SeeBeyond™ eBusiness Integration Suite

# e*Insight Business Process Manager Implementation Guide

*Release 4.5.1*

# 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 expert-level knowledge of Windows NT and UNIX operations and administration, and should be thoroughly familiar with Windows-style GUI environments.

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

## 1.3 Organization of Information

The *e*Insight Business Process Manager Implementation Guide* includes the following information:

- List of Tables - Displays a list of the Tables in the document.
- List of Figures - Displays a list of Figures in the document.
- 1 - Introduction to the various applications included in the SeeBeyond eBusiness Integration Suite and the components of each. Intended audience, writing conventions, Overview of the eBusiness Integration solutions, purpose of guide
- 2 - General overview of the e*Gate components and e*Gate architecture.
- 3 - Explanation of the design and purpose of each of the e*Gate components used in the e*Insight schema.
- 4 - Explanation of the structure, design, and purpose of the ETD used to move data between the e*Insight components.
- 5 - A generalized method for approaching an e*Insight implementation, with explanations of how to accomplish some common implementation tasks.
- 6 - Case study of a simplified order processing e*Insight implementation.
- 7 - Case study showing how to use e*Insight to send out a purchase order created by an e*Insight activity component.
- 8 - Descriptions of the specialized e*Insight Monk functions.
- Appendix A - Example of the XML version of the ETD used by e*Insight to exchange data.
- Appendix B - Tables
- Glossary - Definitions of technical terms specific to the e*Insight Business Process Manager, as well as some industry terms.
- Index - Index of key terms.

## 1.4 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 4**.

### 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.

## 1.5 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*

## 1.6 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*Xchange Partner Manager and eSecurity Manager 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.

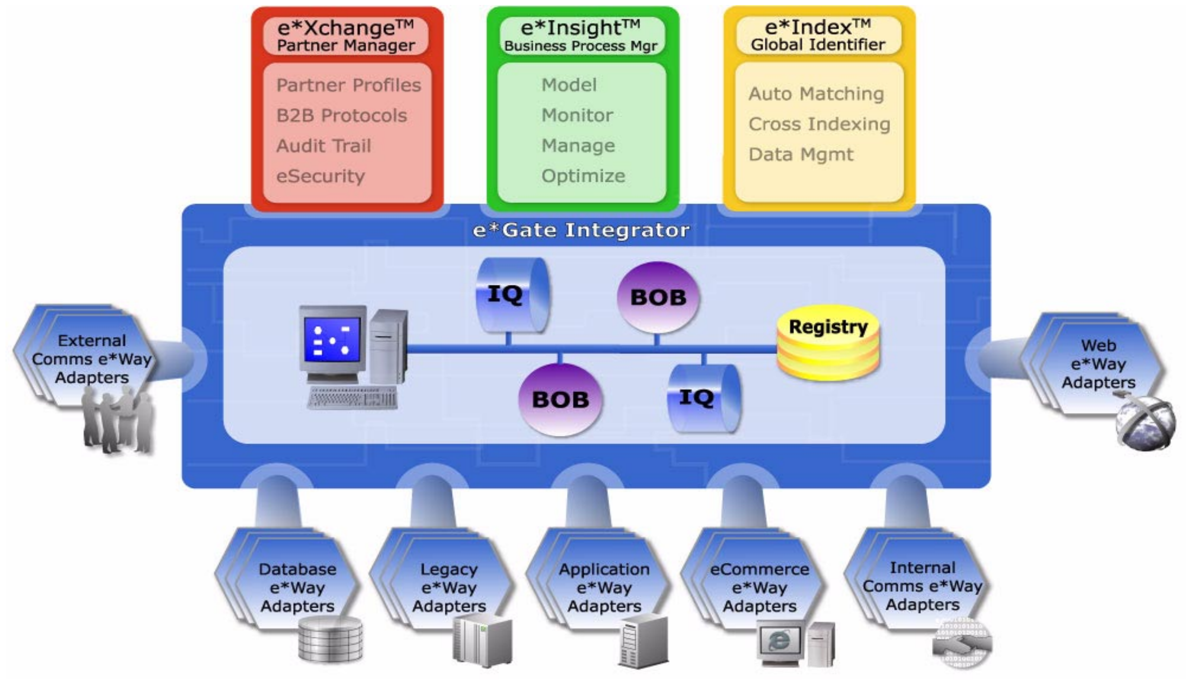
### 2.1.1 SeeBeyond eBusiness Integration Suite Components

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 the e*Insight include business process model design, monitoring, and execution as well as the ability to analyze historical performance.

Using the e*Insight Business Process Manager, business analysts are able to design eBusiness process models through a user-friendly, fully graphical tool. The e*Insight Business Process Manager 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 Business Process Manager 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, the e*Insight Business Process Manager 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.

The e*Insight Business Process Manager 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 three 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.

### 2.4.1 Types of e*Insight Implementations

The e*Insight system is designed for the large-scale integration of information systems, both inside and outside of an enterprise, in order to run and monitor business processes. The details of the business processes themselves depend on the nature of the business.

## 2.5 Implementation Road Map

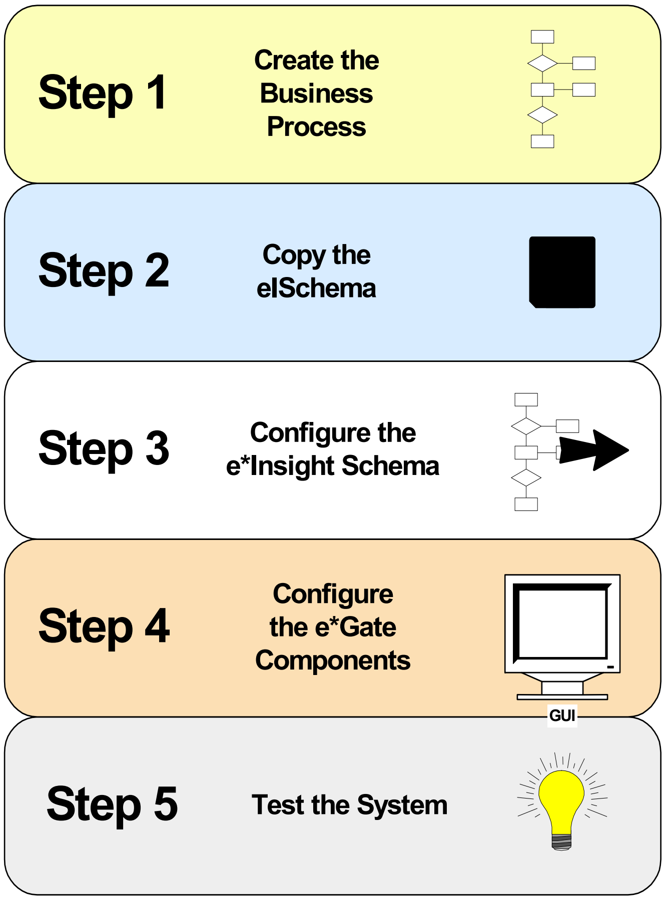
Clearly, each type of implementation involves a different approach. However, at a high level, there are certain similarities.

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
Copy the eISchema
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 eISchema

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

For information on creating a copy of the eISchema, see **"Copy the eISchema" on page 49**.

## 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 will support your business process. Use the e*Gate schema configuration utility in the e*Insight GUI for this step. e*Gate creates placeholders, with the correct names, for all the user-defined components required to implement the business process.

## 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 will:

- add and configure the e*Ways that send data into and out of the e*Insight system
- create all the Collaborations that perform the required business logic (e*Insight)
- 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.

Chapter 3

# e*Insight Schema Components

The e*Insight back end is the e*Gate schema that implements a particular e*Insight installation. The starting point for a working e*Insight back end is the e*Gate schema called **eISchema** created when you install the e*Insight back end 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.

## 3.1 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 back end components actually move and transform the data handled by e*Insight.

### 3.1.1 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.

## 3.2 e*Insight Schema Components Overview

Table 1 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 1 below, 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 |
|---|---|---|---|---|
| eX_eBPM Engine | This is a specially configured java e*Way that runs business processes. | Yes | Some | 3.3.1 on page 17 |
| eX_Resubmitter BOB | A placeholder component used in the e*Insight Event failure handling process. | Yes | Yes | 3.3.2 on page 22 |
| *Start BP* Component | Either an e*Way or BOB that sends the Event that starts a business process instance. | No | Yes | 3.3.4 on page 24 |
| eX_*Activity* e*Way | Implements an e*Insight activity that connects to an external system. | No | Yes | 3.3.5 on page 26 |
| eX_*Activity* BOB | Implements an e*Insight activity that does not connect to an external system. | No | Yes | 3.3.6 on page 29 |

## 3.2.1 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 will use all of these components. Some of the components shown are not provided as part of the e*Insight back end 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

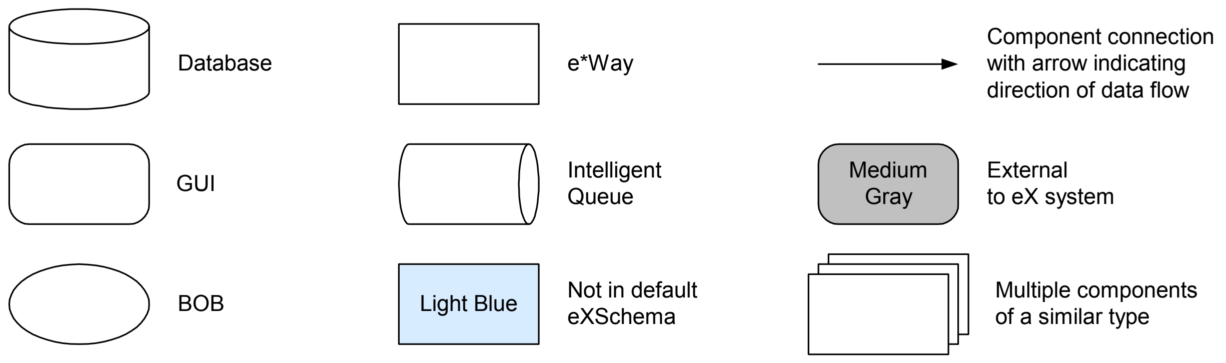
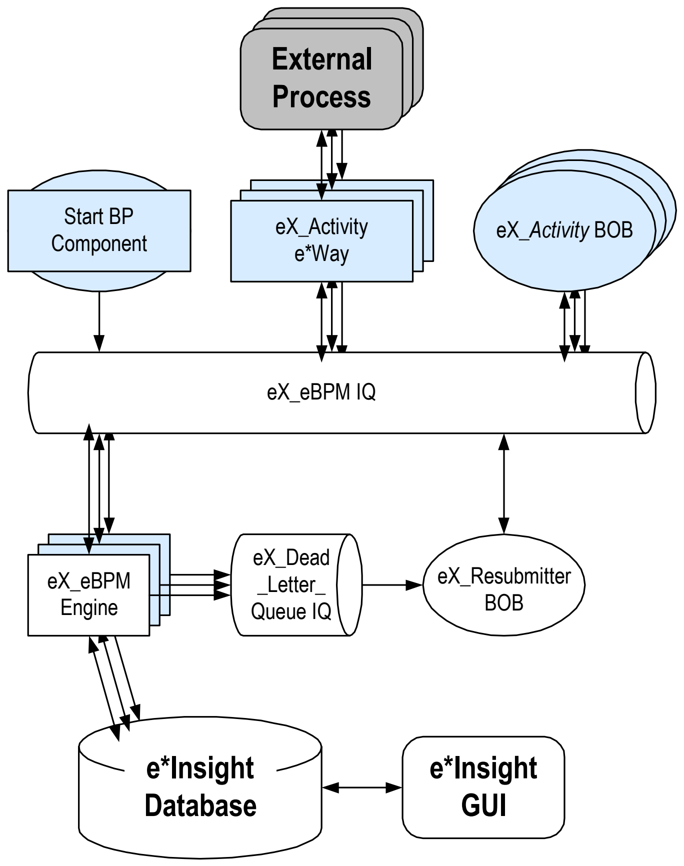

Database
e*Way
Component connection with arrow indicating direction of data flow
GUI
Intelligent Queue
Medium Gray
External to eX system
BOB
Light Blue
Not in default eXSchema
Multiple components of a similar type

**Figure 4** e*Insight 4.5 Components



## 3.3 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 **eX_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 **eX_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.*

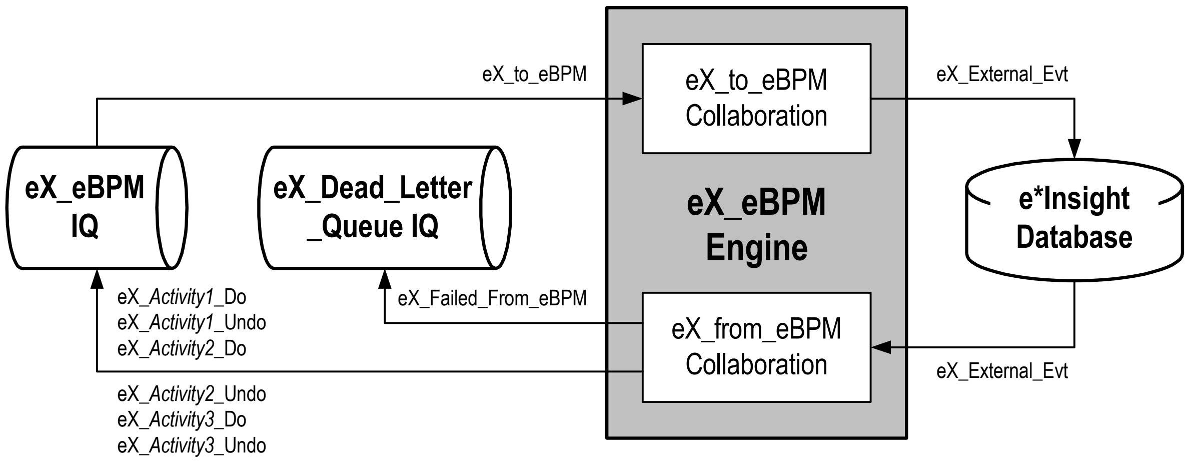
## 3.3.1 e*Insight Engine

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" on page 54**.

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 5 below.

**Figure 5** eX_eBPM Engine Detail



## Configuring the e*Insight Engine

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

**Table 2** 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 will be 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 **eX_Resubmitter** BOB below for more information. |
| | (Others) | (Default) |

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

| Screen | Parameter | Setting |
|---|---|---|
| e*Insight 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).<br><br>For example, an oracle database might use the string: **jdbc:oracle:*thin*:@*machine_name:port:db*** where<br>▪ ***thin*** is the type of oracle client interface. Another possible value is ***oci8***. See your e*Insight database administrator for more information.<br>▪ ***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.<br>▪ ***port*** is correct port for communicating with the e*Insight database (1521 is the default).<br>▪ ***db*** is the service name used to communicate with e*Insight Oracle database from the local machine.<br><br>A SQL Server database connection might use the string:<br>**jdbc:SeeBeyond:sqlserver://<server>:<port#>;DatabaseName=<dbname>;embedded=true**<br>▪ ***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.<br>▪ ***dbname*** is the name of the e*Insight SQL Server database.<br>▪ ***port*** is correct port for communicating with the e*Insight database.<br><br>A Sybase database connection might use the string:<br>**jdbc:sybase:Tds:<server>:<port>**<br>▪ ***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.<br>▪ ***port*** is correct port for communicating with the e*Insight database. |

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

| Screen | Parameter | Setting |
|---|---|---|
| e*Insight Settings | Database Type | Specifies the type of e*Insight database. Select one of the following:<br>▪ **Oracle** when using an Oracle 8i (8.1.5 and above)<br>▪ **SQL Server** when using SQL Server 7.0 or SQL Server 2000<br>▪ **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 will interpret the JDBC URL String specified previously to gain access to the e*Insight backenddatabase. For example, **oracle.jdbc.driver.OracleDriver** can be used with an ORACLE database, and **sun.jdbc.odbc.JdbcOdbcDriver** can be used with a SQLServer 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.<br>Entering the special value of zero (0) implies that caching of Business Process definitions is NOT desired, and thus the eBPM Engine will ALWAYS reload the Business Process definition from the database for EVERY Activity event of a Business Process Instance.  Note, this feature will severely impact performance |
| | Auto Model Reload | Determines if the engine dynamically loads an enabled Business Process Version if the enabled/ disabled status of Business Process Versionchanges. If the value is set to YES then Business Process Versions that are enabled or disabled while the engine is running will be immediately recognized. However, setting this value to YES may degrade performance. |

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

| Screen | Parameter | Setting |
|---|---|---|
| | Instance Caching | Instance Caching is the most efficent way to process Business Process Instances.Setting this value to YES will keep a cache of the instance information throughout the life span of the Business Process Instance. Setting this value to NO will retrieve the information from the database instead. This allows more flexability and fault tolerance at the cost of performance.<br><br>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" on page 55**. |
| | 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. |
| | (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 28** 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 28** 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.

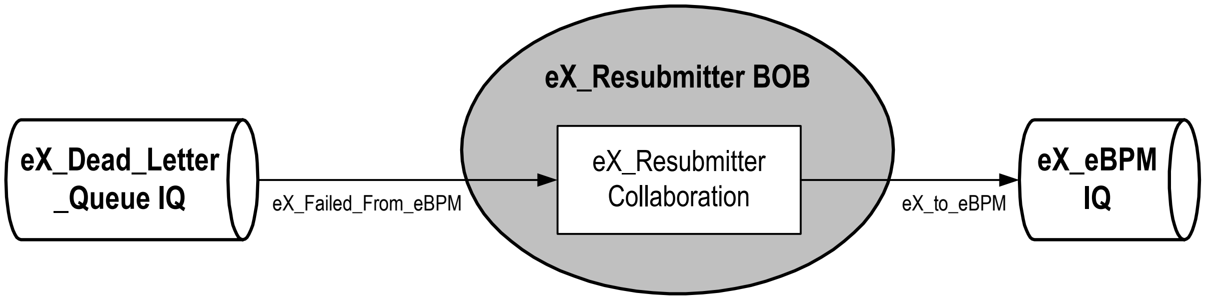
#### Published Event Type: eX_External_Evt

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

## 3.3.2 eX_Resubmitter BOB

The eX_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 eX_Resubmitter BOB's Collaboration must be supplied by the implementor.

**Figure 6** eX_Resubmitter BOB Detail



### Configuring the eX_Resubmitter BOB

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

### eX_Resubmitter Collaboration

The **eX_Resubmitter** Collaboration is a placeholder Collaboration that the implementor can use as a starting point to add logic that will repair and resubmit 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.

## 3.3.3 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 will shut 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.

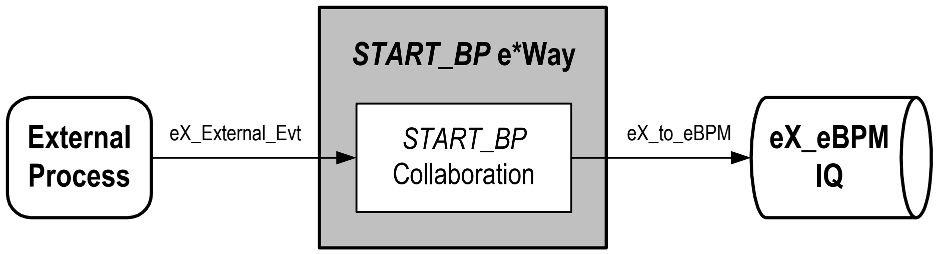
The e*Insight engine publishes the failed Event to the **eX_Dead_Letter_Queue** IQ under the **eX_Failed_From_eBPM** Event Type. The **eX_Resubmitter** BOB subscribes to this Event Type and you can use it to repair the Event and republish it to the **eX_eBPM** IQ. The **eX_Failed_Event.ssc** ETD associated 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.

### 3.3.4 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.

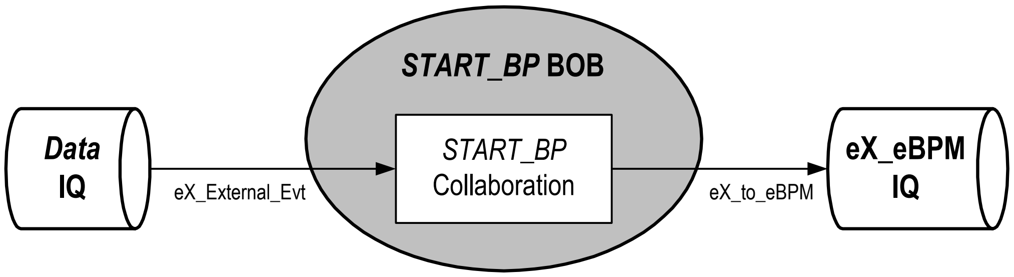
**Figure 7** START_BP e*Way Detail



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 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 8 below.

**Figure 8** 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 will be 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" on page 49** 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.

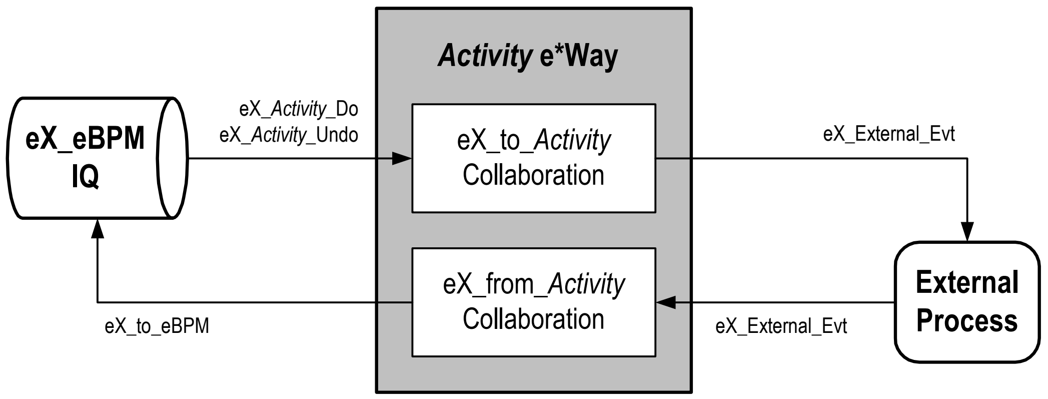
#### Subscribed Event Type: eX_External_Evt

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.

### 3.3.5 Activity e*Way

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

**Figure 9** 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. See **Table 12 on page 86** for an example of this naming convention.

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 will run. 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 will be 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** *on page 52 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 32**.

**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 27** 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** on page 52 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.

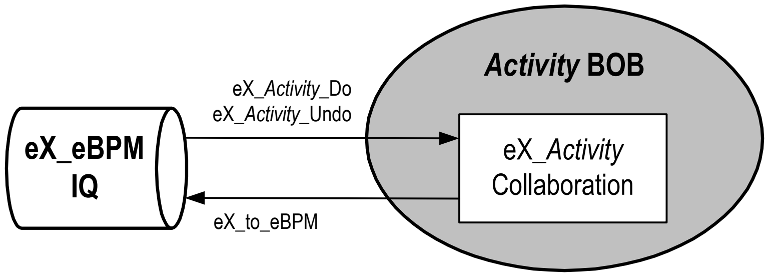
**Published Event Type: eX_to_eBPM**

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

### 3.3.6 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 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 28** 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 28** 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 28** for more information.

Chapter 4

# Understanding the e*Insight ETD

e*Insight uses a single Event Type Definition (ETD) named **eI_Standard_Event.xsc** for Java and **eX_Standard_Event.ssc** for Monk to define Events as they move from one component to another in the e*Insight system. The ETD 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.

## 4.1 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.

### 4.1.1 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.

## 4.2 e*Insight ETD for Monk - eX_Standard_Event

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.

### 4.2.1 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.

below lists these *facilitator* nodes.

**Table 3** 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. |

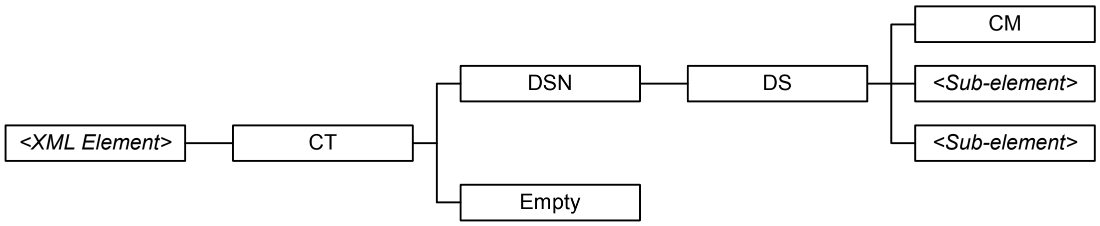
The facilitator nodes always occur in a set order and define the structure of the XML message. In the e*Insight ETD, the facilitator nodes define three types of branches:

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

### 4.2.2 XML Element with Sub-elements

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

**Figure 11** 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:

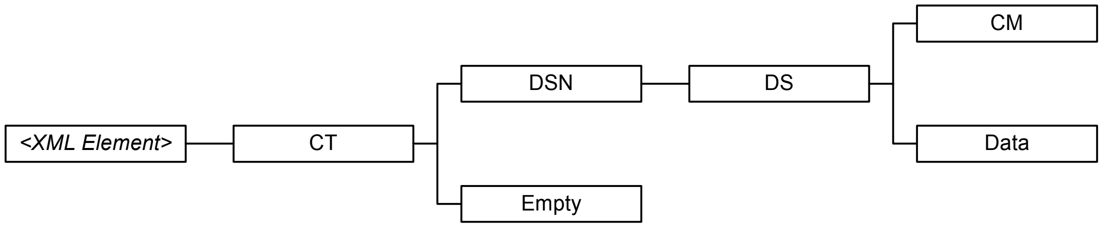
- **CM**, which holds XML comments for the element
- ***<sub-element>***, the name of a sub-element of the parent element

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.

### 4.2.3 XML Element without sub-elements

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

**Figure 12** XML Element without sub-elements

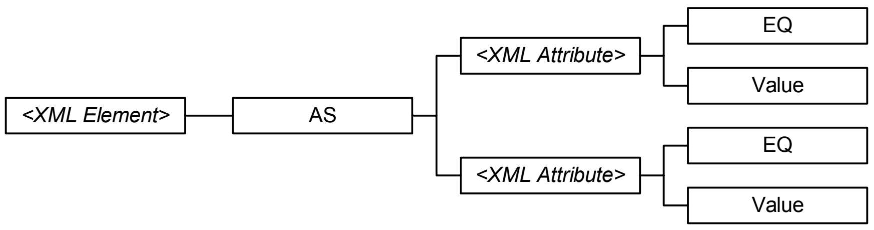


Notice that the only difference between this diagram and the previous diagram is a **Data** child node in place of the ***<sub-element>*** child nodes above. The **Data** node contains the actual data for the XML element that is defined. When creating Collaboration Rules scripts, you must map the XML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

### 4.2.4 XML Attribute

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

**Figure 13** XML Attribute

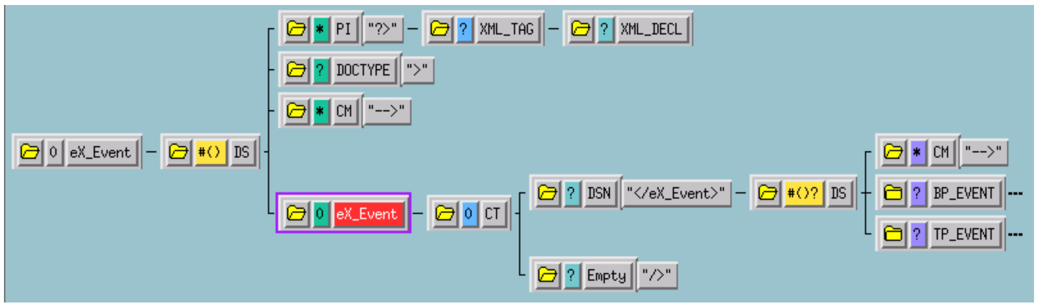


In this case, the XML element contains one child node, **AS**, which identifies the branch as XML attributes of the parent element. The **AS** node contains the ***<XML Attribute>*** nodes as child nodes. Each ***<XML Attribute>*** node has two child nodes: **EQ** to represent the equal sign (=) in the 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.

## 4.2.5 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" on page 33**.

**Figure 14** The e*Insight ETD



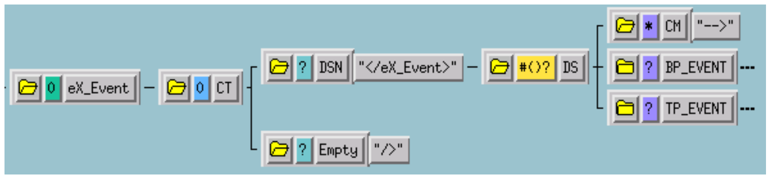
All data pertinent to e*Insight is contained in the XML element **eX_Event**. **eX_Event** contains two distinct "trees": **BP_EVENT** and **TP_EVENT**. **BP_EVENT** contains all of the information pertaining to e*Insight. **TP_EVENT** contains all of the information pertaining to e*Xchange. Both **BP_EVENT** 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 15 below shows the ETD structure for this element.

**Figure 15** 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**.

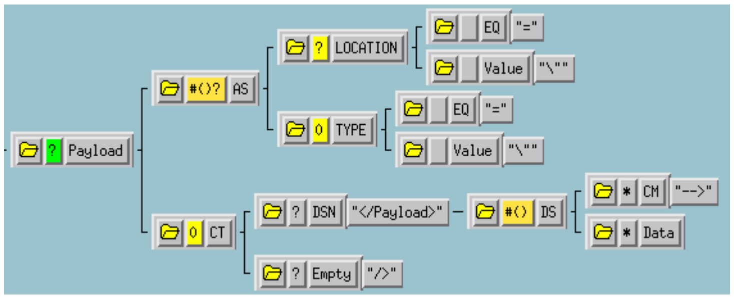
The **DS** node always contains a **CM** child node to hold XML comments. In this example, the eX_Event element does not hold data directly, but contains two sub-elements—**BP_EVENT** and **TP_EVENT**—which have similar facilitator node branches associated with them.

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 16 below shows the ETD structure for these attributes.

**Figure 16** XML Attribute **Type**



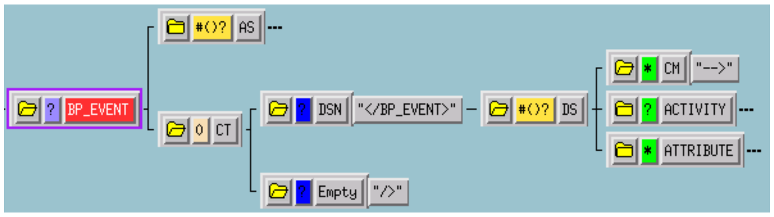
## 4.3 e*Insight ETD for Monk — eI_Standard_Event

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.

### 4.3.1 BP_EVENT

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

**Figure 17** BP_EVENT



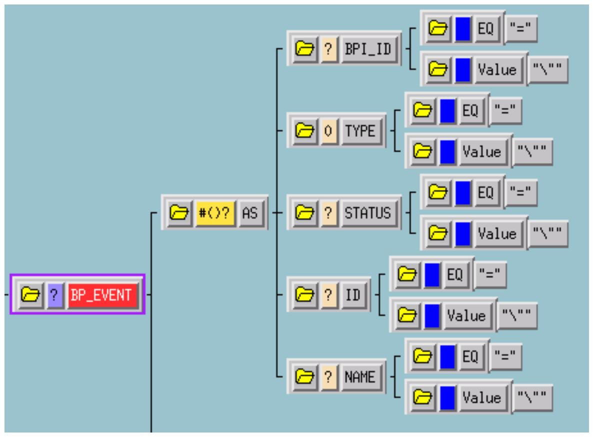
Three of the nodes shown in Figure 17 above are collapsed (followed by three dashes) indicating there are additional nodes underneath these nodes. Each of these nodes contains a different type of information pertinent to e*Insight.

- **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 18 below.

**Figure 18** 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.

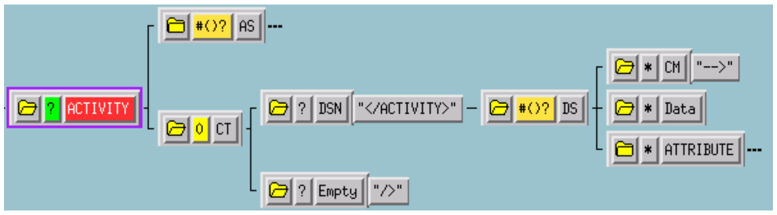
NAME

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 19 below shows the location of these nodes in the e*Insight ETD.

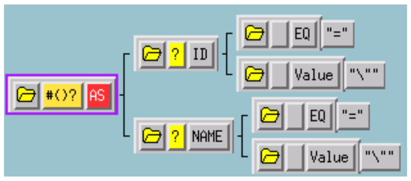
**Figure 19** BP_EVENT.CT.DSN.DS.ACTIVITY



## ACTIVITY.AS Nodes

This location in the e*Insight ETD contains ID information about the current activity in two nodes as shown in Figure 20 below.

**Figure 20** 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.

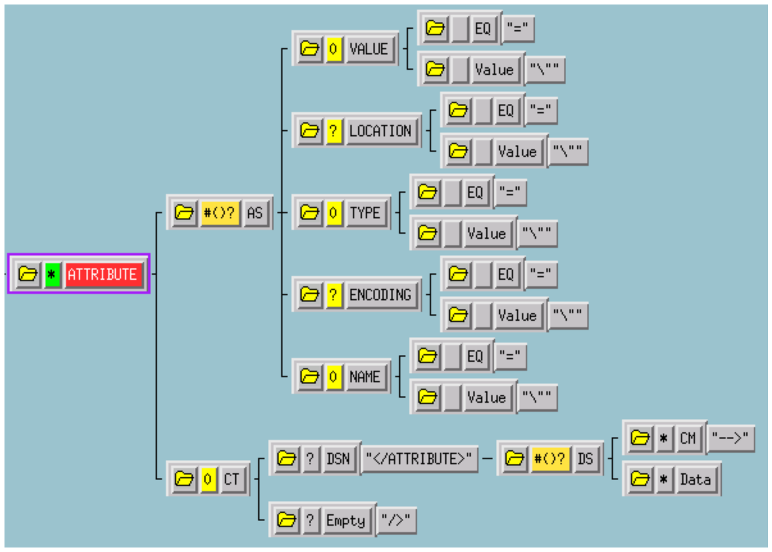
## ACTIVITY.CT.DSN.DS.ATTRIBUTE Nodes

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 21 below:

**Figure 21** 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.

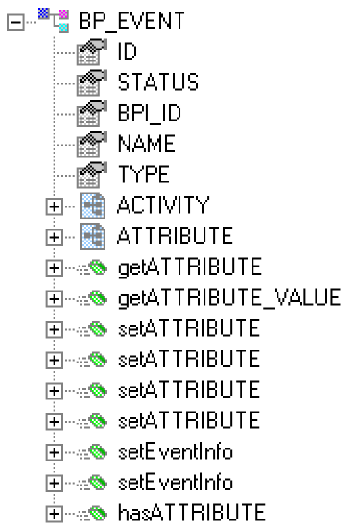
## 4.4 e*Insight ETD for Java — eI_Standard_Event

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

### 4.4.1 BP_EVENT

All data relevant to e*Insight processing is contained in the **BP_EVENT** branch of the ETD. The structure is shown below in Figure 22.

**Figure 22** BP_EVENT



```
BP_EVENT
    ID
    STATUS
    BPI_ID
    NAME
    TYPE
    ACTIVITY
    ATTRIBUTE
    getATTRIBUTE
    getATTRIBUTE_VALUE
    setATTRIBUTE
    setATTRIBUTE
    setATTRIBUTE
    setATTRIBUTE
    setEventInfo
    setEventInfo
    hasATTRIBUTE
```

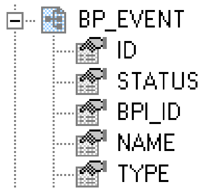
The information is held in three different locations within the Event Type Definition. Each of these nodes contains a different type of information pertinent to e*Insight.

- **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 node 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 18 below.

**Figure 23** 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.

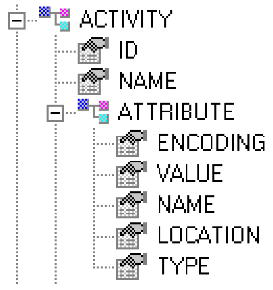
| Value | Purpose |
|---|---|
| “START_BP” | Indicates to the e*Insight engine that this Event starts a BPI. |
| “DO_ACTIVITY” | Indicates that this is a “Do” Event for the current activity. |
| “UNDO_ACTIVITY” | Indicates that this is an “Undo” Event for the current activity. |

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*Xchange 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 24 below shows the location of these nodes in the e*Insight ETD.

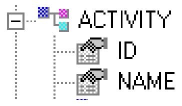
**Figure 24** BP_EVENT.ACTIVITY



## ACTIVITY Element

This location in the e*Insight ETD contains ID information about the current activity in two nodes as shown in Figure 25 below.

**Figure 25** 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.

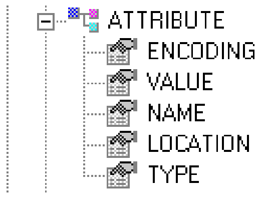
## ACTIVITY.ATTRIBUTE Element

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 26 below:

**Figure 26** 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.

| 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. |

| Value | Purpose |
|---|---|
| “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 attibute 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. |

Chapter 5

# 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 eISchema, creating messages to send to the e*Insight engine.

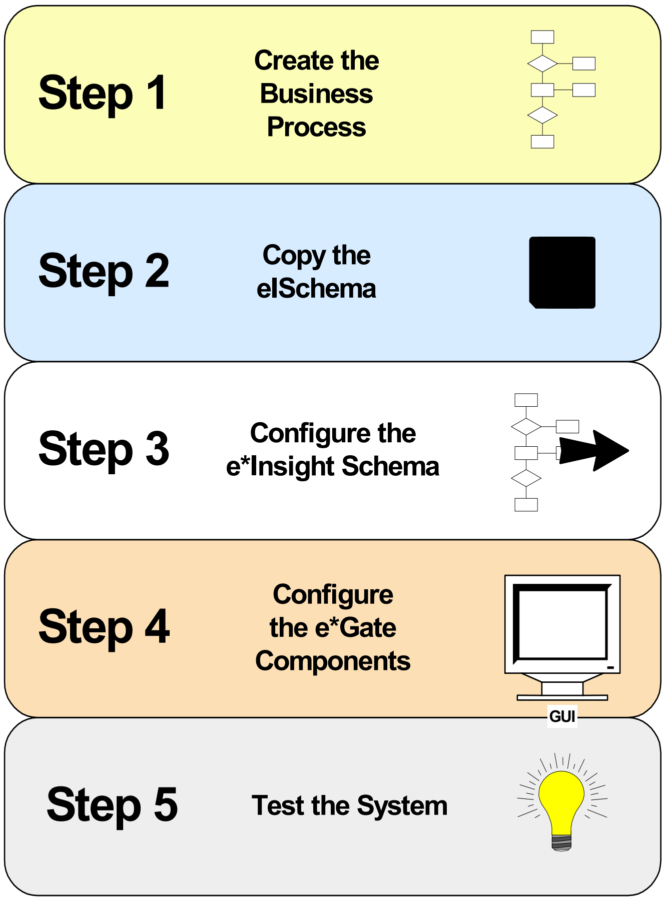
This chapter also contains instructions for deleting business processes and business process instances from the database.

## 5.1 Implementation Road Map

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

Figure 27, illustrates the major steps in the integration process for an e*Insight implementation.

**Figure 27** Integration Road Map


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

## 5.2 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.

### Copy the eISchema

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

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 **c:\eGate\client\<*eISchema 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 <*eISchema backup file name*> 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 <*new e*Insight schema name*> in the text box.
   C Mark the **Create from export** check box.
   D Click **Find** and browse for the <*eISchema backup file name*> file created in step 2 above.
   E Click **Open**.

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

### Starting a Business Process

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 **eX_eBPM** IQ. 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_Standard_Event.xsc** (Java) ETDs are described in Table 4.

**Table 4** Configuring a start message

| eX_Standard_Event.ssc | eI_Standard_Event | 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). |
| 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" on page 50**.

## 5.2.1 Setting Attributes

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:

```
void 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

## 5.2.2 Getting Attributes

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

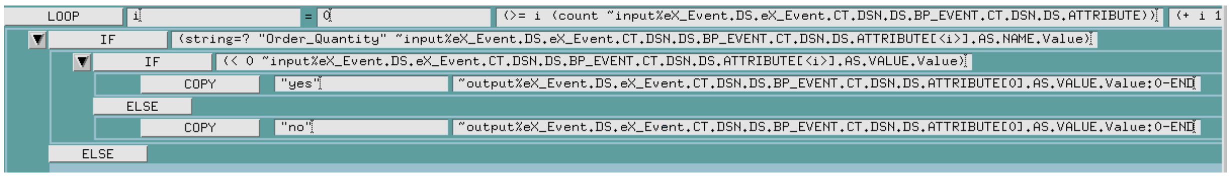
This Collaboration must do one of the following:

- 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 28, on the next page, shows an example of a LOOP rule that does this.

**Figure 28** 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

## 5.2.3 Sending the “Done” Event Back to e*Insight

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 5 describes the required nodes when an activity is set to Active control and Table 6 describes the required nodes when an activity is set to Passive control.

**Table 5** 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 6** 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" on page 50**.

## 5.2.4 Using Multiple e*Insight Engines

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 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.*

## 5.2.5 e*Insight Engine Affinity

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 Engine Affinity can possibly improve the overall message throughput but if an engine is shutdown for some reason, the instances associated with that engine will 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 29.

**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:

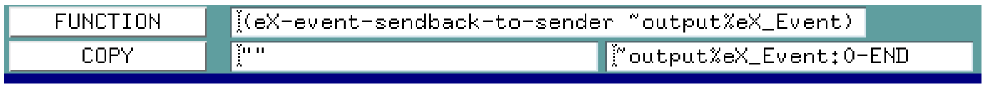
eX-event-sendback-to-sender <root-path>

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 29.

**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 below in Figure 29.

**Figure 29** Monk Collaboration Rule Script for Engine Affinity



```
FUNCTION   (eX-event-sendback-to-sender ~output%eX_Event)
COPY       ""                                ~output%eX_Event:0-END
```

## 5.2.6 Deleting BPIs from a Business Process

To delete all the business process instances from all the business processes stored in e*Insight database use the following procedure for an Oracle database.

1. Log in to SQL*Plus with administrator privileges for the e*Insight database schema (that is, the owner of the e*Insight tables). The default account is ex_admin/ ex_admin.
2. Execute the following commands.

```
truncate table ACTIVITY_INSTANCE;
truncate table ACTIVITY_INSTANCE_ATTR;
truncate table BUSINESS_PROCESS_INSTANCE;
truncate table BUSINESS_PROCESS_INSTANCE_ATTR;
truncate table ebpm_ctrl;
delete long_data;
commit;
```

3. Exit SQL*Plus

## 5.2.7 Deleting Business Processes from the e*Insight Database

To delete all the business processes and their associated BPIs from the e*Insight database use the following procedure for an Oracle database.

1. Log in to SQL*Plus with administrator privileges for the e*Insight database schema (that is, the owner of the e*Insight tables). The default account is ex_admin/ ex_admin.
2. Execute the following commands.

```
truncate table activity_object_attr;
truncate table activity_object;
truncate table activity_instance;
truncate table activity_instance_attr;
truncate table business_process_object;
truncate table business_process_object_attr;
truncate table business_process_instance;
truncate table business_process_instance_attr;
truncate table routing_expression;
truncate table routing_object;
truncate table ebpm_ctrl;
truncate table bp_numeric_attr;
truncate table bp_boolean_attr;
truncate table ai_numeric_attr;
truncate table ai_boolean_attr;
delete long_data;
commit;
```

3. Exit SQL*Plus.

Chapter 6

# e*Insight Implementation

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

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 will be 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.*

## 6.1 Overview

The major tasks in the implementation are shown in Table 7 below.

**Table 7** 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 62** |
| 2 | Use the e*Insight GUI to configure the e*Gate schema that supports e*Insight | **"Configure the Integration Schema" on page 65** |
| 3 | Configure the e*Insight Engine | **"Configure the e*Insight Engine" on page 72** |
| 4 | Add and configure the user-defined e*Gate components | **"Configure User-defined e*Gate Components" on page 73** |
| 5 | Run and test the scenario | **"Run and Test the e*Insight scenario" on page 87** |

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

### 6.1.1 Case Study: Order Processing

The case study discussed in this chapter illustrates a simplified implementation of order processing. In this case, the e*Insight receives an incoming order as a delimited text file. Once the 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 31 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 30** e*Insight Data Flow Diagram

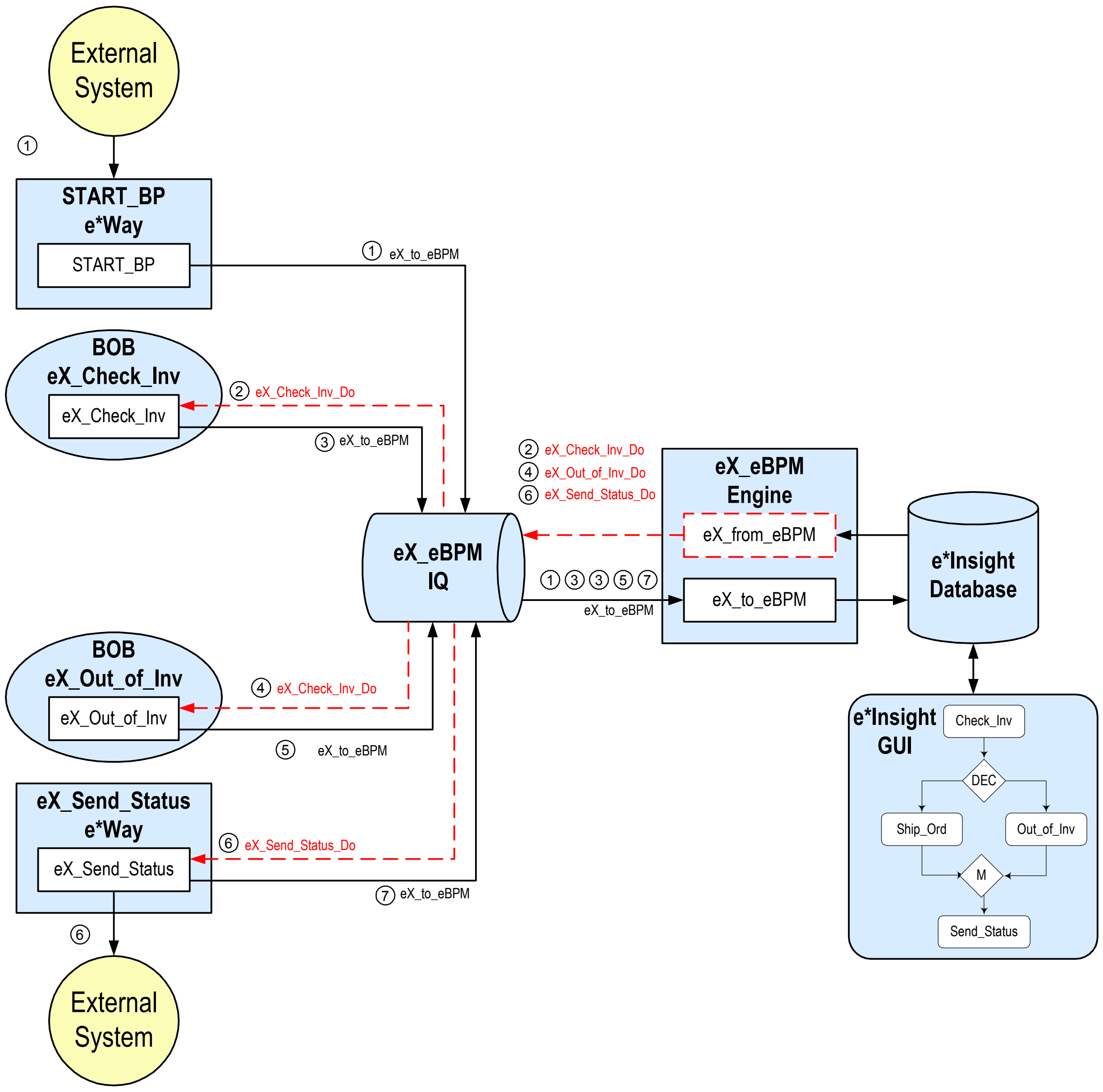
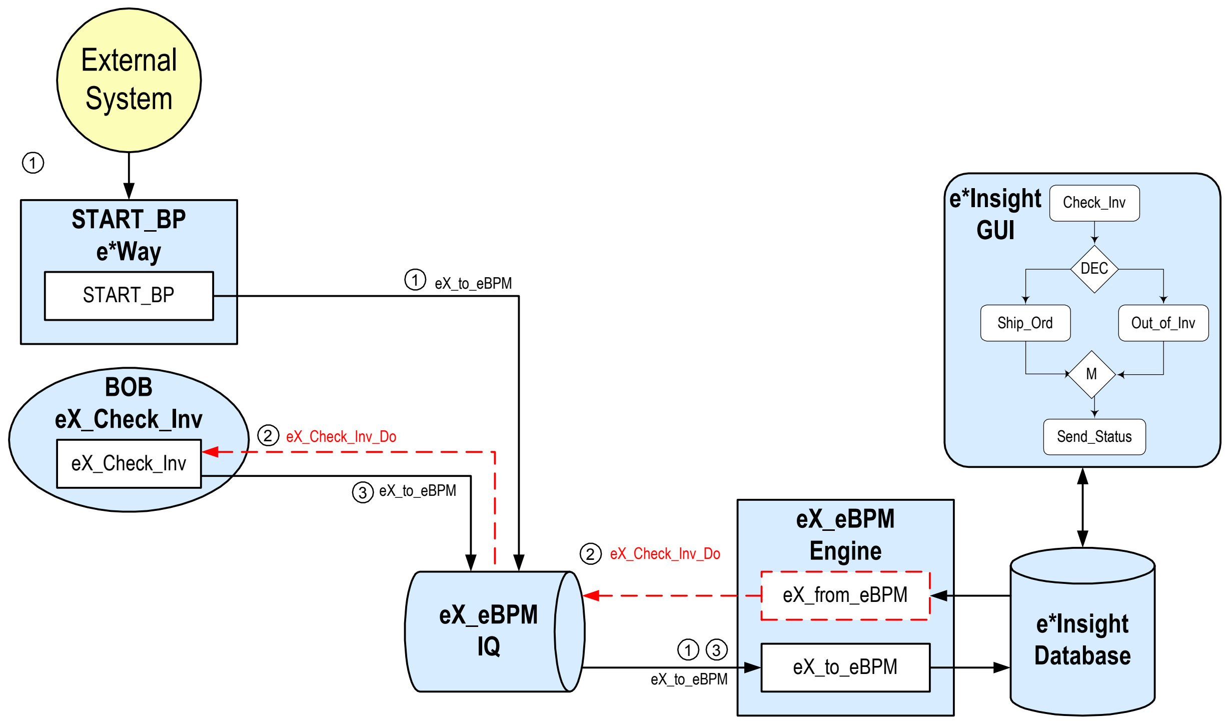


**Figure 31** e*Insight Data Flow Diagram (Part 1)



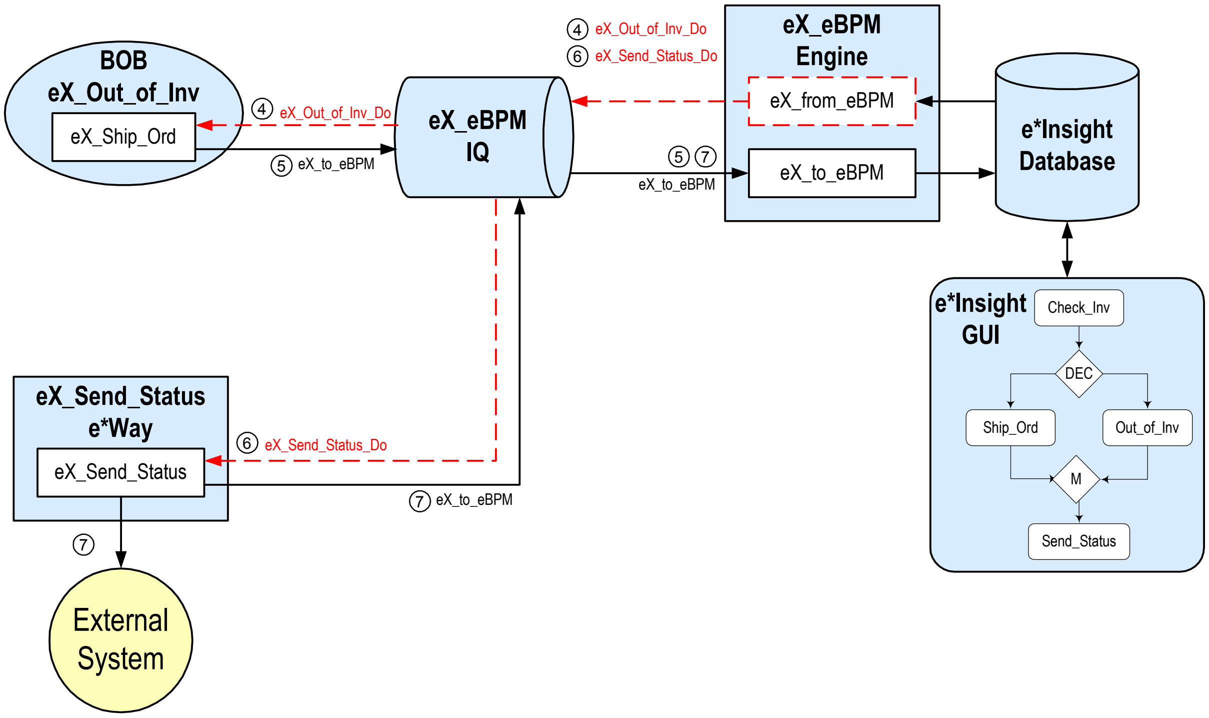
① The user-defined **START_BP** e*Way picks up the text file containing the order information from a shared location on the network, uses the order information to create the e*Insight Event that will cause the e*Insight engine to start a business process instance, and publishes it using the **eX_to_eBPM** 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 will be **Ship_Ord**; otherwise the next activity will be **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 32** 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.

## 6.2 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 33 and the following tables to create the BP in the e*Insight.

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

**Figure 33** ProcessOrder Business Process Model

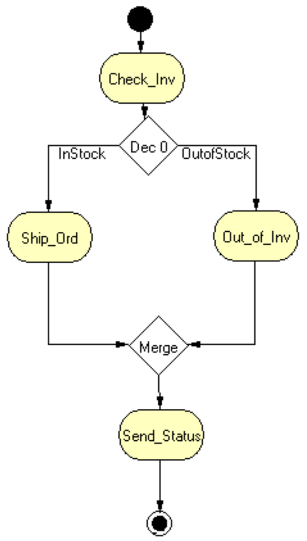


**Table 8** BP Global Attributes

| 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 9** Activity Attributes

| Activity | Attribute(s) | Input/Output |
|---|---|---|
| Check_Inv | Item_Number | Input |
| | 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 10** Decision Gates

| Feeding Activity | Expression |
|---|---|
| Check_Inv | In_Stock==true |

## 6.2.1 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 65**.

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

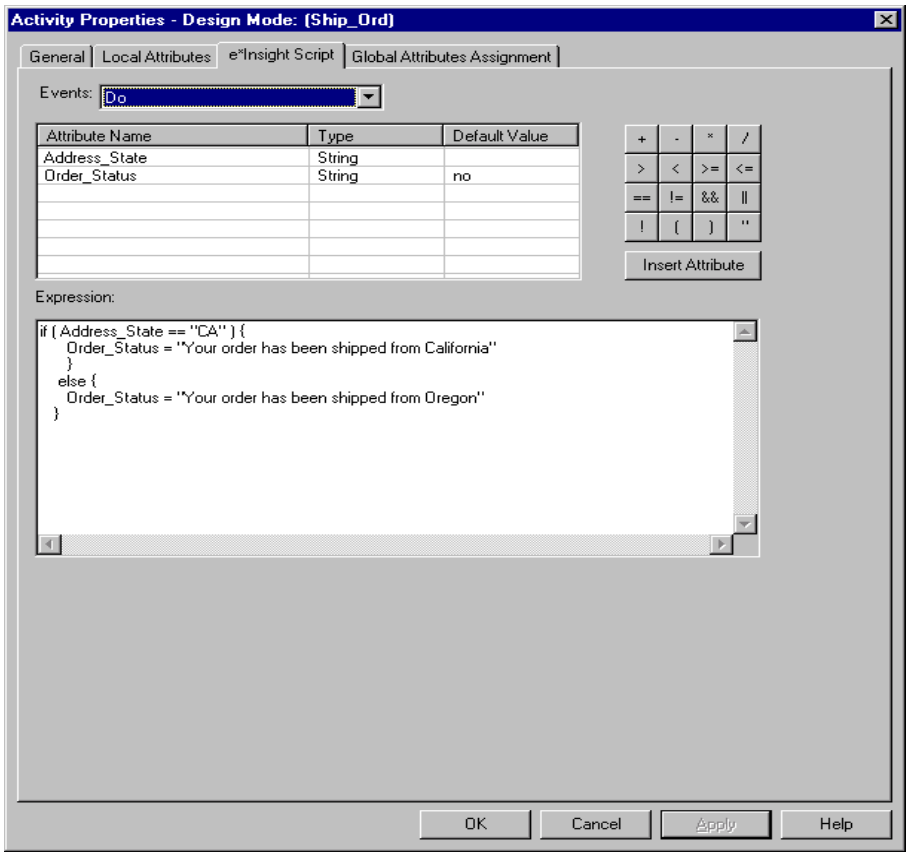
### Configuring the e*Insight Script for Ship_Ord

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 below in Figure 34.

**Figure 34** Ship_Order e*Insight Script Tab



| Attribute Name | Type | Default Value |
|---|---|---|
| Address_State | String | |
| Order_Status | String | no |

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

## 6.3 Configure the Integration Schema

All the activities in this example are carried out using e*Gate components.

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 ProcessOrder 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**.

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 will run.

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 11 below shows a summary of the e*Gate components that support the e*Insight example.

**Table 11** Integration Schema Activity Components

| Name | Type | Participating Host | Configuration Instructions |
|---|---|---|---|
| eX_Check_Inv | BOB | localhost | **"Creating the eX_Check_Inv BOB" on page 66** |
| eX_Out_of_Inv | BOB | localhost | **"Creating the eX_Out_of_Inv BOB" on page 69** |
| eX_Send_Status | e*Way | localhost | **"Send_Status e*Way Configuration" on page 72** |

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 will then implement 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 the e*Insight ending 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”.

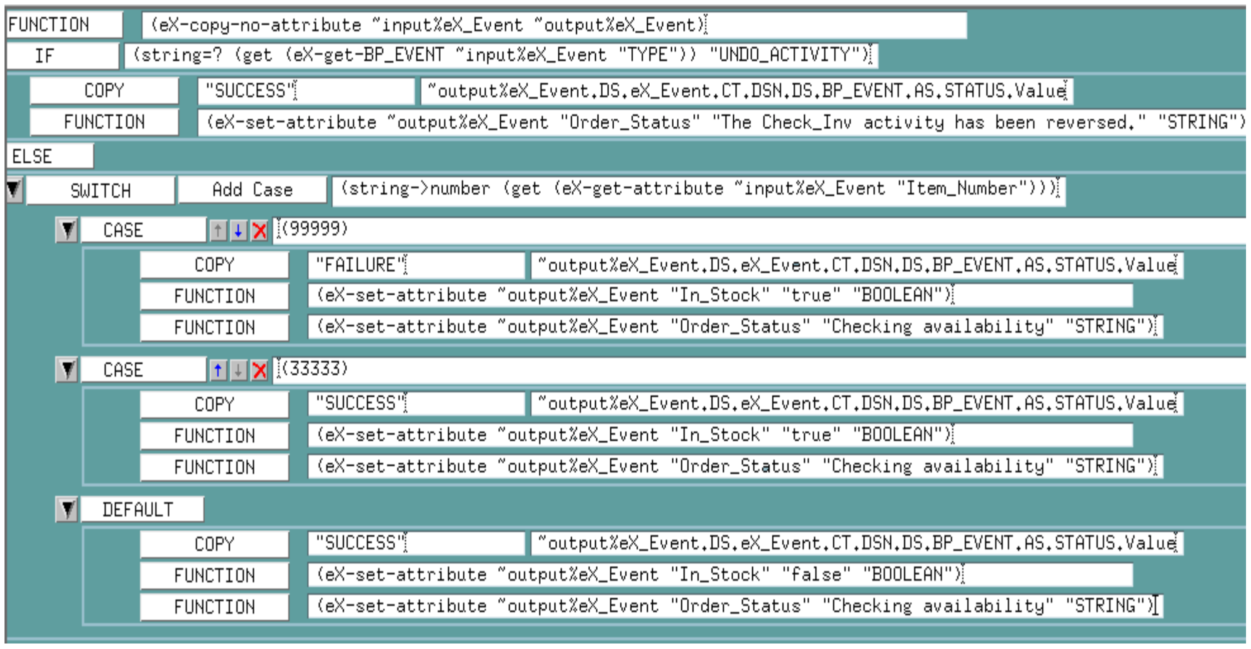
### To configure the Check_Inv activity using Monk

1. In the e*Insight GUI, open the Check_Inv 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.tsc. The source and destination Event Type Definitions are eX_Standard_Event.

   Figure 35, on the following page, shows the **eX_Check_Inv.tsc** CRS used in the e*Insight example.

**Figure 35** eX_Check_Inv.tsc CRS



```
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")
```

6 Validate and save the CRS.

7 Close the editor.

8 In the **Check_Inv** 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 **Check_Inv** Activity properties.

**To configure the Check_Inv activity using Java**

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

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

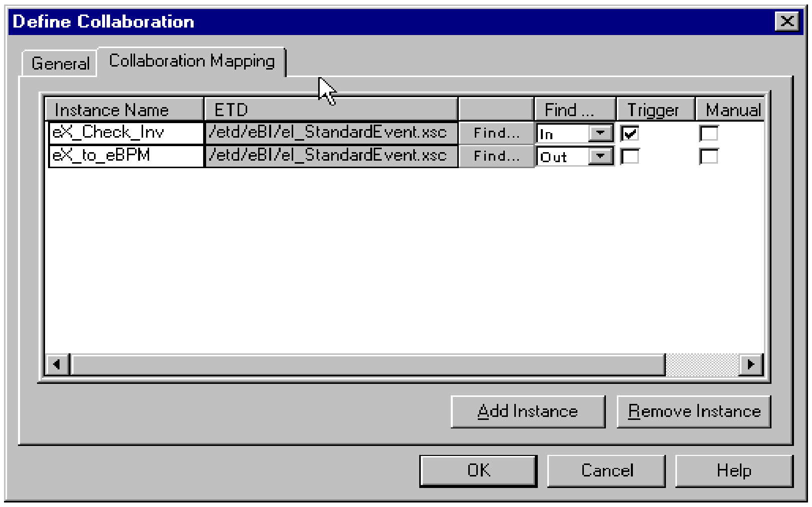
3 Click **New**.

The Define Collaboration dialog appears.

4 Select the **Define Mapping** tab.

5 Add two instances as shown below.

**Figure 36** Define Mapping for eX_Check_Inv



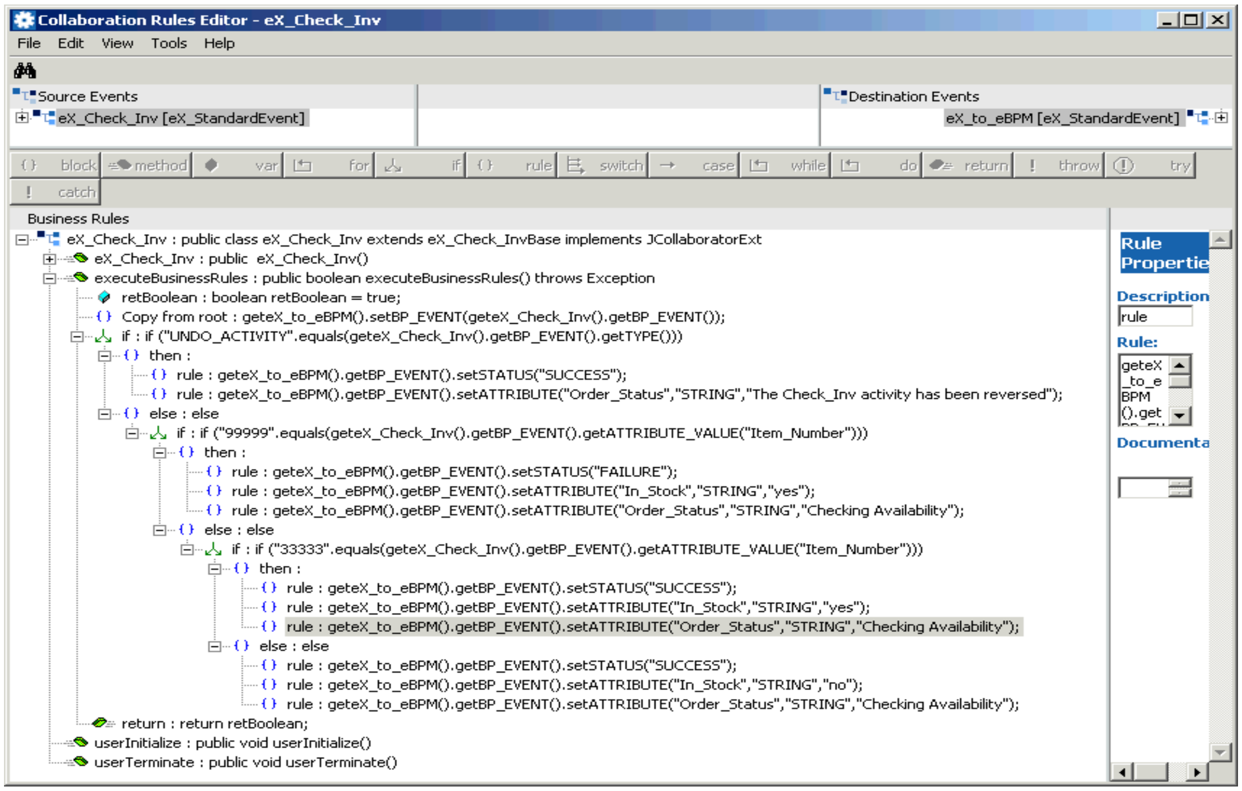
| Instance Name | ETD | | Find ... | Trigger | Manual |
|---|---|---|---|---|---|
| eX_Check_Inv | /etd/eBI/el_StandardEvent.xsc | Find... | In | ✓ | ☐ |
| eX_to_eBPM | /etd/eBI/el_StandardEvent.xsc | Find... | Out | ☐ | ☐ |

6 Click **OK**.

7 Create eX_Check_Inv.xsc.

Figure 35, on the following page, shows the **eX_Check_Inv.xsc** CRS used in the e*Insight example.

**Figure 37** eX_Check_Inv.xsc CRS



```
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 :
        rule : geteX_to_eBPM().getBP_EVENT().setSTATUS("SUCCESS");
        rule : geteX_to_eBPM().getBP_EVENT().setATTRIBUTE("Order_Status","STRING","The Check_Inv activity has been reversed");
      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","STRING","yes");
            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","STRING","yes");
                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","STRING","no");
                rule : geteX_to_eBPM().getBP_EVENT().setATTRIBUTE("Order_Status","STRING","Checking Availability");
    return : return retBoolean;
  userInitialize : public void userInitialize()
  userTerminate : public void userTerminate()
```

Rule: geteX_to_eBPM().get

8 Compile and save the CRS.

9 Close the editor.

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

You may be required to log into e*Gate.

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

12 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 the e*Insight 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, open the **Out_of_Inv** activity properties.

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

3 Click **New**.

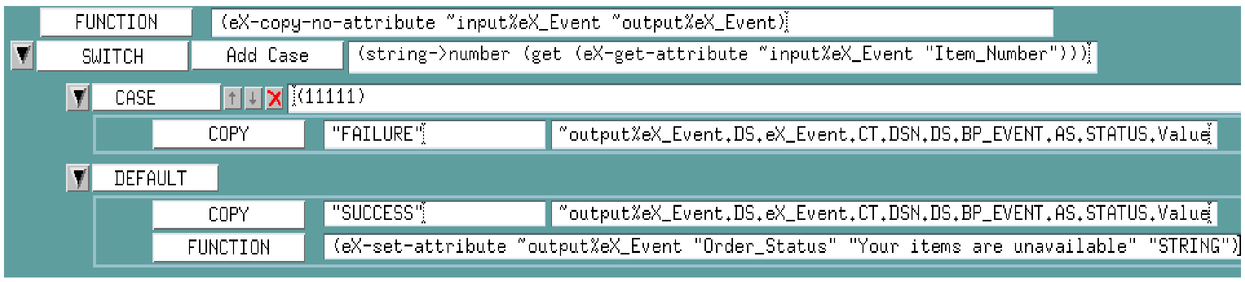
The **Define Collaboration** dialog appears.

4 Click **OK**.

5 Create eX_Out_of_Inv.tsc. The source and destination Event Type Definitions are eX_Standard_Event.ssc.

Figure 38 below shows the eX_Out_of_Inv CRS used in the e*Insight example.

**Figure 38** eX_Out_of_Inv.tsc CRS



```
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")
```

6 Validate and save the CRS.

7 Close the editor.

8 In the **Out_of_Inv** 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 **Out_of_Inv** Activity properties.

**To configure the Out_of_Inv activity using Java**

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

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

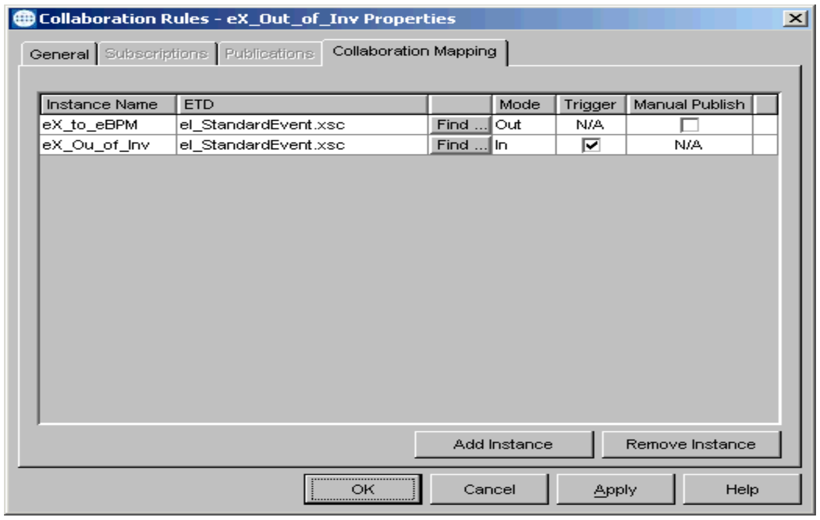
3 Click **New**.

The **Define Collaboration** dialog appears.

4 Select the **Define Mapping** tab.

5 Add two instances as shown below in Figure 39.

**Figure 39** Define Mapping for eX_Out_of_Inv



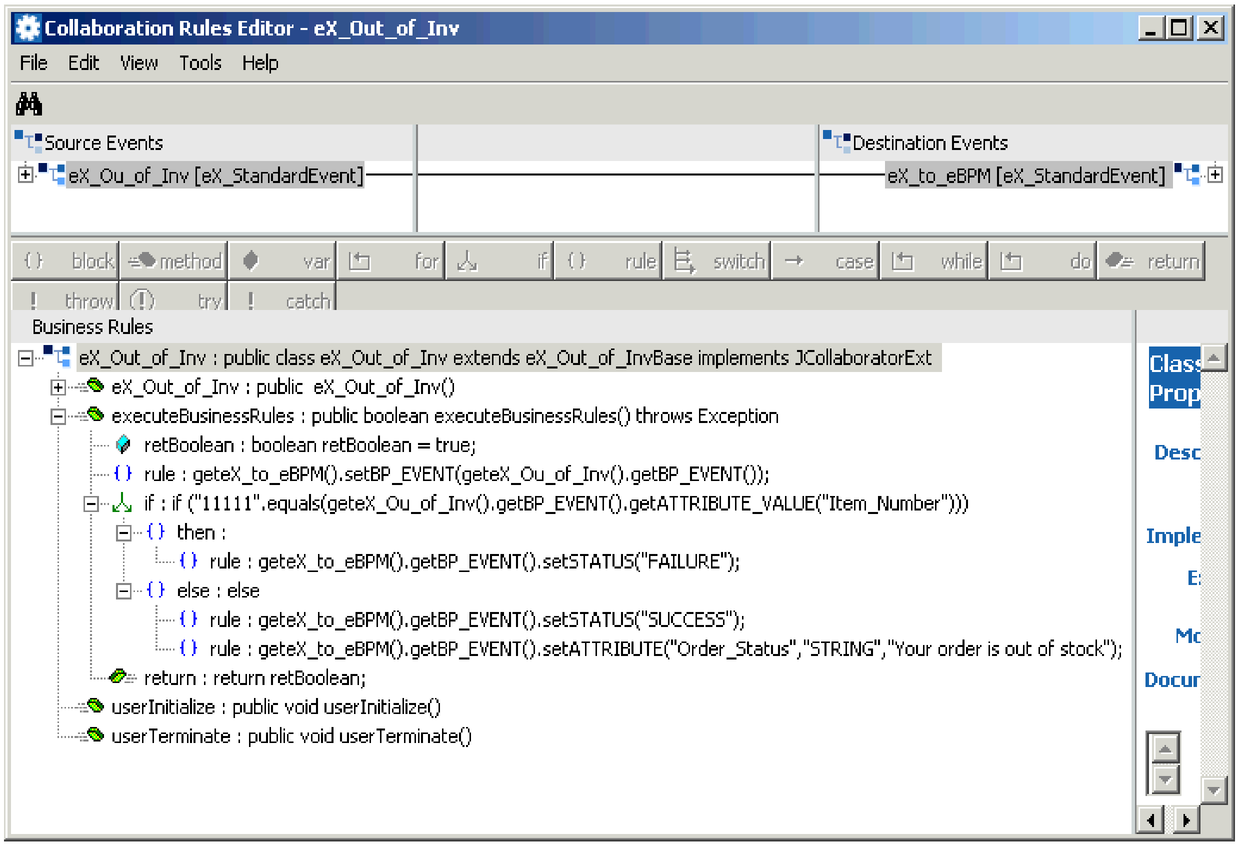
| Instance Name | ETD | | Mode | Trigger | Manual Publish |
|---|---|---|---|---|---|
| eX_to_eBPM | el_StandardEvent.xsc | Find ... | Out | N/A | ☐ |
| eX_Ou_of_Inv | el_StandardEvent.xsc | Find ... | In | ✓ | N/A |

6 Click **OK**.

7 Create eX_Out_of_Inv.tsc.

Figure 40 below shows the eX_Out_of_Inv CRS used in the e*Insight example.

**Figure 40** eX_Out_of_Inv.xsc CRS


Collaboration Rules Editor - eX_Out_of_Inv
File Edit View Tools Help
Source Events
eX_Ou_of_Inv [eX_StandardEvent]
Destination Events
eX_to_eBPM [eX_StandardEvent]
block
method
var
for
if
rule
switch
case
while
do
return
throw
try
catch
Business Rules
Class
Prop
Desc
Imple
Docur

```
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()
```

8 Compile and save the CRS.

9 Close the editor.

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

You may be required to log into e*Gate.

11 Click OK, to close the information dialog.

12 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 81**. 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.

## 6.4 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 will connect.

***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 Java e*Ways.*

### Edit the eX_eBPM Configuration File

Most of the parameter settings in the eX_eBPM engine's configuration file should not be changed. Table 12 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 18** to make the required changes for the ProcessOrder example.

## 6.5 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 12 shows the configuration order for the user-defined components.

**Table 12** 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 73** |
| 2 | Configure the Send_Status e*Way | **"Configure the Send_Status e*Way" on page 81** |

***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.*

### 6.5.1 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 the e*Insight example uses a file e*Way to send the "Start" Event to the e*Insight engine.

Table 13 shows the four steps to configure the **START_BP** e*Way.

**Table 13** 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: Creat the START_BP e*Way using Monk" on page 74 | "Step 1: Create the START_BP e*Way using Java" on page 77 |
| 2 | Create the Input ETD | "Step 2: Create the Input ETD using Monk" on page 74 | "Step 2: Create the Input ETD using Java" on page 77 |
| 3 | Create the START_BP Collaboration Rules script (CRS) | "Step 3: Create the START_BP CRS using Monk" on page 75 | "Step 3: Create the START_BP Collaboration using Java" on page 78 |
| 4 | Configure the Collaboration in the GUI | "Step 4: Configure the START_BP Collaboration in the GUI using Monk" on page 76 | "Step 4: Configure the Collaboration in the GUI using Java" on page 80 |

## Step 1: Creat 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 14** Start e*Way Parameters

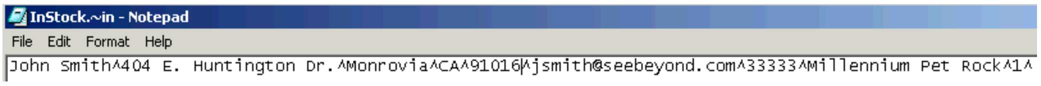
| Screen | Parameter | Setting |
|---|---|---|
| General Settings | (All) | (Default) |
| Outbound (send) settings | (All) | (Default) |
| Poller (inbound) settings | PollDirectory | <eGate>\client\data\ProcessOrder |
| | (All others) | (Default) |
| Performance Testing | (All) | (Default) |

## 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 41 below. Place this data file at the directory location **<eGate>\client\data\ProcessOrder**.

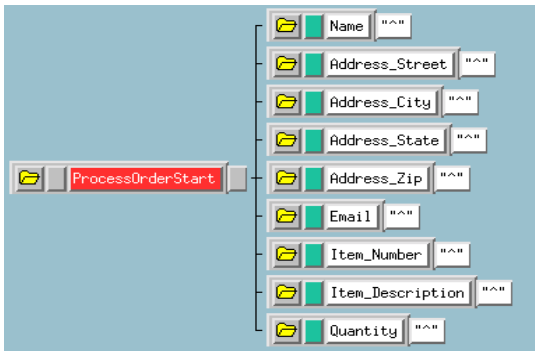
**Figure 41** Input Text File (InStock.~in)



```
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 42 below. For more information on using the ETD Editor see the ETD Editor's online help.

**Figure 42** Input ETD: ProcessOrderStart.ssc



## Step 3: Create the START_BP CRS using Monk

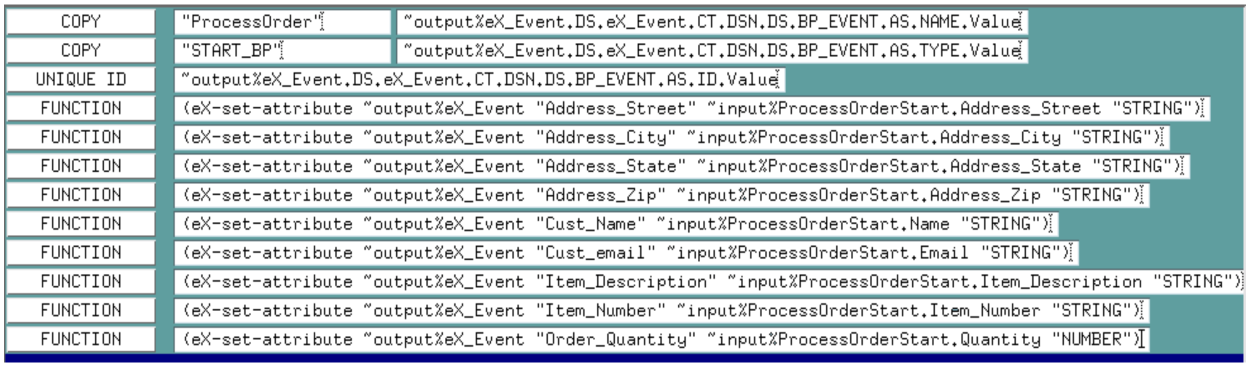
The Collaboration that sends the Event that starts the BPI must do two things:

- 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 43, shows the **START_BP** CRS used in the ProcessOrder example:

**Figure 43** START_BP CRS



| | | |
|---|---|---|
| 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 15** Start e*Way Parameters

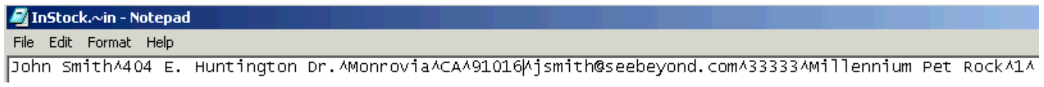
| Screen | Parameter | Setting |
|---|---|---|
| General Settings | (All) | (Default) |
| Outbound (send) settings | (All) | (Default) |
| Poller (inbound) settings | PollDirectory | <eGate>\client\data\ProcessOrder |
| | (All others) | (Default) |
| Performance Testing | (All) | (Default) |

## 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 41 below. Place this data file at the directory location **c:\eGate\client\data\ProcessOrder**.

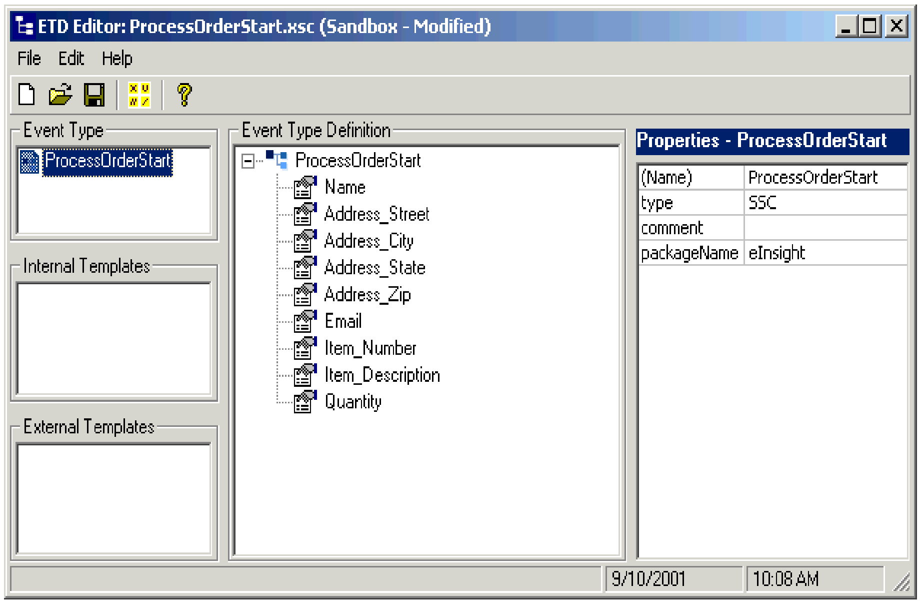
**Figure 44** Input Text File (InStock.fin)



```
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 42 below. Set the global delimiter to a ^ character.For more information on using the ETD Editor see the ETD Editor's online help.

**Figure 45** Input ETD: ProcessOrderStart.xsc



| | |
|---|---|
| (Name) | ProcessOrderStart |
| type | SSC |
| comment | |
| packageName | eInsight |

## Step 3: Create the START_BP Collaboration using Java

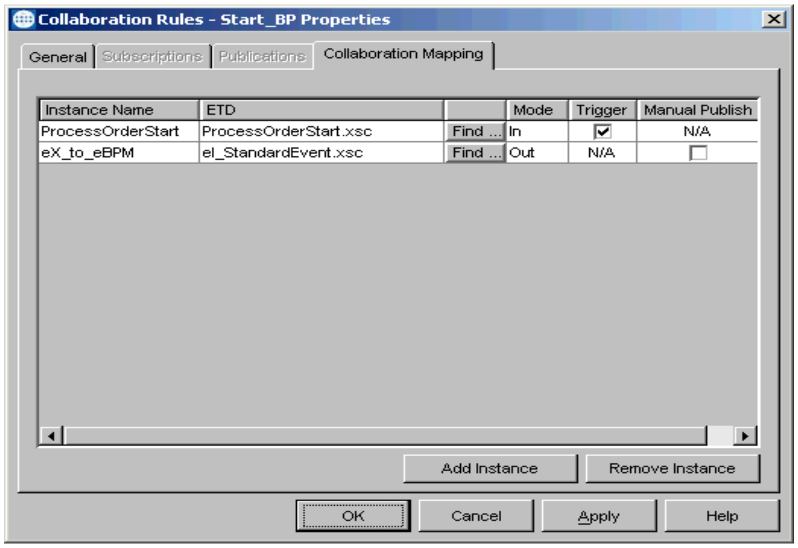
The Collaboration that sends the Event that starts the BPI must do two things:

- 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 below in Figure 46.

**Figure 46** Start_BP Properties, Collaboration Mapping Tab

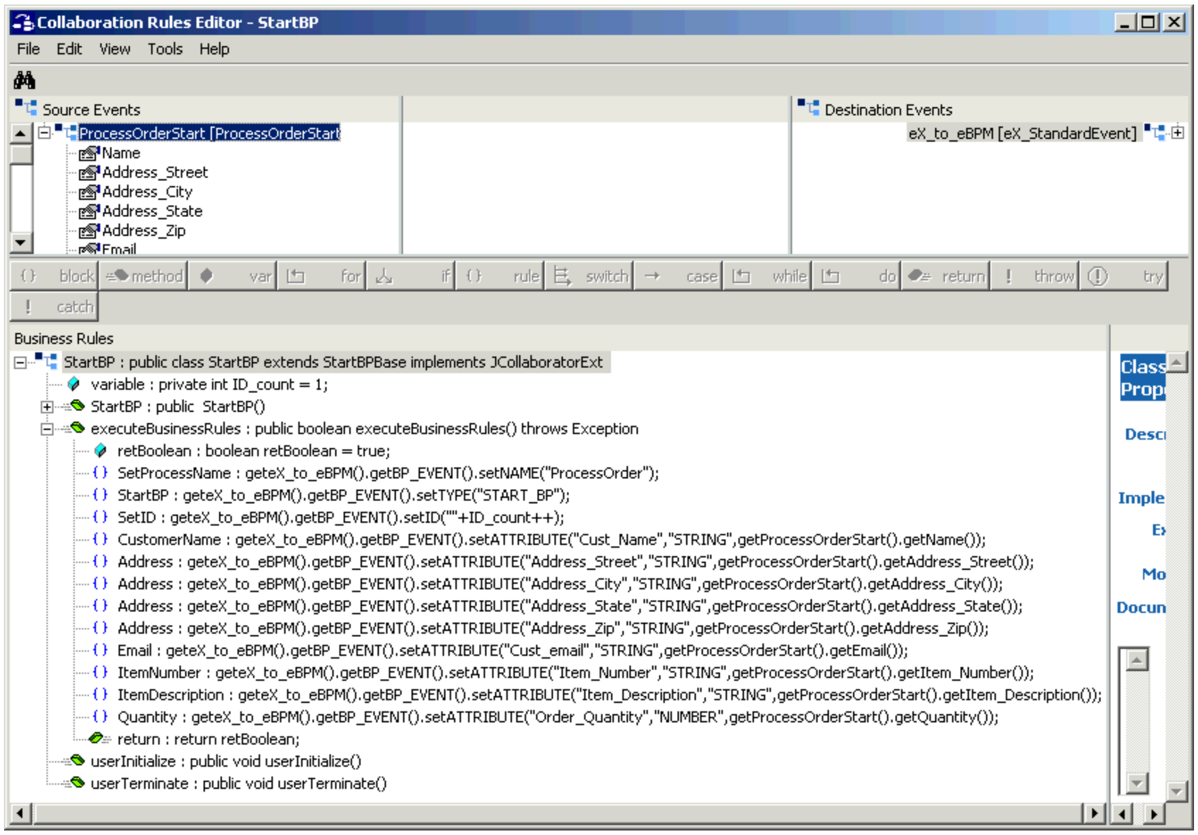


| Instance Name | ETD | | Mode | Trigger | Manual Publish |
|---|---|---|---|---|---|
| ProcessOrderStart | ProcessOrderStart.xsc | Find ... | In | ✓ | N/A |
| eX_to_eBPM | eI_StandardEvent.xsc | Find ... | Out | N/A | ☐ |

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

4 Click **New** to create a new CRS, as show in Figure 47.

**Figure 47** START_BP CRS



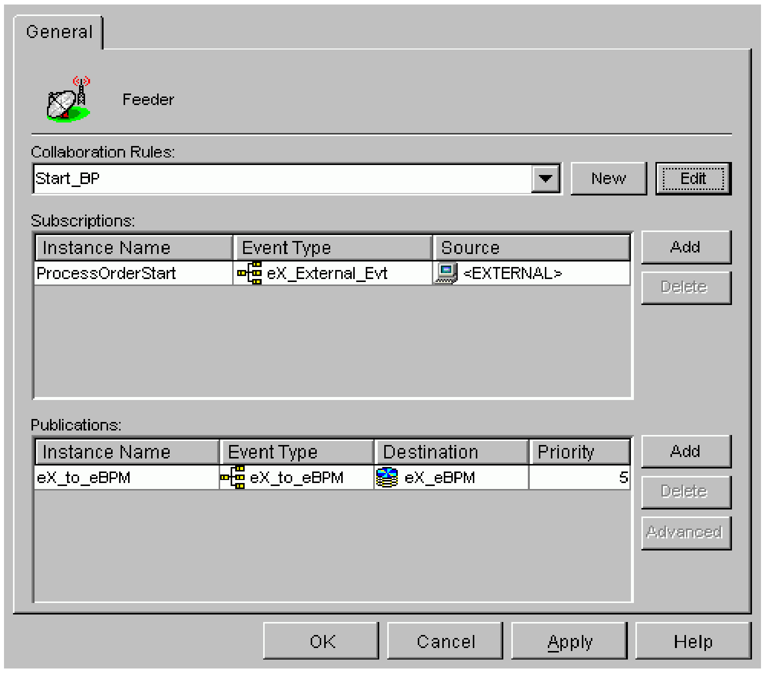
```
StartBP : public class StartBP extends StartBPBase implements JCollaboratorExt
    variable : private int ID_count = 1;
    StartBP : public  StartBP()
    executeBusinessRules : public boolean executeBusinessRules() throws Exception
        retBoolean : boolean retBoolean = true;
        SetProcessName : 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 below.

**Figure 48** 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 |

### 6.5.2 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 will simulate 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 will be returned to the e*Insight engine with the status node set to "SUCCESS".

Table 16 shows the four steps to configure the Send_Status e*Way.

**Table 16** 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 81 | "Step 1: Configure the e*Way using Monk" on page 84 |
| 2 | Create the Output ETD | "Step 2: Create the Output ETD using Monk" on page 82 | "Step 2: Create the Output ETD: SendStatus.xsc using Java" on page 84 |
| 3 | Create the eX_Send_Status.tsc CRS | "Step 3: Create the eX_Send_Status.tsc CRS using Monk" on page 82 | "Step 3: Create the Send_Status Collaboration Rule using Java" on page 85 |
| 4 | Configure the Collaboration in the GUI | "Step 4: Configure the Collaboration using Monk" on page 83 | "Step 4: Configure the Collaboration using Java" on page 86 |

### 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 17** 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 49 below:

**Figure 49** root.ssc ETD

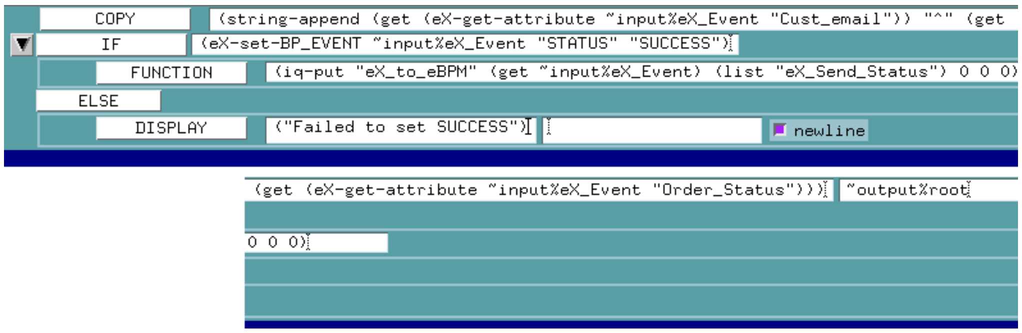
## Step 3: Create the eX_Send_Status.tsc CRS using Monk

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 50 below accomplishes these tasks. The source ETD is eX_Standard_Event.ssc and the destination ETD is root.ssc.

**Figure 50** eX_Send_Status.tsc CRS



```
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 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:

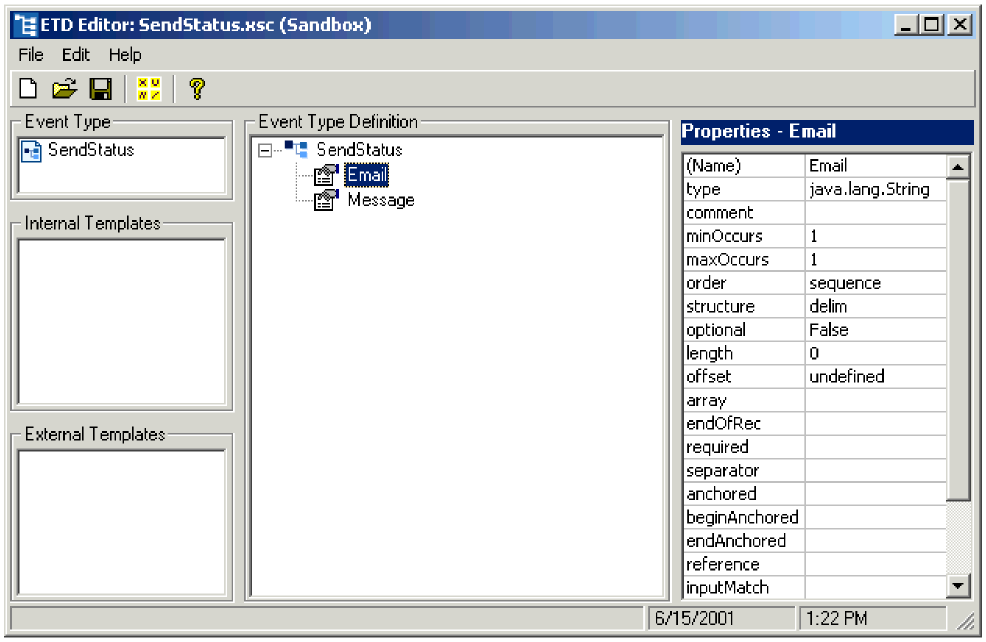
**Table 18** Send_Status e*Way Parameters

| 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 51 below:

**Figure 51** SendStatus.xsc ETD



| | |
|---|---|
| (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

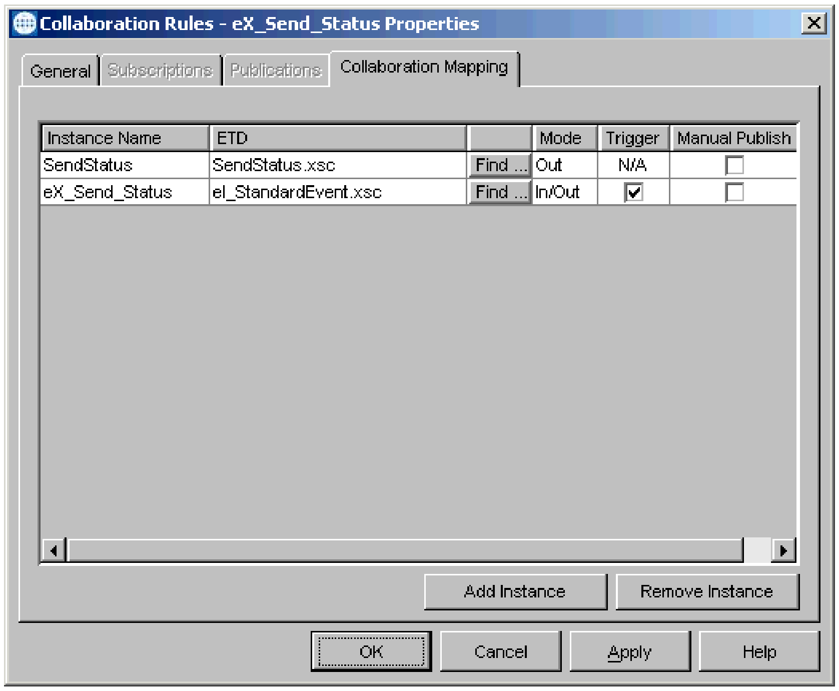
This CRS must accomplish three things:

- 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 below:

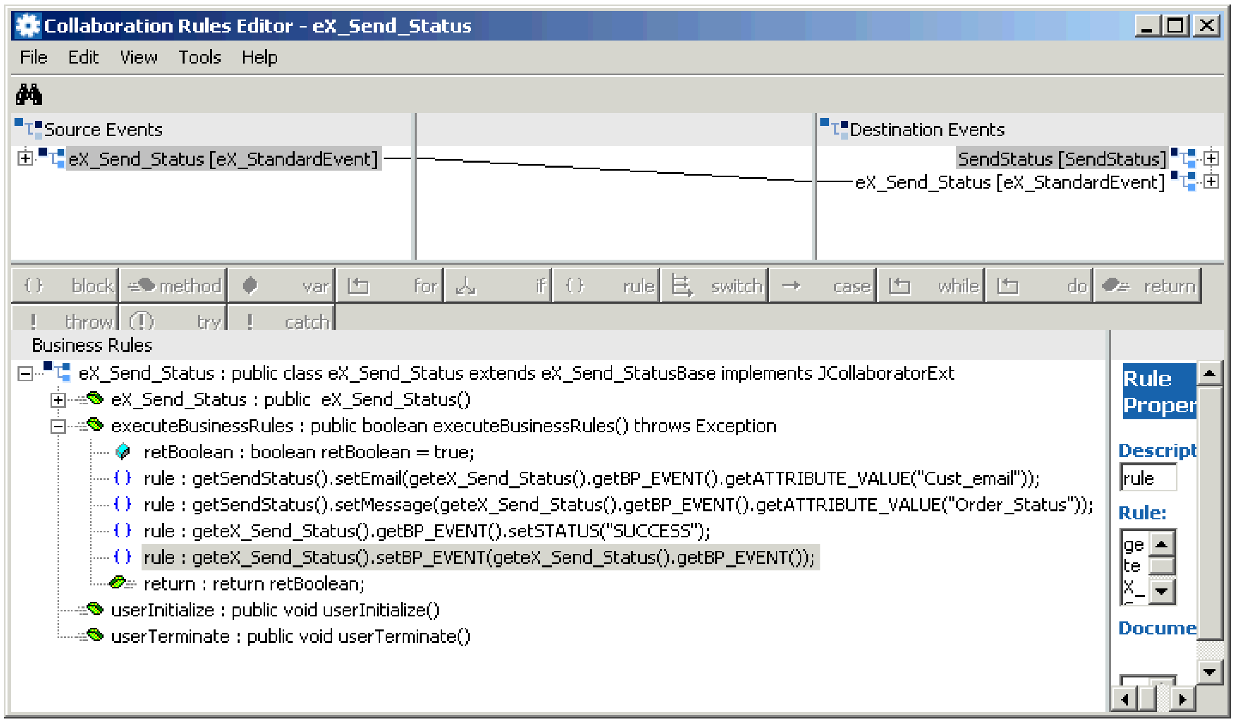
**Figure 52** eX_Send_Status CR Properties, Collaboration Mapping tab



| Instance Name | ETD | | Mode | Trigger | Manual Publish |
|---|---|---|---|---|---|
| SendStatus | SendStatus.xsc | Find ... | Out | N/A | ☐ |
| eX_Send_Status | el_StandardEvent.xsc | Find ... | In/Out | ✓ | ☐ |

3. Click **Apply**, and click the **General Tab**.
4. Click **New** to create a new CRS, as show in Figure 53.

**Figure 53** eX_Send_Status.xsc CRS


Collaboration Rules Editor - eX_Send_Status
File Edit View Tools Help
Source Events
eX_Send_Status [eX_StandardEvent]
Destination Events
SendStatus [SendStatus]
eX_Send_Status [eX_StandardEvent]
block method var for if rule switch case while do return
throw try catch
Business Rules
Rule Proper
Descript
rule
Rule:
Docume

```
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:

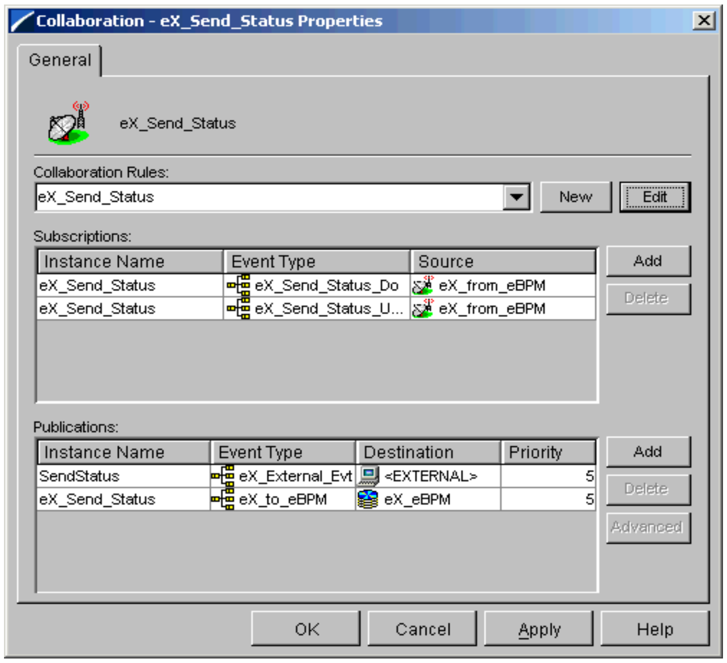
1 Highlight the **eX_Send_Status** e*Way.

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 below:

**Figure 54** eX_Send_Status Collaboration



| 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 |

| Instance Name | Event Type | Destination | Priority |
|---|---|---|---|
| SendStatus | eX_External_Evt | <EXTERNAL> | 5 |
| eX_Send_Status | eX_to_eBPM | eX_eBPM | 5 |

## 6.6 Run and Test the e*Insight scenario

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

### 6.6.1 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) will be 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.

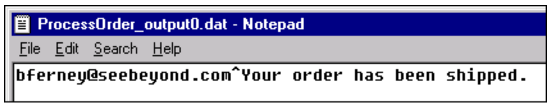
4 Start the e*Gate Monitor, and check the status of all the components. Any components displayed in red are not running. Any e*Insight components that are not running should be investigated before feeding data into the system.

5 Navigate to the location for the input data file, **InStock.~in**, shown in **Figure 41 on page 75** (**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 shown in Figure 55 below appears in the directory indicating successful completion of the BPI.

**Figure 55** In Stock Output File



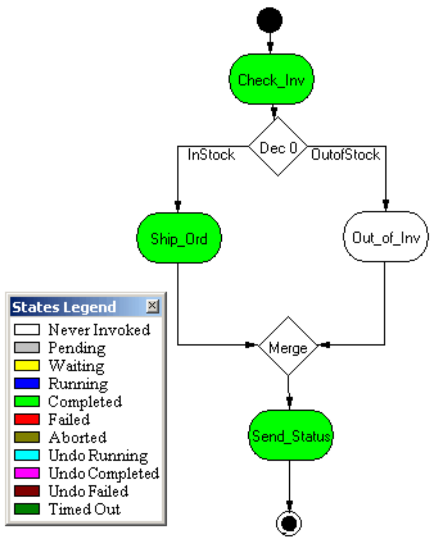
```
bferney@seebeyond.com^Your order has been shipped.
```

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 56 below illustrates how the successfully completed BPI appears in the e*Insight GUI.

**Figure 56** In Stock Completed BPI Diagram

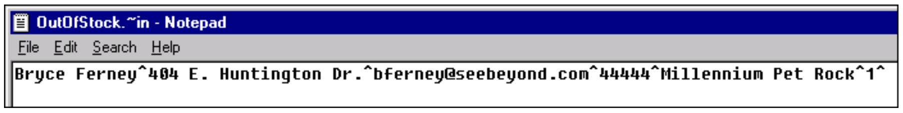


## Out-of-Stock Processing

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 shown in Figure 57 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 58 and the output file shown in Figure 59.

**Figure 57** Out of Stock Input File



```
Bryce Ferney^404 E. Huntington Dr.^bferney@seebeyond.com^44444^Millennium Pet Rock^1^
```

**Figure 58** Out of Stock Completed BPI Diagram

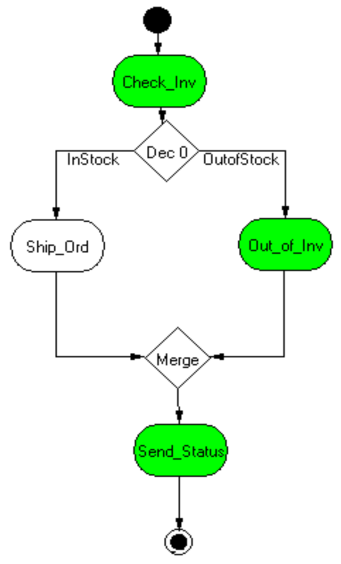
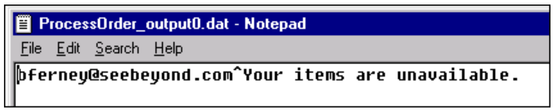


**Figure 59** Out of Stock Output File



```
bferney@seebeyond.com^Your items are unavailable.
```

## 6.6.2 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.

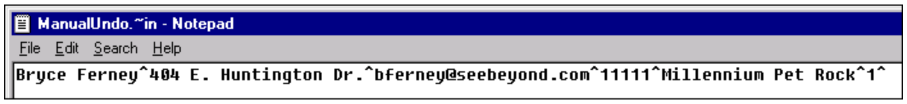
## Manual Undo

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 87**.
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 before you can mark it. See **"Deleting BPIs from a Business Process" on page 57** for information on how to do this.
3. Navigate to the location (**c:\eGate\client\data\ProcessOrder\ManualUndo.~in**) for the input data file with an item number of 11111 as shown in Figure 60 and change the extension to "**.fin**".

**Figure 60** Manual Undo Input File

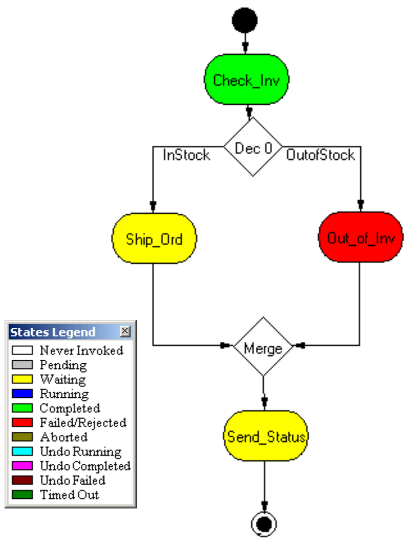

ManualUndo.~in - Notepad

```
Bryce Ferney^404 E. Huntington Dr.^bferney@seebeyond.com^11111^Millennium Pet Rock^1^
```

*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, as shown in Figure 61 on the next page.

**Figure 61** 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.

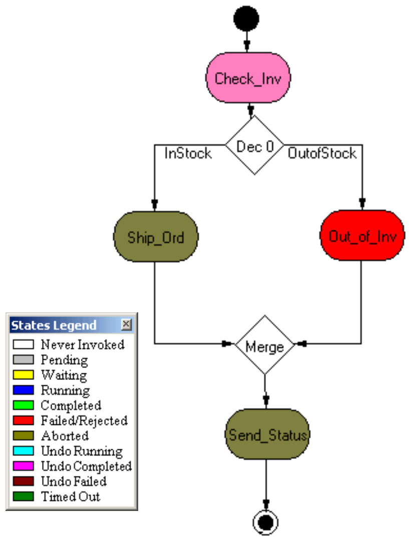
6 Select the **Business Process Attributes** tab.

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 pink indicating that the activity has been successfully undone.

**Figure 62** Manual Undo Completed BPI Diagram



## 6.6.3 Demonstrating Business Process Restart Functionality

An important feature of the e*Insight Business Process Manager 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

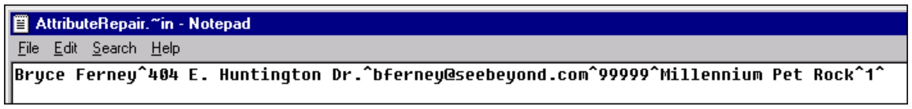
Attributes can be of various types; Boolean, number, string, and XML. The following example shows the procedure to repair an attribute of type string. For information on repairing an attribute with type XML, see the *e*Insight Business Process Manager User's Guide*.

1 Perform steps 1 through 4 outlined in **"In-Stock Processing" on page 87**.

2 Verify that **Manual Restart** has been marked for the activities in the business process. See **“Deleting BPIs from a Business Process” on page 57** for what to do if the **Manual Restart** box is not marked.

3 Navigate to the location (**c:\eGate\client\data\ProcessOrder\AttributeRepair.~in**) for the input data file with an item number of 99999 as shown in Figure 63 below, and change the extension to “**.fin**”.

**Figure 63** Attribute Repair Input File

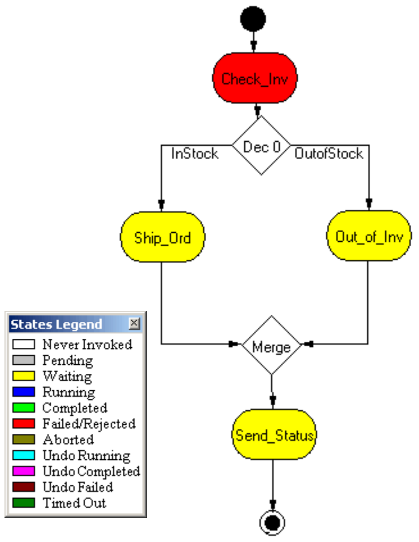


```
Bryce Ferney^404 E. Huntington Dr.^bferney@seebeyond.com^99999^Millennium Pet Rock^1^
```

**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.

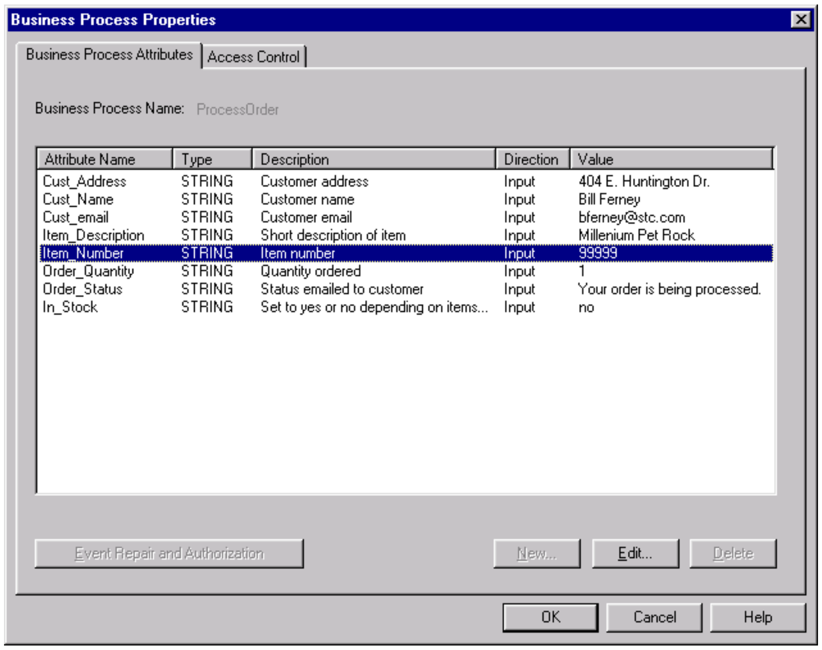
**Figure 64** Attribute Repair—Failed BPI Diagram



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

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

**Figure 65** Attribute Repair—Business Process Properties



Business Process Properties

Business Process Attributes | Access Control

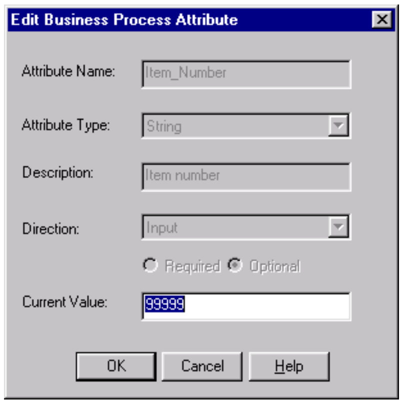
Business Process Name: ProcessOrder

| Attribute Name | Type | Description | Direction | Value |
|---|---|---|---|---|
| Cust_Address | STRING | Customer address | Input | 404 E. Huntington Dr. |
| Cust_Name | STRING | Customer name | Input | Bill Ferney |
| Cust_email | STRING | Customer email | Input | bferney@stc.com |
| Item_Description | STRING | Short description of item | Input | Millenium Pet Rock |
| Item_Number | STRING | Item number | Input | 99999 |
| Order_Quantity | STRING | Quantity ordered | Input | 1 |
| Order_Status | STRING | Status emailed to customer | Input | Your order is being processed. |
| In_Stock | STRING | Set to yes or no depending on items... | Input | no |

6 On the **Business Process Attributes** tab in the **Business Process Properties** dialog box, highlight the **Item_Number** attribute line, and then click **Edit**.

The **Edit Business Process Attribute** dialog box is displayed, as shown in Figure 66 below.

**Figure 66** Attribute Repair—Edit BP Attribute



Edit Business Process Attribute

Attribute Name: Item_Number

Attribute Type: String

Description: Item number

Direction: Input

Required [✓ Optional]

Current Value: 99999

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 **Business Process Properties** dialog box.

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

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

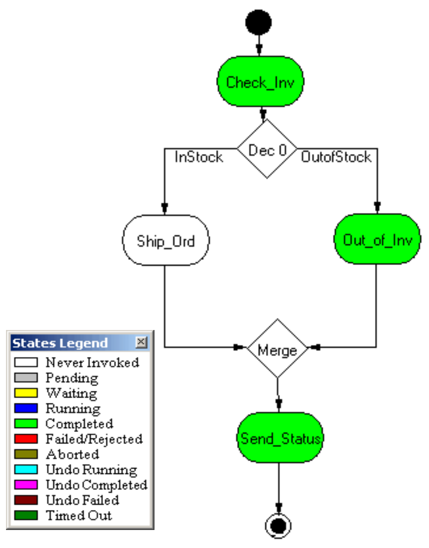
10 Select the **Result/Recourse** tab.

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

12 Highlight the enabled ProcessOrder business process version in the tree view.

The completed BPI diagram is displayed, as shown in Figure 67 on the next page.

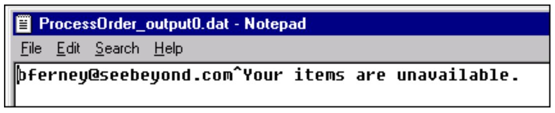
**Figure 67** Attribute Repair Completed BPI Diagram



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.

13 Verify that a text file (**ProcessOrder_output#.dat**) to be sent as e-mail is created indicating that item number 44444 is unavailable.

**Figure 68** Attribute Repair—Output File



```
bferney@seebeyond.com^Your items are unavailable.
```

## 6.7 e*Insight Engine Logging

Set the logging for the e*Insight engine the same way you would for any other e*Gate component.

1 In the Enterprise Manager, highlight the e*Insight Engine component.

2 From the **Edit** menu, select **Properties**.

3. In the **eBPM Engine Properties** dialog box, select the **Advanced** tab.
4. Click **Log**.
5. Set the **Logging level** to the desired level of log masking. See **"Logging Level" on page 96** for guidance on what level is appropriate.
6. In the **Debugging flags** list box, mark the check box next to **e*Way (EWY)**, **e*Way verbose (EWYV)**, or both.
7. Click **OK** to save your changes and exit.

   The new logging will take effect when the e*Insight configuration is reloaded. (You can force this by restarting the e*Insight Engine.)

## e*Insight Logging Guidelines

Use the following guidelines to set the e*Insight engine's logging level.

### Debugging Flags

**e*Way** and **e*Way verbose** are two separate logging channels, each producing separate entries in the log file. One does not include the other.

If the only logging channel selected is **e*Way**, the entries in the log will have the **EWY** label. If both **e*Way** and **e*Way verbose** are selected, *both* will produce entries in the log file, but all entries will have the **EWY** label. If only **e*Way verbose** is selected, then the entries will have the **EWYV** label.

In general, the e*Insight engine uses the **e*Way** logging channel to write brief log entries, and the **e*Way verbose** logging channel to write lengthy entries. The use of one channel or the other by the e*Insight engine does not reflect the severity or importance of the entry.

### Logging Level

There are four levels of e*Insight log masking.

- TRACE means no masking is applied and all entries are written to the log.
- DEBUG is a medium level of masking, that screens out less essential information.
- INFO is a higher level of masking, that screens out everything but the most important log entries.
- Any other logging level (WARNING, ERROR, FATAL, or NONE) will screen out all log entries except those that cannot be masked.

### Additional Information

You must select either **e*Way** or **e*Way verbose** to turn on e*Insight logging.

Database errors are not maskable (that is, they appear at all logging levels as long the log is turn on).

Chapter 7

# e*Insight Authorization Activity Implementation

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" on page 60** for the initial configuration instructions.

## 7.1 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.

### 7.1.1 Case Study: Order Processing

The Order Process will only automatically process orders where less than 100 items are ordered. If 100 or more items are ordered then the order should be manually authorized.

Figure 69, 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 59**.

**Figure 69** e*Insight Data Flow Diagram

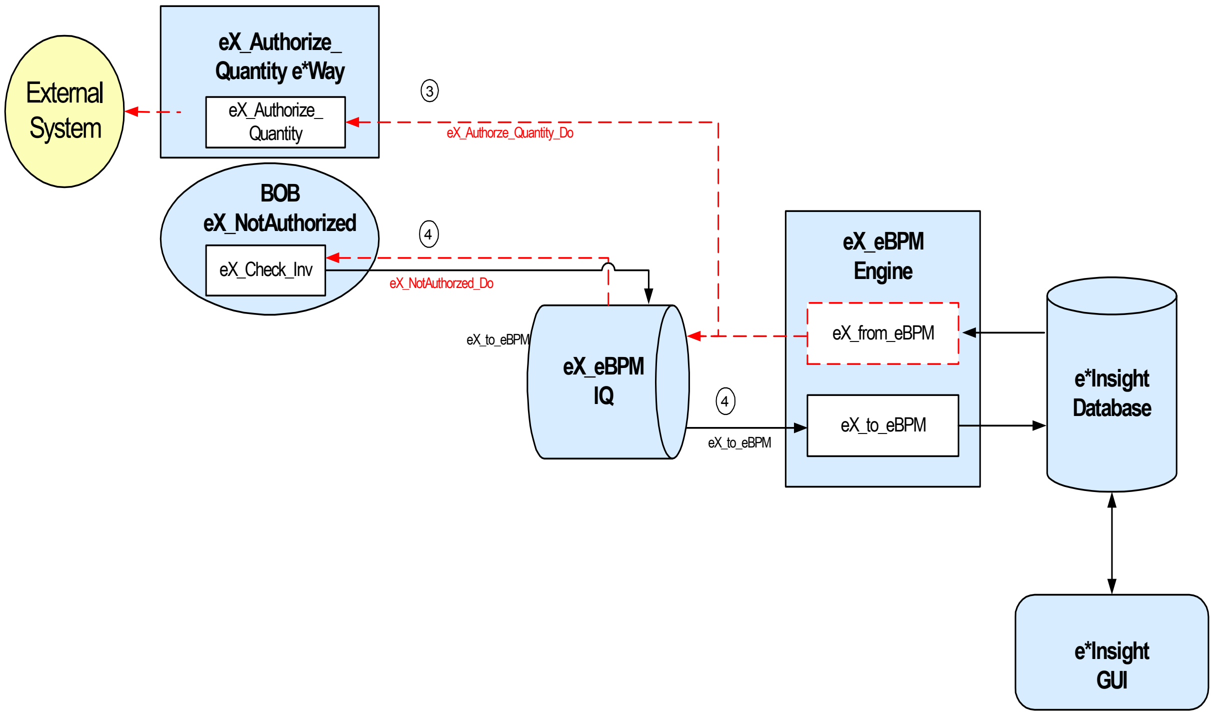


**Figure 69 data flow description**

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 will be another decision gate; otherwise the next activity will be **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 will be **Ship_Ord**; otherwise the next activity will be **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 will be **Not_Authorized** if the quantity was not authorized; otherwise the next activity will be **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.

## 7.2 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.

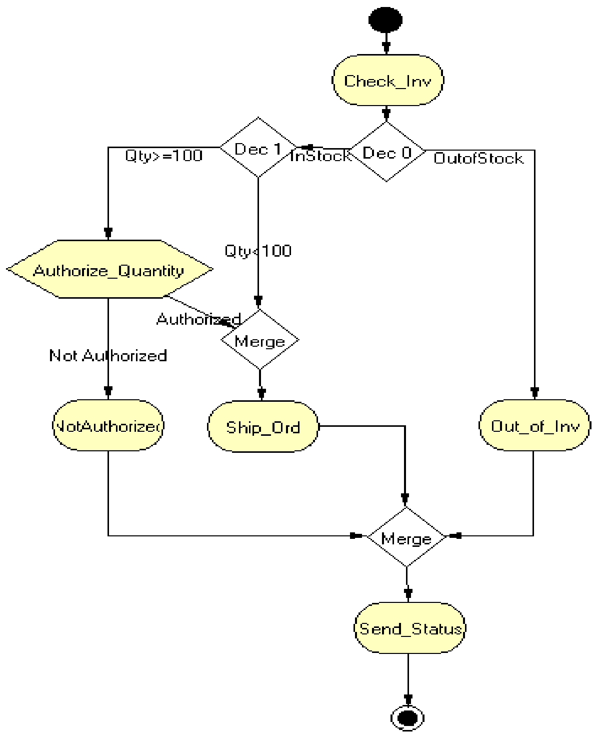
**Note:** *The Authorization Activity has two fixed Local Attributes—assignedTo (the user to whom the Authorization process is assigned) and performedBy (automatically assigned at run time as the user logged in to the e*Insight GUI). These two values must match in order for the user to Authorize, Reject, 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 70 and the following tables to create the BP in the e*Insight.

**Figure 70** ProcessOrder Business Process Model



## 7.3 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 will run.

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 19 below to configure the e*Gate schema that supports the e*Insight example.

**Table 19** 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*.

## 7.4 Step 3: Configure User-defined e*Gate Components

This example requires the configuration of an additional BOB and e*Way.

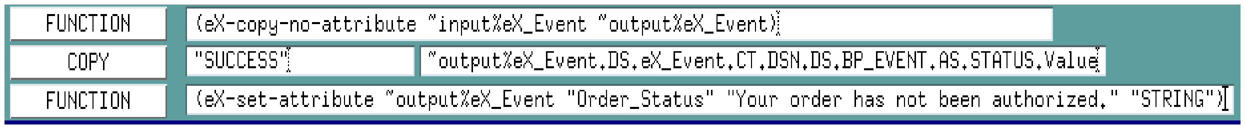
### Configure the Activity BOB CRS in the Enterprise Manager GUI

#### 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 71, on the following page, shows the **Not_Authorized.tsc** CRS used in the e*Insight example. The source and destination ETD's are eX_Standard_Event.ssc.

**Figure 71** 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.

## 7.4.1 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 will simulate 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 will not be returned to the e*Insight engine as it is expecting authorization via the e*Insight GUI in this example.

There are three steps to configuring the Send_Status e*Way:

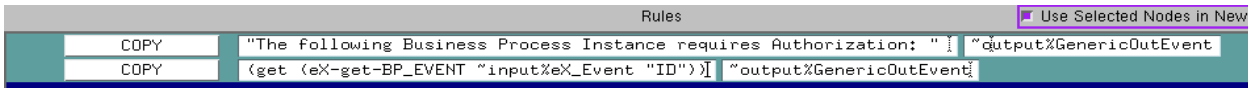
1. Create the **Authorize_Quantity.tsc** CRS.
2. Find the executable and create the e*Way configuration file.
3. Configure the Collaboration in the GUI.

### 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 below in Figure 72. The source ETD is eX_Standar_Event.ssc and the destination ETD is GenericOutEvent.ssc

**Figure 72** 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 20** 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>**.

## 7.5 Step 5: Run and Test the e*Insight scenario

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

### 7.5.1 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.

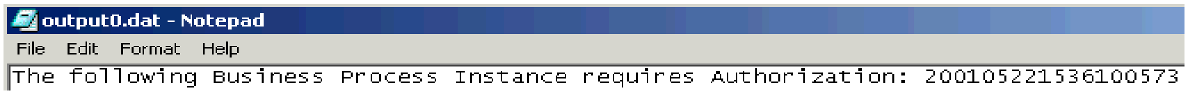
4 Start the e*Gate Monitor, and check the status of all the components. Any components displayed in red are not running. Any e*Insight components that are not running should be investigated before feeding data into the system.

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 73 below appears in the directory indicating that an order requires authorization.

**Figure 73** In Stock Output File

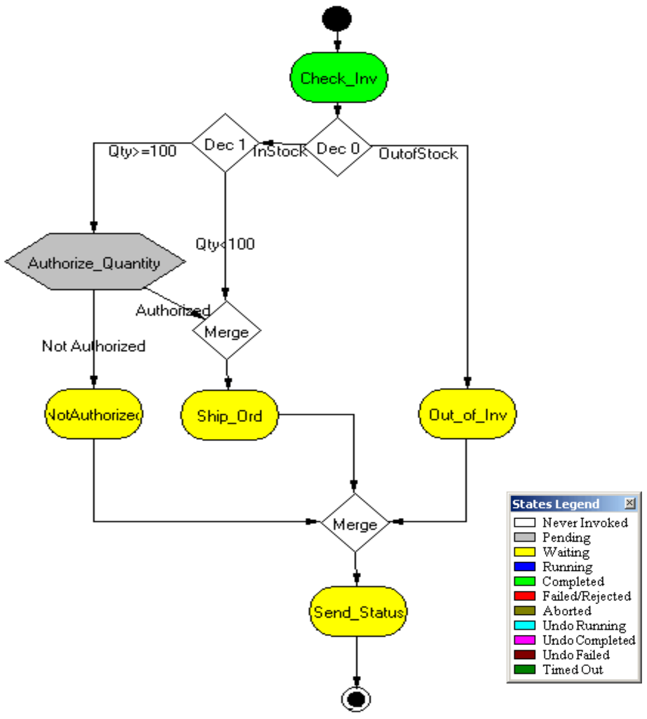


```
The following Business Process Instance requires Authorization: 20010522153610057З
```

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 74** Authorize_Quantity Pending BPI Diagram



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

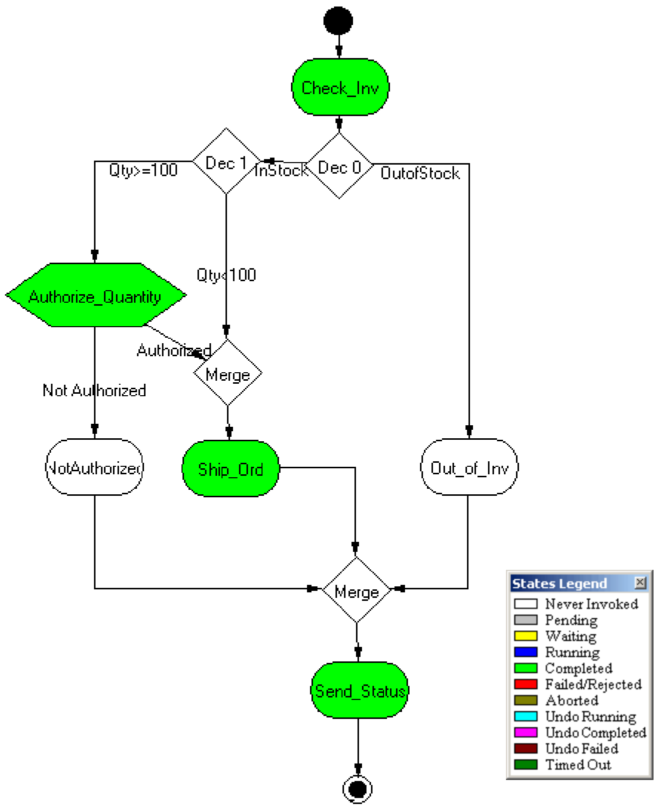
The **Authorization Activity Properties - Monitor Mode: (Authorize_Quantity)** is displayed.

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 will then complete using the route shown below in Figure 75.

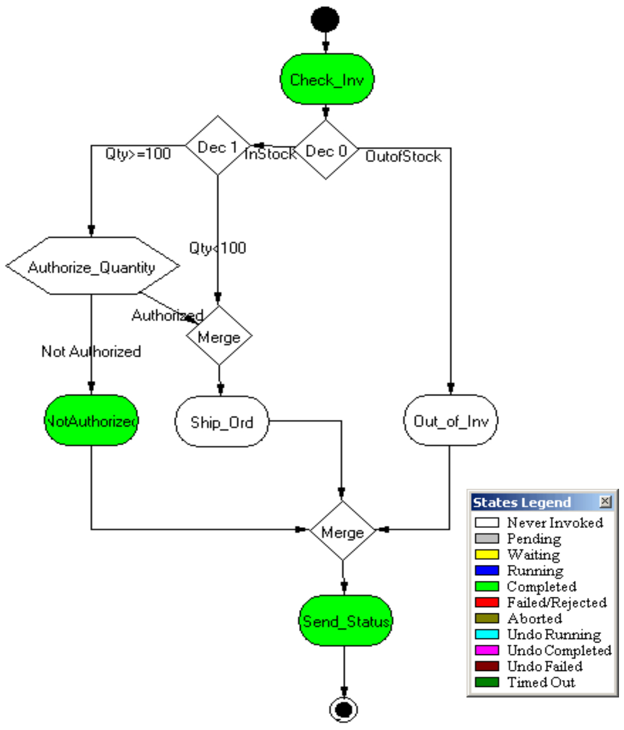
**Figure 75** Authorization_Quantity - Authorized



## Not Authorized Processing

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

**Figure 76** Authorization_Quantity - Not Authorized



Chapter 8

# 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 study could be enhanced to include the User Activity.

Overview of the User Activity

User Activities allow external applications to access attributes in the business process using an Attribute Access API. The API provides a set of functions that allow the external application to access attributes for the UserActivity 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 Functions" on page 270**.

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 infromation, 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 appllication is left up to the developer.

## 8.0.1 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**

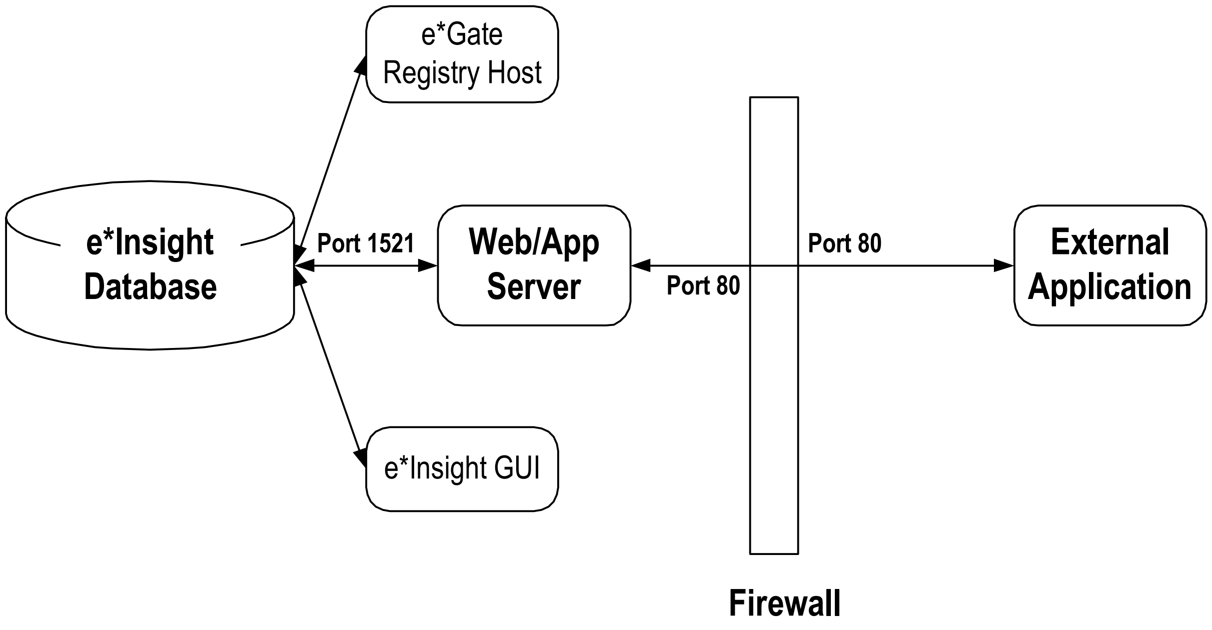
1. Use e*Xchange Administrator to create a user (for example, Connection_User), and assign a password.
2. Do not give this user any authorization rights within e*Insight.

**Note:** *For additional security, create the connection user directly in the database rather than using e*Xchange Administrator.*

### 8.0.2 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 108** is still valid when connecting via a Web/App Server, however there are additional considerations. The diagram below shows how e*Insight, the Web/App Server, and the external application may be deployed.

**Figure 77** 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*.

## 8.1 Overview

In this example, we will assume we have a custom Web application developed for the purpose of allowing various users to authorize 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 will use the Java API's provided with the e*Insight engine (a complete description and list of these APIs is available in **Chapter 15**, "e*Insight User Activity Functions" ). 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.

## 8.2 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.

### 8.2.1 Case Study: Order Processing with User Activity

The Order Process example will use an external application to allow various users to authorize the quantity. The external application will be retrieving new orders, and for those orders that are in stock, the users will evaluate all orders with quantity >100. Based on certain criteria, users will 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.*

## 8.3 Step 1: Create the ProcessOrder BP in e*Insight

The following is a summary of the procedures for adding a User Activity in the e*Insight GUI.

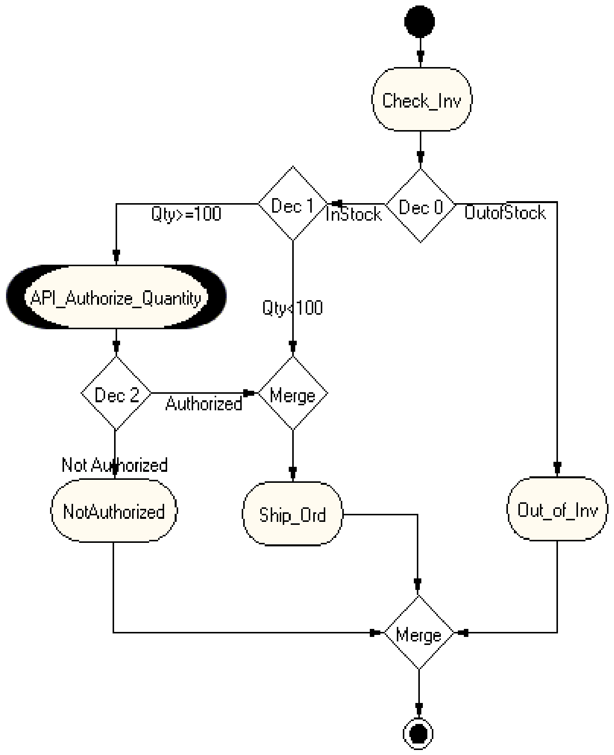
1. Delete the **Authorize_Quantity** Activity.
2. Add a User Activity called **API_Authorize_Order**.
3. Add a Decision gate to receive the output from the User Activity.

4 Make the connections between the Decision gates, Merge gate, and Not Authorized Activity, as shown in **Figure 78 on page 111**.

***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 78** ProcessOrder Business Process Model



## 8.4 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 will be 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 checkbox is not marked.

**Note:** *To send notifications through e*Gate, make sure the checkbox is marked, and configure your e*Way or BOB as in all other cases.*

## 8.5 Step 3: Run and Test the e*Insight scenario

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

### 8.5.1 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.

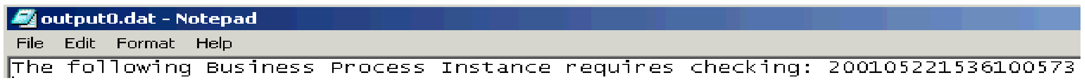
5 Navigate to the location for the input data file, **InStock.~in**, (**c:\eGate\client\data\ProcessOrder**) and change the extension to "**.fin**".

The example instance should have the inventory quantity set to >100. This will ensure 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.*

If everything is working correctly, an output file (**output#.dat**) as shown in Figure 79 below appears in the directory indicating that an order requires interaction from an application.

**Figure 79** User Activity Output File



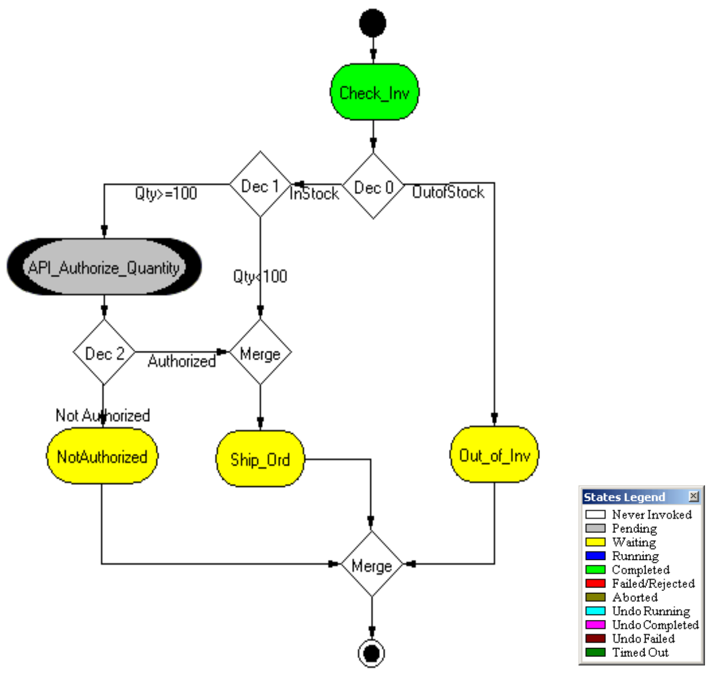
```
The following Business Process Instance requires checking: 20010522153610O573
```

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 will remain in a pending state until the e*Insight engine receives a message from an external application relating to this Business Process Instance.

The diagram below shows how the BPI Diagram appears.

**Figure 80** API_Check_Inv Pending BPI Diagram



7 Start the external application. Return to the e*Insight GUI. The Business Process will then complete using the route shown in **Figure 81 on page 114**.

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.

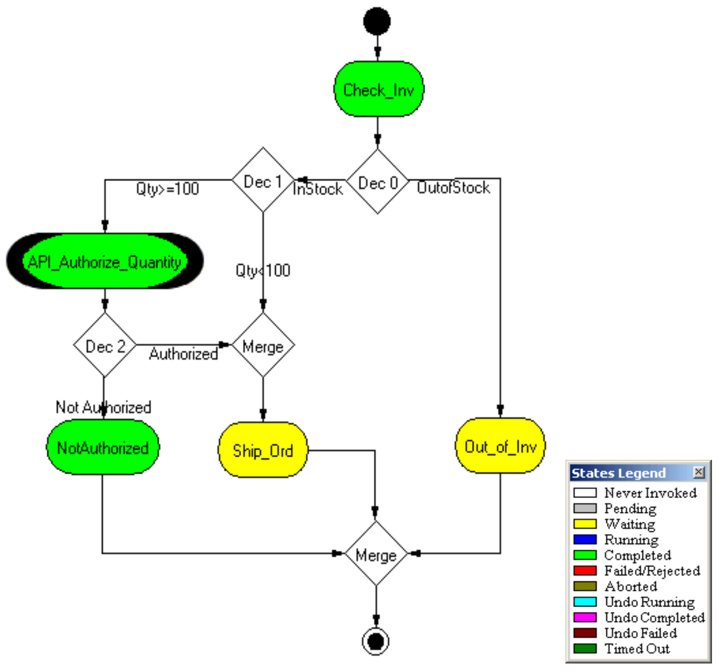
The external application uses an API to communicate with the e*Insight engine. Listed below are some examples of API functions to help illustrate what kind of information is available to the external application (a complete list of API functions can be found in Chapter 16).

- **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 **getLocal AttributeValue**—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. 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 will then send 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 81** User Activity - Item in stock



Chapter 9

# e*Insight Sub-Process Implementation

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" on page 60** for the initial configuration instructions.

## 9.1 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.

## 9.2 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 82 and the following tables to create the BP in the e*Insight.

**Figure 82** CheckInventory Business Process Model

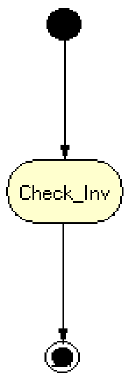


**Table 21** BP Global Attributes

| Attribute | Type | Data Direction |
|---|---|---|
| Item_Number | String | Input |
| Order_Quantity | Number | Input |
| Order_Status | String | Output |
| In_Stock | Boolean | Output |

**Table 22** Activity Attributes

| Activity | Attribute(s) | Input/Output |
|---|---|---|
| Check_Inv | Item_Number | Input |
|  | Order_Quantity | Input |
|  | In_Stock | Output |
|  | Order_Status | Output |

## 9.3 Configure the Integration Schema for CheckInventory

The Check_Inv activity will use the eX_Check_Inv BOB that was created in **“Creating the eX_Check_Inv BOB” on page 66**.

**To configure the Check_Inv Activity**

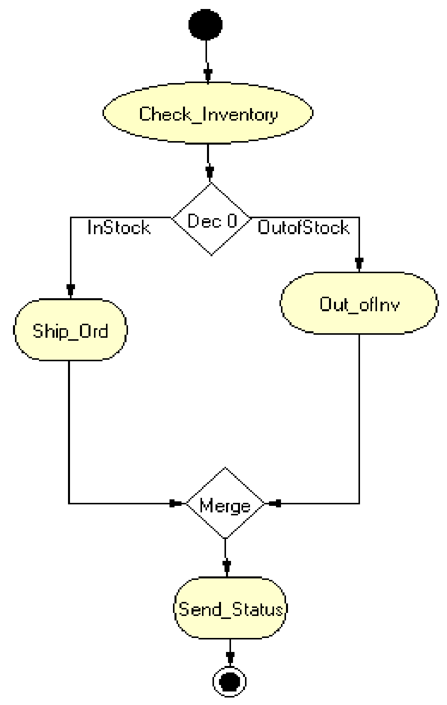
1. In the e*Insight GUI, open the Check_Inv activity properties.
2. On the **General** tab, select the **BOB** e*Gate module.
3. Check the module name is **eX_Check_Inv**, and modify if necessary.
4. Check that the correct Participating Host is selected.
5. Close the **Check_Inv** Activity properties.

## 9.4 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 83.

**Figure 83** 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 23.

**Table 23** 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*.

## 9.5 Configure the Integration Schema for ProcessOrder

The schema created in **"e*Insight Implementation" on page 58** does not need to be modified for this example.

## 9.6 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 87** to test your sub-process.

## 9.7 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.

## 9.8 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 124**.

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.

## 9.9 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 will be created later in **"To create eX_Check_Inv_CA.tsc using Monk" on page 123**.
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**.

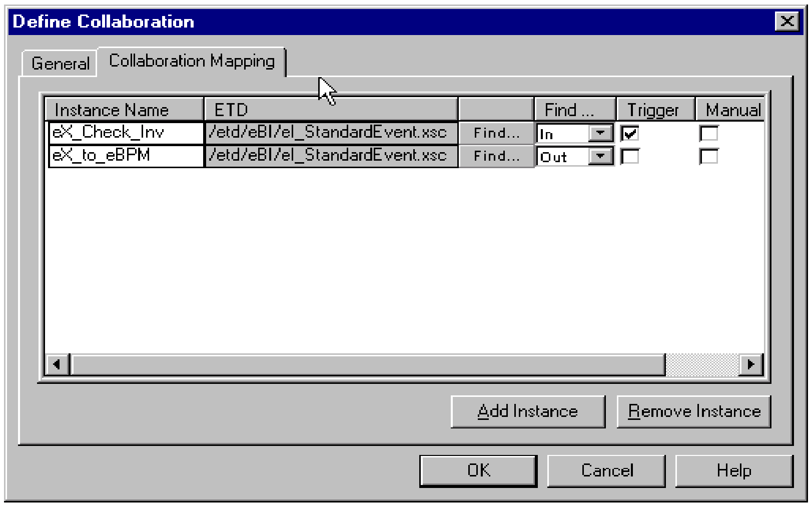
   You may be required to log into e*Gate.
10. Click **OK**, to close the information dialog.
11. Close the **Check_Inv_CA** Activity properties.

### 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. Add two instances as shown below.

**Figure 84** 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 will be created later in **"To create eX_Check_Inv_CA.xsc using Java" on page 123**.
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.

## 9.10 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 121**.
2. Rename the imported CheckInventory business process **OR**.
3. Rename the activity Check_Inv_OR.
4. Enable the business process version.

## 9.11 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 will be created later in **"To create eX_Check_Inv_OR.tsc using Monk" on page 126**.
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**.

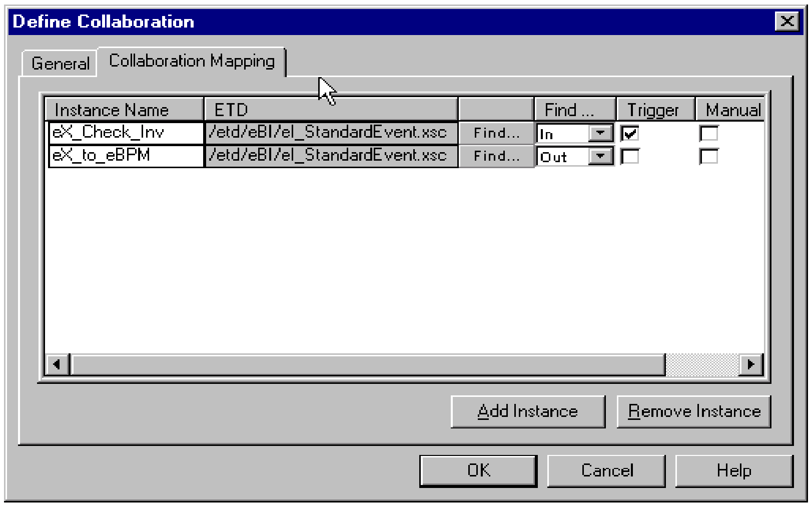
   You may be required to log into e*Gate.
10. Click **OK**, to close the information dialog.
11. Close the **Check_Inv_OR** Activity properties.

**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. Add two instances as shown below.

**Figure 85** 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 will be created later in **"To create eX_Check_Inv_CA.xsc using Java" on page 123**.
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.

## 9.12 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.

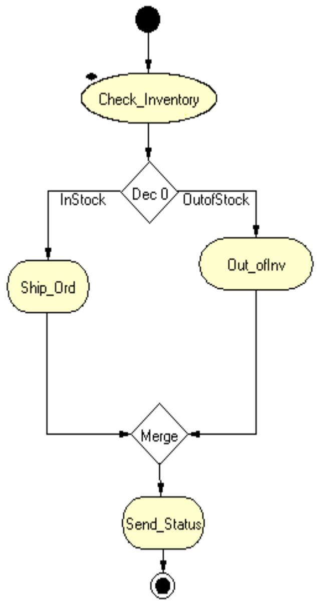
1 Save the enable ProcessOrder business process as a new version.

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 86.

**Figure 86** 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 23.

**Table 24** 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 the 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*.

## 9.13 Configure the Integration Schema for ProcessOrder

The schema created in **"e*Insight Implementation" on page 58** does not need to be modified for this example.

## 9.14 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 87** to test your sub-process.

Chapter 10

# 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 the previous example. See **"e*Insight Implementation" on page 60** for the initial configuration instructions.

## 10.1 Overview

The major topics covered in this chapter are:

- Installation and configuration of Tomcat
- Implementation steps

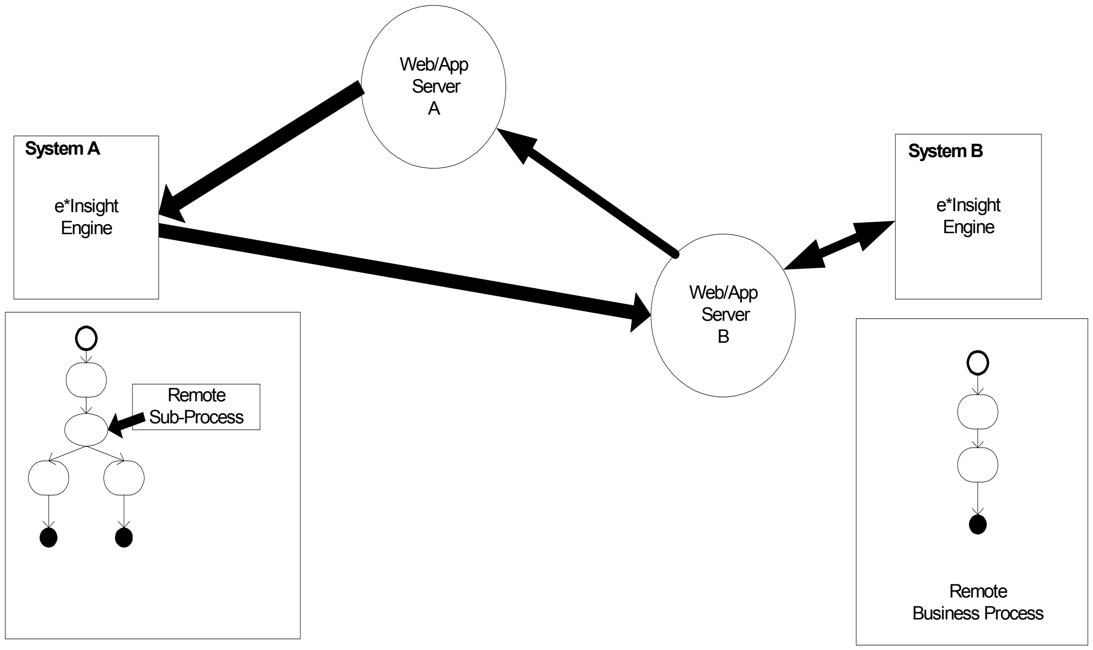
## 10.2 Overview of the Remote Sub-Process

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 132**.

**Figure 87** 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.

## 10.3 Installation and Configuration of Tomcat

This section describes how to install and configure Tomcat. This procedure is divided into three sections:

**Table 25** Steps for Installation and Configuration of Tomcat

| | Step | Procedure |
|---|---|---|
| 1 | Install Tomcat | **“Installing Tomcat” on page 130** |
| 2 | Configure Tomcat | **“Configuring Tomcat” on page 131** |
| 3 | Deploy the SOAP service | **“Deploying the SOAP Service” on page 131** |

### 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.

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.*

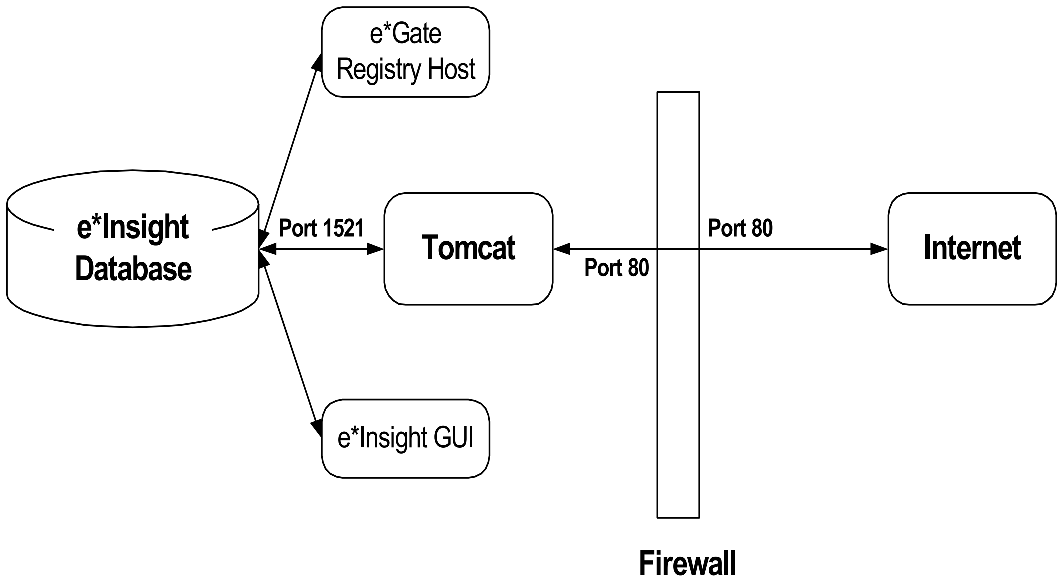
6 Start Internet Explorer and go to url: **http://<webhost>:8080/soap/admin/index.html**

7 Click List to check that the URN has been successfully deployed.

## 10.4 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 88 below 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 88** 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 131**.

## 10.5 Overview of the Remote Sub-Process Example

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 will be 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" on page 60** 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 will run.

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.

## 10.6 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 will assume that Tomcat and e*Insight run on the same machine.

Follow the instructions in **"Installation and Configuration of Tomcat" on page 130** for both machines. Define the URN's as follows:

| | |
|---|---|
| SystemA | urn:ProcessOrder |
| SystemB | urn:CheckInventory |

## 10.7 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 89 and the following tables to create the BP in the e*Insight.

**Figure 89** CheckInventory Business Process Model

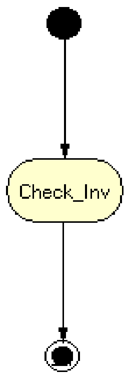


**Table 26** BP Global Attributes

| Attribute | Type | Data Direction |
|---|---|---|
| Item_Number | String | Input |
| Order_Quantity | Number | Input |
| Order_Status | String | Output |
| In_Stock | Boolean | Output |

**Table 27** 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, **SystemA**.

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**.

## 10.8 Configure the Integration Schema for CheckInventory

All the activities in this example are carried out using e*Gate components.

### 10.8.1 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**.

## 10.8.2 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 18** to make the required changes for the CheckInventory example.

## 10.8.3 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**.

## 10.8.4 Configure the Check_Inv activity for e*Gate

1 In the e*Insight GUI, select the CheckInventory business process.

2 From the **File** menu, select **e*Gate Login** and log in to the **CheckInventory** e*Gate schema.

3 In the e*Insight GUI, open the **Check_Inv** activity properties.

4 On the **General** tab, select the **BOB** e*Gate module.

5 Ensure that the participating host is correct, and the module name is eX_Check_Inv, modify if necessary.

6 Click **Configure e*Gate Schema**.

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

8 Close the **Check_Inv** Activity properties.

## 10.9 Modify the ProcessOrder BP in e*Insight

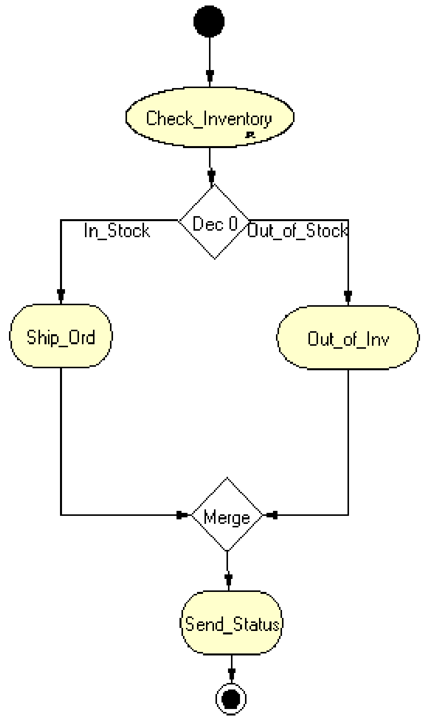
The following is a summary of the procedure for modifying the ProcessOrder BP in the e*Insight GUI.

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 90.

**Figure 90** ProcessOrder BP with Remote Sub-Process



4 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 28.

**Table 28** 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 |

### 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:ProcessOrder**.

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**.

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

## 10.10 Configure the Integration Schema for ProcessOrder

The schema created in **"e*Insight Implementation" on page 58** does not need to be modified for this example.

## 10.11 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.

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. Make sure that all of the e*Insight components, including the user-defined components, are set to start automatically.
3. Start Tomcat on both machines.
4. At the command line on SystemA, 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.

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

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

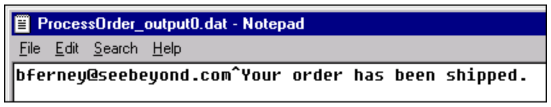
   Substitute the appropriate username and password for your installation.

6. 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.
7. 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 (**ProcessOrder_output#.dat**) as shown in Figure 91 below appears in the directory indicating successful completion of the BPI.

**Figure 91** In Stock Output File



```
bferney@seebeyond.com^Your order has been shipped.
```

8 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.

**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 11

# Activity Control Options

This chapter discusses the difference between implementing an actively controlled and passively controlled activity.

## 11.1 Overview

The major steps in the implementation are:

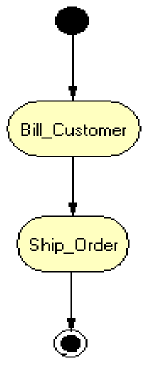
1. Create the business process (BP) in the e*Insight GUI.
2. Use the e*Insight GUI to configure the e*Gate schema that supports e*Insight.
3. Configure the e*Insight Engine.
4. Add and configure the user-defined e*Gate components.
5. Run and test the scenario.

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

The case study discussed in this chapter illustrates a simplified implementation of order processing. In this case, the e*Insight receives an incoming order as a delimited text file. Once the e*Insight has received the order the customer is billed and then the order is shipped. See Figure 92 for the Business Process diagram.

**Figure 92** Simple Order Process



The case study will be 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 29** BP Global Attributes

| Attribute | Type | Data Direction |
|---|---|---|
| Customer_Name | String | Input |
| Item_Number | String | Input |
| Order_Quantity | String | Input |

**Table 30** Activity Attributes

| Activity | Attribute(s) | Input/Output |
|---|---|---|
| Bill_Customer | Customer_Name | Input |
| | Order_Status | Output |
| Ship_Order | Cust_Name | Input |
| | Item_Number | Input |
| | Order_Status | Output |

## 11.1.2 Case Study - Active Control Mode

Figure 93, on the next page, shows the components involved in the business process implementation. Below is a description of how the data flows between these components for an item that is shipped successfully.

### Figure 93 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 will cause 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.

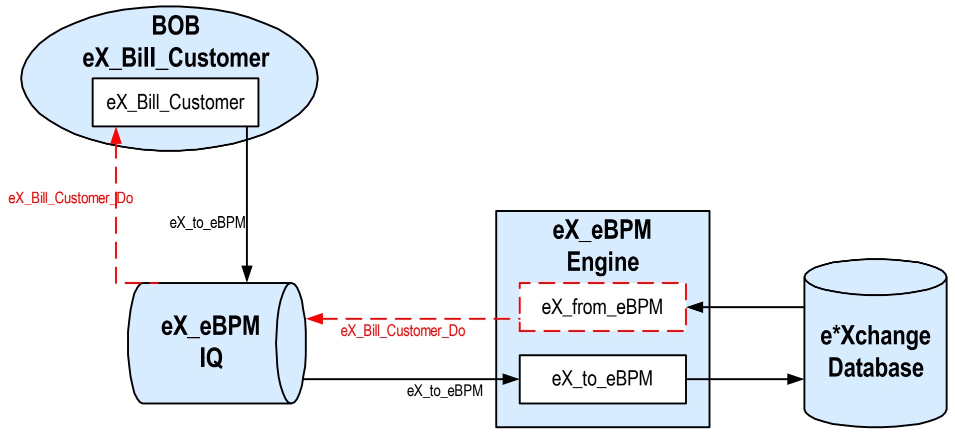
② The e*Insight engine publishes a "Do" Event (**eX_Bill_Customer_Do**) for first activity in the business process (**Bill_Customer**). **eX_Bill_Customer** BOB, the e*Gate component that corresponds to this activity in the business process, retrieves this Event from the **eX_eBPM** IQ and uses the information 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 to the **eX_eBPM** IQ.

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 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_Ship_Ord** BOB publishes a "Done" Event to the **eX_eBPM** IQ indicating that the order has been shipped.

**Figure 93** e*Insight Data Flow Diagram



## 11.2 Step 1: 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*.

## 11.3 Step 2: Configure the Integration Schema

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, eISchema, installed from the CD. Make any configuration changes to this copy and not to eISchema itself. See* **"Copy the eISchema" on page 49** *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 will run.

### Integration Schema Activity Components Summary

Use the information in Table 31 below to configure the e*Gate schema that supports the e*Insight example.

**Table 31** 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*.

## 11.4 Step 3: 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 will connect.

**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 Java e*Ways.*

### 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 18** to make the required changes for the ProcessOrder example.

## 11.5 Step 4: 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

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.*

### 11.5.1 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).

Follow these four steps to configure the **START_BP** e*Way:

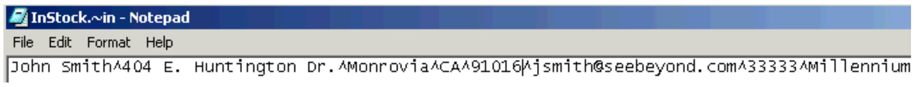
1. Create the Input ETD
2. Create the **START_BP** Collaboration Rules script (CRS)
3. Add the e*Way and create the e*Way configuration file
4. Configure the Collaboration in the GUI

## Step 1: Create the Input ETD

The input ETD is based on the format of the input data. The ProcessOrder 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 94 below. Place this data file at the directory location **c:\eGate\client\data\Order**.

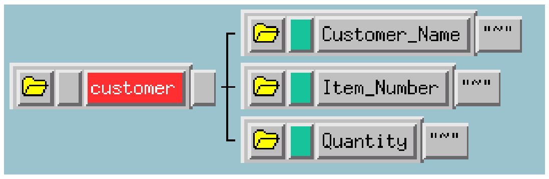
**Figure 94** Input Text File (SimpleOrder.~in)



```
John Smith^404 E. Huntington Dr.^Monrovia^CA^91016^jsmith@seebeyond.com^33333^Millennium
```

Using the ETD Editor and the input data as a guide, create an ETD like the one shown in Figure 95 below. For more information on using the ETD Editor see the ETD Editor's online help.

**Figure 95** Input ETD: Customer.ssc



## Step 2: Create the START_BP Collaboration Rules Script (CRS)

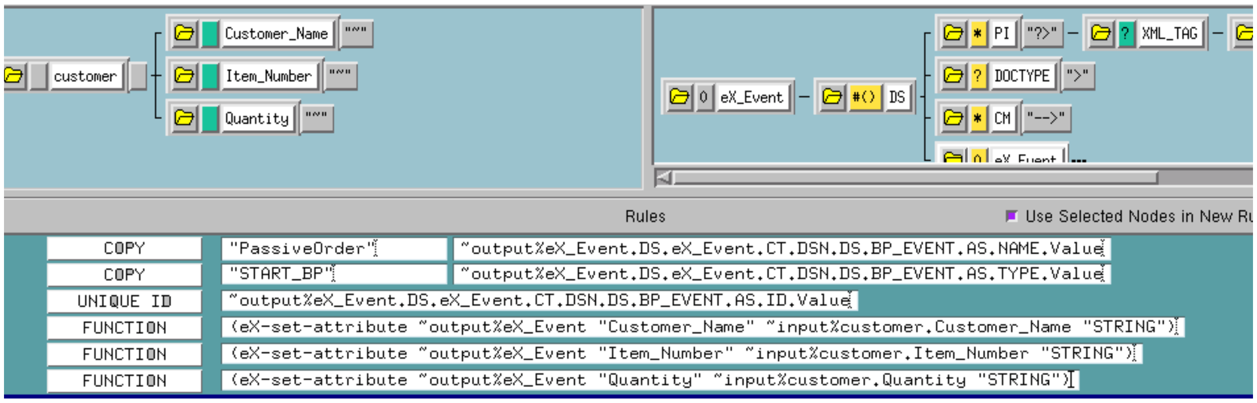
The Collaboration that sends the Event that starts the BPI must do two things:

- 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 96, on the following page, shows the **START_BP** CRS used in the WebOrder example:

**Figure 96** 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 ProcessOrder 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 32** 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.

## 11.5.2 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. Never-the-less, 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.

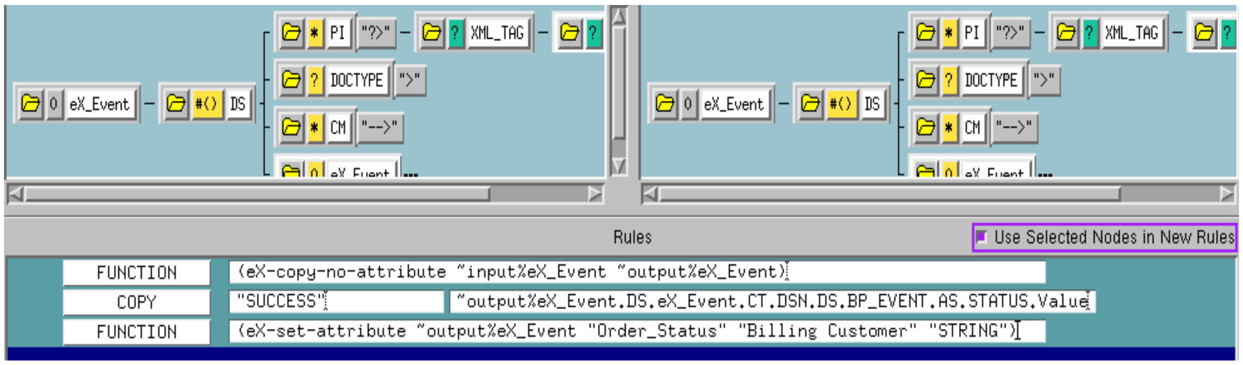
The CRSs associated with the BOBs used in this example are described in the following sections along with a screen capture showing the actual Monk Collaboration Rules Script used.

### Bill Customer CRS

The CRS sets the value of the Order_Status attribute to "Billing Customer" and sends a "SUCCESS" Event back to the e*Insight ending indicating that the activity has completed successfully.

Figure 97, shows the **Billing_Customer.tsc** CRS used in the e*Insight example.

**Figure 97** Billing_Customer.tsc CRS



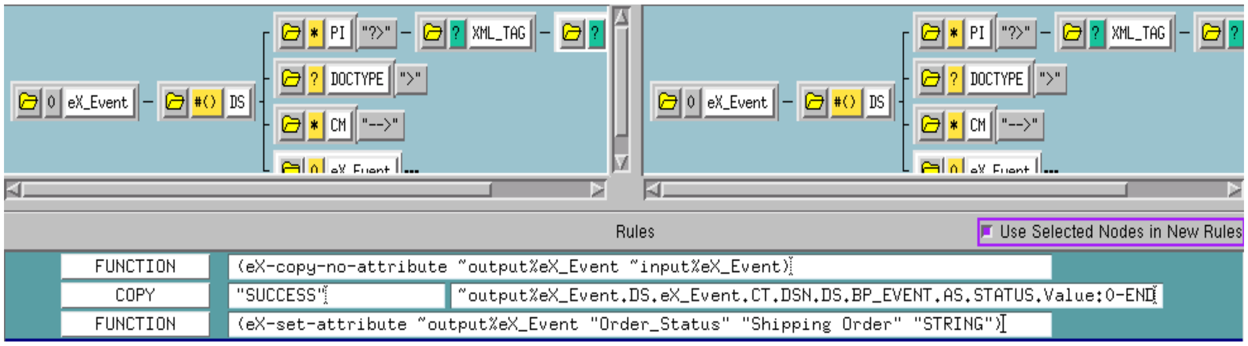
| | | |
|---|---|---|
| FUNCTION | (eX-copy-no-attribute ~input%eX_Event ~output%eX_Event) | |
| COPY | "SUCCESS" | ~output%eX_Event.DS.eX_Event.CT.DSN.DS.BP_EVENT.AS.STATUS.Value |
| FUNCTION | (eX-set-attribute ~output%eX_Event "Order_Status" "Billing Customer" "STRING") | |

### 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 send and returns "SUCCESS" to the e*Insight engine.

Figure 98 below shows the Ship_Order CRS used in the e*Insight example.

**Figure 98** 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.

## 11.6 Step 5: Run and Test the e*Insight scenario

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

### 11.6.1 Testing the Active Schema

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 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 94 on page 146** (**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. Out-of-Stock Processing

## 11.6.2 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.

### Figure 93 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 will cause 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.

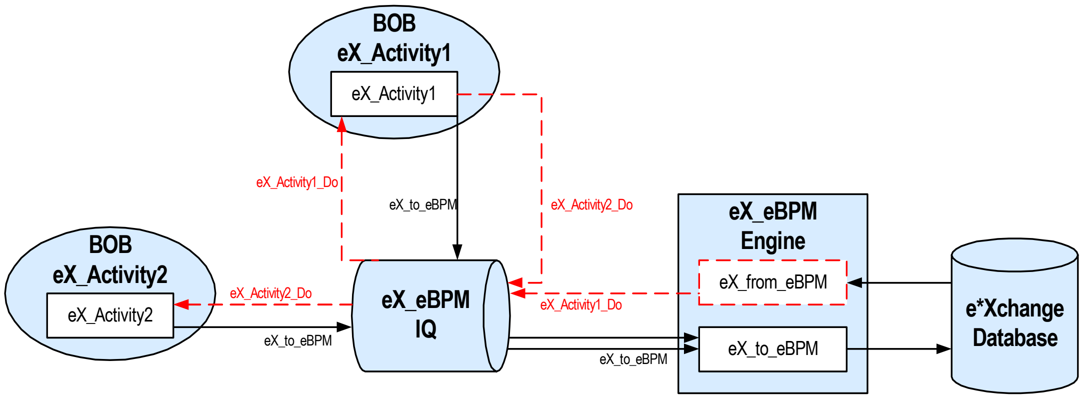
② The e*Insight engine publishes a "Do" Event (**eX_Bill_Customer_Do**) for first activity in the business process (**Bill_Customer**). **eX_Bill_Customer** BOB, the e*Gate component that corresponds to this activity in the business process, retrieves this Event from the **eX_eBPM** IQ and uses the information 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 to the **eX_eBPM** IQ.

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 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_Ship_Ord** BOB publishes a "Done" Event to the **eX_eBPM** IQ indicating that the order has been shipped.

**Figure 99** e*Insight Data Flow Diagram



## 11.7 Step 1: 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.

## 11.8 Step 2: Modify User-defined e*Gate Components

### 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.

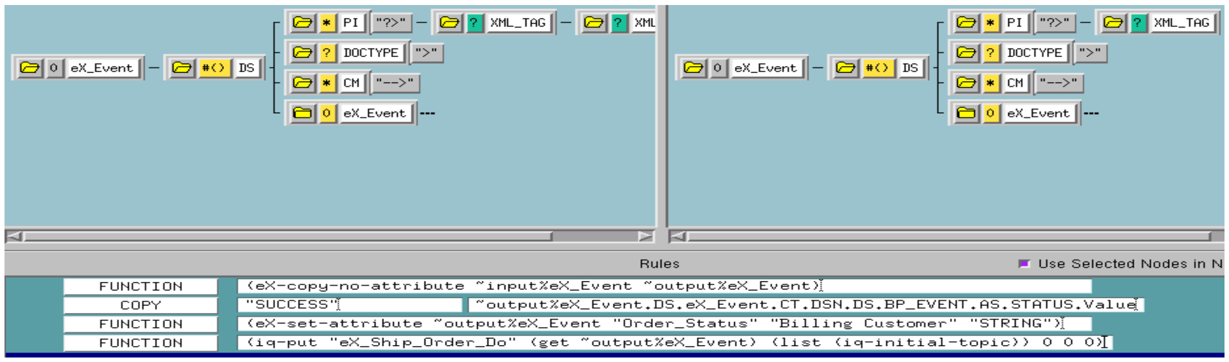
#### Bill Customer CRS

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 ending indicating that the activity has completed successfully.

The script is modified to use an iq-put to publish **eX_Ship_Order_Do**.

Figure 100, shows the **Billing_Customer.tsc** CRS used in the e*Insight example.

**Figure 100** 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)
```

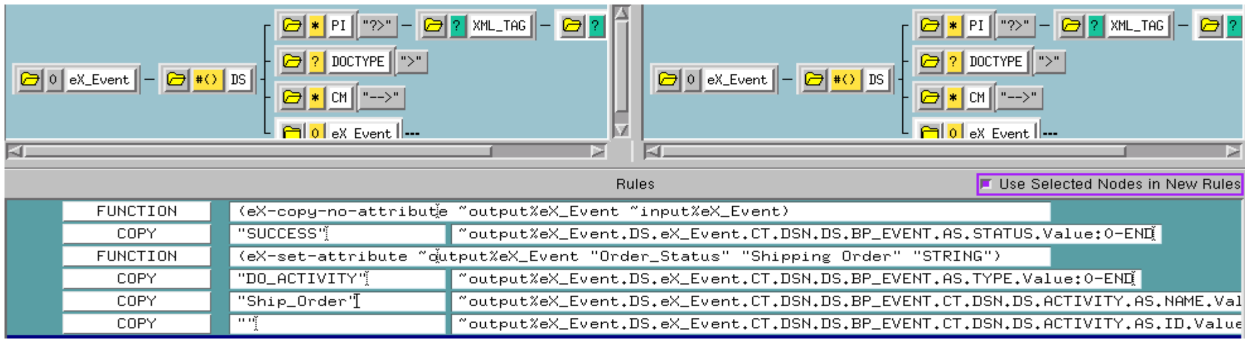
#### Ship_Order CRS

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 send and returns "SUCCESS" to the e*Insight engine.

In the active example, the event is sent from e*Insight with information pertaining to the Ship_Order activity. In the passive example event was originally created for the Bill_Customer activity, so the activity name needs to be update.

Figure 98 below shows the Ship_Order CRS used in the e*Insight example.

**Figure 101** 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 additonally 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.

Run the Schema again as described in **"Step 5: Run and Test the e*Insight scenario" on page 149**.

Chapter 12

# 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.

## 12.1 Log File Locations

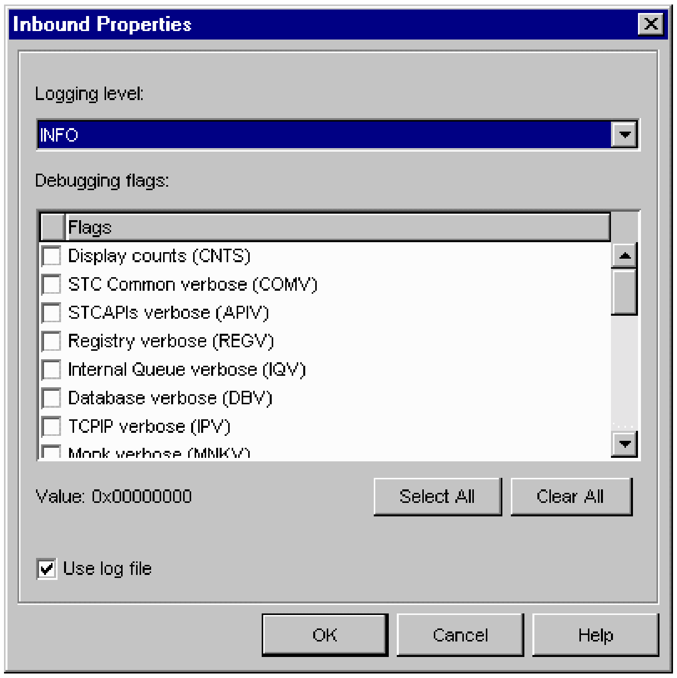
All log files are stored in the **\eGate\client\logs** directory on the Participating Host running the elements that generate the log entries. Logs are named after the component that creates them; for example, the **eX_eBPM** engine creates a log file called **eX_eBPM.log**.

## 12.2 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 102).

**Figure 102** 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 will 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)
- Monk (MNK)

For more information about logging and debugging options, see the *e*Gate Integrator System Administration and Operations Guide*.

## 12.3 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 33 shows a number of common problems and suggested actions you should take. The table also shows where to look for the error message and the logging required to display the error.

**Table 33** Common errors

| 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.<br>Also, check that a business process version has been enabled for this business process.<br>Note: Although the error message appears in the eX_eBPM log, the problem will be 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 33** Common errors

| 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.<br>Note: Although the error appears in the eX_eBPM log, the problem will be 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.<br>Note: Although the error appears in the eX_eBPM log, the problem will be 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.<br>Note: Although the error appears in the eX_eBPM log, the problem will be 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.<br>Also, check that the attribute name matches the name defined in the CRS.<br>Note: This error appears in the log for the e*Way or BOB for the activity. |

## 12.4 General Troubleshooting Tips

### 12.4.1 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.

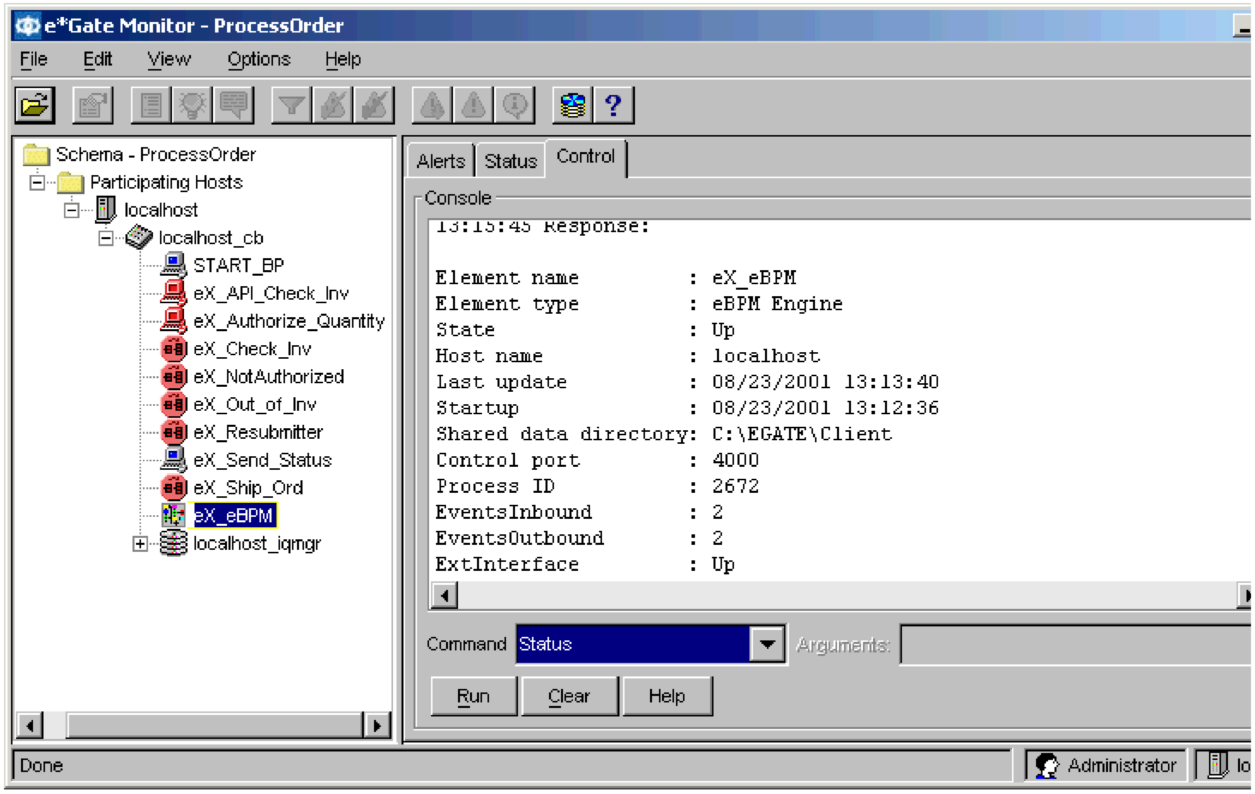
Use the e*Gate Enterprise Monitor GUI to check whether a message has been processed by a component. The number of inbound and outbound messages for a component is displayed by the status command.

**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 103 shows an example status display. EventsInbound and EventsOutbound display the number of messages processed.

**Figure 103** Example status display



```
13:13:45 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".*

## 12.4.2 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 33 31 33 34 34 38 30 34   |  "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 13

# 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 162**.

## 13.1 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:<br>▪ "BIN" - Interpreted as binary, however, must be suitably encoded for XML.<br>▪ "XML" - Interpreted as XML, however, must be Base64 encoded for XML.<br>▪ "STRING" - Interpreted as a string (default).<br>▪ "TRANSIENT" - Interpreted as a transient. The e*Insight engine will not process the value but simply return it as-is.<br>▪ "NUMBER" - Interpreted as a decimal number, however, must be given as a string.<br>▪ "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 will 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" on page 52** 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 14

# 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:

## 14.1 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(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

```
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 the e*Insight modeler for this Activity of 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

```
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(int index)
```

#### Description

**removeATTRIBUTE** removes a specific local Attribute from this ACTIVITY object.

#### Parameters

| Name | Type | Description |
|---|---|---|
| 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:<br>▪ "BIN" - Interpreted as binary, however, must be suitably encoded for XML.<br>▪ "XML" - Interpreted as XML, however, must be Base64 encoded for XML.<br>▪ "STRING" - Interpreted as a string (default).<br>▪ "TRANSIENT" - Interpreted as a transient. The e*Insight engine will not process the value but simply return it as-is.<br>▪ "NUMBER" - Interpreted as a decimal number, however, must be given as a string.<br>▪ "BOOLEAN" - Interpreted as a boolean, such as "true" and "false". |

| Name | Type | Description |
|---|---|---|
| encoding | java.lang.String | The encoding used for the value. Examples:<br>▪ "base64' - Standard MIME Base64 encoding.<br>▪ Null if plain text. |
| location | java.lang.String | The location of the actual data associated with the Attribute value. Examples:<br>▪ "FILE" - Attribute value specifies a file where actual data exists.<br>▪ "DB" - Attribute value is a reference to a row in a database table.<br>▪ "URL" - Attribute value specifies a URL of where actual data exists.<br>▪ "EMBEDDED" - Attribute value is the actual data (default).<br>▪ "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");
```

## setID

### Syntax

```
void setID(java.lang.String val)
```

### Description

**setID** sets the internal unique identifier created by the e*Insight modeler for this Activity of 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

## 14.2 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 will be 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

```
getNAME();
   => "Ship_Order"
```

## 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; will be 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

```
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.ContentHandler | 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:<br>▪ "FILE" - Attribute data is the name of a file where actual data is stored.<br>▪ "DB" - Attribute data is a reference such as "ROWID" to a row in a table.<br>▪ "URL" - Attribute data is the URL to where the actual data is stored.<br>▪ "EMBEDDED" - Attribute data is the actual data (this is the default).<br>▪ "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:<br>▪ "BIN" - Attribute data is binary in nature. However, must be safely encoded for XML.<br>▪ "XML" - Attribute data represents a XML message. However, must be base64 encoded.<br>▪ "STRING"- Attribute data appears as clear-text string (this is the default).<br>▪ "TRANSIENT" - Attribute data is not persisted in the e*Insight database; will be passed-thru.<br>▪ "NUMBER" - Attribute data represents a decimal numeric string.<br>▪ "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.

## 14.3 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 233

**hasID** on page 234

**hasNAME** on page 235

**hasSTATUS** on page 236

**marshal** on page 237

**setSTATUS** on page 251

**setTYPE** on page 252

**toString** on page 253

**unmarshal** on page 254

### 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(int index)
```

#### Description

**removeATTRIBUTE** removes a specific global Attribute from the Business Process object.

#### Parameters

| Name | Type | Description |
|---|---|---|
| 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:<br>▪ "BIN" - Interpreted as binary, however, must be suitably encoded for XML.<br>▪ "XML" - Interpreted as XML, however, must be Base64 encoded for XML.<br>▪ "STRING" - Interpreted as a string (default).<br>▪ "TRANSIENT" - Interpreted as a transient. The e*Insight engine will not process the value but simply return it as-is.<br>▪ "NUMBER" - Interpreted as a decimal number, however, must be given as a string.<br>▪ "BOOLEAN" - Interpreted as a boolean, such as "true" and "false". |
| encoding | java.lang.String | The encoding used for the value. Examples:<br>▪ "base64' - Standard MIME Base64 encoding.<br>▪ Null if plain text. |

| Name | Type | Description |
|---|---|---|
| location | java.lang.String | The location of the actual data associated with the Attribute value. Examples:<br>▪ "FILE" - Attribute value specifies a file where actual data exists.<br>▪ "DB" - Attribute value is a reference to a row in a database table.<br>▪ "URL" - Attribute value specifies a URL of where actual data exists.<br>▪ "EMBEDDED" - Attribute value is the actual data (default).<br>▪ "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:<br>▪ "SUCCESS" - Indicates that the current Activity has successfully completed.<br>▪ "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:<br>▪ "START_BP" - Instructs the e*Insight engine to start a Business Process Instance.<br>▪ "DO_ACTIVITY" - Indicates a "Do" Event for the current Activity.<br>▪ "UNDO_ACTIVITY" - Indicates an "Undo" Event for the current Activity.<br>▪ "UNDO_BPI" - Indicates an "Undo" for the entire Business Process Instance.<br>▪ "RESTART_ACTIVITY" - Indicates restarting the current Activity after it has paused.<br>▪ "SKIP_ACTIVITY" - Indicates the current Activity should be skipped.<br>▪ "RELOAD_BP" - Indicates the Business Process definition should be reloaded.<br>▪ "AUTHORIZE" - Indicates that the current Activity should be authorized.<br>▪ "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.

## 14.4 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 Management 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** on page 257

**getBP_EVENT** on page 258

**getTP_EVENT** on page 259

**hasBP_EVENT** on page 260

**hasTP_EVENT** on page 261

**marshal** on page 262

**omitBP_EVENT** on page 263

**omitTP_EVENT** on page 264

**setBP_EVENT** on page 265

**setTP_EVENT** on page 266

**to_eBPMConvert** on page 267

**toString** on page 268

**unmarshal** on page 269

## 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.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

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 15

# e*Insight User Activity API Functions

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.

## 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 271**
- **"UserActivityMessage Class" on page 295**
- **"IClient Interface" on page 296**
- **"EbpmMonitor Class" on page 334**

## 15.1 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>
```

## 15.2 UserActivityMessage Class

Implements IMessage.

See **“Imessage Interface” on page 271**.

## 15.3 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”
```

## 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.<br>For example:<br>dbURL=jdbc:oracle:thin:@hostname:1521:dbname<br>userID=ex_admin<br>password=xxxxx<br>driverName=oracle.jdbc.driver.OracleDriver<br>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”);
```

## 15.4 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 296, for details of methods contained in this interface.

Additional methods are described in detail on the following pages:

**checkUserPrivileges** on page 335

## 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