”);
    => “2001-05-23 14:31:57”
```

getActivityInstanceStatus

Syntax

```
java.lang.String getActivityInstanceStatus(java.lang.String
businessModelInstanceId, java.lang.String businessModelId,
java.lang.String activityName, java.lang.String activityType)
```

Description

getActivityInstanceStatus retrieves the status of the activity instance on the e*Insight server.

Parameters

Name	Type	Description
businessModelInstanceId	java.lang.String	The business model instance id.
businessModelId	java.lang.String	The business model id.
activityName	java.lang.String	The activity name
activityType	java.lang.String	The activity type

Return Values

java.lang.String

Returns the status of the activity instance.

Throws

java.lang.Exception

Example

```
getActivityInstanceEndTime(“602”, “12345”, “API_Check_Inv”, “USER”);
    => “pending”
```

getActivityNames

Syntax

```
java.lang.String[] getActivityNames(java.lang.String businessModelId)
```

Description

getActivityNames gets the activity names.

Parameters

Name	Type	Description
businessModelId	java.lang.String	The business model id.

Return Values

java.lang.String

Returns an string[] containing the activity names.

Throws

None.

Example

```
getActivityNames("12345");
    => {"API_Check_Inv", "Send_Status"}
```

getAssignedBPIIdByState

Syntax

```
java.lang.String[] getActivityNames(java.lang.String businessModelId,
java.lang.String activityName, java.lang.String status)
```

Description

getAssignedBPIIdByState gets the business process instance ids of all business process instances in a particular state.

Parameters

Name	Type	Description
businessModelId	java.lang.String	The business model id.
activityName	java.lang.String	The activity name.
status	java.lang.String	The activity status.

Return Values

java.lang.String

Returns an string[] containing the business process instance ids.

Throws

None.

Example

```
getAssignedBPIIdByState("Pending");
    => {"603", "604"}
```

getAuthorizationActivityNames

Syntax

```
java.lang.String[] getAuthorizationActivityNames(java.lang.String
businessModelId)
```

Description

getAuthorizationActivityNames gets a String[] of authorization acitivity names.

Parameters

Name	Type	Description
businessModelId	java.lang.String	The business model id.

Return Values

java.lang.String[]

Returns the authorization acitivity names.

Throws

java.lang.Exception

Example

```
getAuthorizationActivityNames(“12345”);
    => {“Authorize_Quantity”, “Authorize_Total”}
```

getBPIStack

Syntax

```
java.lang.String getBPIStack(java.lang.String businessInstanceId)
```

Description

getBPIStack gets the business model instance stack.

Parameters

Name	Type	Description
businessInstanceId	java.lang.String	The business model instance id.

Return Values

java.lang.String

The business model instance stack. For example, "637.0.133:".

Throws

java.lang.Exception

Example

```
getBPIStack();
    =>("637.0.133")
```

getBusinessModelInstancesIds

Syntax

```
java.lang.String[] getBusinessModelInstancesIds(java.lang.String
businessModelId)
```

Description

getBusinessModelInstancesIds gets a String[] of business instance ids.

Parameters

Name	Type	Description
businessModelId	java.lang.String	The business model id.

Return Values

java.lang.String[]

Returns the business instance ids.

Throws

java.lang.Exception

Example

```
getBusinessModelInstanceIds(“12345”);
    => {“602”, “603”}
```

getBusinessModelInstanceName

Syntax

```
java.lang.String getBusinessModelInstanceName(java.lang.String
businessModelId)
```

Description

getBusinessModelInstanceName retrieves the business model instance name.

Parameters

Name	Type	Description
businessModelId	java.lang.String	The business model id.

Return Values

java.lang.String

Returns the business instance model instance name.

Throws

java.lang.Exception

Example

```
getBusinessModelInstanceName(“12345”);
    => {“bp_603”}
```

getBusinessModelInstanceStatus

Syntax

```
java.lang.String[] getBusinessModelInstanceStatus(java.lang.String
businessModelId)
```

Description

getBusinessModelInstancesIds gets the status of business model instance.

Parameters

Name	Type	Description
businessModelId	java.lang.String	The business model id.

Return Values

java.lang.String

Returns the status of a business instance.

Throws

java.lang.Exception

Example

```
getBusinessModelInstanceIds(“12345”);
    => “Pending”
```

getBusinessModelName

Syntax

```
java.lang.String getBusinessModelName(java.lang.String
businessModelName)
```

Description

getBusinessModelName gets the business model name.

Parameters

Name	Type	Description
businessModelName	java.lang.String	The business model name.

Return Values

java.lang.String

Returns the business model name.

Throws

java.lang.Exception

Example

```
getBusinessModelName();
    => “ProcessOrder”
```

getEnabledBusinessModelId

Syntax

```
java.lang.String getEnabledBusinessModelId(java.lang.String
businessModelName)
```

Description

getEnabledBusinessModelId gets the enabled business model id.

Parameters

Name	Type	Description
businessModelName	java.lang.String	The business model name.

Return Values

java.lang.String
Returns the business model id.

Throws

java.lang.Exception

Example

```
getEnabledBusinessModelId();
    => “12345”
```

getEnabledBusinessModelsIds

Syntax

```
java.lang.String[] getEnabledBusinessModelsIds()
```

Description

getEnabledBusinessModelsIds gets a String[] of enabled business model ids.

Parameters

None.

Return Values

java.lang.String[]
Returns the enabled business model ids.

Throws

java.lang.Exception

Example

```
getEnabledBusinessModelsIds();
    => { “12345”, “12386”}
```

getGlobalAttributeDefaultValue

Syntax

```
java.lang.String getGlobalAttributeDirection(java.lang.String
businessModelId, java.lang.String attributeName)
```

Description

getGlobalAttributeDefaultValue gets the business model attribute default value.

Parameters

Name	Type	Description
businessModelId	java.lang.String	The business model id.
attributeName	java.lang.String	The attribute name.

Return Values

java.lang.String
Returns a String containing the global attribute default value.

Throws

java.lang.Exception

Example

```
getGlobalAttributeDefaultValue( “In_Stock”);
    => “no”
```

getGlobalAttributeDirection

Syntax

```
java.lang.String getGlobalAttributeDirection(java.lang.String
businessModelId, java.lang.String activityName, java.lang.String
attributeName)
```

Description

getGlobalAttributeDirection gets the business model attribute direction.

Parameters

Name	Type	Description
businessModelId	java.lang.String	The business model id.
activityName	java.lang.String	The activity name.
attributeName	java.lang.String	The attribute name.

Return Values

java.lang.String
Returns a String containing the global attribute direction.

Throws

java.lang.Exception

Example

```
getGlobalAttributeDirection( “12345”,“API_Check_Inv”,“In_Stock”);
    => “OUTPUT”
```

SeeBeyond Proprietary and Confidential

getGlobalAttributeNames

Syntax

```
java.lang.String[] getGlobalAttributeNames(java.lang.String
businessModelId)
```

Description

getGlobalAttributeNames gets a String[] of global attribute names.

Parameters

Name	Type	Description
businessModelId	java.lang.String	The business model id.

Return Values

java.lang.String[]
Returns global attribute names.

Throws

java.lang.Exception

Example

```
getGlobalAttributeNames();
    => { “Customer_Name”,“In_Stock”}
```

getGlobalAttributeType

Syntax

```
java.lang.String getGlobalAttributeType(java.lang.String
businessModelId, java.lang.String attributeName)
```

Description

getGlobalAttributeType gets the business model attribute type.

Parameters

Name	Type	Description
businessModelId	java.lang.String	The business model id.
attributeName	java.lang.String	The attribute name.

Return Values

java.lang.String
Returns a global attribute type.

Throws

java.lang.Exception

Example

```
getGlobalAttributeType(“12345”,“In_Stock”);
    => “String”
```

getGlobalAttributeValue

Syntax

```
java.lang.String getGlobalAttributeValue(java.lang.String
businessInstanceId, java.lang.String businessModelId,
java.lang.String attributeName)
```

Description

getGlobalAttributeValue gets the business model attribute value.

Parameters

Name	Type	Description
businessInstanceId	java.lang.String	The business model instance id.
businessModelId	java.lang.String	The business model id.
attributeName	java.lang.String	The attribute name.

Return Values

java.lang.String
Returns a global attribute value.

Throws

java.lang.Exception

Example

```
getGlobalAttributeValue( “602”,“12345”,“In_Stock”);
    => “yes”
```

getLocalAttributeNames

Syntax

```
java.lang.String[] getLocalAttributeNames(java.lang.String
businessModelId, java.lang.String activityId)
```

Description

getLocalAttributeNames is used to retrieve activity attribute names.

Parameters

Name	Type	Description
businessModelId	java.lang.String	The business model id.
activityId	java.lang.String	The activity id.

Return Values

java.lang.String
Returns the local attribute names.

Throws

java.lang.Exception

Example

```
getLocalAttributeNames("12345","123");
    => { "Customer_Temp_Id","Previous_Customer"}
```

getLocalAttributeType

Syntax

```
java.lang.String getLocalAttributeType(java.lang.String
businessModelId, java.lang.String activityName, java.lang.String
attributeName)
```

Description

getLocalAttributeType gets a String of local attribute type.

Parameters

Name	Type	Description
businessModelId	java.lang.String	The business model id.
activityName	java.lang.String	The activity name.
attributeName	java.lang.String	The attribute name.

Return Values

java.lang.String

Returns the local attribute type.

Throws

java.lang.Exception

Example

```
getGlobalAttributeType( “12345”,“API_Check_Inv”,“Previous_Customer”);
    => “String”
```

getLocalAttributeValue

Syntax

```
java.lang.String getLocalAttributeValue(java.lang.String
businessInstanceId, java.lang.String businessModelId,
java.lang.String activityName, java.lang.String attributeName)
```

Description

getLocalAttributeValue retrieves local attribute value.

Parameters

Name	Type	Description
businessInstanceId	java.lang.String	The business model instance id.
businessModelId	java.lang.String	The business model id.
activityName	java.lang.String	The activity name.
attributeName	java.lang.String	The attribute name.

Return Values

java.lang.String
Returns the local attribute value.

Throws

java.lang.Exception

Example

```
getGlobalAttributeValue(“602”, “12345”,“API_Check_Inv”,
“Previous_Customer”);
    => “yes”
```

getMessageStatus

Syntax

```
java.lang.String getMessageStatus(java.lang.String msgId)
```

Description

getMessageStatus gets the status state of the message sent to the e*Insight server.

Parameters

Name	Type	Description
msgId	java.lang.String	The message id.

Return Values

java.lang.String
Returns the status of the message.

Throws

java.lang.Exception

Example

```
getMessageStatus(“99” );
    => “unprocessed”
```

getUser

Syntax

```
java.lang.String getUser()
```

Description

getUser is used to get the User id.

Parameters

None.

Return Values

java.lang.String
Returns the User id.

Throws

None.

Example

```
getUser();
    => “ex_admin”
```

getUserActivityNames

Syntax

```
java.lang.String[] getUserActivityNames(java.lang.String
businessModelId)
```

Description

getUserActivityNames gets the user activity names.

Parameters

Name	Type	Description
businessModelId	java.lang.String	The business model id.

Return Values

java.lang.String[]
Returns the user activity names.

Throws

java.lang.Exception

Example

```
getUserActivityNames(“12345”);
    => { “API_Check_Inv”, “Send_Order”}
```

getUUID

Syntax

```
java.lang.String[] getUUID ()
```

Description

getUUID retrieves the UUID.

Parameters

None.

Return Values

java.lang.String
Returns the UUID.

Throws

None.

Example

```
getUUID();
```

initialize

Syntax

```
void initialize(java.util.Properties p)
```

Description

initialize establishes a connection to the e*Insight engine and database. The user needs to provide the appropriate e*Insight connection property configuration.

Parameters

Name	Type	Description

p	Properties	The e*Insight configuration properties.
		For example:
		dbURL=jdbc:oracle:thin:@hostname:1521:dbname
		userID=ex_admin
		password=xxxxx
		driverName=oracle.jdbc.driver.OracleDriver
		DBServerType=Oracle

Return Values

None.

Throws

java.lang.Exception

Example

```
Properties p = new Properties();
p.setProperty("dbURL", "jdbc:oracle:thin:@localhost:1521:eXchange");
p.setProperty("userID", "ex_admin");
p.setProperty("password", "ex_admin");
p.setProperty("driverName", "oracle.jdbc.driver.OracleDriver");
p.setProperty("DBServerType", "Oracle");
initialize( p );
```

refreshCachedMemory

Syntax

```
void refreshCachedMemory()
```

Description

refreshCachedMemory refreshes the cached memory used by this client.

Parameters

None.

Return Values

None.

Throws

java.lang.Exception

Example

```
refreshCachedMemory();
```

releaseActivityInstance

Syntax

```
void releaseActivityInstance(java.lang.String businessModelId,
java.lang.String businessModelInstanceId, java.lang.String
activityName)
```

Description

releaseResources releases the usage on the activity instance by this current userId.

Parameters

Name	Type	Description
businessModelId	java.lang.String	The business model id.
businessModelInstanceId	java.lang.String	The business model instance id.
activityName	java.lang.String	The activtity name.

Return Values

java.lang.String

Returns the userId of the user who is using this activity instance. Returns null if the current userId did not checkout the activity instance.

Throws

java.lang.Exception

Example

```
releaseActivityInstance("12345", "605", "API_Check_Inventory");
    => "jo_smith"
```

releaseResources

Syntax

```
void releaseResources()
```

Description

releaseResources releases the resource used by the client. This method should be called after all executions are complete.

Parameters

None.

Return Values

None.

Throws

java.lang.Exception

Example

```
releaseResources();
```

resetUser

Syntax

```
void resetUser()
```

Description

resetUser resets the user.

Parameters

None.

Return Values

None.

Throws

None.

Example

```
resetUser();
```

sendMessage

Syntax

```
java.lang.String sendMessage(IMessage msg)
```

Description

sendMessage sends an e*Insight message to the e*Insight server for processing. The message will be sent to the e*Insight server queue.

Parameters

Name	Type	Description
msg	IMessage	An e*Insight message.

Return Values

java.lang.String
Returns an acknowledge message id sent to the e*Insight server queue.

Throws

java.lang.Exception

Example

```
IMessage msg = new UserActivityMessage();
msg.setBusinessModelId(act.bpoId);
msg.setBusinessModelInstanceId(bpiId);
msg.setBusinessModelName(act.bpoName);
msg.setActivityName(act.bpoName);
sendMessage(msg);
    => “99”
```

setGlobalAttributeValue

Syntax

```
void setGlobalAttributeValue(java.lang.String businessInstanceId,
java.lang.String businessModelId, java.lang.String attributeName,
java.lang.String attributeValue, java.lang.String attributeType)
```

Description

setGlobalAttributeValue sets a global attribute value.

Parameters

Name	Type	Description
businessInstanceId	java.lang.String	The business model instance id.
businessModelId	java.lang.String	The business model id.
attributeName	java.lang.String	The attribute name.
attributeValue	java.lang.String	The attribute value.
attributeType	java.lang.String	The attribute type.

Return Values

None.

Throws

java.lang.Exception

Example

```
setGlobalAttributeValue(“602” , “12345” , “In_Stock” , “no” , “String”);
```

setLocalAttributeValue

Syntax

```
void setLocalAttributeValue(java.lang.String businessInstanceId,
java.lang.String businessModelId, java.lang.String activityName,
java.lang.String attributeName, java.lang.String attributeValue,
java.lang.String attributeType)
```

Description

setLocalAttributeValue sets the local attribute value.

Parameters

Name	Type	Description
businessInstanceId	java.lang.String	The business model instance id.
businessModelId	java.lang.String	The business model id.
activityName	java.lang.String	The activity name.
attributeName	java.lang.String	The attribute name.
attributeValue	java.lang.String	The attribute value.
attributeType	java.lang.String	The attribute type.

Return Values

None.

Throws

java.lang.Exception

Example

```
setLocalAttributeValue(“602”, “12345”, “API_Check_Inv”, “In_Stock”, “no”,
“String”);
```

setUser

Syntax

```
void setUser(java.lang.String userId)
```

Description

setUser sets the user to connect to e*Insight server.

Parameters

Name	Type	Description
userId	java.lang.String	The user id.

Return Values

None.

Throws

java.lang.Exception

Example

```
setUser(“ex_admin”);
```

21.4 EbpmMonitor Class

This is the e*Insight monitor client to the e*Insight engine. The user can access the e*Insight business model with sufficient user privileges. This class also allows the user to send e*Insight messages to the e*Insight engine.

The EbpmMonitor Class implements IClient interface. See **IClient Interface** on page 420, for details of methods contained in this interface.

Additional methods are described in detail on the following pages:

checkUserPrivileges on page 459

checkUserPrivileges

Syntax

```
void checkUserPrivileges(int bpoId)
```

Description

checkUserPrivileges checks user privileges for bpoId. Throws InsufficientPrivilegesException if userId is not set properly or does not have sufficient privileges to bpoId.

Parameters

Name	Type	Description
bpoId	integer	The business process object id.

Return Values

None.

Throws

java.sql.SQLException, InsufficientPrivilegesException

Example

```
checkUserPrivileges(“123”);
```

Appendix A

XML Structure for the e*Insight Event

This appendix shows the XML structure for the **e*Insight** Event Type Definition. If your data conforms to this structure, you do not need to convert it upon entry to the e*Insight system.

Note: *BP_EVENT attribute names must match the attributes defined in the business process created in the e*Insight GUI. These names are case-sensitive. If the attributes defined in the e*Insight GUI do not match the incoming XML structure you must create a Collaboration to map the incoming data to the correct attribute names; otherwise, the e*Insight system will not function correctly.*

A.1 XML Structure

```
<<!-- edited with XML Spy v3.0 NT (http://www.xmlspy.com) by STC (STC)
-->
<!--DTD for eX_Standard_Event.ssc   $Id: eX_event.dtd,v 1.1.2.10
2000/09/07 04:43:14 galbers Exp $-->
<!ELEMENT eX_Event (BP_EVENT?, TP_EVENT?)>
<!--eBusiness Process Manager Event section-->
<!ELEMENT BP_EVENT (ACTIVITY?, ATTRIBUTE*)>
<!ATTLIST BP_EVENT
    TYPE (START_BP | DO_ACTIVITY | UNDO_ACTIVITY) #REQUIRED
    STATUS (SUCCESS | FAILURE) #IMPLIED
    NAME CDATA #IMPLIED
    ID CDATA #IMPLIED
    BPI_ID CDATA #IMPLIED
>
<!ELEMENT ATTRIBUTE EMPTY>
<!--ENCODING=base64 or whatever; eBPM only recognizes base64 for
TYPE=XML-->
<!ATTLIST ATTRIBUTE
    TYPE (BIN | XML | STRING | TRANSIENT | NUMBER | BOOLEAN) #REQUIRED
    NAME CDATA #REQUIRED
    VALUE CDATA #REQUIRED
    ENCODING CDATA #IMPLIED
    LOCATION (FILE | DB | URL | EMBEDDED | AUTO) #IMPLIED
>
<!ELEMENT ACTIVITY (#PCDATA | ATTRIBUTE)*>
<!ATTLIST ACTIVITY
    NAME CDATA #IMPLIED
    ID CDATA #IMPLIED
>
<!--ePartner Manager Input/Output Event section-->
<!ELEMENT TP_EVENT (PartnerName?, InternalName?, Direction?,
MessageID?, OrigEventClass?, UsageIndicator?, Payload?, CommProt?,
Url?, SSLClientKeyFileName?, SSLClientKeyFileType?,
SSLClientCertFileName?, SSLClientCertFileType?, MessageIndex?,
TPAttribute?)>
<!--External Partner Name-->
<!ELEMENT PartnerName (#PCDATA)>
<!--Internal Sending ERP (ex. SAP)-->
<!ELEMENT InternalName (#PCDATA)>
<!--Direction of Transaction to/from Trading Partner (ex. Outbound=O
Inbound=I)-->
<!ELEMENT Direction (#PCDATA)>
<!--Original Request ID from Internal Sending ERP-->
<!ELEMENT MessageID (#PCDATA)>
<!--Original Event Classification (ex. QAP for Query Price and
Availability)-->
<!ELEMENT OrigEventClass (#PCDATA)>
<!--Usage Indicator of EDI message by Trading Partner (Production=P
Test=T)-->
<!ELEMENT UsageIndicator (#PCDATA)>
<!--Payload to carry EDI message-->
<!ELEMENT Payload (#PCDATA)>
<!ATTLIST Payload
    TYPE (RAW | PROCESSED | ENCRYPTED) #REQUIRED
    LOCATION (FILE | DB | URL | EMBEDDED | AUTO) #IMPLIED
>
<!--RAW=Need translation  PROCESSED=Already X12 or RN  ENCRYPTED=from
Trading Partner-->
<!--Communication Protocol (ex. BATCH, HTTP) for sending to Trading
Partner-->
<!ELEMENT CommProt (#PCDATA)>
<!--URL for EDI message to be exchanged with Trading Partner-->
<!ELEMENT Url (#PCDATA)>
<!--SSL information-->
<!ELEMENT SSLClientKeyFileName (#PCDATA)>
<!ELEMENT SSLClientKeyFileType (#PCDATA)>
<!ELEMENT SSLClientCertFileName (#PCDATA)>
<!ELEMENT SSLClientCertFileType (#PCDATA)>
<!--Message Index for Batched delivery, ex. 1|20 means 1 of 20-->
<!ELEMENT MessageIndex (#PCDATA)>
<!--TP Attribute will contain optional repeating name value pair for
storing of TP data -->
<!ELEMENT TPAttribute (NameValuePair*)>
<!ELEMENT NameValuePair (Name, Value)>
<!ELEMENT Name (#PCDATA)>
<!ELEMENT Value (#PCDATA)>
```

Glossary

attribute
Attributes pass user-defined control information (programming arguments) to and from the e*Insight Business Process Manager and its activities.

activity
An activity is an organizational unit for performing a specific function.

activity states
Activity states are the stages that activities within the business process instance go through as the business process version is being run.

business process
A business process is a collection of actions and messages, revolving around a specific business practice, that flow in a specific pattern to produce an end result.

business process attributes
Business process attributes pass user-defined control information (programming arguments) to and from the e*Insight Business process manager, external sources, and internal components.

business process expression
Business process expressions allow you to dictate business process logic flow based on the ability to perform various types of logic on business process instance attributes.

business process instance (BPI)
A unique instantiation of a business process.

business process version
A form or variant of the original business process model.

Collaboration
A component of an e*Way or BOB that receives and processes Events and forwards the output to other e*Gate components. Collaborations perform three functions: they subscribe to Events of a known type, they apply business rules to Event data, and they publish output Events to a specified recipient. Collaborations use Monk translation script files with the extension "**.tsc**" to do the actual data manipulation.

Design mode
The mode used during the design phase of the business process. Design mode allows you access to the drawing canvas, enabling you to create or modify a business process version, based on its status.

Diagram pane

The Diagram pane is used in Monitor mode to review the status of a business process instance, using a pictorial representation of it. Various colors assigned to the activities within the instance inform you of the status of each, during the cycle of the instance.

documentation box

The documentation box displays comments and free-text descriptions about the business process version.

e*Insight Business Process Manager (e*Insight)

The component within the SeeBeyond eBusiness Integration Suite product suite that facilitates the automation of the business process flow of eBusiness activities.

e*Xchange Partner Manager (e*Xchange)

An application that allows you to set up and maintain trading partner profiles and view processed messages. e*Xchange also processes inbound and outbound messages according to certain eBusiness protocols and your validation Collaborations.

eSecurity Manager (eSM)

An application that is used with the e*Xchange Partner Manager to secure transmission of business-to-business exchanges over public domains such as the Internet.

Event (Message)

Data to be exchanged, either within e*Gate or between e*Gate and external systems, which has a defined data structure; for example, a known number of fields, with known characteristics and delimiters. Events are classified by type and exchanged within e*Gate using Event Type Definitions.

Event Type Definition (ETD)

An Event Type template, defining Event fields, field sequences, and delimiters. Event Type Definitions enable e*Insight systems to identify and transform Event Types. They are Monk script files with an extension of **ssc** and Java script files with an extension of **xsc**.

e*Xchange Administrator

An application within the eBusiness Integration Solutions suite of products that you use to establish user security for e*Xchange Partner Manager (e*Xchange) and e*Insight Business Process Manager (e*Insight).

gate

Gates control the logical flow of data-based decisions in the business process model. A gate outputs specific information when specified input conditions are met.

Graph Wizard

The Graph Wizard is used in Monitor mode to display custom graphs, based on instance data.

GUI

Graphical User Interface. A type of computer interface that enables the user to perform actions via the use of symbols, visual metaphors and pointing devices.

List pane

The List pane is used in Monitor mode to review the status of a business process version, by reviewing the instances created by it.

modeling canvas

The modeling canvas is the portion of the e*Insight Business Process Manager where you create the business process model, in the form of a flow chart.

Monitor mode

Monitor mode is used during the monitoring and reporting phase of the process, and allows you to view the status of the business process.

schema

Schemas are files and associated stores created by e*Xchange that contain the parameters of all the components that control, route, and transform data as it moves through e*Xchange.

security

Security is the ability to limit user access to specific items based on a pre-determined profile of the user.

String

A sequence of text characters.

sub-process

A sub-process is a business process version which is called, or used by, another business process, as a sub-component.

tree view

The tree view displays a hierarchical representation of all the business process models, and their activities.

user account

A user account is information about a particular user that is stored in a database for security purposes.

user group

User groups allow you to grant access permissions to a set of users with similar processing needs without having to specify individual privileges for each user.

XML

Extensible Markup Language. XML is a language that is used in Events or messages in e*Insight, containing structured information. XML is different from String in that XML messages can contain both content, and information about the content.

Index