SeeBeyond ICAN Suite

e*Insight Business Process Manager Implementation Guide

Release 5.0.5 for Schema Run-time Environment (SRE)

Version 20050502110609.

Contents

Chapter 1

Introduction 15

Document Purpose and Scope 15
Intended Audience 15
Writing Conventions 16
SeeBeyond Web Site 17

Chapter 2

Introduction to the SeeBeyond Business Integration Suite 18

SeeBeyond Business Integration Suite 18
SeeBeyond Business Integration Suite Components 19
e*Gate Integrator Components 20
Introducing e*Insight Business Process Manager (e*Insight) 21
Building an Integration Application 22
Basic Information 22

Chapter 3

Implementation Overview 24

Basic Information 24
Implementation Road Map 24
Step 1: Create a Business Process 25
Step 2: Create the e*Insight Schema from a Template 25
Step 3: Configure the e*Insight Schema 25
Step 4: Configure the e*Gate Components 26
Step 5: Test and Tune the System 26
The e*Insight Schema 26
The eIJSchema (Java) 26
The eISchema (Classic) 27

Chapter 4

e*Insight Schema Components (eIJSchema) 28

e*Insight Schema Components Overview 28
Schema Component Types 28
e*Insight Schema Component Relationships 29
e*Insight Business Process Manager Components 30
Start a Business Processes 30
Run a Business Processes 30
Implement Business Process Activities 31
e*Insight Engine 31
Configuring the e*Insight Engine 31

Using XA 32
How XA Protocol Works 32
Database Requirements for XA Compliance 32
Oracle 32
Sybase 32
Microsoft SQL Server 33
e*Way Connection Configuration Changes 33
ORACLE DataSource Section 33
Sybase DataSource Section 34
Microsoft SQL Server DataSource Section 35
eBPM Settings Section 35
Performance Improvements 35
eBPM Tuning 35
Parameter: Maximum Commit Frequency 36
Parameter: Receive Wait Timeout 36
Parameter: Stability Threshold 36
Parameter: Show Change Audit 37
Connection Management 37
Connector Section Changes 37
Manual Database XA Recovery 38
Engine Connection Configuration 38
eIcr_eBPM Collaboration 46
eI_Resubmitter BOB 47
Configuring the eI_Resubmitter BOB 47
eI_Resubmitter Collaboration 48
Failed Event Handling by the e*Insight Engine 48
Error Types 48
Error Handling 48
START_BP Component 49
Configuring the START_BP Component 49
START_BP Collaboration 49
Multi-Mode Activity e*Way 50
eX_*Activity* Collaboration 50
Single-Mode Activity e*Way 51
Configuring the eX_to_*Activity* e*Way 52
eX_to_*Activity* Collaboration 52
eX_from_*Activity* Collaboration 53
Activity BOB 54
eX_*Activity* Collaboration 54

Using Monk in eIJSchema 55

Chapter 5

Understanding the e*Insight ETD 57

Selecting an ETD 57
About Business Process Attributes 58
Global Attributes 58
Local Attributes 59

Activity Specific ETDs 60
GlobalAttributes 60
LocalAttributes 60
EventSpecifics 61

eI_StandardEvent 62
BP_EVENT 62
BP_EVENT Element 63
BP_EVENT.ACTIVITY Nodes 64
ACTIVITY Element 65
ACTIVITY.ATTRIBUTE Element 65
BP_EVENT.ATTRIBUTE Nodes 66

Chapter 6

e*Insight Implementation (eIJSchema) 68

Overview 68
Case Study: Payroll Processing 69

Create the Payroll BP in e*Insight 72
Creating the processes performing the Activities 74
Configuring the e*Insight Script for Update_Status 74

Configure the Integration Schema (e*Insight) 75
Integration Schema Activity Components Summary 75
Creating the eX_Check_Eligibility Multi-Mode e*Way 76
Creating the eX_Calculate_Bonus BOB 78
Process_Payroll e*Way Configuration 79

Configure the Integration Schema (e*Gate) 79
Configure the e*Insight Engine 79
Edit the eIcp_eInsightEngine Connection Configuration File 79
Configure the JMS Connection 79

Configure User-defined e*Gate Components 80
Configuration Order for the User-defined Components 80
Configure the START_BP e*Way 80
Step 1: Create the START_BP e*Way 81
Step 2: Create the Input ETD 81
Step 3: Create the START_BP Collaboration 82
Step 4: Configure the Collaboration in the GUI 85
Configure the Process_Payroll e*Way 85

Step 1: Create the Output ETD:PayrollProcess.xsc using Java 86
Step 3: Create the Process_Payroll Collaboration Rule 86
Step 4: Configure the Collaboration 87
Step 1: Configure the e*Way 88
Run and Test the e*Insight Scenario 89
Testing the Standard Business Logic 89
Payroll Processing 89
Not Eligible Processing 90
Demonstrating Business Process Undo Functionality 92
Manual Undo 93
Demonstrating Business Process Restart Functionality 94
Repairing a String Attribute 94

Chapter 7

e*Insight Authorization Activity Implementation (eIJSchema) 97

Overview 97
Case Study: Payroll Processing 97
Step 1: Update the Payroll BP in e*Insight 98
Creating the processes performing the Activities 99
Configuring the e*Insight Script for Bonus_Refused 99
Step 2: Run and Test the e*Insight scenario 100
Testing the Standard Business Logic 100
Authorized Processing 100
Not Authorized Processing 102

Chapter 8

e*Insight User Activity Implementation 104

Overview of the User Activity 104
User Activity Security 104
Deployment of the User Activity 105
Overview of the Payroll BP 105
Overview 106
Case Study: Payroll Processing with User Activity 106
Step 1: Update the Payroll BP in e*Insight 106
Step 2: Configure the Integration Schema 107
Step 3: Run and Test the e*Insight scenario 107
Testing the User Activity 108
Overview of the ProcessOrder BP 110
Overview 111
Case Study: Order Processing with User Activity 111
Step 1: Update the ProcessOrder BP in e*Insight 111

Step 2: Configure the Integration Schema 112
Step 3: Run and Test the e*Insight scenario 113
Testing the User Activity 113

Chapter 9
e*Insight Sub-Process Implementation (eIJSchema) 116
Overview of the Sub-Process Example 116
Create the CalculateBonus BP in e*Insight 117
Configure the Integration Schema for CalculateBonus 118
Modify the Payroll BP in e*Insight 119
Configure the Integration Schema for Payroll 120
Run and Test the e*Insight scenario 120
Overview of the Dynamic Sub-Process Example 121
Create the accounts BP in e*Insight 122
Configure the Integration Schema for accounts 122
Creating the CRS in e*Gate 123
Create the marketing BP in e*Insight 124
Configure the Integration Schema for marketing 124
Creating the CRS for eX_Calculate_Bonus_marketing in e*Gate 124
Modify the Payroll BP in e*Insight 126
Configure the Integration Schema for Payroll 126
Run and Test the e*Insight scenario 127

Chapter 10
e*Insight Sub-Process Implementation (eISchema) 128
Overview of the Sub-Process Example 128
Create the CheckInventory BP in e*Insight 129
Configure the Integration Schema for CheckInventory 130
Modify the ProcessOrder BP in e*Insight 131
Configure the Integration Schema for ProcessOrder 132
Run and Test the e*Insight scenario 132
Overview of the Dynamic Sub-Process Example 133
Create the CA BP in e*Insight 134
Configure the Integration Schema for CA 134
Creating the CRS in e*Gate 136

Create the OR BP in e*Insight 137
Configure the Integration Schema for OR 137
Creating the CRS in e*Gate 139
Modify the ProcessOrder BP in e*Insight 140
Configure the Integration Schema for ProcessOrder 141
Run and Test the e*Insight scenario 141

Chapter 11
e*Insight Remote Sub-Process Implementation 142
Overview 142
Overview of the Remote Sub-Process 142
Installation and Configuration of Tomcat 143
Installing Tomcat 143
Configuring Tomcat 144
Deploying the SOAP Service 144
Installation of Tomcat and e*Insight on Different Hosts 145
Overview of the Remote Sub-Process Example (eIJSchema) 146
Install and configure Tomcat 146
Create the CalculateBonus BP in e*Insight 147
Configure the Integration Schema for CalculateBonus 148
Create the CalculateBonus Schema 148
Configure the CalculateBonus Schema 149
Create the Calculate_Bonus activity BOB 149
Edit the eIcp_eInsightEngine Connection Configuration File 149
Configure the JMS Connection 149
Modify the Payroll BP in e*Insight 150
Configure the Integration Schema for Payroll 151
Run and Test the e*Insight scenario 152
Overview of the Remote Sub-Process Example (eISchema) 152
Install and configure Tomcat 153
Create the CheckInventory BP in e*Insight 153
Configure the Integration Schema for CheckInventory 154
Create the CheckInventory Schema 154
Configure the e*Insight engine 155
Create the Check_Inv activity BOB 155
Modify the ProcessOrder BP in e*Insight 156
Configure the Integration Schema for ProcessOrder 157
Run and Test the e*Insight scenario 157

In-Stock Processing 157

Chapter 12

Active and Passive Modes 159

Overview 159
Case Study 159
Case Study - Active Control Mode 160
Case Study - Passive Control Mode 161

Create the Order BP in e*Insight 163

Configure the Integration Schema (eIJSchema) 164
Integration Schema Activity Components Summary 164
Creating the eX_Bill_Customer BOB 165
Creating the eX_Ship_Order BOB 166

Configure the e*Insight Engine (eIJSchema) 167
Edit the eIcp_eInsightEngine Connection Configuration File 167
Configure the JMS Connection 167

Configure User-defined e*Gate Components (eIJSchema) 167
Configuration Order for the User-defined Components 168
Configure the START_BP e*Way 168
Step 1: Create the START_BP e*Way 168
Step 2: Create the Input ETD 169
Step 3: Create the START_BP Collaboration 170
Step 4: Configure the Collaboration in the GUI 172

Configure the Integration Schema (eISchema) 173
Integration Schema Activity Components Summary 173

Configure the e*Insight Engine (eISchema) 173
Edit the eX_eBPM Engine's Configuration File 173

Configure User-defined e*Gate Components (eISchema) 174
Configuration Order for the User-defined Components 174
Configure the START_BP e*Way 174
Step 1: Create the Input ETD 174
Step 2: Create the START_BP Collaboration Rules Script (CRS) 175
Step 3: Add the e*Way and Create the e*Way Configuration File 175
Step 4: Configure the Collaboration in the GUI 176
Configure the Activity BOBs 176
Create the Activity BOB CRSs 176
Configure the Activity BOB Collaborations in the Schema Designer 177

Run and Test the e*Insight scenario 178

Case Study - Passive Control Mode 179

Passive Control Mode (eIJSchema) 179
Modify the Order BP in e*Insight (eIJSchema) 179
Modify User-defined e*Gate Components (eIJSchema) 179
Configuration Order for the User-defined Components 179
Configure the Bill_Customer Collaboration Rule and Collaboration 179
Run and Test the e*Insight scenario 182

Passive Control Mode (eISchema) 183
Modify the Order BP in e*Insight (eISchema) 183
Modify User-defined e*Gate Components (eISchema) 183
Configuration Order for the User-defined Components 183
Configure the Bill_Customer Collaboration Rule and Collaboration 184
Run and Test the e*Insight scenario 184

Chapter 13
e*Insight Performance 185
Performance Improvements Using eIJSchema 185
Instance Caching 185
Using Multiple e*Insight Engines 186
e*Insight Engine Affinity (eIJSchema) 189
Using Engine Affinity with e*Gate — Active Mode 189
Configuring the Engine Affinity JMS Properties 190
Using Engine Affinity with e*Gate — Passive Mode 190
Using e*Xchange with e*Insight (eIJSchema) 191
Using Binary XML (eIJSchema) 191
Subscribing to Event Types 192
Subscribing to a Single "Go" Event 192
Configuring a Separate Collaboration for Do and Undo Events 192
Removing Unnecessary Subscriptions 193
Event Type "get" Interval — JMS Server 193
Event Type "get" Interval — e*Insight Engine 193
Review JVM Settings 193
Performance Improvements Using eISchema 194
Instance Caching 194
Using Multiple e*Insight Engines (eISchema) 194
e*Insight Engine Affinity (eISchema) 195
Manually Publishing Events using eX-event-sendback-to-sender 196
Exchange Data Interval (eISchema) 197
Review JVM Settings 197
General e*Insight Performance Tips 197

Chapter 14
Troubleshooting 199
Log Files 199
Generating Log Files 199
Reading Log Files 200
Common Problems 202
General Troubleshooting Tips 206
Locating the problem 206
Viewing the Message Content 207

Chapter 15

Java Helper Methods 209

ACTIVITY Class 210

ATTRIBUTE Class 230

BP_EVENT Class 248

eX_StandardEvent Class 284

Chapter 16

e*Insight User Activity API Methods 299

User Activity Security 299
Defining the Classpath 300

Imessage Interface 301
clearMessage 301

UserActivityMessage Class 324

IClient Interface 325

EbpmMonitor Class 363
checkUserPrivileges 363

Appendix A

e*Insight Schema Components (eISchema) 364

The Purpose of the e*Gate Schema for e*Insight 364
e*Insight Components 364

e*Insight Schema Components Overview 364
Additional Components 365
e*Insight Schema Component Relationships Diagram 366

e*Insight Business Process Manager Components 367
Components That Run Business Processes 367
Components that Start Business Processes 368
Components that Implement Business Process Activities 368
e*Insight Engine 368
Configuring the e*Insight Engine 369
eX_from_eBPM Collaboration 373
eX_to_eBPM Collaboration 373
eI_Resubmitter BOB 373
Configuring the eI_Resubmitter BOB 374
eI_Resubmitter Collaboration 374
Failed Event Handling by the e*Insight Engine 374
Error Types 374
Error Handling 375
START_BP Component 375
Configuring the START_BP Component 376

START_BP Collaboration 376
Activity e*Way 377
Configuring the eX_to_*Activity* e*Way 378
eX_to_*Activity* Collaboration 379
eX_from_*Activity* Collaboration 380
Activity BOB 380
eX_*Activity* Collaboration 381

Appendix B

The e*Insight ETD for Monk 382
ETD Structure 382
XML Element with Sub-elements 383
XML Element without sub-elements 383
XML Attribute 384
Element Overview 384
Example: XML Element with Sub-elements 385
Example: XML Element with Attributes 385
Using eX_Standard_Event.ssc 386
BP_EVENT 386
BP_EVENT.AS Nodes 386
BP_EVENT.CT.DSN.DS.ACTIVITY Nodes 388
ACTIVITY.AS Nodes 388
ACTIVITY.CT.DSN.DS.ATTRIBUTE Nodes 388
BP_EVENT.CT.DSN.DS.ATTRIBUTE.AS Nodes 389

Appendix C

Common Configuration Tasks 391
Copy the e*Insight Schema 391
Using the e*Insight GUI 391
Copying the Schema from the Registry Host 392
Installing from the CD 393
Sending Messages to the e*Insight Engine (eIJSchema) 393
Starting a Business Process (eIJSchema) 393
Setting Attributes (eIJSchema) 394
Getting Attributes (eIJSchema) 395
Sending the "Done" Event Back to e*Insight (eIJSchema) 395
Sending Messages to the e*Insight Engine (eISchema) 396
Starting a Business Process (eISchema) 396
Setting Attributes (eISchema) 397
Setting Attributes in a Monk Collaboration 397
Setting Attributes in a Java Collaboration 398
Getting Attributes (eISchema) 398
Getting Attributes in a Monk Collaboration 398
Getting Attributes in a Java Collaboration 399
Sending the "Done" Event Back to e*Insight (eISchema) 399

Appendix D

e*Insight Authorization Activity Implementation (eISchema) 402

Overview 402
Case Study: Order Processing 402
Step 1: Create the ProcessOrder BP in e*Insight 405
Step 2: Configure the Integration Schema 406
Integration Schema Activity Components Summary 406
Step 3: Configure User-defined e*Gate Components 406
Configure the Activity BOB CRS in the Schema Designer 406
Configure the Activity BOB Collaborations in the Schema Designer 407
Configure the Authorize_Quantity e*Way 407
Step 2: Create the Authorize_Quantity.tsc CRS 407
Step 3: Configure the e*Way 408
Step 4: Configure the Collaboration 408
Step 5: Run and Test the e*Insight scenario 409
Testing the Standard Business Logic 409
Authorized Processing 409
Not Authorized Processing 411

Appendix E

e*Insight Implementation (eISchema) 413

Overview 413
Case Study: Order Processing 414
Create the ProcessOrder BP in e*Insight 417
Creating the processes performing the Activities 419
Configuring the e*Insight Script for Ship_Ord 419
Configure the Integration Schema 420
Integration Schema Activity Components Summary 420
Creating the eX_Check_Inv BOB 421
Creating the eX_Out_of_Inv BOB 424
Send_Status e*Way Configuration 426
Configure the e*Insight Engine 427
Edit the eX_eBPM Configuration File 427
Configure User-defined e*Gate Components 427
Configuration Order for the User-defined Components 427
Configure the START_BP e*Way 428
Step 1: Create the START_BP e*Way using Monk 428
Step 2: Create the Input ETD using Monk 429
Step 3: Create the START_BP CRS using Monk 429
Step 4: Configure the START_BP Collaboration in the GUI using Monk 430
Step 1: Create the START_BP e*Way using Java 431
Step 2: Create the Input ETD using Java 431
Step 3: Create the START_BP Collaboration using Java 432

Step 4: Configure the Collaboration in the GUI using Java 434
Configure the Send_Status e*Way 435
Step 1: Configure the eX_Send_Status e*Way using Monk 435
Step 2: Create the Output ETD using Monk 436
Step 3: Create the eX_Send_Status.tsc CRS using Monk 436
Step 4: Configure the Collaboration using Monk 437
Step 1: Configure the e*Way using Java 438
Step 2: Create the Output ETD: SendStatus.xsc using Java 438
Step 3: Create the Send_Status Collaboration Rule using Java 439
Step 4: Configure the Collaboration using Java 440
Run and Test the e*Insight scenario 441
Testing the Standard Business Logic 441
In-Stock Processing 441
Out-of-Stock Processing 443
Demonstrating Business Process Undo Functionality 444
Manual Undo 444
Demonstrating Business Process Restart Functionality 446
Repairing a String Attribute 446

Appendix F

XML Structure for the e*Insight Event 449
XML Structure 449

Appendix G

e*Insight Helper Monk Functions 451
e*Insight Helper Monk Functions 451

Glossary 469

Index 472

Chapter 1

Introduction

This guide provides comprehensive information on implementing business 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 application.

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 you 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. You should have experience of Windows and UNIX operations and administration, and should be thoroughly familiar with Windows-style GUI operations.

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

1.3 Writing Conventions

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

Hypertext Links

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

Command Line

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

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

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

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

Code and Samples

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

```
Configuration for BOB_Promotion
```

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

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

Notes and Cautions

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

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

User Input

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

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

File Names and Paths

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

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

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

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

Parameter, Function, and Command Names

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

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

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

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

1.4 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 Business Integration Suite

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

2.1 SeeBeyond Business Integration Suite

Complex and dynamic partner relationships, and the management of various processes present tremendous challenges to business. Organizations and their trading partners must manage disparate component applications and align proprietary software requirements. Organizations and their trading partners must also agree on data exchange and security standards.

The SeeBeyond Business Integration Suite merges traditional Enterprise Application Integration and Business-to-Business (B2B) interactions into a multi-enterprise business integration product suite.

This suite allows you to:

- Leverage your existing technology and applications.
- Create an application consisting of component applications that are managed by your organization or your trading partners.
- Rapidly execute business 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 through the public domain.

This suite also provides:

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

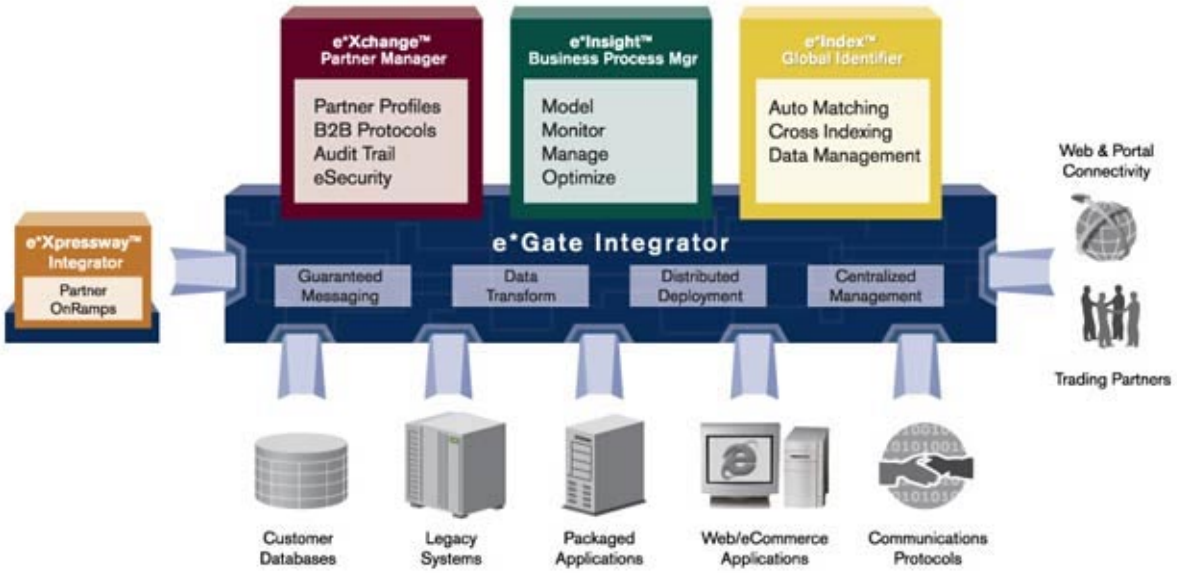
2.1.1 SeeBeyond Business Integration Suite Components

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

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

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

Figure 1 SeeBeyond Business Integration Suite



e*Insight Business Process Manager

The e*Insight Business Process Manager facilitates the automation and administration of business process flow across business activities. Graphical modeling and monitoring enables users to instantly assess the state of a business process instance and identify any bottlenecks in the process.

e*Xchange Partner Manager

The e*Xchange Partner Manager manages trading partner profiles and supports standard business message format and enveloping protocols, including RosettaNet,

UN/EDIFACT, ASC X12, NCPDP-HIPAA, and CIDX. The e*Xchange Partner Manager includes a Validation Rules Builder to aid in the creation of X12 and UN/EDIFACT message validations based on industry implementation guides.

Security Manager

The optional Security Manager works with e*Xchange to ensure the integrity of message data sent to and from trading partners, which is imperative when conducting business in the public domain. The Security Manager uses public key infrastructure (PKI) to authenticate the origin of a message sender. Encryption then ensures that business messages remain secure and private.

e*Xpressway Integrator

e*Xpressway Integrator (e*Xpressway) enables rapid trading partner connectivity and integration through a comprehensive B2B implementation methodology, Web-based graphical configuration wizards, and downloadable integration software. Trading partners follow a quick step-by-step process for registering their company profile, installing customized integration software, and configuring connectivity.

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 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 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. This multi-threaded processing allows for exceptional 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 once-only message delivery.

Business Object Brokers

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

- BOBs only communicate with IQs within e*Gate. They do not communicate with external applications as e*Ways do.
- BOBs 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 Business Integration Suite that facilitates the automation of the business process flow of business activities. The functions of e*Insight include business process model design, monitoring, and execution as well as the ability to analyze historical performance.

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

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

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

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

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

2.3 Building an Integration Application

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

Implementing e*Insight involves the following steps:

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

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

2. Obtain a working knowledge of e*Insight.

   Read chapters 1 through 3 of this Guide to comprehend the technical architecture of e*Insight, its components, and how they work together with e*Gate back-end components. This provides the foundation for implementing a working end-to-end business 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 your job 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 require some modification. The components that you add are completely user-defined. However, the e*Insight GUI and

this guide provide a framework for integrating these user-defined components into a working e*Insight system.

Chapter 3

Implementation Overview

3.1 Basic Information

Implementing an e*Insight system is the process of translating the vision of the business analyst into a functioning system. Once the analyst has determined that a certain business task must be accomplished with e*Insight, it is your job 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 but may require some modification. The components that you add are completely user-defined. However, the e*Insight GUI and this guide provide a framework for integrating these user-defined components into a working e*Insight system.

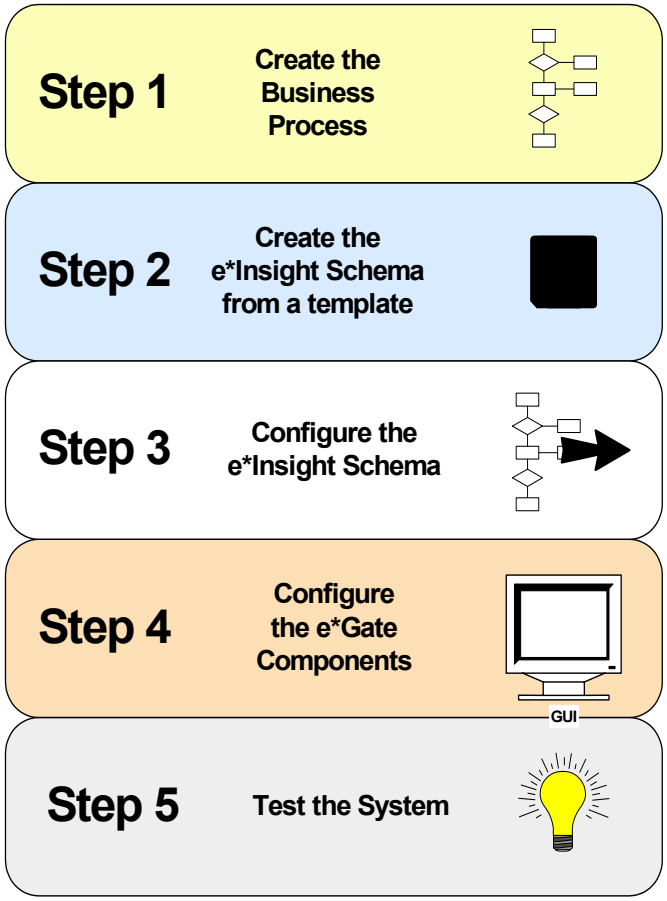
3.2 Implementation Road Map

Each type of implementation involves a different approach, however, there are certain similarities. 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 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: Create the e*Insight Schema from a Template

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

For information on creating a copy of the e*Insight schema, see **"Copy the e*Insight Schema" on page 391**.

Step 3: Configure the e*Insight Schema

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

Step 4: Configure the e*Gate Components

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

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

Step 5: Test and Tune the System

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

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

3.3 The e*Insight Schema

The e*Insight schema is actually an e*Gate schema that implements a particular e*Insight installation. Two default schemas are installed when you install e*Insight as a starting point for your schema creation. You should base your schema off of one of these schemas when implementing e*Insight. These schemas are:

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

The schemas contain a number of pre-configured and partially configured e*Gate components used by e*Insight. In addition to the components that are provided, a complete e*Insight implementation requires several other e*Gate components that are added to the e*Insight schema during the implementation process.

The pre-configured components that are used, as well as the additional e*Gate components that are added, make up the final working e*Insight schema.

3.3.1 The eIJSchema (Java)

This schema is designed specifically to be used in a Java environment and all the components provided are Java based. However, Monk components will still work with the schema. The following sections in this guide provide information for the **eIJSchema**:

- Configuration Information
  - **"e*Insight Schema Components (eIJSchema)" on page 28**
  - **"Common Configuration Tasks" on page 391**
  - **"e*Insight Performance" on page 185**

- **"Troubleshooting" on page 199**
- **"Java Helper Methods" on page 209**

- Sample Implementations
  - **"e*Insight Implementation (eIJSchema)" on page 68**
  - **"e*Insight Authorization Activity Implementation (eIJSchema)" on page 97**
  - **"e*Insight User Activity Implementation" on page 104**
  - **"e*Insight Sub-Process Implementation (eIJSchema)" on page 116**
  - **"e*Insight Remote Sub-Process Implementation" on page 142**
  - **"Active and Passive Modes" on page 159**

3.3.2 The eISchema (Classic)

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

- Configuration Information
  - **"e*Insight Schema Components (eISchema)" on page 364**
  - **"Using eX_Standard_Event.ssc" on page 386**
  - **"Common Configuration Tasks" on page 391**
  - **"e*Insight Performance" on page 185**
  - **"Troubleshooting" on page 199**
  - **"e*Insight Helper Monk Functions" on page 451**
- Sample Implementations
  - **"e*Insight Authorization Activity Implementation (eISchema)" on page 402**
  - **"e*Insight User Activity Implementation" on page 104**
  - **"e*Insight Sub-Process Implementation (eISchema)" on page 128**
  - **"e*Insight Remote Sub-Process Implementation" on page 142**
  - **"Active and Passive Modes" on page 159**

Chapter 4

e*Insight Schema Components (eIJSchema)

The purpose of this chapter is to describe the e*Gate components provided with the eIJSchema as well as those that are added in the implementation process. This chapter also discusses how each component fits into and supports a working e*Insight implementation.

4.1 e*Insight Schema Components Overview

The purpose of the e*Gate schema for e*Insight (eIJSchema) is to provide the working portion of e*Insight within e*Gate. The e*Insight GUI is used to configure and monitor the e*Insight system. The e*Gate schema components actually perform the business processes that you define in the e*Insight GUI.

The e*Insight schema components :

- determine workflow
- persist data
- publish messages to the e*Gate schema, which moves and transforms the data.

Schema Component Types

The module types used by e*Insight, as well as the components provided in the e*Insight installation, are listed in **Table 1**. The components that the user adds in the implementation process are also listed. The column headings are as follows:

- **Component**—The e*Gate logical name for the component. *Italics* indicates that the name varies by association or is user-defined.
- **Description**—A brief description of what the component does in e*Insight.
- **In Default eIJSchema**—Whether or not this component is provided as part of the schema installation of e*Insight.
- **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?	More Information
e*Insight Engine	This is a specially configured Multi-Mode e*Way that runs business processes, using the e*Insight e*Way Connection.	Yes	4.1.3 on page 31
eI_Resubmitter	A placeholder component used in the e*Insight Event failure handling process.	Yes	4.2.7 on page 47
Start BP Component	An e*Way that sends the Event that starts a business process instance.	No	4.2.9 on page 49
eX_*Activity* e*Way	This Multi-Mode e*Way implements an e*Insight activity that connects to an external system.	No	4.2.11 on page 51
eX_*Activity* BOB	Implements an e*Insight activity that does not connect to an external system.	No	ß on page 54

4.1.1 e*Insight Schema Component Relationships

e*Insight implementations do not *always* use all components available. Some components that you may need, are not provided as part of the e*Gate schema and must be added to the base e*Insight schema. Figure 4 illustrates the relationships between e*Insight schema components.

Note: Figure 3 *is the legend for the diagram in Figure 4.*

Figure 3 e*Insight Overview Legend

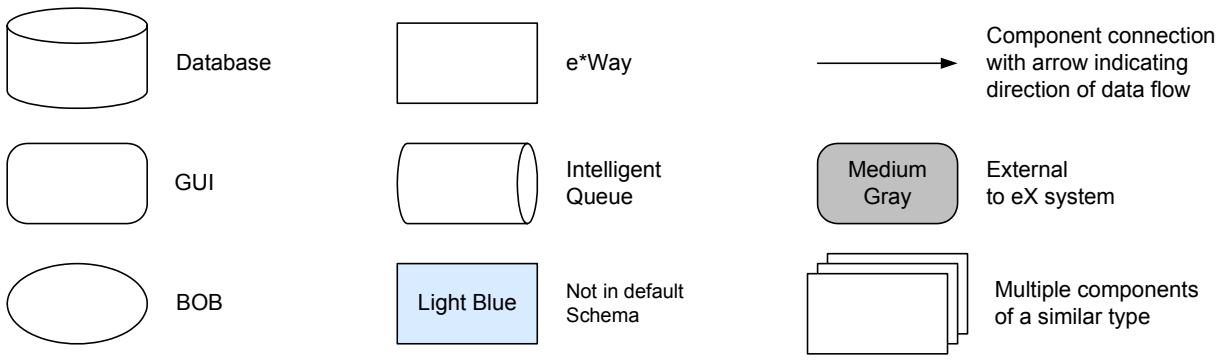
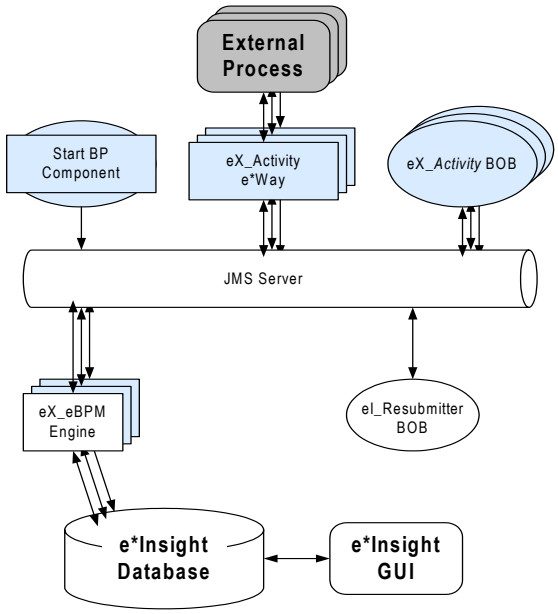


Figure 4 e*Insight Components



4.1.2 e*Insight Business Process Manager Components

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 schema.

Start a Business Processes

The Multi-Mode e*Way component that starts a business process is:

- The ***START_BP*** component (the component name is editable).

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

Run a Business Processes

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

- One or more e*Insight engines

  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.

Note: *For information on using multiple engines, see* **"Using Multiple e*Insight Engines" on page 186**.

- The **eI_Resubmitter** e*Way

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

Implement Business Process Activities

The Multi-Mode e*Way is the component that implements business process activities:

- **eX_*Activity*** e*Way(s)

By default, when an activity component is created, the activity name is prefixed by "eX_". Activity components are added to the e*Insight schema when you create a business process in the e*Insight GUI .

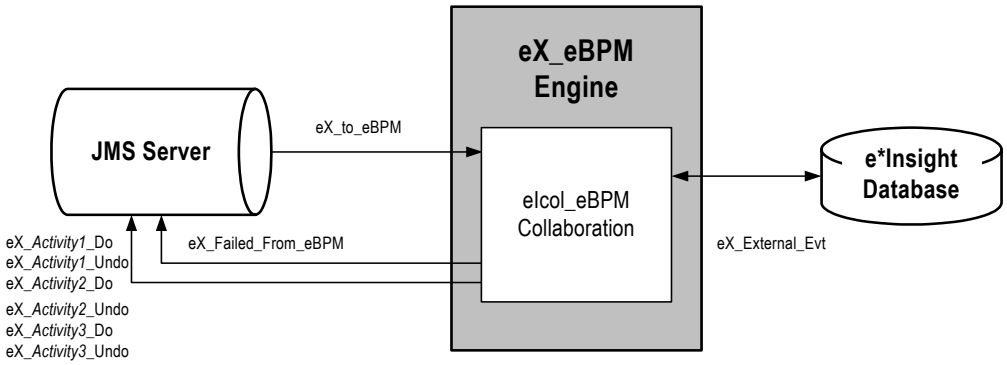
An **eX_Activity** e*Way can connect to an external system if required by the business activity. You must supply the programming to carry out the business logic of the activity and return an activity completed message (the "Done" Event) to the e*Insight engine.

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

4.1.3 e*Insight Engine

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

Figure 5 The e*Insight Engine (Java)



Configuring the e*Insight Engine

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

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

4.2 Using XA

XA or 2-Phase Commit capability is available for use with the XA-compliant JMS e*Way Connection.

How XA Protocol Works

First Phase

Once a transaction is completes but is not yet committed, the neutral third-party Transaction Coordinator polls the Resource Manager for each XA-compliant Connection to verify if a commit is possible.

Second Phase

All Resource Managers must vote "Yes" unanimously before the Transaction Coordinator asks each for a commit; otherwise each is asked to rollback the transaction.

4.2.1 Database Requirements for XA Compliance

Oracle

Oracle XA-compliancy is available with version Oracle 8i and above. To enable an Oracle Database for XA transactions, a Database Administrator must:

- Grant SELECT privileges to PUBLIC for the SYS.DBA_PENDING_TRANSACTIONS. View, by logging in as SYS and issuing:

```
GRANT SELECT ON DBA_PENDING_TRANSACTIONS TO PUBLIC
```

- Grant SELECT and DELETE privileges to PUBLIC for the SYS.PENDING_TRANS$ table by logging in as SYS and issuing:

```
GRANT SELECT, DELETE ON PENDING_TRANS$ TO PUBLIC
```

- Change the Shared Global Area (SGA) parameters to a minimum of:
  - Shared Pool of 42 MB
  - Buffer Cache of 4 MB
  - Large Pool of 1024 KB
  - Java Pool of 20 MB

Sybase

To use XA with Sybase Database server, you must have:

- An XA license installed for the Sybase Database server.

- Merant DataDirect 2.2 JDBC Driver for Sybase.

Note: *For more information, please see the* ***DataDirect Connect JDBC User's Guide and Reference****.*

Microsoft SQL Server

To use XA with Microsoft SQL Server, you must:

- Install Merant DataDirect 2.2 JDBC Driver for SQL Server.
- Copy the sqljdbc.dll from the SQLServer\JTA directory to the SQLServer\bin directory.
- From the command shell, execute the instjdbc.sql script that's provided by DataDirect.

Note: *For more information, please see the DataDirect Connect JDBC User's Guide and Reference.*

4.2.2 e*Way Connection Configuration Changes

XA-compliant Database connections require the use of the JDBC DataSource mechanism instead of the JDBC URL String methodology. Non-XA connections can be created using JDBC DataSources as well.

You use the DataSource methodology by setting the JDBC URL String value to <NONE> in the eBPM Settings Section and choosing the appropriate Database Type. The e*Insight Engine e*Way Connection uses the specified parameters from the appropriate DataSource Section.

ORACLE DataSource Section

Parameter: XA Compliant

Choosing "YES" enables XA 2-Phase Commits Compliancy for the e*Insight Engine.

Note: *The JDBC URL String parameter (under the eBPM Settings Section) is not in use, if XA is enabled.*

Parameter: Driver Type

Choosing "thin" indicates that Oracle JDBC Pure Java Drivers are to be used, whereas "oci8" implies the use of Type 2 OCI Transports.

Parameter: ServerName

This specifies the hostname or IP address of the computer where the Database Server TNS Listener is running.

Parameter: PortNumber

This specifies the TCP/IP port number that the TNS Listener is using. The default Default is 1521.

Parameter: DatabaseName

This specifies the name of the database instance and is either the Service Name (version Oracle 8i and above) or SID (version Oracle 8.0 or below).

Parameter: Row Prefetch Size

This pertains to Oracle Performance Tuning . The value specified controls the number of rows in a Result Set that is prefetched by the JDBC Driver, each time the e*Insight Engine queries the backend database. If not specified, a default of 1000 is used.

Parameter: Execute Batch Size

This pertains to Oracle Performance Tuning . The value specified controls the number of sequential database commands issued by the e*Insight Engine, such as INSERT, UPDATE and DELETE, that are executed in a batch. If not specified, a default of 1 is used.

Parameter: Attribute Value Pair Separator

DataSources consist of Attributes (such as ServerName and PortNumber). In the event that new Attributes become available you can specify additional Attributes. This entry specifies the character separator used to separate the Attribute from the Value in an Attribute-Value Pair. For example, the Attribute-Value pair "ServerName^myHost" has "^" as a separator. One should select a separator that will NOT be part of the Attribute-name or the Attribute-value. The default value is "^".

Parameter: Attribute Value Pairs

Multiple Attribute-Value pairs can be used to specify supplemental connection properties. The Attribute-name should be exactly the same as the one that is specified in the driver documentation and the value should be a valid one. For example:

```
PortNumber^8888
```

To disable an Attribute, de-select it. The separator used in this parameter should match the one specified in the "Attribute Value Pair Separator". By default, the separator used is "^".

Parameter: Standard Class

This entry specifies the name of the Java class in the JDBC driver that implements the `ConnectionPoolDataSource` interface. It is not editable.

Parameter: XA Class

This entry specifies the name of the Java class in the JDBC driver that implements the `XADataSource` interface. It is not editable.

Sybase DataSource Section

All Sybase Parameters are the same as the Oracle Parameters, unless listed below:

Parameter: ServerName

This is the database server hostname or IP address.

Parameter: PortNumber

This is the I/O port number of the database server. Default is 4100.

Parameter: DatabaseName

This is the name of the database instance.

Parameter: Attribute Value Pairs

Same function as in Oracle, however, the "SelectMethod^cursor" Attribute Value pair is predefined, since XA mode for the Merant DataDirect JDBC Driver requires it.

Microsoft SQL Server DataSource Section

Only parameters that differ from the Oracle DataSource and Sybase DataSource Sections will be mentioned here.

Parameter: PortNumber

This is the I/O port number of the database server. Default is 1433.

eBPM Settings Section

New XA related parameters are described below:

Parameter: JDBC URL String

The user must specify "<NONE>", in order to use this DataSource methodology for Database connection.

4.2.3 Performance Improvements

With each committed transaction that a Database server processes, there is a Redo log entry written. This is so the Database can be incrementally recreated from the Redo logs. Additionally, as Redo log entries accumulate, these need to be Archived. During this process, all database activities on the affected tables are suspended.

Each processed transaction that is not committed, is written to the Rollback segment. As these accumulate, they are stored temporarily to disk, causing more disk overhead. This must be taken into consideration to maintain the balance between processing many transactions together and the frequency of the database commits.

In addition to providing XA-compliance, the e*Insight Engine e*Way Connection can process groups of e*Insight command messages before issuing a singular commit, in order to enhance performance. This is called eBPM Tuning and is located in the e*Insight Engine e*Way Connection configuration that controls this "Commit Frequency" or how often Database commits are performed. The Engine can also be configured to adjust the Commit Frequency to maximize transaction throughput.

eBPM Tuning

Parameter: Commit Frequency

The number of transactions (START_BP's, DO_ACTIVITY statuses) to process before committing the e*Insight Engine Database as well as the JMS Queue. It serves as a way to optimize database performance by batching the number of transactions before calling commit.

Specifying zero (0) means that commits are done after each transaction is processed by the Engine.

Parameter: Maximum Commit Frequency

Dynamic Commit Frequency Optimization

By specifying a number greater than zero (0), the Commit Frequency will be adjusted to increase performance. The Commit Frequency will never be adjusted greater than the stated Maximum Commit Frequency nor below the initial Commit Frequency.

As each incoming transaction (START_BPs, ACTIVITY Do and Undo Statuses) to the Engine is processed, a running command processing rate is computed. At each Commit Frequency interval when the transactions are committed, the current command processing rate is compared with the rate at the previous commit.

If the current rate is higher, the Commit Frequency is advanced cumulative-incrementally, that is, for each consecutive positive change in the processing rate, the increment is one greater than the previous, starting with one. The Commit Frequency is increased incrementally to the specified maximum then it is reset to the running average Commit Frequency. This is continually calculated at each Commit Frequency interval.

If the current rate is less than the previous rate, the Commit Frequency is decremented by one (always), but never below a minimum given by the initial Commit Frequency. There, the Commit Frequency is increased incrementally by one regardless of a negative change in rate.

If the rates are the same, the Commit Frequency is increased incrementally by one, to "prod" the performance over a hump, subject to whether the Stability Threshold has been exceeded; after which it is no longer adjusted.

Static Commit Frequency

Specifying zero (0) means that the Commit Frequency is a constant.

Parameter: Receive Wait Timeout

This is the maximum time, in milliseconds, the e*Insight Engine waits for a new transaction to come from the JMS Queue when the Commit Frequency specified number of messages have not yet been all received. Specifying zero (0) means that the standard Event Type "get" interval set in the JMS e*Way Connection is used.

Parameter: Stability Threshold

By specifying a number greater than zero (0) for this and if adaptive adjustment of Commit Frequency is enabled, the adjustments will be temporarily suspended once there is no more change in performance after the specified threshold of adjustments are made. Specifying zero (0) means that adjustments will always be made even though there are no performance improvements.

Parameter: Show Change Audit

Set this value to YES only if on shutdown of the e*Insight Engine, an audit trail of all the changes to the Commit Frequency is to be shown in the e*Way log file.

4.2.4 Connection Management

The e*Insight Engine e*Way Connection is now also Connection Management enabled, and can be controlled by the Connection Manager. Some of the benefits this brings include:

- Automatic Alerts sent to the Schema Manager should the Database connections go down.
- Continual verification of Database connection status on a periodic basis.

Connector Section Changes

Parameter: Connection Establishment Mode

This parameter specifies how the connection with the database server is established and closed:

- Automatic indicates that the connection is automatically established when the collaboration is started, and it keeps the connection available.
- OnDemand indicates that the connection is established on demand, as business rules requiring a connection to the external system are performed. The connection is closed after the methods complete.
- Autonomous indicates that connection with database is maintained without the aid of the Connection Manager.
- Automatic is the default setting.

Parameter: Connection Inactivity Timeout

This value is used to specify timeout (in milliseconds) for the Automatic connection establishment mode. If this is not set or if it is set to 0, the connection will not be brought down due to inactivity. The connection is always kept alive.

If the connection is lost, it will attempt to re-establishing the connection automatically. If a non-zero value is specified, the connection manager will try to monitor for inactivity.

Parameter: Connection Verification Interval

This value is used to specify the minimum period of time (milliseconds) between checks for connection status to the database server. If the connection to the server is detected to be down during verification, an Alert is sent to the Schema Manager indicating the Down status.

If the connection comes from a previous connection error, a compensating Alert is sent to the Schema Manager to indicate the connection is Up again. If no value is specified, 60000 ms is used.

4.2.5 Manual Database XA Recovery

When a database that is running in XA mode suddenly halts due to a power failure or disk crash, some transactions are left pending. When all components are available again, the e*Insight Engine e*Way Connection may have problems establishing an XA session with the database and may require manual intervention by the Database Administrator.

Oracle Database

1. Login as the System Administrator.
2. Obtain the local Transaction ID of the pending transactions by executing:

```
SELECT local_trans_id FROM sys.dba_2pc_pending;
```

3. Forcibly commit or rollback each local Transaction ID by executing:

```
COMMIT FORCE or ROLLBACK FORCE <local Transaction ID>;
```

4. Delete any rows left in the sys.pending_sessions$ and sys.pending_trans$ tables by executing:

```
TRUNCATE TABLE sys.pending_sessions$ or sys.pending_trans$;
```

4.2.6 Engine Connection Configuration

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

Table 2 e*Insight Engine Connection Configuration Settings

Screen	Parameter	Setting
ORACLE DataSource	XA Compliant	Determines whether the e*Insight engine is XA 2-Phase Commits Compliant. If the value is set to YES, then XA Compliancy is enabled. However, setting this value to YES may degrade performance. *Note: If XA Compliancy is enabled, then the JDBC URL String parameter in the eBPM Settings section is ignored.*
	DriverType	Specifies the driver type. Select one of the following: ▪ oci8 — uses Type 2 OCI Transports ▪ thin (default) — uses Oracle JDBC Pure Java Drivers.
	ServerName	Specifies the hostname or IP address of the computer where the Database Server TNS Listener is running.
	PortNumber	Specifies the TCP/IP port number that the TNS Listener uses. Default value is 1521.
	DatabaseName	Specifies the Service Name of the database instance for Oracle 8i or above
	SID	Specifies the SID of the database instance for Oracle 8.0 or below.

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

Screen	Parameter	Setting
	Row Prefetch Size	Controls the number of rows in a Result Set that are prefetched by the JDBC Driver each time the e*Insight engine queries the database. The default value is 1000. *Changing this value may improve performance. You should follow the guidance of a Database Administrator when changing this value.*
	Execute Batch Size	Controls the number of sequential database commands issued by the e*Insight engine that are executed in a batch, such as INSERT, UPDATE, and DELETE. *Changing this value may improve performance. You should follow the guidance of a Database Administrator when changing this value.*
	Attribute-Value Pair Separator	Specifies the character separator used to separate the Attribute from the Value in an Attribute-Value pair.
	Attribute-Value Pairs	Specifies supplemental connection properties as a list of Attribute-Value pairs. The Attribute name should be as specified in the driver documentation. The separator used in this parameter should match the one specified in the **Attribute-Value Pair Separator**. The default separator is ^. If the Oracle native JDBC OCI Driver Type is used, the Attribute-Value pairs that should be defined include: ▪ NetworkProtocol — Default value is **tcp**. Can be set to all possible protocols that Net8 supports. ▪ TNSEntryName — TNS entry name. This assumes that the Oracle Client is installed and TNS_ADMIN environmental variable is set properly.
	Standard Class	Specifies the name of the Java class in the JDBC driver that implements the **ConnectionPoolDataSource** interface.
	XA Class	Specifies the name of the Java class in the JDBC driver that implements the **XADataSource** interface.
Sybase DataSource	XA Compliant	Determines whether the e*Insight engine is XA 2-Phase Commits Compliant. If the value is set to YES, then XA Compliancy is enabled. However, setting this value to YES may degrade performance. *Note: If XA Compliancy is enabled, then the JDBC URL String parameter in the eBPM Settings section is ignored.*
	ServerName	Specifies the hostname or IP address of the database server.
	PortNumber	Specifies the I/O port number of the database server. Default value is 4100.

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

Screen	Parameter	Setting
	DatabaseName	Specifies the name of the database instance.
	Attribute-Value Pair Separator	Specifies the character separator used to separate the Attribute from the Value in an Attribute-Value pair.
	Attribute-Value Pairs	Specifies supplemental connection properties as a list of Attribute-Value pairs. The Attribute name is specified in the driver documentation. The separator used in this parameter matches the one specified in the **Attribute-Value Pair Separator**. The default separator is **^**. The "SelectMethod^cursor" Attribute-Value pair is predefined as XA mode because the Merant DataDirect JDBC Driver requires it.
	Standard Class	Specifies the name of the Java class in the JDBC driver that implements the **ConnectionPoolDataSource** interface.
	XA Class	Specifies the name of the Java class in the JDBC driver that implements the **XADataSource** interface.
SQLServer Datasource	XA Compliant	Determines whether the e*Insight engine is XA 2-Phase Commits Compliant. If the value is set to YES, then XA Compliancy is enabled. However, setting this value to YES may degrade performance. *Note: If XA Compliancy is enabled, then the JDBC URL String parameter in the eBPM Settings section is ignored.*
	ServerName	Specifies the hostname or IP address of the database server.
	PortNumber	Specifies the I/O port number of the database server. Default value is 1433.
	DatabaseName	Specifies the name of the database instance.
	Attribute-Value Pair Separator	Specifies the character separator used to separate the Attribute from the Value in an Attribute-Value pair.
	Attribute-Value Pairs	Specifies supplemental connection properties as a list of Attribute-Value pairs. The Attribute name is specified in the driver documentation. The separator used in this parameter should match the one specified in the **Attribute-Value Pair Separator**. The default separator is **^**. The "SelectMethod^cursor" Attribute-Value pair is predefined as XA mode because the Merant DataDirect JDBC Driver requires it.
	Standard Class	Specifies the name of the Java class in the JDBC driver that implements the **ConnectionPoolDataSource** interface.

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

Screen	Parameter	Setting
	XA Class	Specifies the name of the Java class in the JDBC driver that implements the **XADataSource** interface.
	ServerName	Specifies the hostname or IP address of the database server.
	PortNumber	Specifies the I/O port number of the database server.
	DatabaseName	Specifies the name of the database instance.
	CollectionId	Specifies the collection (group of packages) to which the package is bound.
	PackageName	Specifies the name (7-character limit) of the package that the driver uses to process static and dynamic SQL.
	Attribute-Value Pair Separator	Specifies the character separator used to separate the Attribute from the Value in an Attribute-Value pair.
	Standard Class	Specifies the name of the Java class in the JDBC driver that implements the **ConnectionPoolDataSource** interface.
	XA Class	Specifies the name of the Java class in the JDBC driver that implements the **XADataSource** interface.
eBPM Settings	Database Type	Specifies the type of e*Insight database. Select one of the following: ▪ **Oracle** when using Oracle 8.1.7 or 9i. ▪ **SQL Server** when using Microsoft SQL Server 2000. ▪ **Sybase** when using Sybase 12.5
	JDBC URL String	This parameter is required if you do not use the DataSource mechanism to connect to the e*Insight database. If you have defined the DataSource section, then this parameter should be set to <NONE>. This is the connection string used by the e*Insight engine to communicate with the e*Insight database. Use the connection string that is appropriate for the database client setup on the machine running the e*Insight engine. (Refer to the relevant driver documentation for more details on configuring your system).

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

Screen	Parameter	Setting
eBPM Settings	JDBC URL String	An Oracle database connection will use the string: **jdbc:oracle:*thin*:@*machine_name*:*port*:*db*** where ▪ ***thin*** is the type of Oracle client interface. ▪ ***machine_name*** is the network name of the computer running the e*Insight database. If the database is on the same machine as the e*Insight engine you can use "localhost" instead of the machine's network name. ▪ ***port*** is the port for communicating with the e*Insight database (1521 is the default port). ▪ ***db*** is the service name used to communicate with e*Insight Oracle database from the local machine. If you are using XML data and the Model Specific database option, then you should use the **OCI** driver. You will use this string to connect: **jdbc:oracle:*oci*:@*db*** where ▪ ***oci*** is the type of oracle client interface. ▪ ***db*** is the service name used to communicate with e*Insight Oracle database from the local machine. A SQL Server database connection uses the string: **jdbc:SeeBeyond:sqlserver://<server>:<port#>;DatabaseName=<dbname>;embedded=true;SelectMethod=cursor** ▪ ***server*** is the network name of the computer running the e*Insight database. If the database is on the same machine as the e*Insight engine you can use "localhost" instead of the machine's network name. ▪ ***dbname*** is the name of the e*Insight SQL Server database. ▪ ***port*** is the port for communicating with the e*Insight database. A Sybase database connection uses the string: **jdbc:sybase:Tds:<server>:<port>** ▪ ***server*** is the network name of the computer running the e*Insight database. If the database is on the same machine as the e*Insight engine you can use "localhost" instead of the machine's network name. ▪ ***port*** is the port for communicating with the e*Insight database.

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

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

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

Screen	Parameter	Setting
eBPM Settings	Auto Model Reload	Determines if the engine dynamically loads an enabled Business Process Version if the enabled/ disabled status of Business Process Version changes. If the value is set to YES then Business Process Versions that are enabled or disabled while the engine is running are immediately recognized. However, setting this value to YES may degrade performance.
	Instance Caching	Instance caching is the most efficient way to process Business Process Instances. Setting this value to YES keeps a cache of the instance information throughout the life span of the Business Process Instance. Setting this value to NO retrieves the information from the database instead. This allows more flexibility and fault tolerance at the cost of performance. To improve performance it is recommended to set this parameter to YES and use multiple e*Insight engines. To use both instance caching and multiple engines it is necessary to ensure that a single instance is always processed by the same engine. This is achieved by using engine affinity. For information using instance caching with multiple engines see **"e*Insight Engine Affinity (eIJSchema)" on page 189**.
	Brand By Collaboration Name	This parameter allows you to name Event Types based on the Collaboration rather than the e*Way name. This is used for engine affinity when multiple e*Insight engine Collaborations are used in a single e*Way. For more information, see **"e*Insight Engine Affinity (eIJSchema)" on page 189**.
	Business Processes to Preload	This parameter allows you to load all or a subset of all the business processes stored in the e*Insight database. The default is ALL.
	Using e*Xchange with e*Insight	If an Event Type Definition is used that contains both e*Insight and e*Xchange sections, setting this to NO causes the e*Xchange section to be ignored. This reduces the time taken by the e*Insight engine at runtime to parse the Event.

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

Screen	Parameter	Setting
eBPM Tuning	Commit Frequency	Specifies the number of transactions (START_BP and DO_ACTIVITY statuses) to process before committing the changes and updates to the e*Insight database as well as the JMS Queue. This can optimize database performance by batching the number of transactions before committing. Specifying zero (0) means that commits are performed after each transaction processed by the engine.
	Maximum Commit Frequency	By specifying a number greater than zero (0) for this parameter, the Commit Frequency is adjusted upwards or downwards in order to increase performance. The value remains between the initial and maximum value. Specifying zero (0) means that Commit Frequency is not changed.
	Receive Wait Timeout	Specifies the maximum time, in milliseconds, that the e*Insight engine waits for a new transaction to come from the JMS Queue when the Commit Frequency specified number of messages have not yet been received. Specifying zero (0) means that the standard Event Type “get” interval set in the JMS e*Way Connection is used.
	Stability Threshold	By specifying a number greater than zero (0) for this and if adaptive adjustment of Commit Frequency is enabled, the adjustments are temporarily suspended once there are no more changes in performance after the specified threshold of adjustments are made. Specifying zero (0) means that adjustments are always made, even though there are no performance improvements.
	Show Change Audit	Setting this value to YES creates an audit trail of all changes to the Commit Frequency in the e*Insight engine log file.
	Echo Passive START_BP	The default setting for this parameter is “NO”. Set this parameter to "YES" if backwards- compatibility is necessary. Otherwise, do not change it.

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

Screen	Parameter	Setting
Connections	Connection Establishment Mode	Specifies how the connection to the e*Insight database is established and closed. The options include: ▪ **Automatic** — the connection is automatically established when the Collaboration is started and the connection is kept alive as necessary. ▪ **OnDemand** — the connection is established on demand, as business rules requiring a connection to the external system are performed. The connection is closed after the methods are completed. ▪ **Autonomous** — the connection is maintained without the aid of the Connection Manager.
	Connection Inactivity Timeout	Specifies the timeout (in milliseconds) for the Automatic connection establishment mode. If this is not set or set to 0, the connection is not closed due to inactivity, the connection is always kept alive and if it goes down, re-establishing the connection is automatically attempted. If a non-zero value is specified, the connection manager tries to monitor for inactivity so the connection is closed if the value specified is reached.
	Connection Verification Interval	Specifies the minimum period of time (in milliseconds) between checking the connection status to the database. If the connection is down, an alert is sent to e*Gate Monitor. If the connection is re-established after a connection error, then an alert is sent to Schema Manager indicating the connection has been restored. If now value is specified, 6000 ms is used.

eIcr_eBPM Collaboration

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

Subscribed Event Type

- **eX_External_Evt**—This Event carries the data retrieved from the e*Insight database to the e*Insight engine.
- **eX_to_eBPM**—This Event Type carries all Events intended for e*Insight. These include Start BP Events, "Done" Events, and other Events that must be processed by the e*Insight engine. All data sent to the e*Insight engine must use the Activity Specific ETD generated by e*Insight or **eI_StandardEvent.xsc**.

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

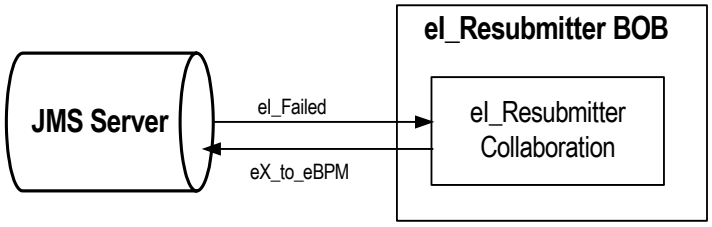
Published Event Types

- **eX_DynamicET**—This Event is used to enable the Collaboration to automatically generate one of the following Events:
  - **eX_*Activity*_Do**—This Event causes the subscribing Collaboration to execute the "Do" logic of the activity with the same name in the business process. See **"Subscribed Event Type: eX_Activity_Do" on page 53** 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 53** for more information.
- **eX_Failed_From_eBPM**—This Event contains the failed Event along with error information.
- **eX_External_Evt**—This Event carries the data that is written to the e*Insight database.

4.2.7 eI_Resubmitter BOB

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

Figure 6 eI_Resubmitter Detail (Java)



Configuring the eI_Resubmitter BOB

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

eI_Resubmitter Collaboration

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

Subscribed Event Type: eX_Failed_From_eBPM

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

Published Event Type: eX_to_eBPM

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

4.2.8 Failed Event Handling by the e*Insight Engine

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

Error Types

Connection errors

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

Data errors

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

Error Handling

Connection Failure Handling

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

Data Failure Handling

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

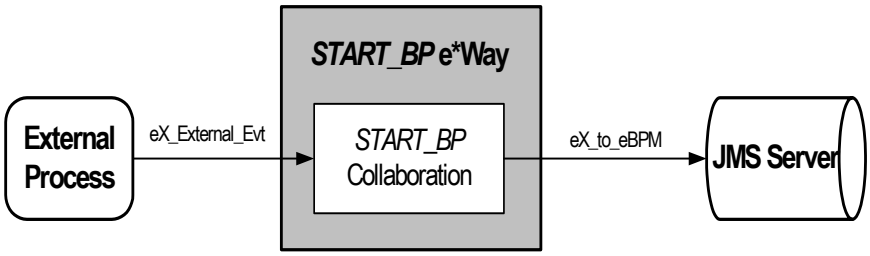
The e*Insight engine publishes the failed Event to the JMS Server under the **eX_FailedEvent** Event Type. The **eI_Resubmitter** BOB subscribes to this Event Type and you can use it to repair the Event and republish it to the e*Insight Engine. The **eX_FailedEvent.xsc** ETD associated with the **eX_FailedEvent**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

Note: *Please see "Exception Handling" in the* ***e*Insight Business Process Manager User's Guide****, for information about configuring e*Insight to handle errors.*

4.2.9 START_BP Component

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

Figure 7 START_BP e*Way Detail



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

If the data is held internally, in e*Gate, then a JMS Connection is used. Alternatively, it is possible to retrieve data from a JMS IQ. The Collaboration must change the data it receives into the eI_StandardEvent or eIX_StandardEvent format.

Configuring the START_BP Component

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

START_BP Collaboration

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

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

See **"Starting a Business Process (eIJSchema)" on page 393** for more information on how to start a BPI.

Subscribed Event Type: eX_External_Evt

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

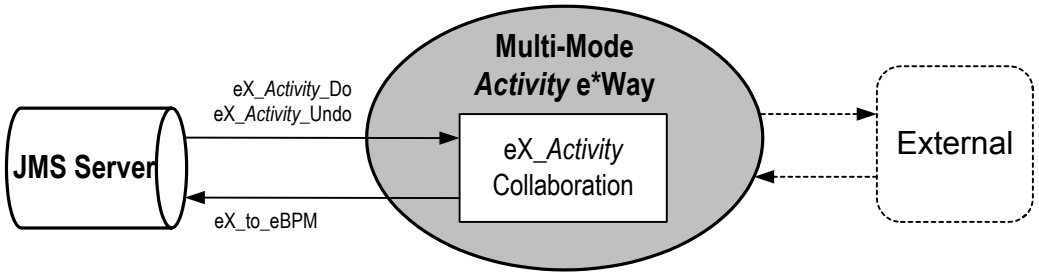
Published Event Type: eX_to_eBPM

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

4.2.10 Multi-Mode Activity e*Way

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

Figure 8 Multi-Mode Activity e*Way Detail



eX_*Activity* Collaboration

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

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

Subscribed Event Types:

- **eX_*Activity*_Do**—This Event causes the subscribing Collaboration to execute the "Do" logic of the activity with the same name in the business process. It is the same Event Type as that subscribed to by the **eX_to_*Activity* Collaboration**. See **"Subscribed Event Type: eX_Activity_Do" on page 53** 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 53** for more information.

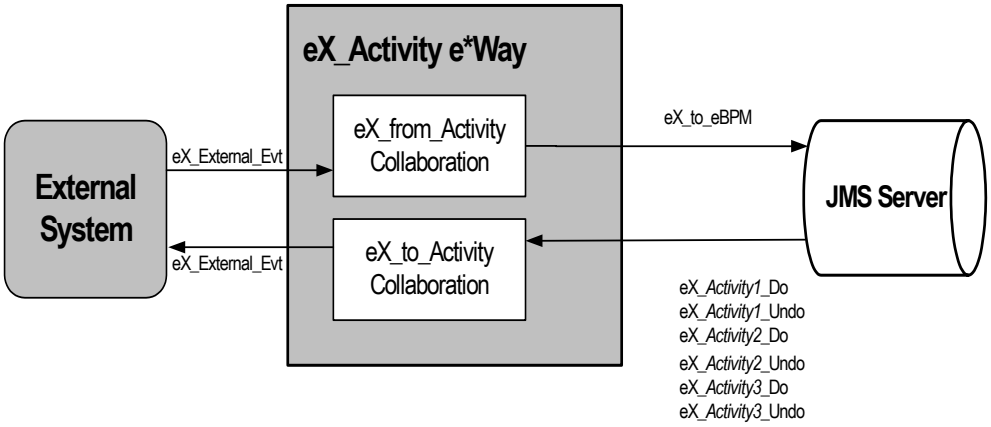
Published Event Type: eX_to_eBPM

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 54** for more information.

4.2.11 Single-Mode Activity e*Way

The single-mode e*Way is another e*Way option. However, the Multi-Mode e*Way is the default and the most flexible for most configurations.

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 or a single Collaboration. The choice to use a single Collaboration or multiple Collaboration depends on your preference.

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

The Collaboration Rule created in this process is only a placeholder. Implementors must configure the Collaboration Rules by writing the Collaboration Rules Script and

choosing the service under which this script runs. The CRS contains the programming that implements the business logic for the corresponding activity in the business process.

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

Configuring the eX_to_*Activity* e*Way

Some of the configuration for the activity e*Way is done for you when you use the e*Insight GUI to configure the schema. This includes setting up the component relationships and Event Type routing in the e*Insight schema, but not the actual business logic programming or the type of e*Way that is used.

You must write the business logic code in the Collaboration Rules Scripts used by the activity Collaborations. 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.

eX_to_*Activity* Collaboration

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

Important: *In addition to passing the attribute data it receives from the e*Insight engine to the external system, the **eX_to_Activity** Collaboration must preserve the e*Insight Business Process Instance tracking information contained in the **eX_Activity_Do** or **eX_Activity_Undo** Events. This information is used to send the return or "Done" Event back to the e*Insight engine, when the activity completes. See* **Sending the "Done" Event Back to e*Insight (eIJSchema)** *on page 395* *for more information on what information is required in the "Done" Event.*

Do and Undo logic in an Activity Collaboration

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

By default, the eX_to_Activity and eX_Activity Collaboration in an activity e*Way subscribe to both the "Do" and "Undo" Events. Consequently the CRS must contain logic to handle both the activity and the compensating transaction for the activity. You

may 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 59**.

Subscribed Event Type: eX_*Activity*_Do

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

Subscribed Event Type: eX_*Activity*_Undo

This Event causes the subscribing Collaboration to execute the "Undo" logic of the corresponding activity in the business process. That is, it causes a compensating transaction to occur that "undoes" the completed activity within a BPI (see**"Do and Undo logic in an Activity Collaboration" on page 52** for an explanation of "undoing" an activity). This Event Type is in standard e*Insight format, and contains the current values of any global variables designated as "Input" by the activity in the appropriate location in the **eI_StandardEvent.xsc** ETD. Also, the **eX_*Activity*_Undo** Event contains any local variables set by the Collaboration executing the "Do" logic associated with this activity.

Published Event Type: eX_External_Evt

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

eX_from_*Activity* Collaboration

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

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

See **Sending the "Done" Event Back to e*Insight (eIJSchema)** on page 395 for more information.

Subscribed Event Type: eX_External_Evt

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

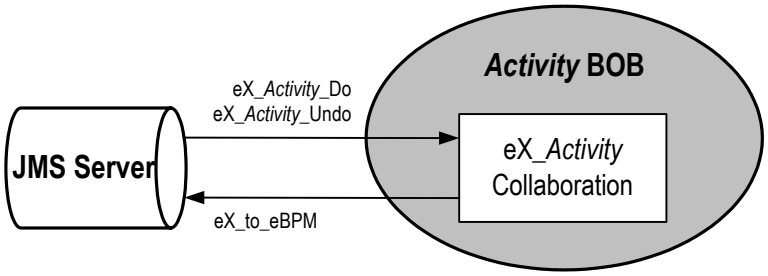
Published Event Type: eX_to_eBPM

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

4.2.12 Activity BOB

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

Figure 10 Activity BOB Detail



eX_*Activity* Collaboration

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

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

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

Subscribed Event Types:

- **eX_*Activity*_Do**—This Event causes the subscribing Collaboration to execute the "Do" logic of the activity with the same name in the business process. It is the same Event Type as that subscribed to by the **eX_to_*Activity* Collaboration**. See **"Subscribed Event Type: eX_Activity_Do" on page 53** 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 53** 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 54** for more information.

4.3 Using Monk in eIJSchema

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

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

Updating an eISchema to use the eIJSchema engine

1. Backup your schema.
2. Open eIJSchema and export the **eX_eBPM** module.
3. Open the schema that you are updating.
4. Rename **eX_eBPM** to **eX_eBPM_old**.
5. Import the eX_eBPM module that you exported in step 2.
6. Configure e*Insight engine connection.
7. Open the **eIcr_eBPM** Collaboration Rule properties.
   A. Delete initialization string.

   You must perform this step because Monk does not support binary marshalling of XML.
8. Open the **eIcol_eBPM** Collaboration properties.

A Change the subscription source to <ANY>.

B Delete the publication of Event Type eX_DynamicET.

Add a publication for every activity in your e*Gate schema as described in **Table 3**.

Table 3 Publication for eIJSchema update

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

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

10 Delete eX_eBPM_old.

You are now able to use your eIJSchema with Monk components.

Chapter 5

Understanding the e*Insight ETD

e*Insight uses an Event Type Definition (ETD) to define Events as they move from one component to another in the e*Insight system. The ETD contains information used to parse the data moving through e*Insight components . Understanding this ETD is the key to creating the Collaboration Rules scripts necessary to process the data defined by the business process.

5.1 Selecting an ETD

In most cases, we recommend that you use the Activity Specific ETD because it simplifies development and provides less room for errors.

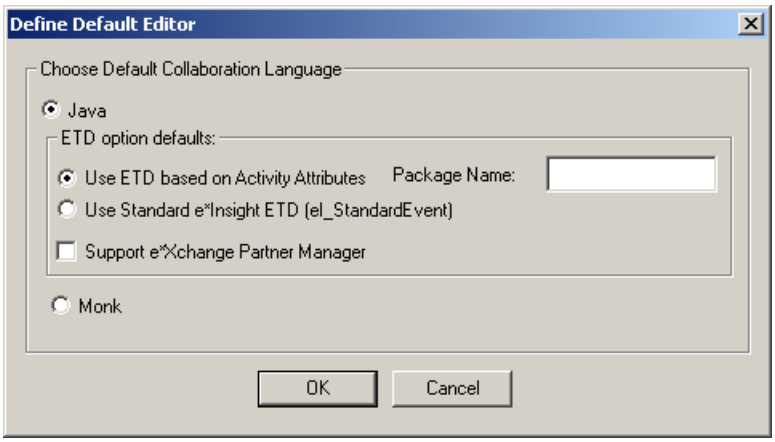
The Activity Specific ETD contains nodes for every attribute that you assign the Activity; these nodes are automatically generated and contain information including the name, type, and direction of the attribute.

The Activity Specific ETD is based on the **eI_StandardEvent** unless you need to support e*Xchange. In that case, your Activity Specific ETD is based on the **eIX_StandardEvent**. The eIX_StandardEvent has the Trading Partner Event (TP_EVENT) location that is used for e*Xchange Partner Manager.

Note: *For more information about e*Xchange Partner Manager, refer to the* ***e*Xchange Partner Manager Implementation Guide****.*

If you choose to manually configure the **eI_StandardEvent.xsc** then you need to know the name, type, and direction of the attribute in order to write your code. You determine whether to use Activity Specific ETDs or **eI_StandardEvent.xsc** as a default, in the **Define Default Editor** dialog box shown in Figure 11.

Figure 11 Define Default Editor



To configure ETD option defaults

1. From the **Options** menu, click **Define Default Editor**.
2. Choose **Java**.
3. Select **Use ETD based on Activity Attributes**.

   The package name is created for you.
4. (Optional) Choose the **Support e*Xchange Partner Manager** option if you plan to use e*Xchange Partner Manager in this implementation.

5.1.1 About Business Process Attributes

The e*Insight engine uses the e*Insight ETD to carry out the business process. When you use an Activity Specific ETD, business process attributes are automatically added to the ETD as nodes. When using the **eI_StandardEvent**, the **BP_EVENT** location contains attribute data and data that the e*Insight engine uses to track the business process instance.

Business process attributes are defined in the e*Insight GUI as global or local. Global attributes are used and shared by any of the activities in the business process. Local attributes are only 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. 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. This takes place in the "Done" Event that is sent to the e*Insight engine when an activity completes.

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 is set by the activity Collaboration in the "Done" Event that is sent when the "Undo" portion of the activity completes.

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.

5.2 Activity Specific ETDs

There are five sections in an Activity Specific ETD:

- **GlobalAttributes** — This section contains a node for every global attribute that has been assigned to this activity.
- **LocalAttribute** — This section contains a node for every local attribute defined in this activity.
- **EventSpecifics** — This section contains nodes for general information pertaining to the instance, for example, business process name, event type, and status.

Important: *It is recommended that you copy the* ***EventSpecifics*** *node from the Source Event to the Destination Event, prior to editing your CRS in any other way.*

- **Tools** — This sections contains methods that are generally available within an ETD.
- **eI_StandardEvent** — This section contains all information available to use if necessary. For more information about using eI_StandardEvent, see **"eI_StandardEvent" on page 62**.

5.2.1 GlobalAttributes

Every global attribute that is assigned to the activity appears as a node under the GlobalAttributes node. Figure 12 shows an example where three global attributes are assigned to an activity, GlobalBoolean, GlobalString, and GlobalNumber.

Figure 12 Activity Specific ETD — GlobalAttributes

5.2.2 LocalAttributes

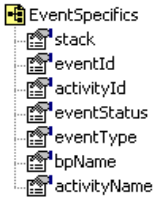
Every local attribute that is assigned to the activity appears as a node under the LocalAttributes node. Figure 13 shows an example where three global attributes are assigned to an activity, LocalB, LocalS, and LocalN.

Figure 13 Activity Specific ETD — LocalAttributes

5.2.3 EventSpecifics

This section contains general information about the activity and business process in the seven nodes as shown in **Figure 14**. It is recommended that you copy **EventSpecifics** from the Source Event to the Destination Event, prior to editing your CRS in any other way.

Figure 14 Activity Specific ETD — EventSpecifics



stack

This node is used to contain information used by the engine.

eventID

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, which occurs when you copy the EventSpecifics node from the Source Event to the Destination Event.

activityId

This node contains a number assigned by the e*Insight engine for the current activity within a BPI. The e*Insight engine uses this to adjust processing speed. It is not recommended that you manually set this number.

eventStatus

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 contain logic to set the value of this node in the "Done" Event sent to the e*Insight engine.

eventType

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

Value	Purpose
"START_BP"(eI_StandardEvent only.)	Indicates to the e*Insight engine that this Event starts a BPI if you are using the eI_StandardEvent only. (The Activity Specific ETD does not use a START_BP.)
"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.

bpName

This node contains the name of the business process, as it appears in the e*Insight GUI.

activityName

This node contains the name of the current activity, as it appears in the e*Insight GUI.

Note: *An Event Type Definition can be used to define the structure of XML data. This Event Type Definition must be created from a .xsd (XML Schema Definition) or .dtd (Document Type Definition) file. In order to create these Event Type Definitions you must install the XML ETD Builders with e*Gate.*

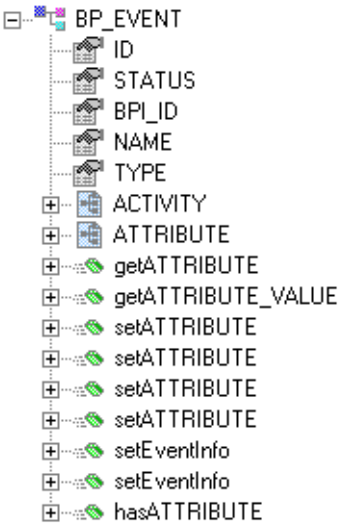
5.3 eI_StandardEvent

If you use eI_StandardEvent.xsc instead of an Activity Specific ETD, then you need to know the name, type, and direction of the attribute in order to write your code. You must program the logic into your Collaboration. It is recommended that you copy the BP_EVENT from the Source Event to the Destination Event, prior to editing your CRS in any other way.

5.3.1 BP_EVENT

All data relevant to e*Insight processing is contained in the **BP_EVENT** branch of the ETD. The structure is shown in Figure 15.

Figure 15 BP_EVENT



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

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

BP_EVENT Element

This location in the e*Insight ETD contains general information about the current business process in the five nodes as shown in Figure 16.

Figure 16 BP_EVENT Element

ID

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

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

STATUS

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

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

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

BPI_ID

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

NAME

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

TYPE

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

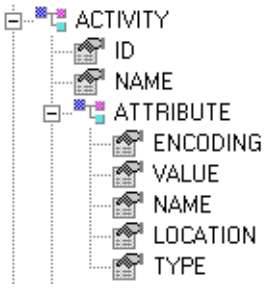
Value	Purpose
"START_BP"	Indicates to the e*Insight engine that this Event starts a BPI.
"DO_ACTIVITY"	Indicates that this is a "Do" Event for the current activity.
"UNDO_ACTIVITY"	Indicates that this is an "Undo" Event for the current activity.

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

BP_EVENT.ACTIVITY Nodes

This location in the e*Insight ETD contains information about the current activity. The **ACTIVITY** node contains information of a general nature about the current activity. The **ACTIVITY.ATTRIBUTE** node contains information about any local attributes that have been defined for the current activity. Figure 17 shows the location of these nodes in the e*Insight ETD.

Figure 17 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 18.

Figure 18 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.

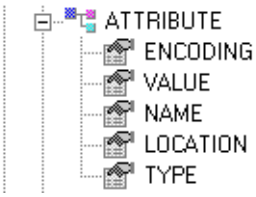
ACTIVITY.ATTRIBUTE Element

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 19.

Figure 19 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 attribute is actual data but storage in e*Insight is automatically determined.

TYPE

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

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

Chapter 6

e*Insight Implementation (eIJSchema)

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

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

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

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

6.1 Overview

The major tasks in the implementation are shown in Table 4.

Table 4 Overview of implementation tasks

	Task	Section
1	Create the business process (BP) in the e*Insight GUI	**"Create the Payroll BP in e*Insight" on page 72**
2	Use the e*Insight GUI to configure the e*Gate schema that supports e*Insight	**"Configure the Integration Schema (e*Insight)" on page 75**
3	Configure the e*Insight Engine	**"Configure the e*Insight Engine" on page 79**
4	Add and configure the user-defined e*Gate components	**"Configure User-defined e*Gate Components" on page 80**
5	Run and test the scenario	**"Run and Test the e*Insight Scenario" on page 89**

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

6.1.1 Case Study: Payroll Processing

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

Figure 20 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 20 e*Insight Data Flow Diagram

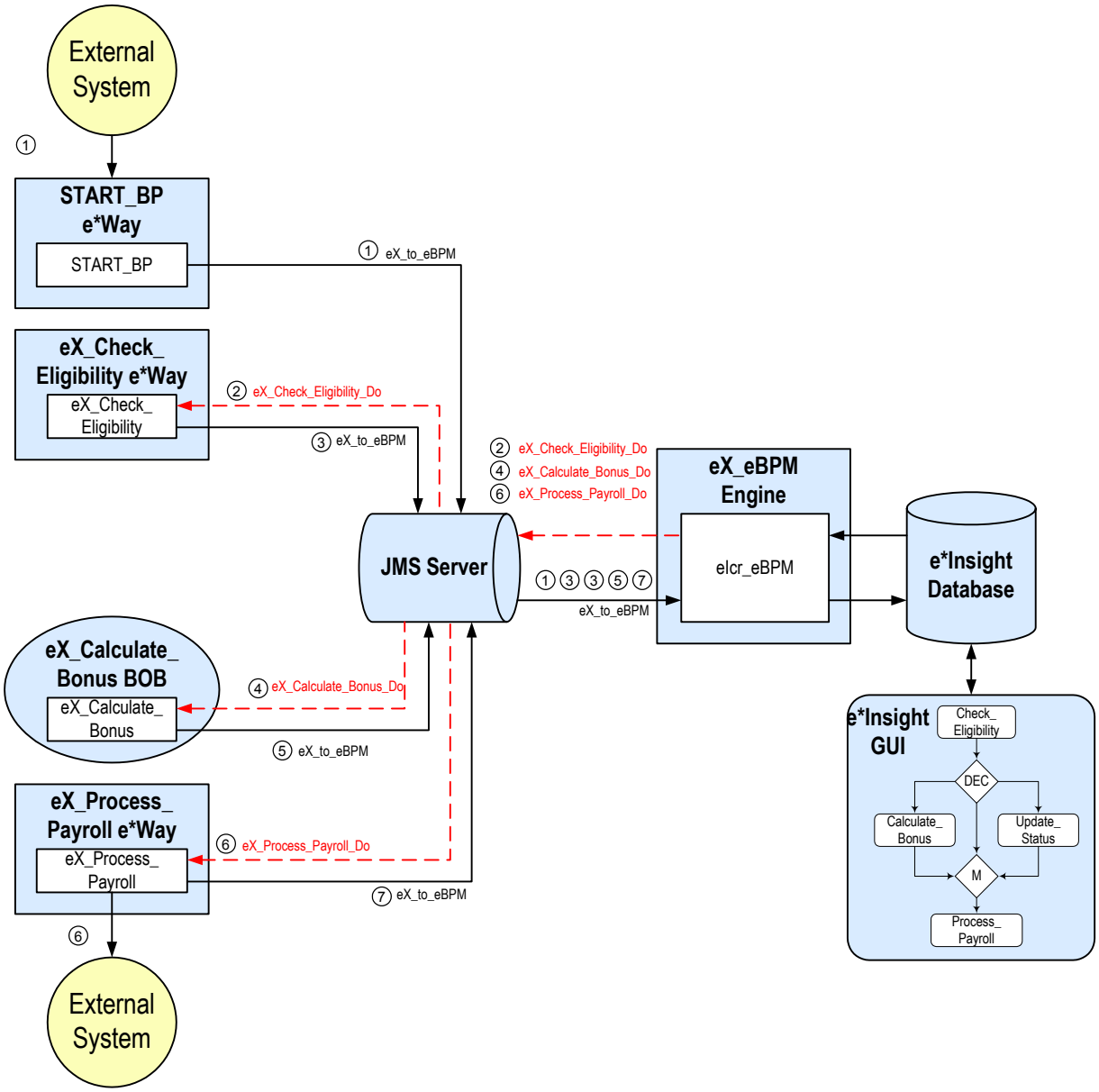
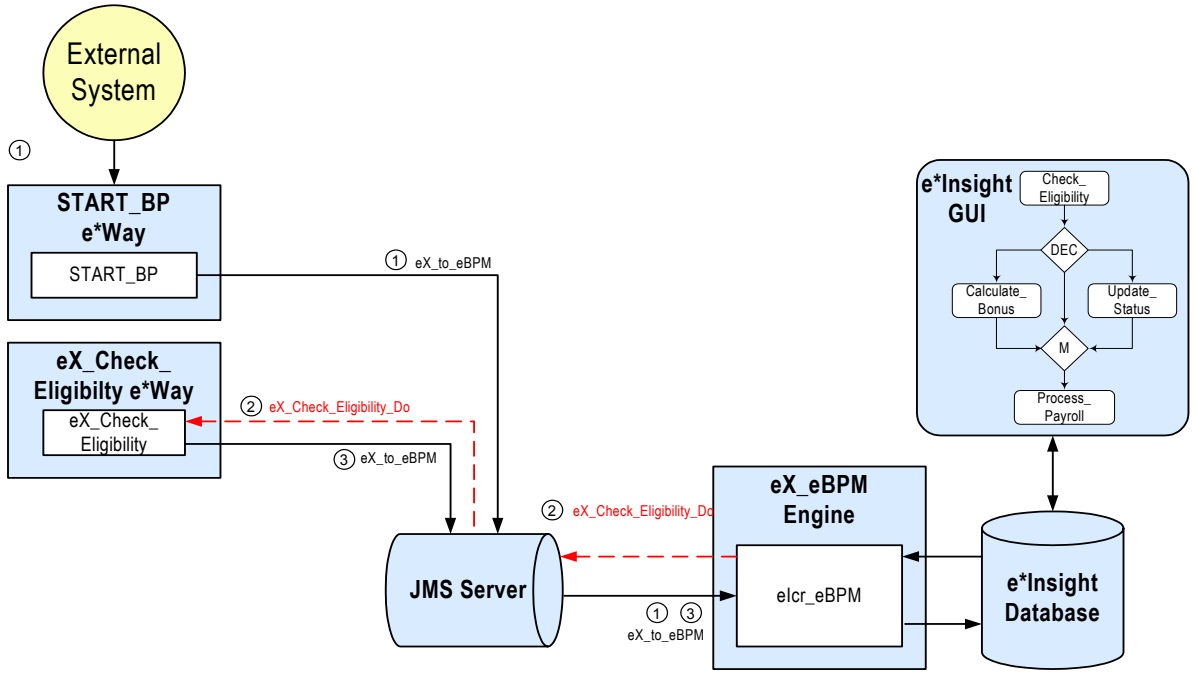


Figure 21 e*Insight Data Flow Diagram (Part 1)



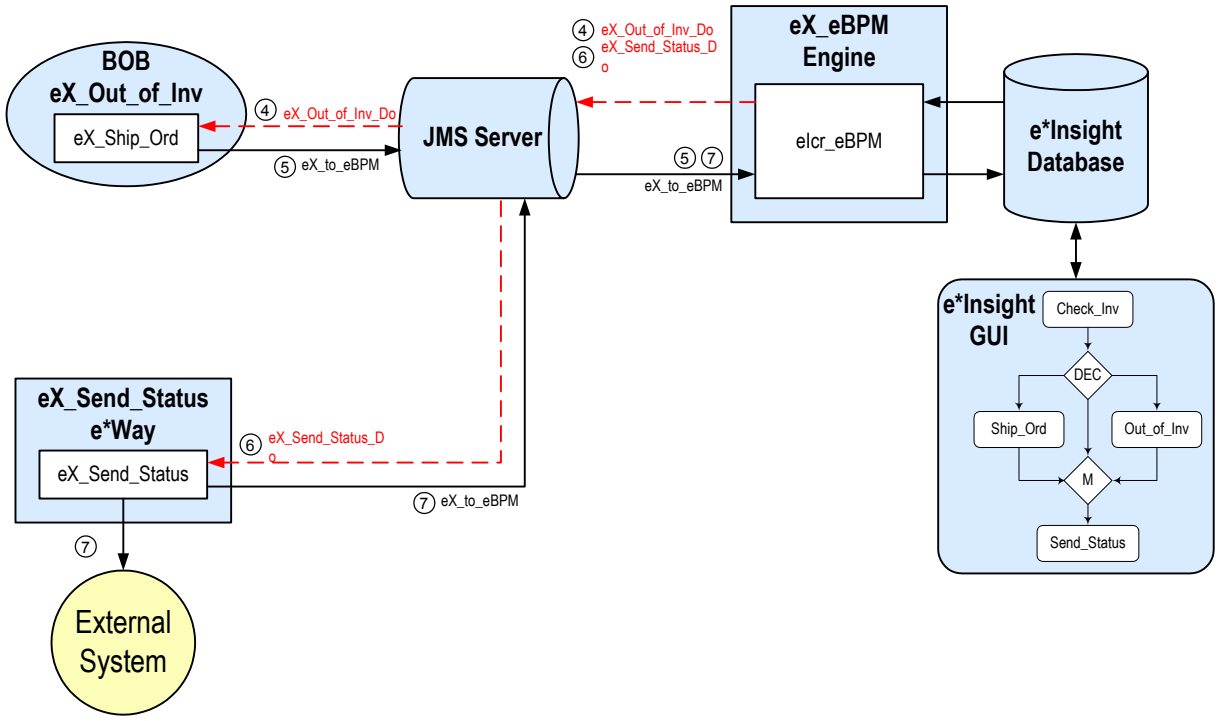
① The user-defined **START_BP** e*Way picks up the text file containing the employee's information from a shared location on the network, uses the payroll information to create the e*Insight Event that causes the e*Insight engine to start a business process instance, and publishes it using the **eX_to_eBPM**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

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

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

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

Figure 22 e*Insight Data Flow Diagram (Part 2)



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

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

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

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

6.2 Create the Payroll BP in e*Insight

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

1. Create a Business Process named **Payroll**.
2. Add the activities.
3. Add the decision gates.
4. Make the connections between the activities and gates.
5. Add all the global attributes (see Table 5).
6. Assign global attributes to activities (see Table 6).
7. Add the logic to the decision gates (see Table 7).
8. 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 23 and the following tables to create the BP in e*Insight.

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

Figure 23 Payroll Business Process Model

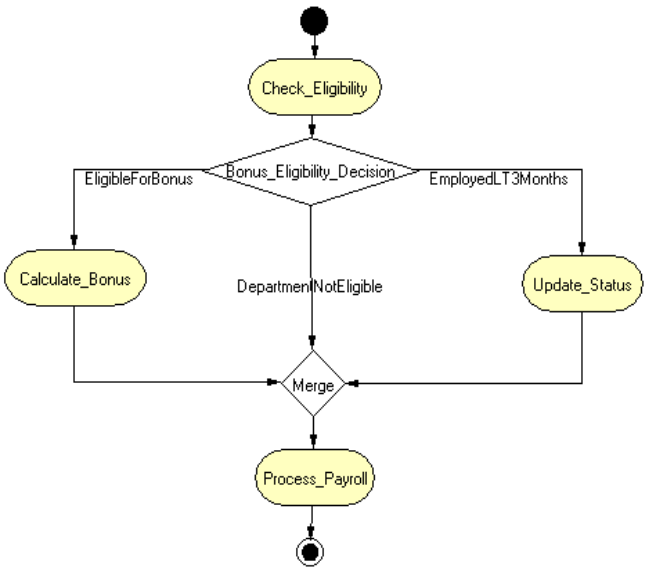


Table 5 BP Global Attributes

Attribute	Type	Data Direction	Default Value
FirstName	String	Input	
LastName	String	Input	
Department	String	Input	
Salary	Number	Input	
StartDate	String	Input	
PayPeriod	String	Input	
Grade	String	Input/Output	1
Probation	Boolean	Input	
Eligibility	String	Internal	null
Comments	String	Internal	No comment
Bonus	Number	Internal	0

Table 6 Activity Attributes

Activity	Attribute(s)	Input/Output
Check_Eligibility	Department	Input
	Probation	Input
	Eligibility	Output
	Comments	Output
Calculate_Bonus	Salary	Input
	Grade	Input
	Bonus	Output
	Comments	Output
Update_Status	Comments	Output
Process_Payroll	FirstName	Input
	LastName	Input
	Department	Input
	Salary	Input
	PayPeriod	Input
	Bonus	Input
	Comments	Input

Table 7 Decision Gate

Link	Target Activity	Expression
EmployedLT3Months	Update_Status	Probation==true
DepartmentNotEligible	Merge	Department==”sales”

Table 7 Decision Gate

Link	Target Activity	Expression
EligibleForBonus	Calculate_Bonus	(Default)

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_Eligibility**, **Calculate_Bonus**, and **Process_Payroll**) use e*Gate. This is described in **"Configure the Integration Schema (e*Insight)" on page 75**.

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

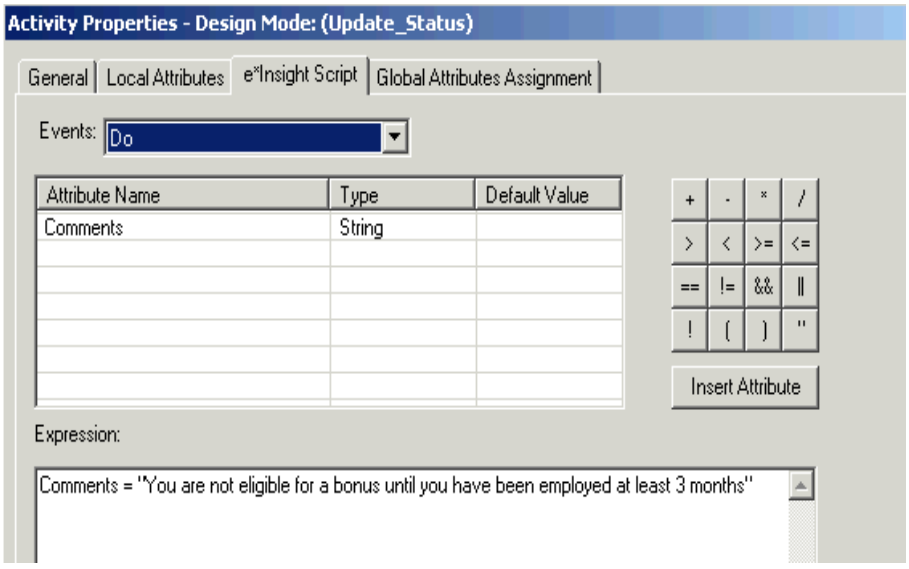
Configuring the e*Insight Script for Update_Status

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

To configure the e*Insight Script for Update_Status

1. From the **Update_Status** properties, **Activity Performed by** area, select **e*Insight Script**.
2. Select the **e*Insight Script** tab.
3. Configure the script as shown in Figure 24.

Figure 24 Update_Status e*Insight Script Tab



6.3 Configure the Integration Schema (e*Insight)

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

To create a copy of eIJSchema

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

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

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

Integration Schema Activity Components Summary

The information in Table 8 shows a summary of the e*Gate components that support this example.

Table 8 Integration Schema Activity Components

Name	Type	Participating Host	Configuration Instructions
eX_Check_Eligibility	Multi-Mode e*Way	localhost	**"Creating the eX_Check_Eligibility Multi-Mode e*Way" on page 76**
eX_Calculate_Bonus	BOB	localhost	**"Creating the eX_Calculate_Bonus BOB" on page 78**
eX_Process_Payroll	Single-Mode e*Way	localhost	**"Process_Payroll e*Way Configuration" on page 79**

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

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

Creating the eX_Check_Eligibility Multi-Mode e*Way

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

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

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

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

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

To configure the Check_Eligibility activity

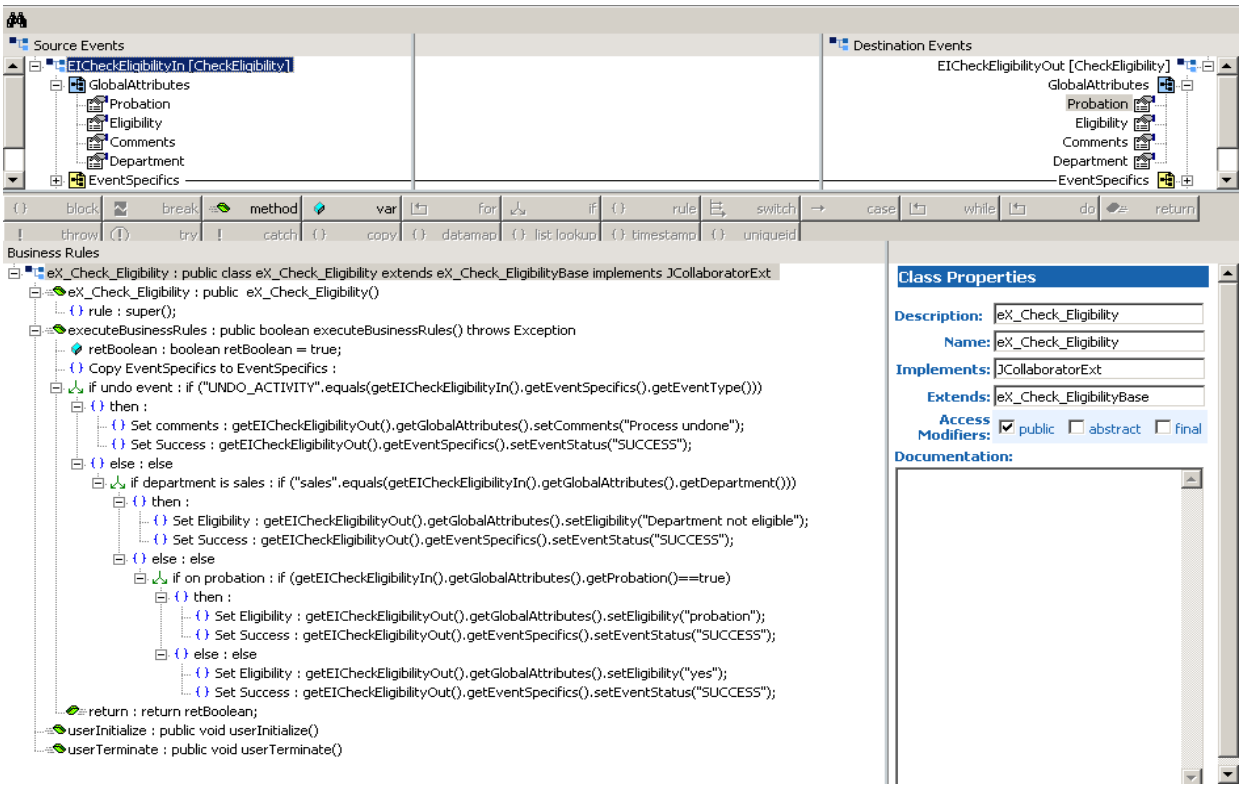
1. In the e*Insight GUI, open the **Check_Eligibility** activity properties.
2. On the **General** tab, **e*Gate Module** section, select a **Module Type** of **Multi-Mode e*Way**.
3. Click **New**.

   You may be required to log into e*Gate.

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

   Figure 25, on the following page, shows the **eX_Check_Eligibility.xpr** CRS used in this example.

Figure 25 eX_Check_Eligibility.xpr CRS (Java)



6 Compile and save the CRS.

7 Close the editor.

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

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

10 Close the **Check_Eligibility** Activity properties.

Creating the eX_Calculate_Bonus BOB

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

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

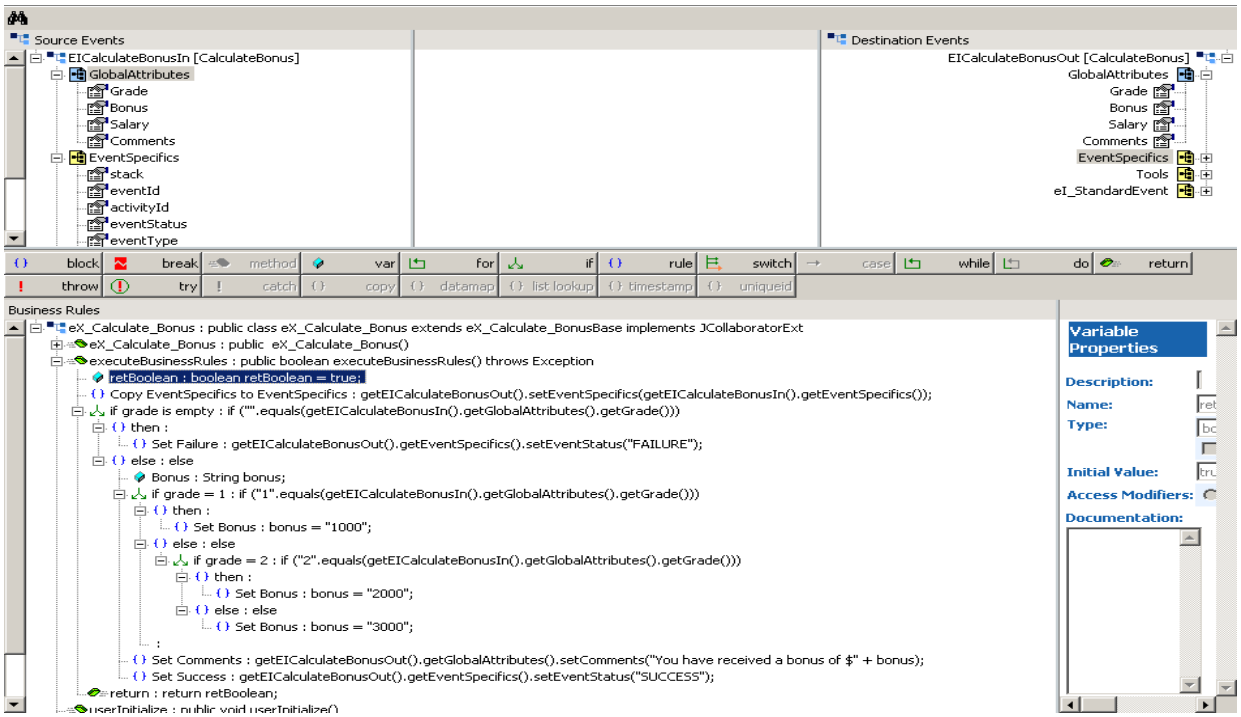
To configure the Calculate_Bonus activity

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

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

   Figure 26 shows the eX_Calculate_Bonus CRS used in this example.

Figure 26 eX_Calculate_Bonus.xpr CRS



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

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

10 Close the Calculate_Bonus Activity properties.

Process_Payroll e*Way Configuration

The Process_Payroll Collaboration is configured later, see **"Configure the Process_Payroll e*Way" on page 85**.

6.4 Configure the Integration Schema (e*Gate)

The remaining components used in this implementation are configured from the e*Gate Enterprise Manager. You must start the e*Gate Schema Designer and open the **Payroll** schema that you created in **"To create a copy of eIJSchema" on page 75**.

6.4.1 Configure the e*Insight Engine

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

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

Edit the eIcp_eInsightEngine Connection Configuration File

Most of the parameter settings in the eIcp_eInsightEngine connection's configuration file should not be changed. **"Using XA" on page 32** discusses the parameters that may need to be changed depending on the implementation. Use the e*Way Editor and the information in **"Using XA" on page 32** to make the required changes for the Payroll example.

6.4.2 Configure the JMS Connection

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

To configure the JMS Connection

1 From the e*Gate Schema Designer components view, select the **e*Way Connections** folder.

2 Select the **eI_cpeInsightJMS** connection, and click the **Properties** tool.

3 In the **e*Way Connection Configuration File** section, click **Edit**.

4 From the **Goto Section** list, select **Message Service**.

5 Enter a **Server Name** and **Host Name** where your JMS server resides.

Note: *Depending on your configuration and the volume of your message traffic, there may be a performance benefit if you use multiple JMS Servers in your schema. The message traffic can also be distributed to different JMS Servers. All publishers/subscribers for a particular topic should be attached to the same JMS Server to simplify the schema.*

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

Configuration Order for the User-defined Components

Table 9 shows the configuration order for the user-defined components.

Table 9 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 80**
2	Configure the Process_Payroll e*Way	**"Configure the Process_Payroll e*Way" on page 85**

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 **eIJSchema** in the e*Gate Enterprise Manager. In an actual implementation, the choice of e*Way (or BOB) would depend on the requirements of the situation. For example, if the data were coming from an SAP system, you might select an SAP ALE e*Way; or if the data were

already in the e*Gate system, you could use a BOB to start the BPI. In the present case, a text file on the local system provides the input data, therefore this example uses a file e*Way to send the "Start" Event to the e*Insight engine.

Table 10 shows the steps to configure the **START_BP** e*Way.

Table 10 Configuration steps for the START_BP e*Way

	Step	Section
1	Add the e*Way and create the e*Way configuration file	**"Step 1: Create the START_BP e*Way" on page 81**
2	Create the Input ETD	**"Step 2: Create the Input ETD" on page 81**
3	Create the START_BP Collaboration Rules script (CRS)	**"Step 3: Create the START_BP Collaboration" on page 82**
4	Configure the Collaboration in the GUI	**"Step 4: Configure the Collaboration in the GUI" on page 85**

Step 1: Create the START_BP e*Way

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

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

Table 11 Start e*Way Parameters

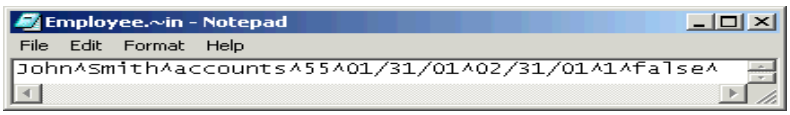
Screen	Parameter	Setting
General Settings	(All)	(Default)
Outbound (send) settings	(All)	(Default)
Poller (inbound) settings	PollDirectory	<eGate>\client\data\Payroll
	(All others)	(Default)
Performance Testing	(All)	(Default)

Step 2: Create the Input ETD

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

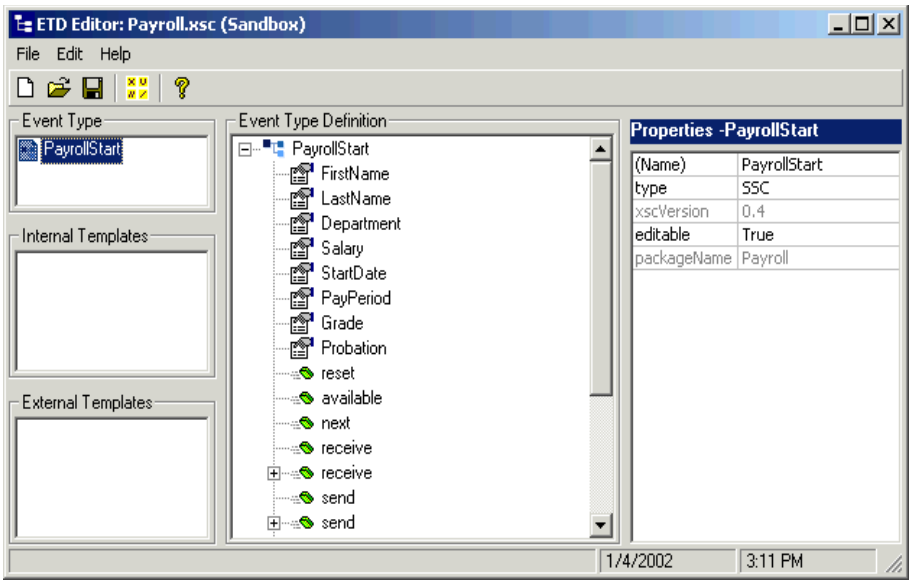
The input data file used in this example is shown in Figure 27. Place this data file at the directory location **c:\eGate\client\data\Payroll**.

Figure 27 Input Text File (Employee.fin)



Using the ETD Editor and the input data as a guide, create an ETD like the one shown in Figure 28. Set the end delimiter to a ^ character. For more information on using the ETD Editor see the ETD Editor's online help.

Figure 28 Input ETD: PayrollStart.xsc (Java)



Step 3: Create the START_BP Collaboration

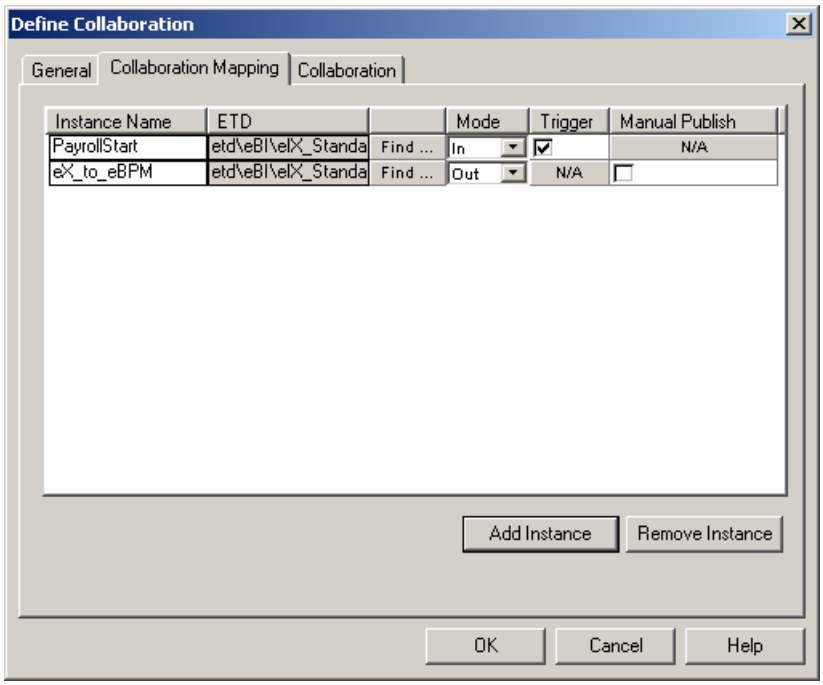
The Collaboration that sends the Event that starts the BPI must do two things:

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

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

1. Create a Collaboration Rule, **START_BP**, that uses the Java service.
2. Configure the Collaboration Mapping tab, as shown in Figure 29.

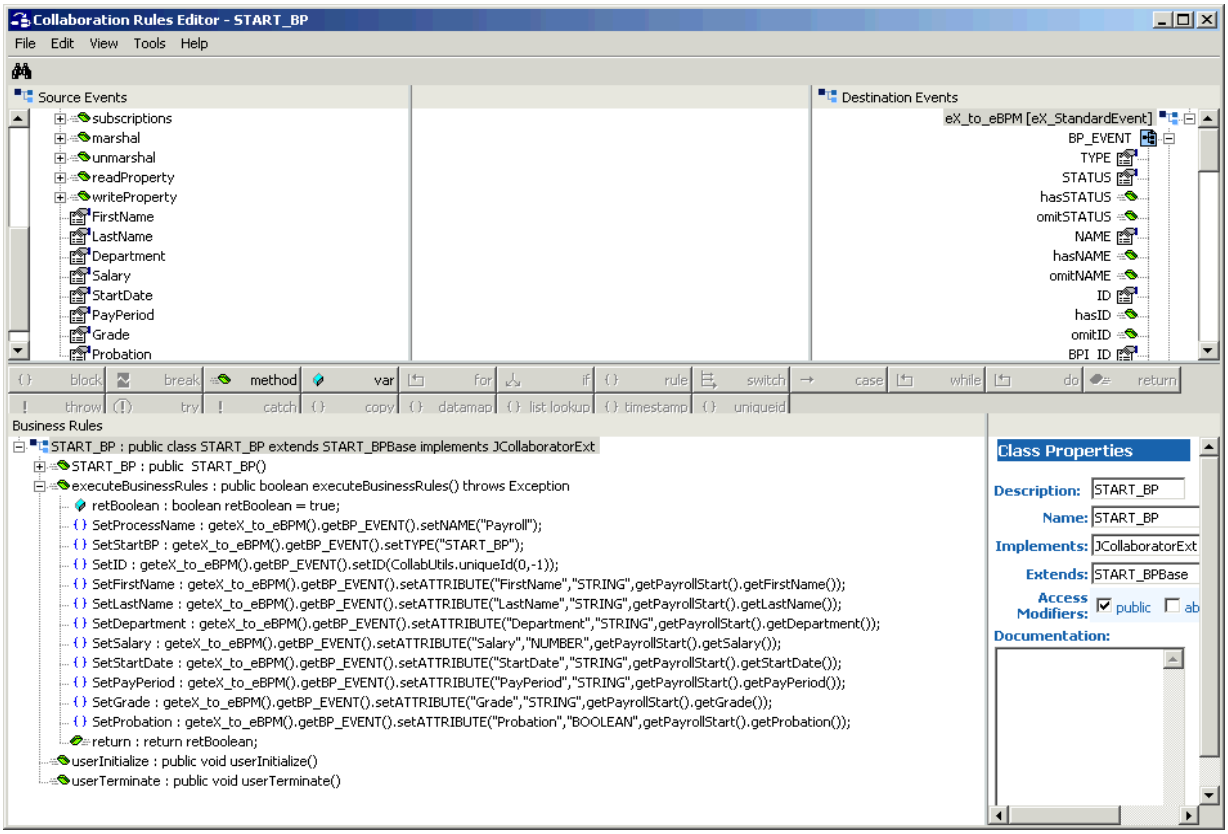
Figure 29 Start_BP Properties, Collaboration Mapping Tab



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

4 Click **New** to create a new CRS, as show in Figure 30.

Figure 30 START_BP CRS

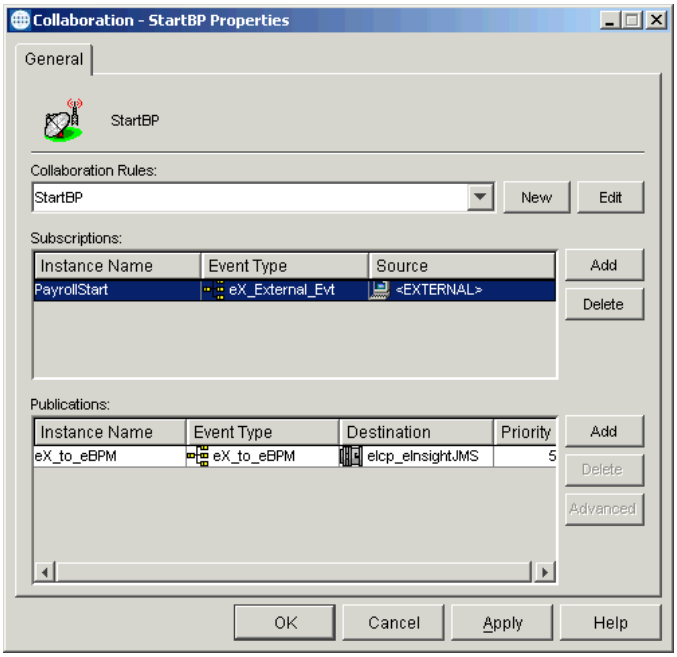


Step 4: Configure the Collaboration in the GUI

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 Schema Designer.

1 Create a Collaboration for the **Start_BP** e*Way configured as shown in Figure 31.

Figure 31 Start_BP Collaboration



6.5.2 Configure the Process_Payroll e*Way

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

This e*Way must accomplish two tasks:

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

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

Table 12 shows the steps to configure the Process_Payroll e*Way.

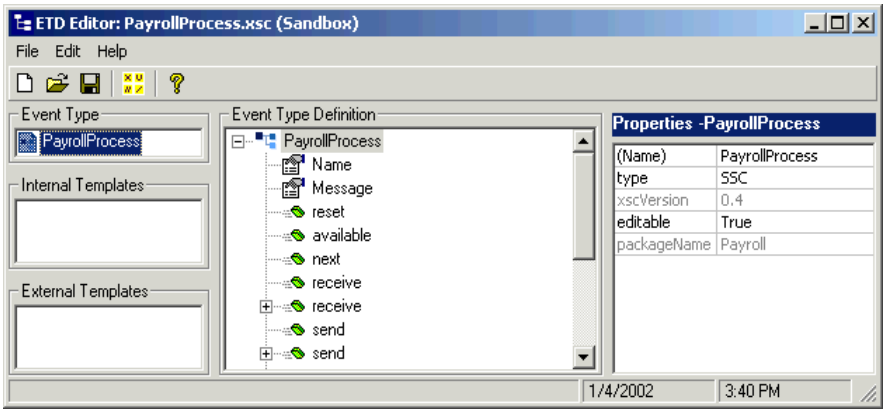
Table 12 Process_Payroll e*Way configuration steps

	Step	Section
1	Find the executable and create the e*Way configuration file	**"Step 1: Configure the e*Way" on page 88**
2	Create the Output ETD	**"Step 1: Create the Output ETD:PayrollProcess.xsc using Java" on page 86**
3	Create the eX_Process_Payroll.tsc CRS	**"Step 3: Create the Process_Payroll Collaboration Rule" on page 86**
4	Configure the Collaboration in the GUI	**"Step 4: Configure the Collaboration" on page 87**

Step 1: Create the Output ETD:PayrollProcess.xsc using Java

Use the e*Gate ETD Editor to create an ETD like that shown in Figure 32. Set the global delimiter to a , (comma) character. For more information on using the ETD Editor see the ETD Editor's online help.

Figure 32 PayrollProcess.xsc ETD



Step 3: Create the Process_Payroll Collaboration Rule

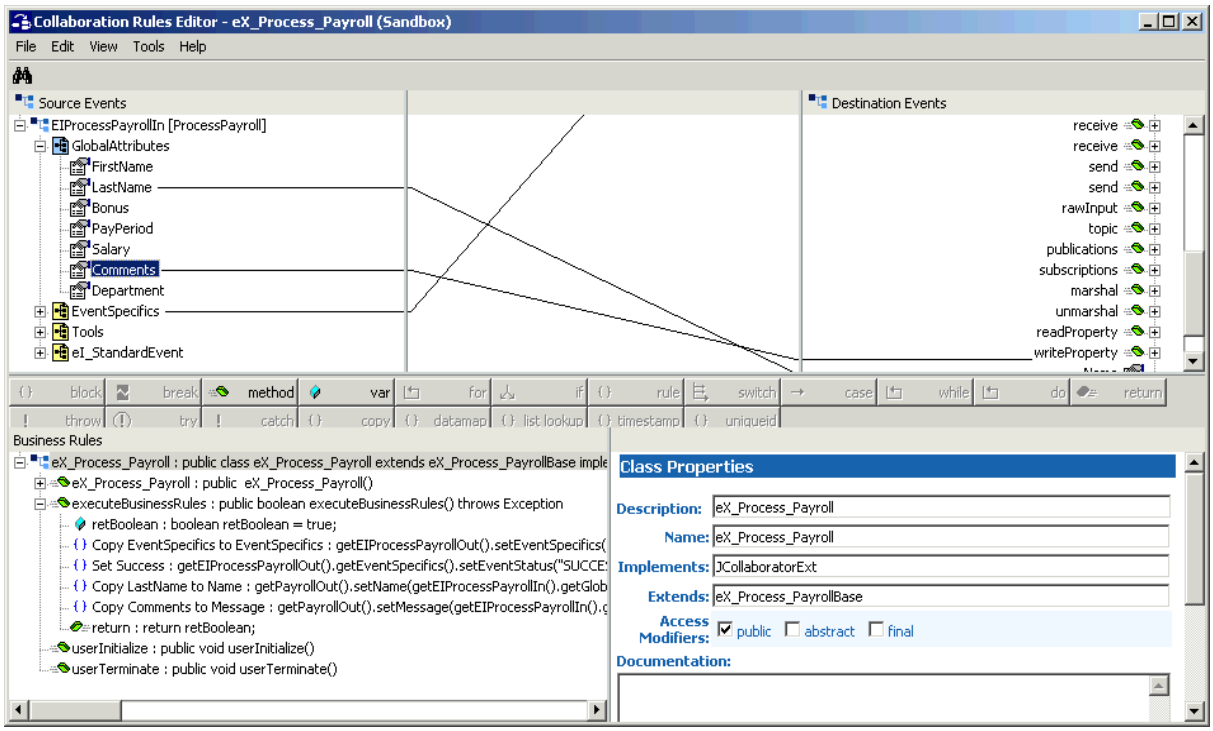
This CRS must accomplish three things:

- put the output data into a readable format that can be written to a file
- set the BP status node to "SUCCESS"
- send the "Done" Event back to the e*Insight engine

To create the Process_Payroll Activity e*Way and Collaboration Rule

1. In the e*Insight GUI, open the **Process_Payroll** activity properties.
2. On the **General** tab, **e*Gate Module** section, from the **Module Type** list select the **Multi-Mode e*Way**.
3. Click **New**.
4. From the eX_Process_Payroll Collaboration Rule **Collaboration Mapping** tab, add the new ProcessPayroll instance.
5. Click **Apply**, and click the **General Tab**.
6. Click **OK**, to open the Collaboration Editor, and add code as show in Figure 33.

Figure 33 eX_to_Process_Payroll.xpr CRS



7. Compile and save the CRS.
8. Close the editor.
9. In the **Process_Payroll** Activity properties, click **Configure e*Gate Schema**.
10. Click **OK** to close the information dialog.
11. Close the **Process_Payroll** Activity properties.

Step 4: Configure the Collaboration

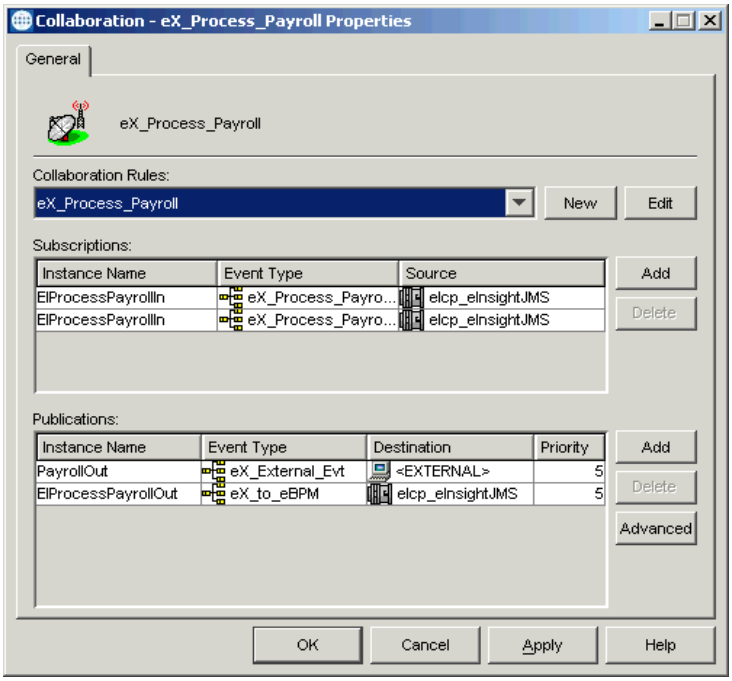
In the Enterprise Manager:

1. Refresh the Schema to show the new components.

2 Highlight the **eX_Process_Payroll** e*Way.

3 Configure the Collaboration, as shown in Figure 34.

Figure 34 eX_Process_Payroll Collaboration



Step 1: Configure the e*Way

First change the executable, then create the configuration file.

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

Table 13 Process_Payroll e*Way Parameters

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

6.5.3 Run and Test the e*Insight Scenario

Once the schema is set up in e*Gate, you can run the scenario.

Testing the Standard Business Logic

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

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

Payroll Processing

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

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

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

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

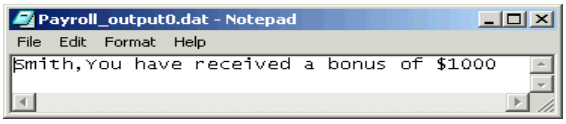
```
stccb.exe -rh localhost -rs Payroll -ln localhost_cb
     -un username -up password
```

   Substitute the appropriate username and password for your installation.
4. Start the e*Gate Monitor, and check the status of all the components. Any components displayed in red are not running. Any e*Insight components that are not running should be investigated before feeding data into the system.
5. Navigate to the location for the input data file, **Employee.~in**, shown in **Figure 27 on page 82** (**c:\eGate\client\data\Payroll**) and change the extension to "**.fin**".

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

If everything is working correctly, an output file (**Payroll_output#.dat**) as shown in Figure 35 appears in the directory indicating successful completion of the BPI.

Figure 35 In Stock Output File

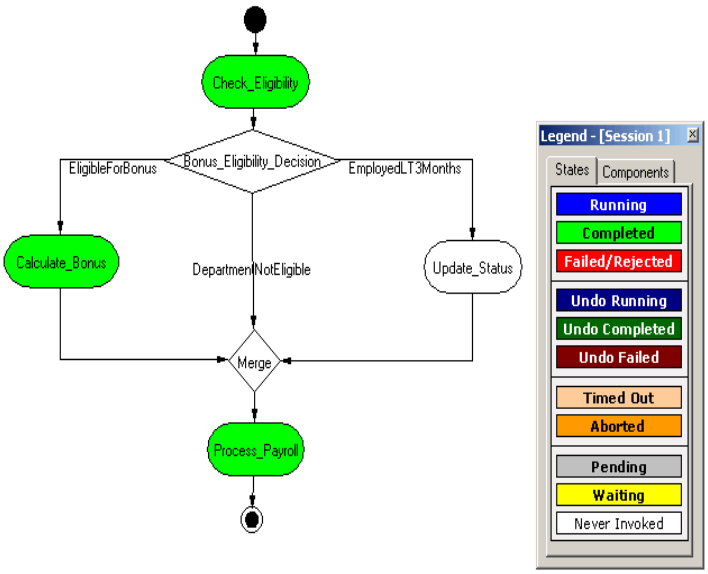


6 Switch to the e*Insight GUI and, while in monitor mode, select the most recent BPI from the **List** tab, and then select the **Diagram** tab to observe the path that the data has taken.

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

In the Payroll example, an activity that stays blue for more than couple minutes indicates a problem, and the e*Gate component associated with that activity should be investigated for the cause of the problem. Figure 36 illustrates how the successfully completed BPI appears in the e*Insight GUI.

Figure 36 Eligible for Bonus Completed BPI Diagram



Not Eligible Processing

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

- Verify that sending in the data shown in Figure 37 with a department of sales causes the business process to take the "EmployedLT3Months" branch of the decision gate and create the diagram shown in Figure 38 and the output file shown in Figure 39.

Figure 37 EmployedLT3Months Input File

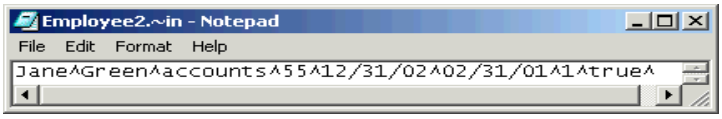


Figure 38 Employed Less Than 3 months Completed BPI Diagram

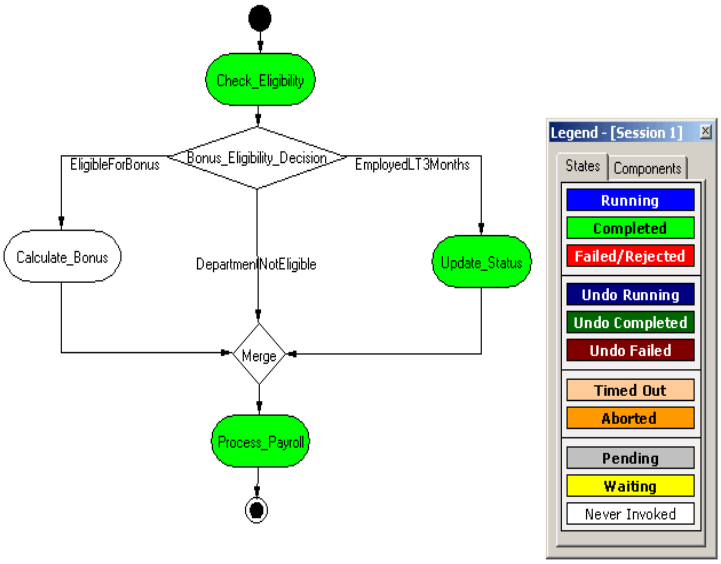
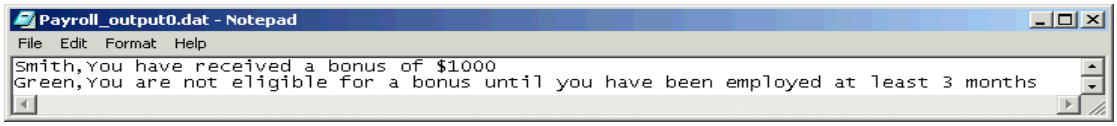


Figure 39 Out of Stock Output File



- Verify that sending in the data shown in Figure 40 with a department of sales causes the business process to take the "DepartmentNotEligible" branch of the decision gate and create the diagram shown in Figure 41 and the output file shown in Figure 42.

Figure 40 DepartmentNotEligible Input File

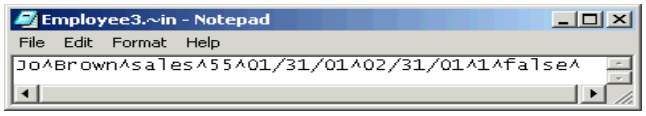


Figure 41 Department Not Eligible Completed BPI Diagram

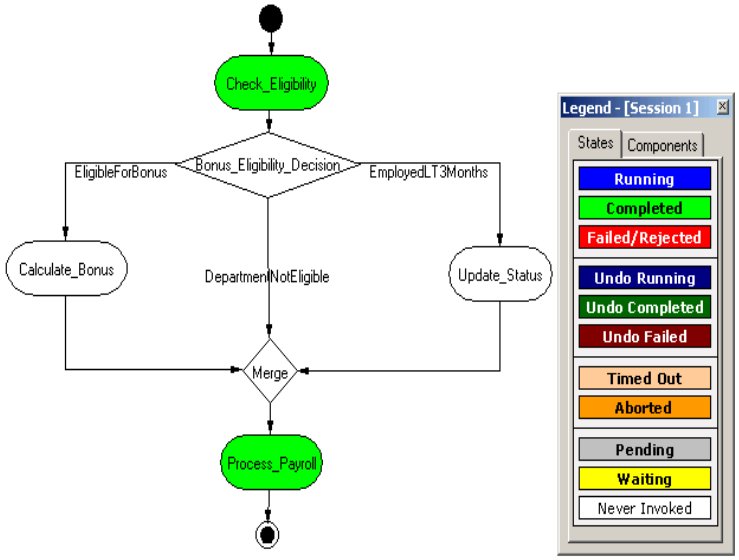
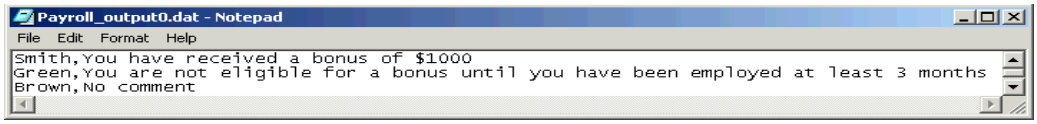


Figure 42 Department Not Eligible Output File



6.5.4 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** option is selected on the **General** tab of the **Activity Properties** dialog box for that activity, then e*Insight marks the activity as "Failed" in the GUI. Then the activity waits for the user to initiate the next course of action; skip, restart, or undo. If the user selects undo, then the BPI is undone as described in the paragraph above.

Manual Undo

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

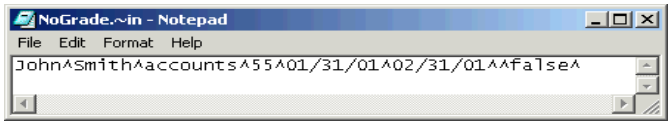
1 Perform steps 1 through 4 outlined in **"Payroll Processing" on page 89**.

2 Verify that **Manual Restart** is selected for the activities in the business process.

If **Manual Restart** is not selected, you must delete the BPIs for the business process, or save the business process as a new version, before selecting it. Refer to the *e*Insight Business Process Manager User's Guide* for information on how to do this.

3 Navigate to the location (**c:\eGate\client\data\Payroll\NoGrade.~in**) for the input data file with no grade defined as shown in Figure 43 and change the extension to "**.fin**".

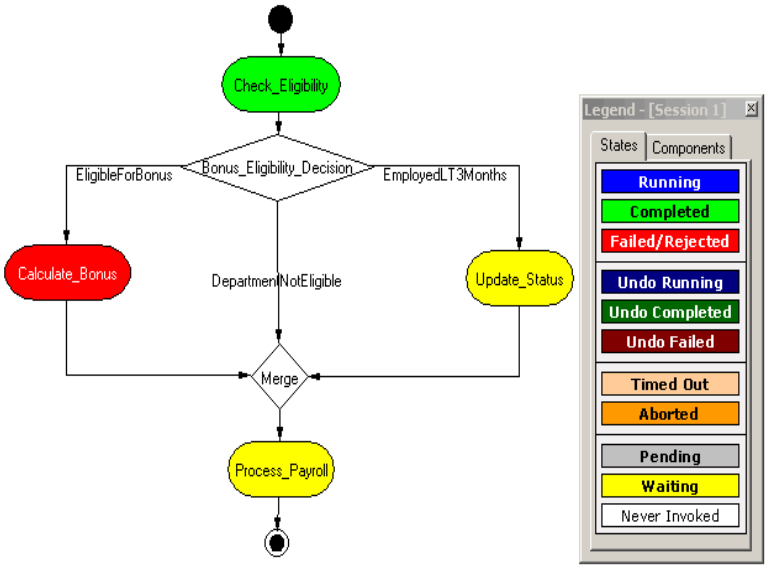
Figure 43 Manual Undo Input File



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

4 Switch to the e*Insight GUI and, while in monitor mode, select the most recent business process instance. Observe the path that the data has taken, as shown in Figure 44 on the next page.

Figure 44 Manual Undo—Failed BPI Diagram



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

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

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

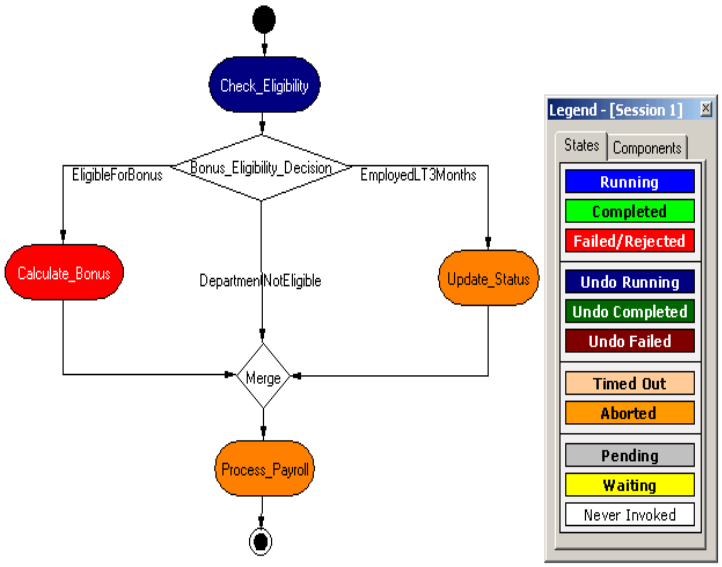
6 Select the **Business Process Attributes** tab.

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

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

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

Figure 45 Manual Undo Completed BPI Diagram



6.5.5 Demonstrating Business Process Restart Functionality

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

Repairing a String Attribute

Attributes can be of various types; Boolean, number, string, and XML. The following example shows the procedure to repair an attribute of type string. For information on

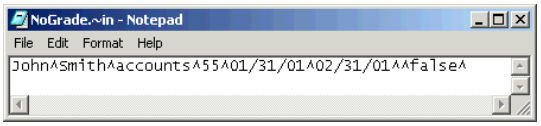
repairing an attribute with type XML, see the *e*Insight Business Process Manager User's Guide*.

1 Perform steps 1 through 4 outlined in **"Payroll Processing" on page 89**.

2 Verify that **Manual Restart** is selected for the activities in the business process.

3 Navigate to the location (**c:\eGate\client\data\Payroll\NoGrade.~in**) for the input data file with the grade not defined as shown in Figure 46, and change the extension to "**.fin**".

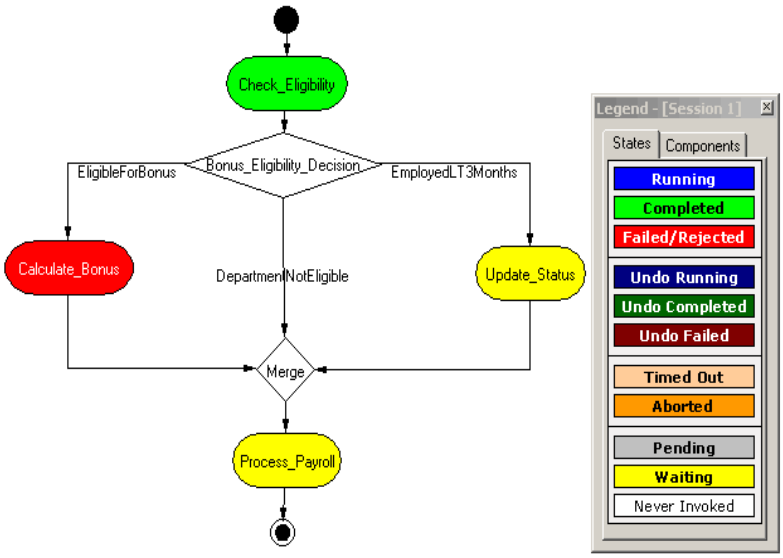
Figure 46 Attribute Repair Input File



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

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

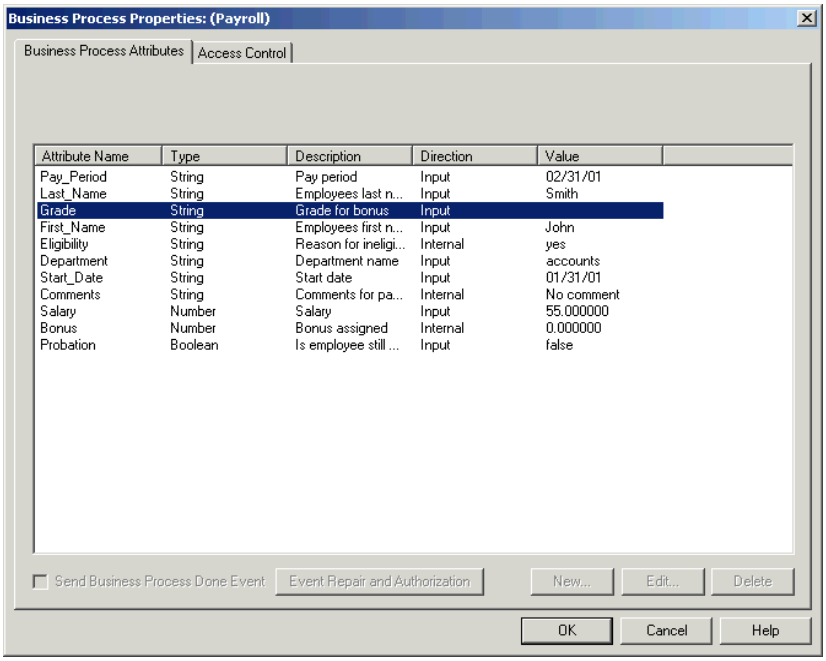
Figure 47 Attribute Repair—Failed BPI Diagram



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

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

Figure 48 Attribute Repair—Business Process Properties



6 Select the **Business Process Attributes** tab in the **Business Process Properties** dialog.

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

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

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

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

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

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

The BPI now completes successfully.

Chapter 7

e*Insight Authorization Activity Implementation (eIJSchema)

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

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

This case study is a continuation of the previous example. See **"e*Insight Implementation (eIJSchema)" on page 68** for the initial configuration instructions.

7.1 Overview

The major steps in the implementation are:

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

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

7.1.1 Case Study: Payroll Processing

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

Important Considerations

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

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

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

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

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

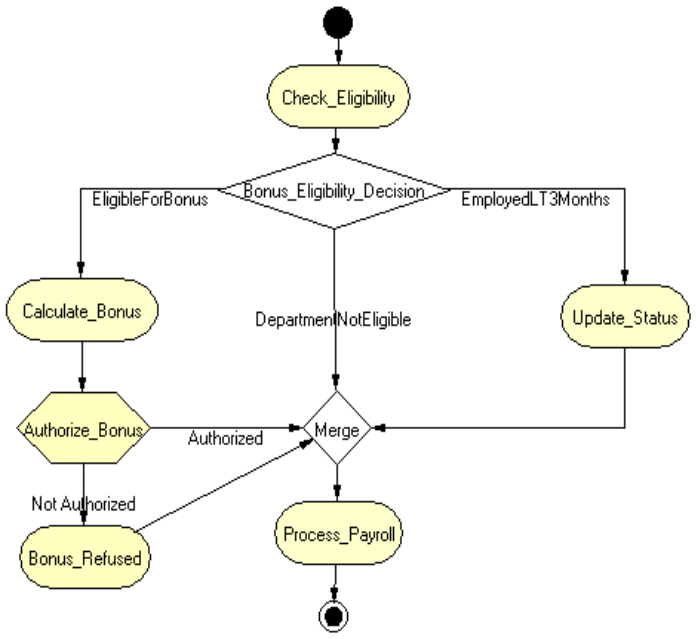
Note: *The Authorization Activity has two fixed Local Attributes—assignedTo (the user to whom the Authorization process is assigned) and performedBy (automatically assigned at run time as the user logged in to the e*Insight GUI). These two values must match in order for the user to Authorize, Reject,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

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

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

Use the diagram shown in Figure 49 and the following tables to create the BP in e*Insight.

Figure 49 Payroll Business Process Model



7.2.1 Creating the processes performing the Activities

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

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

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

Configuring the e*Insight Script for Bonus_Refused

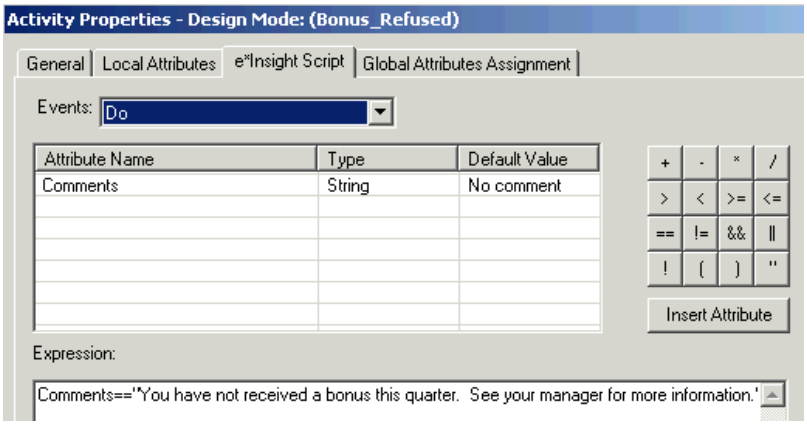
This script defines a message that appears on the pay slip. It sets the value of the **Comments** attribute to a short message indicating that the employee has not been paid a bonus.

To configure the e*Insight Script for Update_Status

1. From the **Bonus_Refused** properties, **Activity Performed by** area, select **e*Insight Script**.
2. Select the **e*Insight Script** tab.

3 Configure the script as shown in Figure 50.

Figure 50 Update_Status e*Insight Script Tab



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

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

7.3.1 Testing the Standard Business Logic

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

Authorized Processing

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

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

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

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

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

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

Substitute the appropriate username and password for your installation.

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

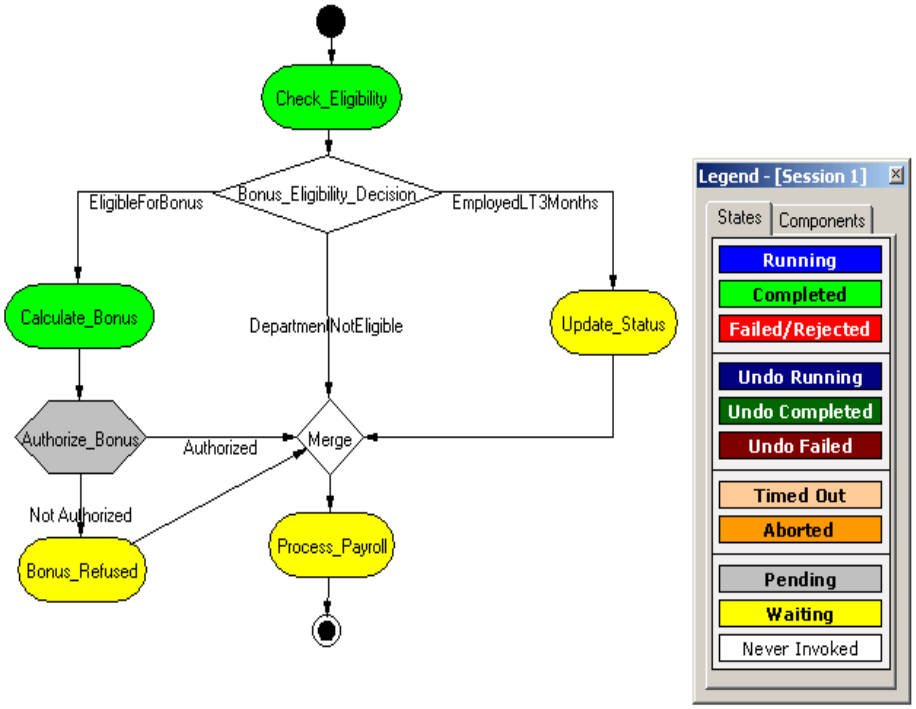
5 Navigate to the location for the input data file, **Employee.~in**, (**c:\eGate\client\data\Payroll**) and change the extension to "**.fin**".

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

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

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

Figure 51 Authorize_Bonus Pending BPI Diagram



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

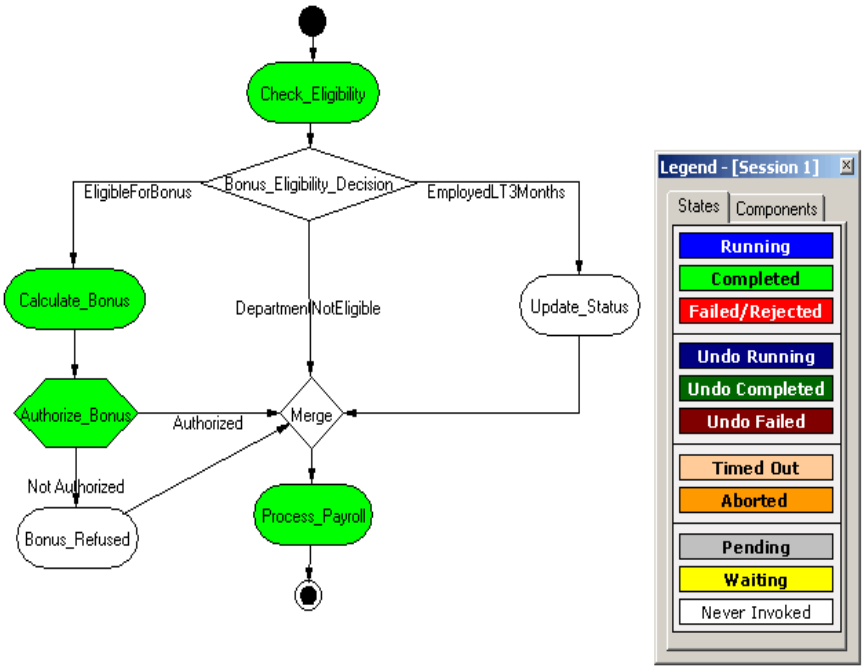
The **Authorization Activity Properties - Monitor Mode: (Authorize_Bonus)** is displayed.

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

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

10 The Business Process then completes using the route shown in Figure 52.

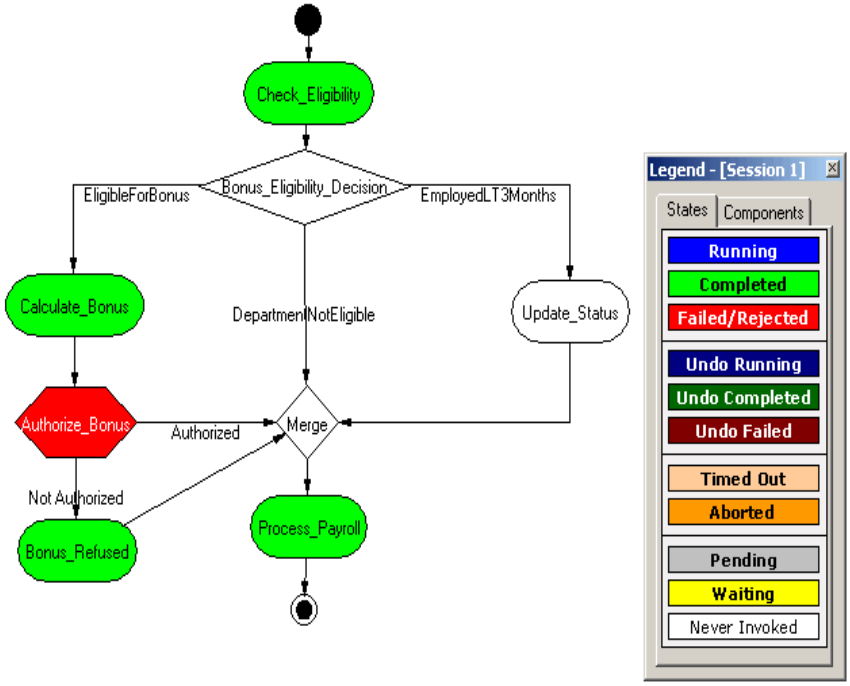
Figure 52 Authorization_Quantity - Authorized



Not Authorized Processing

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

Figure 53 Authorize_Bonus - 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 studies could be enhanced to include the User Activity.

8.1 Overview of the User Activity

User Activities allow external applications to access attributes in the business process using an Application Programming Interface (API). The e*Insight engine uses the returned value of the attributes to continue the business process.

The role of the API is to allow the external application to communicate with the e*Insight engine for the purpose of setting security and verifying users, retrieving business model related information, getting and returning attribute types and values, tracking instances, and many other functions related to the business process instances. However, design and development of the external application is left up to the developer.

For more information on the API, see **"e*Insight User Activity API Methods" on page 299**.

8.1.1 User Activity Security

Three security checks are performed when connecting to the database using the User Activity methods. First, use the **initialize** method to connect to the database. You should use a user that has no authority to access any of the Business Processes.

Once that connection has been made, use the **authenticate** method to pass the user name and password for a user that has privileges for the Business Process. This user should have the necessary authority for the Business Processes that they are accessing. For subsequent messages sent during the session, use the **setUser** method to re-establish the user security, or **resetUser** to establish security for a new user.

To create a user for the initial connection

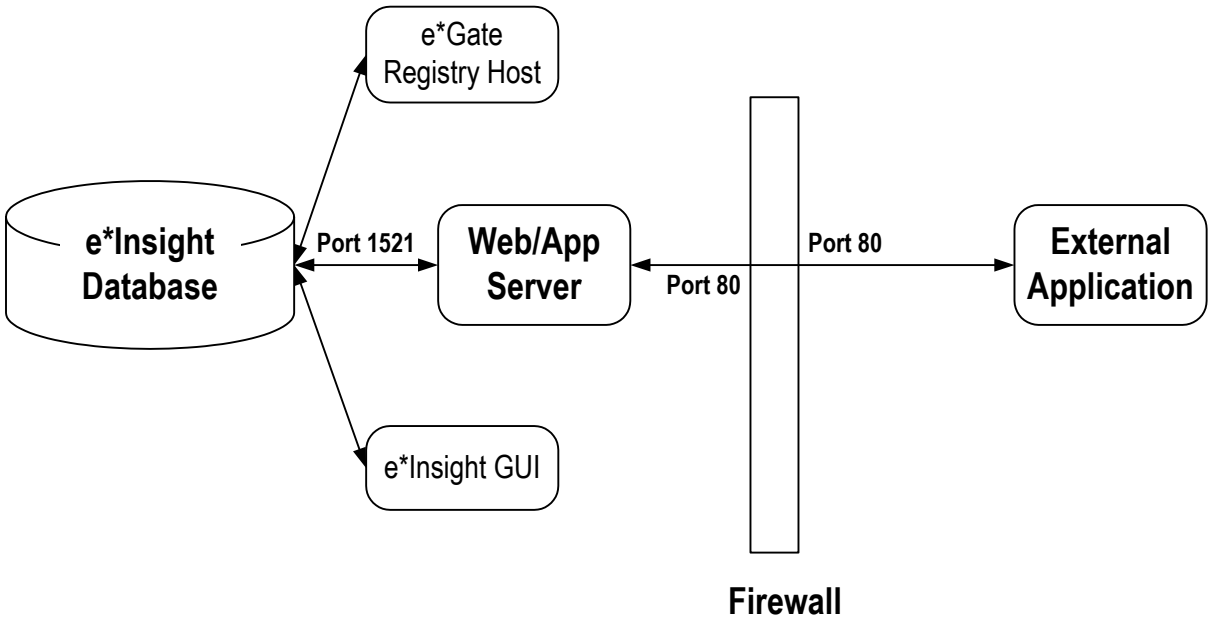
1. Use e*Insight Administrator to create a user (for example, Connection_User), and assign a password.
2. Do not give this user any authorization rights within e*Insight.

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

8.1.2 Deployment of the User Activity

The application used with the User Activity may connect directly to the database, or it may be a Web-based application that connects to the database via a Web/Application Server. The security described in **"User Activity Security" on page 104** is still valid when connecting via a Web/App Server, however there are additional considerations. Figure 54 shows how e*Insight, the Web/App Server, and the external application may be deployed.

Figure 54 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*.

8.2 Overview of the Payroll BP

In this example, we assume we have a custom Web application developed for the purpose of allowing various users to set the bonus values. The current example uses an Automated Activity to do a similar function from within the e*Insight GUI. By replacing the Automated Activity with a User Activity, the business process can now integrate with an external application such as a Web-based application. This application

would allows users to login, view lists of the business process instances, make appropriate changes, and have those changes re-enter the e*Insight business model for continued processing.

While this example specifically uses a custom Web application, design and development of the external application is left up to the developer. The external application uses the Java APIs provided with the e*Insight engine (a complete description and list of these APIs is available in **"e*Insight User Activity API Methods" on page 299**). 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.

8.3 Overview

The major steps in the implementation are:

1. Add the User Activity to the Business Process.
2. Use the e*Insight GUI to configure the e*Gate schema that supports e*Insight.
3. Add and configure the user-defined e*Gate components.
4. Run and test the scenario.

8.3.1 Case Study: Payroll Processing with User Activity

The Payroll example uses an external application to allow various users to determine the bonus for an employee. The external application retrieves new instances, and for those employees that are eligible for a bonus, the manager enters a bonus value. The user of the external application could also perform other changes to the order based on capabilities developed into the external application.

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

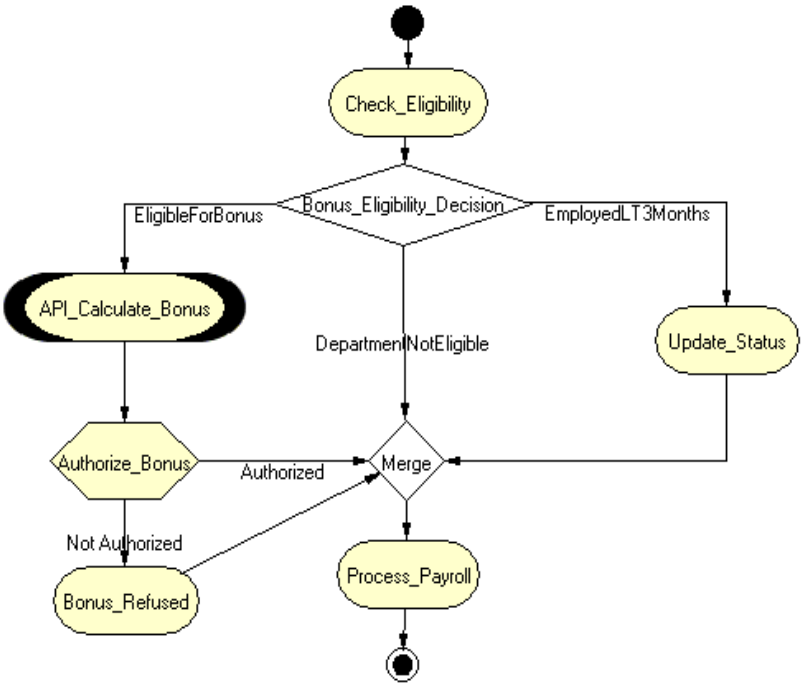
The following is a summary of the procedures for adding a User Activity in the e*Insight GUI.

1. Delete the **Calculate_Bonus** Activity.
2. Add a User Activity called **API_Calculate_Bonus**.
3. Make the connections between the Decision gates, Merge gate, and User Activity, as shown in Figure 55.

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 55 Payroll Business Process Model



8.5 Step 2: Configure the Integration Schema

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

You can send notifications through e*Gate to the external application but in this example, the same result is achieved using only the e*Insight engine. To use only the e*Insight engine, double click the User Activity, and make sure the "Send Notifications through e*Gate check box is not marked.

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

8.6 Step 3: Run and Test the e*Insight scenario

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

8.6.1 Testing the User Activity

Use the following procedure to test the functionality of the User Activity.

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

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

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

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

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

Substitute the appropriate username and password for your installation.

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

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

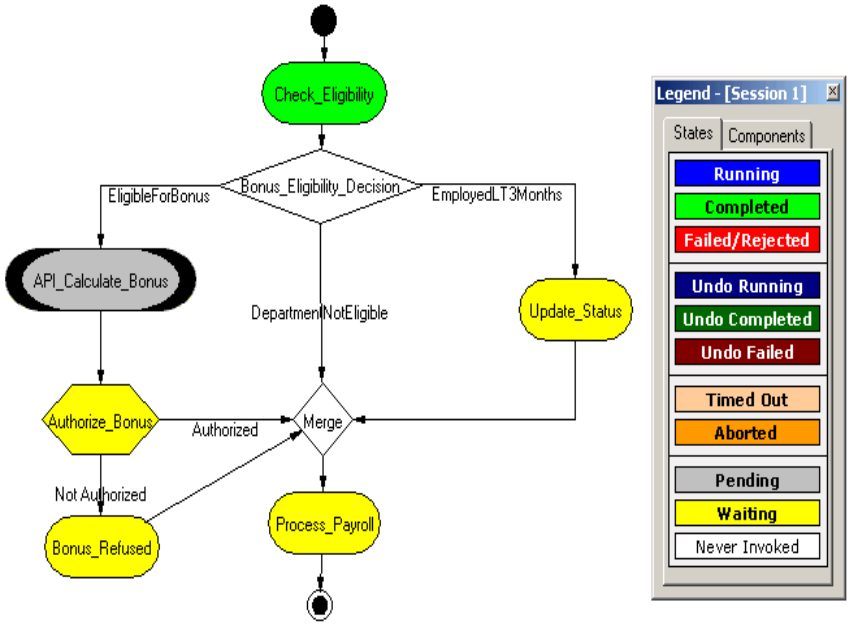
Note: *The change of the extension to "**.~in**" indicates that the data file has been picked up by the **START_BP** e*Way.*

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

The **API_Authorize_Quantity** should be in a pending state. This remains in a pending state until the e*Insight engine receives a message from an external application relating to this Business Process Instance.

Figure 56 shows how the BPI Diagram appears.

Figure 56 API_Calculate_Bonus Pending BPI Diagram



7 Start the external application. Return to the e*Insight GUI. The Business Process then completes using the route shown in **Figure 57 on page 110**.

The external application refers to any application developed by you to be inserted into the business process. A common example could be a method by which users can view lists of business instances assigned to them. These instances may require a simple review/authorize/reject process or, a more complex task using a custom-developed GUI interface to adjust or introduce attribute values. After the application completes its job, the updated business process instance information re-enters the User Activity and the business process continues.

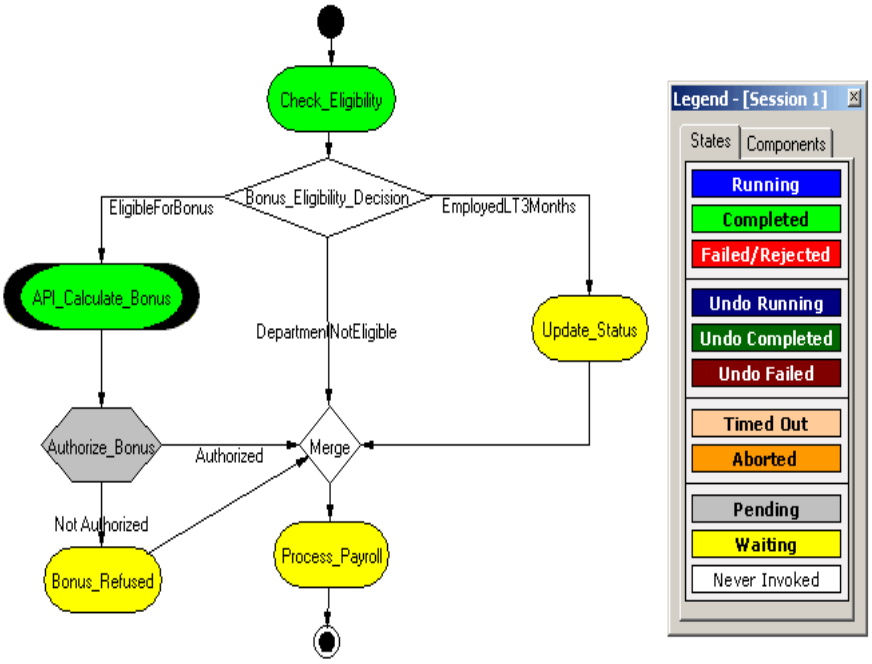
The external application uses an API to communicate with the e*Insight engine. Listed below are some examples of API functions to help illustrate what kind of information is available to the external application (a complete list of API functions can be found in Chapter 16).

- **setUser**—This function passes the name of the user to the e*Insight engine if the external application uses login/password.
- **checkUserPrivileges**—This function establishes the security rights for the session using security levels as set within e*Insight.
- **getGlobalAttributeValue** and **getLocalAttributeValue**—This function receives Attribute values for a business process instance for use within the external application.
- **getBusinessModelInstanceIds**—The external application uses this function to retrieve identification information for business model instances and then uses this information to appropriately track instances when sending information back into the business process.

An example of a business model including a User Activity and an external application is shipped with e*Insight. This example can be installed by you and set up and used to demonstrate a business process using a Web-based application to allow users to log in, and view lists of instances that need to be reviewed and adjusted. The external application then sends the information back to e*Insight, thus allowing the business process to continue.

Note: *For information on installing the sample business model see the e*Insight Business Process Manager Installation Guide.*

Figure 57 User Activity - Bonus set



8.7 Overview of the ProcessOrder BP

In this example, we assume we have a custom Web application developed for the purpose of allowing various users to set the quantities. The current example uses an Authorization Activity to do a similar function from within the e*Insight GUI. By replacing the Authorization Activity with a User Activity, the business process can now integrate with an external application such as a Web-based application. This application would allows users to login, view lists of the business process instances, make appropriate changes, and have those changes re-enter the e*Insight business model for continued processing.

While this example specifically uses a custom Web application, design and development of the external application is left up to the developer. The external application uses the Java APIs provided with the e*Insight engine (a complete description and list of these APIs is available in **"e*Insight User Activity API**

Methods" on page 299). 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.

8.8 Overview

The major steps in the implementation are:

1. Add the User Activity to the Business Process.
2. Use the e*Insight GUI to configure the e*Gate schema that supports e*Insight.
3. Add and configure the user-defined e*Gate components.
4. Run and test the scenario.

8.8.1 Case Study: Order Processing with User Activity

The Order Process example uses an external application to allow various users to authorize the quantity. The external application retrieves new orders, and for those orders that are in stock, the users evaluate all orders with quantity >100. Based on certain criteria, users either authorize the order for shipment or reject the order. The user of the external application could also perform other changes to the order based on capabilities developed into the external application.

Note: *The external application can also retrieve orders and deliver them to the correct user if the developer of the application uses specific API functions.*

8.9 Step 1: Update the ProcessOrder BP in e*Insight

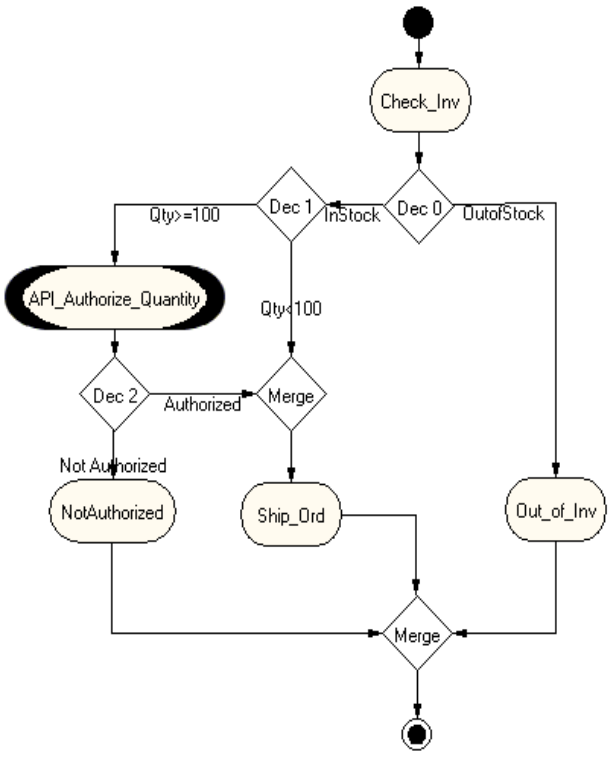
The following is a summary of the procedures for adding a User Activity in the e*Insight GUI.

1. Delete the **Authorize_Quantity** Activity.
2. Add a User Activity called **API_Authorize_Order**.
3. Add a Decision gate to receive the output from the User Activity.
4. Make the connections between the Decision gates, Merge gate, and Not Authorized Activity, as shown in **Figure 55 on page 107**.

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 58 ProcessOrder Business Process Model



8.10 Step 2: Configure the Integration Schema

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

You can send notifications through e*Gate to the external application but in this example, the same result is achieved using only the e*Insight engine. To use only the e*Insight engine, double click the User Activity, and make sure the "Send Notifications through e*Gate check box is not marked.

Note: *To send notifications through e*Gate, make sure the check box is marked, and configure your e*Way or BOB as required.*

8.11 Step 3: Run and Test the e*Insight scenario

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

8.11.1 Testing the User Activity

Use the following procedure to test the functionality of the User Activity.

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

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

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

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

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

Substitute the appropriate username and password for your installation.

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

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

The example instance should have the inventory quantity set to >100. This ensures that the business process instance follows the correct path to test the User Activity.

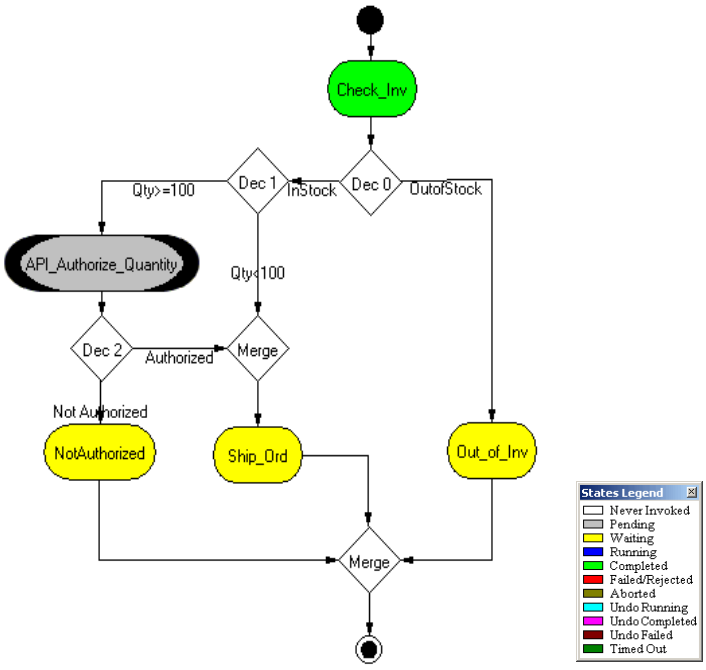
Note: *The change of the extension to "**.~in**" indicates that the data file has been picked up by the **START_BP** e*Way.*

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

The **API_Authorize_Quantity** should be in a pending state. This remains in a pending state until the e*Insight engine receives a message from an external application relating to this Business Process Instance.

Figure 59 shows how the BPI Diagram appears.

Figure 59 API_Authorize_Quantity Pending BPI Diagram



7 Start the external application. Return to the e*Insight GUI. The Business Process then completes using the route shown in **Figure 60 on page 115**.

The external application refers to any application developed by you to be inserted into the business process. A common example could be a method by which users can view lists of business instances assigned to them. These instances may require a simple review/authorize/reject process or, a more complex task using a custom-developed GUI interface to adjust or introduce attribute values. After the application completes its job, the updated business process instance information re-enters the User Activity and the business process continues.

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

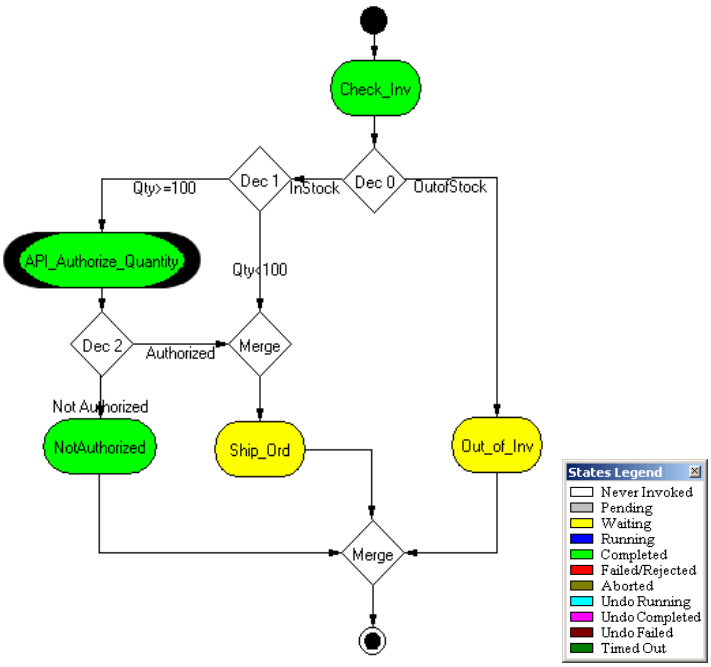
- **setUser**—This function passes the name of the user to the e*Insight engine if the external application uses login/password.
- **checkUserPrivileges**—This function establishes the security rights for the session using security levels as set within e*Insight.
- **getGlobalAttributeValue** and **getLocalAttributeValue**—This function receives attribute values for a business process instance for use within the external application.
- **getBusinessModelInstanceIds**—The external application uses this function to retrieve identification information for business model instances and then uses

this information to appropriately track instances when sending information back into the business process.

An example of a business model including a User Activity and an external application is shipped with e*Insight. This example can be installed by you and set up and used to demonstrate a business process using a Web-based application to allow users to log in, and view lists of instances that need to be reviewed and adjusted. The external application then sends the information back to e*Insight, thus allowing the business process to continue.

Note: *For information on installing the sample business model see the e*Insight Business Process Manager Installation Guide.*

Figure 60 User Activity - Item in stock



Chapter 9

e*Insight Sub-Process Implementation (eIJSchema)

This chapter discusses the steps involved to enhance the previous case study to include the Sub-Process.

The implementation starts with a local Sub-Process and then enhanced to demonstrate the use of the Remote Sub-Process and Dynamic Sub-Process.

This case study is a continuation of the previous example. See **"e*Insight Implementation (eIJSchema)" on page 68** 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 Payroll business process to replace the Calculate_Bonus activity with a Sub-Process.
3. Use the e*Insight GUI to configure the e*Gate schema that supports e*Insight.
4. Add and configure the user-defined e*Gate components.
5. Run and test the scenario.

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

9.2 Create the CalculateBonus BP in e*Insight

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

1. Create a business process named CalculateBonus.
2. Add the activities.
3. Make the connections between the activities.
4. Enable the business process version.

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

Use the diagram shown in Figure 61 and the following tables to create the BP in e*Insight.

Figure 61 CalculateBonus Business Process Model

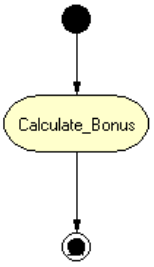


Table 14 BP Global Attributes

Attribute	Type	Data Direction	Default Value
Salary	Number	Input	
Grade	String	Input	
Bonus	Number	Output	0

Table 15 Activity Attributes

Activity	Attribute(s)	Input/Output
Calculate_Bonus	Salary	Input
	Grade	Input
	Bonus	Output

9.3 Configure the Integration Schema for CalculateBonus

The Calculate_Bonus activity uses the eX_Calculate_Bonus BOB that was created in **"Creating the eX_Calculate_Bonus BOB" on page 78**.

To configure the Calculate_Bonus Activity

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

9.4 Modify the Payroll BP in e*Insight

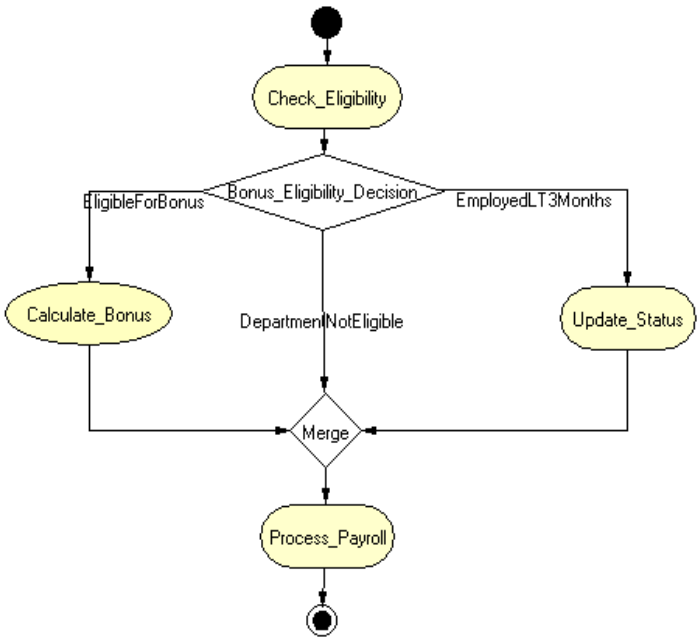
The following is a summary of the procedure for modifying the Payroll BP in the e*Insight GUI.

1 Delete the Calculate_Bonus activity.

2 Add the Calculate_Bonus Sub-Process.

3 Make the connections between the sub-process, decision gate and activity as shown in Figure 62.

Figure 62 Payroll BP with Sub-Process



4 Configure the Calculate_Bonus sub-process properties.

A Assign the CalculateBonus business process as the sub-process.

B Map the sub-process attributes to business process attributes as defined in Table 16.

Table 16 Sub-Process attribute mapping

Sub-Process Attributes	Business Process Attributes	Direction
Salary	Salary	Input
Grade	Grade	Input
Bonus	Bonus	Output

Note: *In this example, the sub-process attribute names and the business process attributes names are the same. This is not a requirement for the sub-process, however, it is a requirement for the dynamic sub-process.*

9.5 Configure the Integration Schema for Payroll

The schema created in **"e*Insight Implementation (eIJSchema)" on page 68** does not need to be modified for this example.

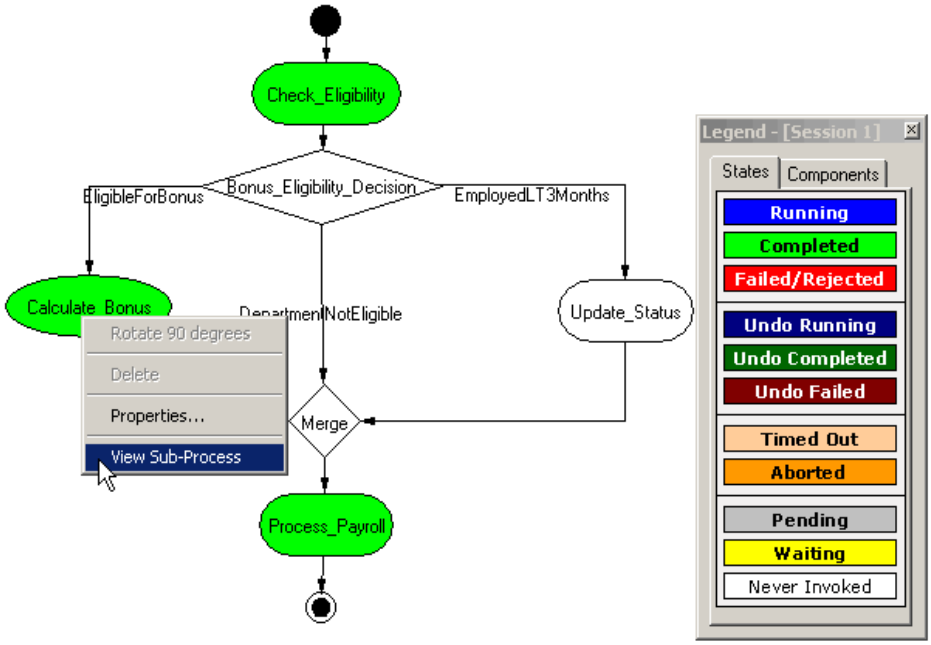
9.6 Run and Test the e*Insight scenario

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 89** to test your sub-process.

Note: *You can access the CalculateBonus business process directly from the Payroll business process. Right-click on the Sub-Process in the parent business process as shown in Figure 63.*

Figure 63 Accessing the CalculateBonus business 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 department that the employee works for. If the department is accounts, then the **Calculate_Bonus_accounts** Business Process is called, and if the department is marketing, then the **Calculate_Bonus_marketing** Business Process is called.

The major steps in the implementation are:

1. Create and configure two new business processes (**accounts** for accounts and **marketing** for marketing) to perform the activities required for calculating the bonus.
2. Update the Payroll business process to replace the Calculate_Bonus sub-process with a dynamic sub-process.
3. Use the e*Insight GUI to configure the e*Gate schema that supports e*Insight.
4. Add and configure the user-defined e*Gate components.
5. Run and test the scenario.

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

9.8 Create the accounts BP in e*Insight

The following is a summary of the procedure for creating a Business Process for California orders.

1 Export business process named CalculateBonus.

Note: *The exported file is used in* **"Create the marketing BP in e*Insight" on page 124**.

2 Rename the CalculateBonus business process **accounts**.

3 Select the enabled version, and from the **File** menu, select **Save as New Version**.

4 Rename the activity **Calculate_Bonus_accounts**.

5 Enable the business process version.

9.9 Configure the Integration Schema for accounts

To configure the Calculate_Bonus_accounts activity

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

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

3 Click **New**.

The Define Collaboration dialog appears.

4 Click **OK**.

5 Create eX_Calculate_Bonus_marketing.xpr.

The actual CRS is created later in **"To create eX_Calculate_Bonus_accounts.xpr" on page 123**.

6 Save the CRS.

7 Close the editor.

8 Click **Configure e*Gate**.

You may be required to log into e*Gate.

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

10 Close the **Calculate_Bonus_accounts** Activity properties.

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

Creating the CRS in e*Gate

Since the CRS for eX_Calculate_Bonus_accounts BOB is very similar to the eX_Calculate_Bonus CRS previously created you can copy this instead of creating a new script from scratch.

To create eX_Calculate_Bonus_accounts.xpr

1. Open the Collaboration Editor.
2. Open eX_Calculate_Bonus.xpr.
3. From the **File** menu, select **Save As**, and enter the name **eX_Calculate_Bonus_accounts.xpr**.

Important: *You should replace the existing file in* ***Collaboration_Rules\Calculate_Bonus_accounts****.*

4. Compile and then close the editor.

9.10 Create the marketing BP in e*Insight

The following is a summary of the procedure for creating a Business Process for marketing.

1. Import the business process that was exported in **"Create the accounts BP in e*Insight" on page 122**.
2. Rename the imported CalculateBonus business process **marketing**.
3. Rename the activity Calculate_Bonus_marketing.
4. Enable the business process version.

9.11 Configure the Integration Schema for marketing

To configure the Calculate_Bonus_marketing activity

1. In the e*Insight GUI, open the **Calculate_Bonus_marketing** activity properties.
2. On the **General** tab, select the **BOB** e*Gate module.
3. Click **New**.

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

   The actual CRS script is created later in **"To create eX_Calculate_Bonus_marketing.xpr" on page 125**.
6. Save the CRS.
7. Close the editor.

Note: *You do not need to compile the script yet.*

8. In the **Calculate_Bonus_marketing** activity properties, check the module name is eX_Calculate_Bonus_marketing, and modify if necessary.
9. Click **Configure e*Gate**.

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

Creating the CRS for eX_Calculate_Bonus_marketing in e*Gate

Since the CRS for eX_Calculate_Bonus_marketing BOB is very similar to the eX_Calculate_Bonus CRS previously created you can copy this instead of creating a new script from scratch.

To create eX_Calculate_Bonus_marketing.xpr

1. Open the Collaboration Editor.
2. Open eX_Calculate_Bonus_marketing.xpr.
3. From the **File** menu, select **Save As**, and enter the name **eX_Calculate_Bonus_marketing.xpr**.

Important: *You should replace the existing file in* ***Collaboration_Rules\Calculate_Bonus_marketing****.*

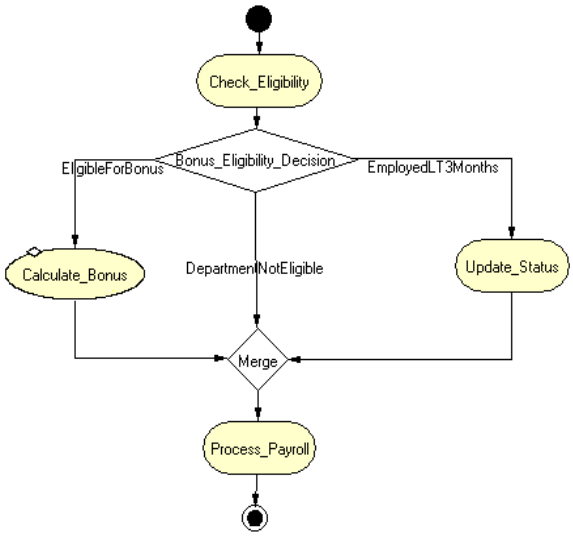
4. Update the script to define grade 1 as 5000.
5. Compile and then close the editor.

9.12 Modify the Payroll BP in e*Insight

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

1. Save the enabled **Payroll** business process as a new version.
2. Delete the **Calculate_Bonus** activity.
3. Add the **Calculate_Bonus** Dynamic Sub-Process.
4. Make the connections between the dynamic sub-process, decision gate and activity as shown in Figure 64.

Figure 64 Payroll BP with Dynamic Sub-Process



5. Configure the **Calculate_Bonus** dynamic sub-process properties.
   A. Ensure that the dynamic sub-process uses the **Incoming Global Attribute Value**.
   B. From the **Local Attributes** tab, edit the **subBPName**, and select **Department** from the Value drop down list.

9.13 Configure the Integration Schema for Payroll

The schema created in **"e*Insight Implementation (eIJSchema)" on page 68** 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 89** to test your sub-process.

Chapter 10

e*Insight Sub-Process Implementation (eISchema)

This chapter discusses the steps involved to enhance the previous case study to include the Sub-Process.

The implementation starts with a local Sub-Process and then enhanced to demonstrate the use of the Remote Sub-Process and Dynamic Sub-Process.

This case study is a continuation of the previous example. See **"e*Insight Implementation (eISchema)" on page 413** for the initial configuration instructions.

10.1 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.

10.2 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 65 and the following tables to create the BP in the e*Insight.

Figure 65 CheckInventory Business Process Model

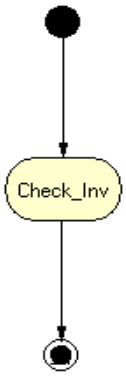


Table 17 BP Global Attributes

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

Table 18 Activity Attributes

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

10.3 Configure the Integration Schema for CheckInventory

The Check_Inv activity uses the eX_Check_Inv BOB that was created in **"Creating the eX_Check_Inv BOB" on page 421**.

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.

10.4 Modify the ProcessOrder BP in e*Insight

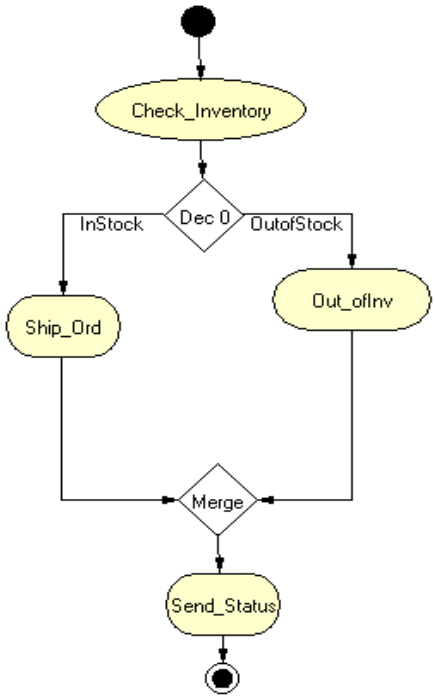
The following is a summary of the procedure for modifying the ProcessOrder BP in the e*Insight GUI.

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 66.

Figure 66 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 19.

Table 19 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*.

10.5 Configure the Integration Schema for ProcessOrder

The schema created in **"e*Insight Implementation (eISchema)" on page 413** does not need to be modified for this example.

10.6 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 441** to test your sub-process.

10.7 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.

10.8 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 137**.

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.

10.9 Configure the Integration Schema for CA

To configure the Check_Inv_CA activity using Monk

1. In the e*Insight GUI, open the **Check_Inv_CA** activity properties.
2. On the **General** tab, select the **BOB** e*Gate module.
3. Click **New**.

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

   The actual CRS script is created later in **"To create eX_Check_Inv_CA.tsc using Monk" on page 136**.
6. Save the CRS.
7. Close the editor.
8. In the **Check_Inv_CA** activity properties, check the module name is eX_Check_Inv_CA, and modify if necessary.
9. Click **Configure e*Gate Schema**.

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

To configure the Check_Inv_CA activity using Java

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

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

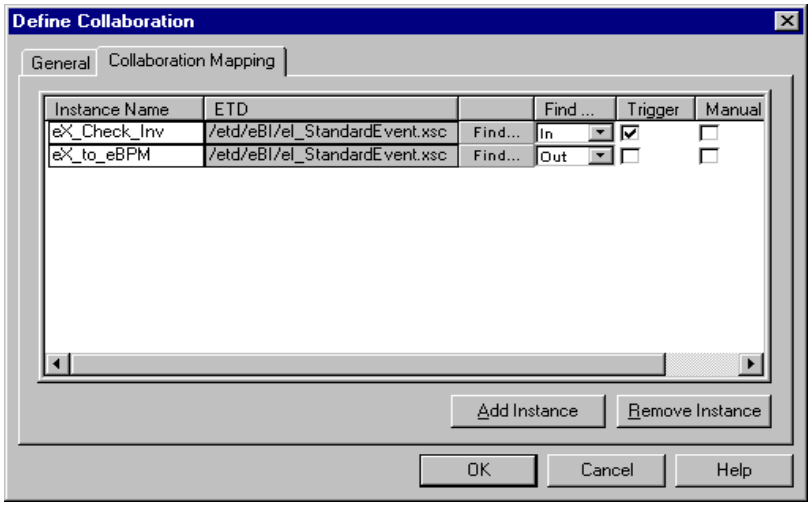
3 Click **New**.

The Define Collaboration dialog appears.

4 Select the **Define Mapping** tab.

5 Configure the instances as shown in Figure 67.

Figure 67 Define Mapping for eX_Check_Inv



6 Click **OK**.

7 Create eX_Check_Inv_CA.xsc.

The actual CRS is created later in **"To create eX_Check_Inv_CA.xsc using Java" on page 136**.

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.

10.10 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 134**.
2. Rename the imported CheckInventory business process **OR**.
3. Rename the activity Check_Inv_OR.
4. Enable the business process version.

10.11 Configure the Integration Schema for OR

To configure the Check_Inv_OR activity using Monk

1. In the e*Insight GUI, open the **Check_Inv_OR** activity properties.
2. On the **General** tab, select the **BOB** e*Gate module.
3. Click **New**.

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

   The actual CRS script is created later in **"To create eX_Check_Inv_OR.tsc using Monk" on page 139**.
6. Save the CRS.
7. Close the editor.
8. In the **Check_Inv_OR** activity properties, check the module name is eX_Check_Inv_OR, and modify if necessary.
9. Click **Configure e*Gate**.

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

To configure the Check_Inv_OR activity using Java

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

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

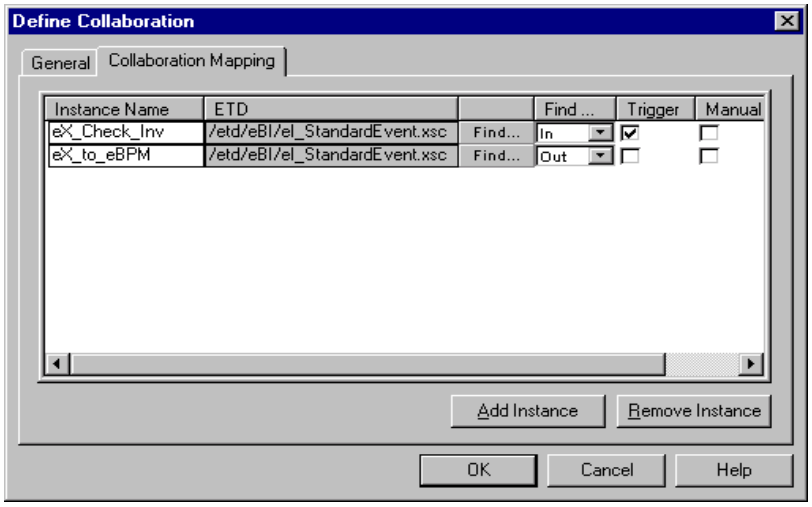
3 Click **New**.

The Define Collaboration dialog appears.

4 Select the **Define Mapping** tab.

5 Configure the instances as shown in Figure 68.

Figure 68 Define Mapping for eX_Check_Inv



6 Click **OK**.

7 Create eX_Check_Inv_OR.xsc.

The actual CRS is created later in **"To create eX_Check_Inv_OR.xsc using Java" on page 139**.

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.

10.12 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.

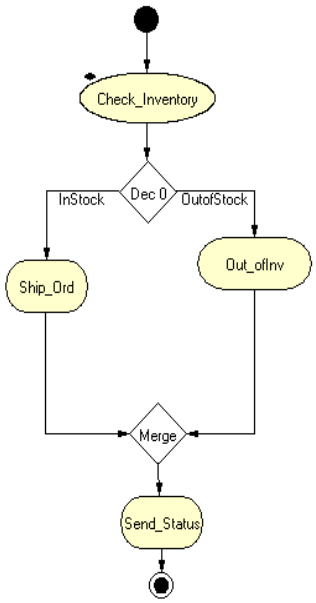
1 Save the enable ProcessOrder business process as a new version.

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 69.

Figure 69 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 19.

Table 20 Global attribute mapping

Global Attributes	Direction
Address_State	Input
Item_Number	Input
Order_Quantity	Input
In_Stock	Output
Order_Status	Output

C From the Local Attributes tab, edit **subBPName**, and select Address_State from the Value drop down list.

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

10.13 Configure the Integration Schema for ProcessOrder

The schema created in **"e*Insight Implementation (eISchema)" on page 413** does not need to be modified for this example.

10.14 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 441** to test your sub-process.

Chapter 11

e*Insight Remote Sub-Process Implementation

This chapter discusses the steps involved to configure the Remote Sub-Process. It covers the installation and configuration of Tomcat on single and multiple machines, and also works through a detailed implementation.

This case study is a continuation of a previous example. See either **"e*Insight Implementation (eIJSchema)" on page 68** or **"e*Insight Implementation (eISchema)" on page 413** for the initial configuration instructions.

11.1 Overview

The major topics covered in this chapter are:

- Installation and configuration of Tomcat
- Implementation steps

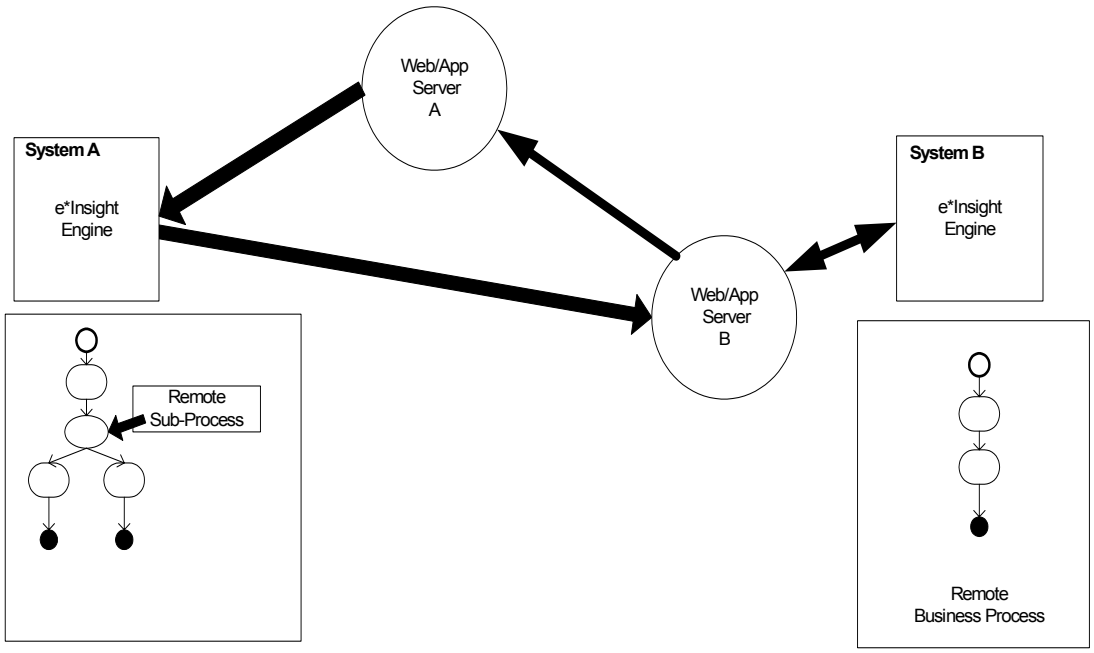
11.2 Overview of the Remote Sub-Process

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 145**.

Figure 70 Remote Sub-Process Overview



The Remote Sub-Process defined in this scenario uses Apache/Tomcat as the Web/Application Server. Tomcat is provided with e*Insight, and this needs to be installed and configured before using the Remote Sub-Process with SOAP.

11.3 Installation and Configuration of Tomcat

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

Table 21 Steps for Installation and Configuration of Tomcat

	Step	Procedure
1	Install Tomcat	**"Installing Tomcat" on page 143**
2	Configure Tomcat	**"Configuring Tomcat" on page 144**
3	Deploy the SOAP service	**"Deploying the SOAP Service" on page 144**

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 sub-folders.*

Configuring Tomcat

This section describes how to configure Tomcat once it has been installed. The procedure updates and copies files that are required for an e*Insight/Soap implementation.

To configure Tomcat

1 Open **<eInsight>\Soap\eInsight_tomcat.bat**. Locate the EINSIGHT variable and ensure that it is set to <eInsight>\Soap.

2 Copy the following files from <eInsight>\Soap.

- copy eInsight_startup.bat to <tomcat>\bin
- copy eInsight_tomcat.bat to <tomcat>\bin
- copy soap.war to <tomcat>\webapps

3 Edit EInsightBridge.properties to connect to correct database.

Note: *You should make sure that when you set parameter values, you do not add any trailing spaces, as these are interpreted as part of the parameter.*

4 Set system environment classpath c:\eInsight\soap.

A Go to Control Panel

B Select System

C Select Advanced tab

D Select Environment Variables

E Edit CLASSPATH on system variables

F Add "c:\eInsight\soap" to Variable Value to the front

5 From a command prompt, change directory to <tomcat> folder and run

```
bin\eInsight_startup.bat
```

Deploying the SOAP Service

The Tomcat Server has to be made aware of the SOAP service and the methods that it can use. This is achieved by deploying the SOAP service using the procedure described below.

To Deploy the SOAP Service

1 Open **<eInsight>\Soap\registerSoapService.cmd**.

2 Edit the following variables for your system:

```
SOAP_SERVICE
INTEGRATOR
TOMCAT_HOME
JAVA_HOME
```

3 Open **eInsightDeploymentDescripter.xml** in any text editor. Modify the following parameter:

```
id="<URN>"
```

where <URN> should be in the format **urn:<user defined name>**.

4 In a command prompt, change directory to <einsight>\soap.

5 Run **RegisterSoapService.cmd**.

Note: *Before you run registerSoapService.cmd, make sure your Apache Tomcat service is running.*

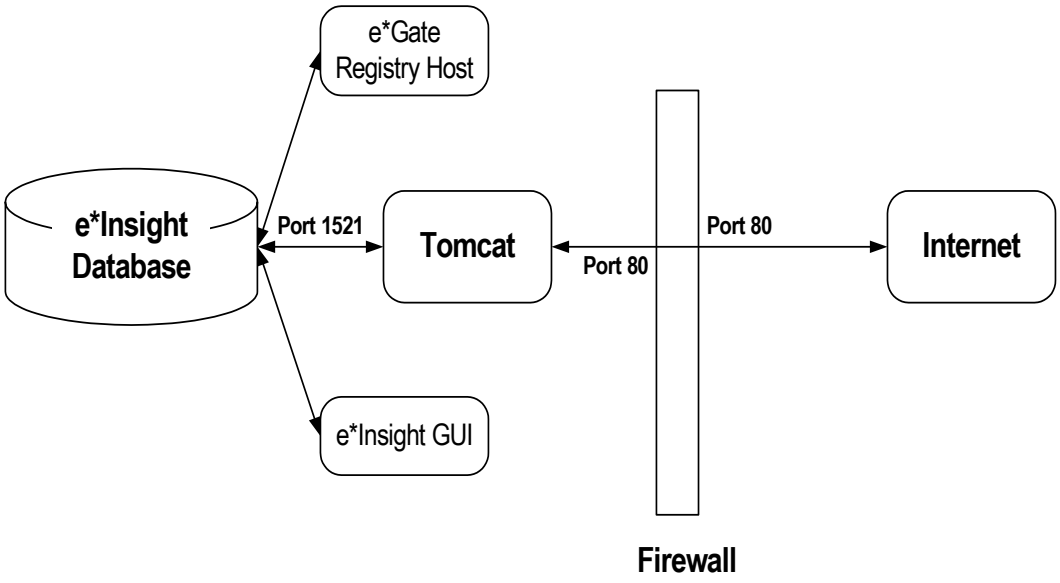
6 Start Internet Explorer and go to url: **http://<webhost>:8080/soap/admin/index.html**

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

11.4 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 71 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 71 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 144**.

11.5 Overview of the Remote Sub-Process Example (eIJSchema)

This case study is a continuation of the Payroll example. The procedure that checked the inventory is going to be moved to a remote system and SOAP messages is used to send the data to and from the remote process.

The Payroll example must be completed before the remote sub-process example. See **"e*Insight Implementation (eIJSchema)" on page 68** for the initial configuration instructions.

For this example, SystemA is the machine where the Payroll example was created. SystemB is the machine where the business process that calculates the bonus runs.

The major steps in the implementation are:

1 Configure Tomcat on both machines.

2 Create the CalculateBonus business process.

3 Create the e*Gate schema that supports the CalculateBonus business process.

4 Update the Payroll business process.

5 Run and test the scenario.

11.6 Install and configure Tomcat

Tomcat needs to be installed on both SystemA and SystemB. In a real scenario, it may be necessary to run Tomcat on a different machine to e*Insight, but in this example we assume that Tomcat and e*Insight run on the same machine.

Follow the instructions in **"Installation and Configuration of Tomcat" on page 143** for both machines. Define the URN's as follows:

SystemA (Original system)	urn:Payroll
SystemB (Added for the Remote Sub-Process example)	urn:CalculateBonus

11.7 Create the CalculateBonus BP in e*Insight

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

1. Create a business process named CalculateBonus.
2. Add the activities.
3. Make the connections between the activities.
4. Create and assign the global attributes.

Use the diagram shown in Figure 74 and the following tables to create the BP in e*Insight.

Figure 72 CalculateBonus Business Process Model

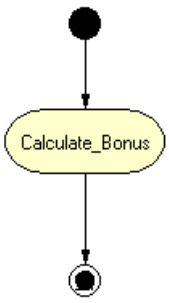


Table 22 BP Global Attributes

Attribute	Type	Data Direction	Default Value
Salary	Number	Input	
Grade	String	Input	
Bonus	Number	Output	0

Table 23 Activity Attributes

Activity	Attribute(s)	Input/Output
Calculate_Bonus	Salary	Input
	Grade	Input
	Bonus	Output

5 In the business process properties, select **Send Business Process Done Event**.

6 Enable the business process version.

To configure the Partner Information

1 From the **Options** menu, select **Define Information for Partners**.

2 Enter **UUID**.

This is a unique name that identifies your e*Insight database to other Trading Partners. For example, **SystemB**.

3 Enter the URN.

This corresponds to the URN setup in the SOAP client. For example, **urn:CalculateBonus**.

4 Enter the URL.

This is the location of the SOAP client. By default this is **http://localhost:8080/soap/servlet/rpcrouter**.

5 Click **OK**.

11.8 Configure the Integration Schema for CalculateBonus

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

11.8.1 Create the CalculateBonus Schema

Use the following procedure to create a copy of the eIJSchema:

1 From the e*Insight GUI **File** menu, select **New e*Gate Schema**.

2 Enter or select a **Registry Host** on which to create the schema.

3 Enter a **Username** and **Password** that is valid on the Registry Host.

4 From the **Based on** list, select **eIJSchema (Java)**.

5 In the **Name** box, enter **CalculateBonus**.

6 Click **OK**.

11.8.2 Configure the CalculateBonus Schema

Since the module for the Calculate_Bonus activity has been created in a previous case study, you only need to import the module into the CalculateBonus schema and configure the e*Insight engine and JMS server. These three procedures are described below.

Create the Calculate_Bonus activity BOB

1. In the **SystemA** e*Gate Schema Designer, open the **Payroll** schema.
2. Select the eX_Calculate_Bonus **BOB**.
3. From the File menu select **Export Module Definitions to File**.

   The **Select Archive File** dialog appears.
4. Enter eX_Calculate_Bonus, and then click **OK**.

   The file **eX_Calculate_Bonus.zip** is created.
5. Log into the **SystemB** e*Gate Schema Designer and open the **CalculateBonus** schema.
6. From the File menu select **Import Module Definitions from File**.
7. Locate the file created in step 4.
8. Click **OK**.

Edit the eIcp_eInsightEngine Connection Configuration File

Most of the parameter settings in the eIcp_eInsightEngine connection's configuration file should not be changed. **"Using XA" on page 32** discusses the parameters that may need to be changed depending on the implementation. Use the e*Way Editor and the information in **"Using XA" on page 32** to make the required changes for the CalculateBonus example.

Configure the JMS Connection

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

To configure the JMS connection

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

11.9 Modify the Payroll BP in e*Insight

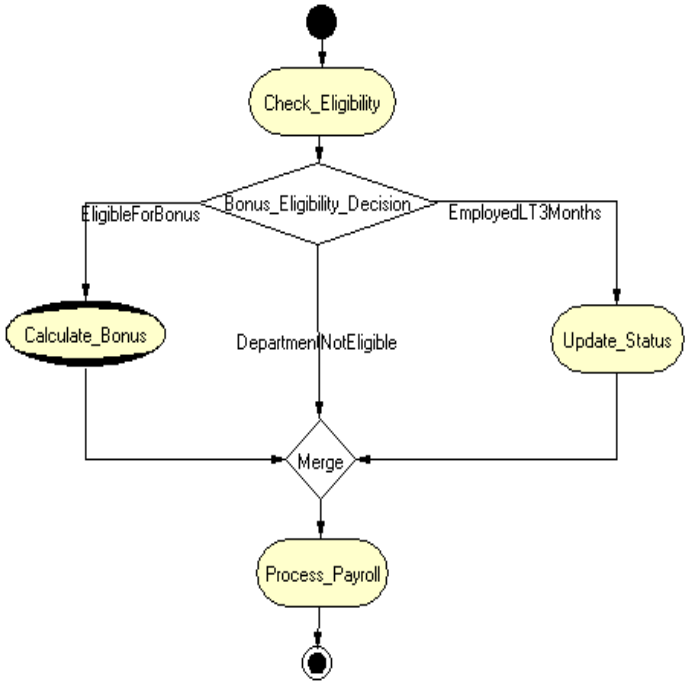
The following is a summary of the procedure for modifying the Payroll BP in the e*Insight GUI.

1 Delete the Calculate_Bonus activity.

2 Add the Calculate_Bonus Remote Sub-Process.

3 Make the connections between the remote sub-process, decision gate and activity as shown in Figure 75.

Figure 73 Payroll BP with Remote Sub-Process



4 Assign **EligibleForBonus** as the default link in **Bonus_Eligibility_Decision**.

5 Configure the Partner Information

A From the **Options** menu, select **Define Information for Partners**.

B Enter **UUID**.
This is a unique name that identifies your e*Insight database to other Trading Partners. For example, **SystemA**.

C Enter the URN.
This corresponds to the URN setup in the SOAP client. For example, **urn:Payroll**.

D Enter the URL.
This is the location of the SOAP client. By default this is **http://localhost:8080/soap/servlet/rpcrouter**.

E Click **OK**.

6 Configure the **Calculate_Bonus** remote sub-process properties.

A Enter the relevant information for the remote e*Insight database including URL, URN, user name and password. Use the following values:

URL	http://<SystemB>:8080/soap/servlet/rpcrouter
URN	urn:CalculateBonus
User Name	Anonymous
Password	

Note: *To use an Anonymous login you must create a user called Anonymous on the remote system. Although e*Insight Administrator requires that a password is defined to create the user, this does not need to be supplied when connecting via SOAP.*

Click **Connect**, and select the **CalculateBonus** business process from the Remote Sub-Process drop-down list

B Map Attributes as shown in Table 27.

Table 24 Sub-Process attribute mapping

Sub-Process Attributes	Business Process Attributes	Direction
Salary	Salary	Input
Grade	Grade	Input
Bonus	Bonus	Output

7 Enable the business process version.

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

11.10 Configure the Integration Schema for Payroll

The schema created in **"e*Insight Implementation (eISchema)" on page 413** does not need to be modified for this example.

11.11 Run and Test the e*Insight scenario

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

To test the Remote Sub-Process scenario

1. Start Tomcat on both machines.
2. Start the control broker for Payroll, and check all components are running.
3. Start the control broker for CalculateBonus, and check all components are running.
4. Rename the data file.

To monitor a remote business process

1. Select the remote business process, and then select the down arrow.
2. If prompted, enter a user name and password provided by the partner.

11.12 Overview of the Remote Sub-Process Example (eISchema)

This case study is a continuation of the Process Order example. The procedure that checked the inventory is going to be moved to a remote system and SOAP messages are used to send the data to and from the remote process.

The Process Order example must be completed before the remote sub-process example. See **"e*Insight Implementation (eISchema)" on page 413** for the initial configuration instructions.

For this example, SystemA is the machine where the Process Order example was created. SystemB is the machine where the business process that checks the inventory runs.

The major steps in the implementation are:

1. Configure Tomcat on both machines.
2. Create the CheckInventory business process.
3. Create the e*Gate schema that supports the CheckInventory business process.
4. Update the ProcessOrder business process.
5. Run and test the scenario.

11.13 Install and configure Tomcat

Tomcat needs to be installed on both SystemA and SystemB. In a real scenario, it may be necessary to run Tomcat on a different machine to e*Insight, but in this example we assume that Tomcat and e*Insight run on the same machine.

Follow the instructions in **"Installation and Configuration of Tomcat" on page 143** for both machines. Define the URN's as follows:

SystemA	urn:ProcessOrder
SystemB	urn:CheckInventory

11.14 Create the CheckInventory BP in e*Insight

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

1. Create a business process named CheckInventory.
2. Add the activities.
3. Make the connections between the activities.
4. Create and assign the global attributes.

Use the diagram shown in Figure 74 and the following tables to create the BP in e*Insight.

Figure 74 CheckInventory Business Process Model

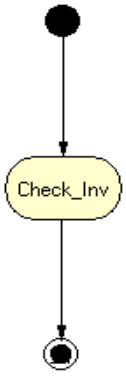


Table 25 BP Global Attributes

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

Table 26 Activity Attributes

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

5 In the business process properties, select **Send Business Process Done Event**.

6 Enable the business process version.

To configure the Partner Information

1 From the **Options** menu, select **Define Information for Partners**.

2 Enter **UUID**.

This is a unique name that identifies your e*Insight database to other Trading Partners. For example, **SystemB**.

3 Enter the URN.

This corresponds to the URN setup in the SOAP client. For example, **urn:CheckInventory**.

4 Enter the URL.

This is the location of the SOAP client. By default this is **http://localhost:8080/soap/servlet/rpcrouter**.

5 Click **OK**.

11.15 Configure the Integration Schema for CheckInventory

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

11.15.1 Create the CheckInventory Schema

Use the following procedure to create a copy of the eISchema:

1 Open the eISchema in the e*Gate Schema Designer.

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**.

11.15.2 Configure the e*Insight engine

Most of the parameter settings in the eX_eBPM engine's configuration file should not be changed. Use the e*Way Editor and the information in **"Configuring the e*Insight Engine" on page 369** to make the required changes for the CheckInventory example.

11.15.3 Create the Check_Inv activity BOB

1 In the **SystemA** e*Gate Schema Designer, 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 Schema Designer 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**.

11.16 Modify the ProcessOrder BP in e*Insight

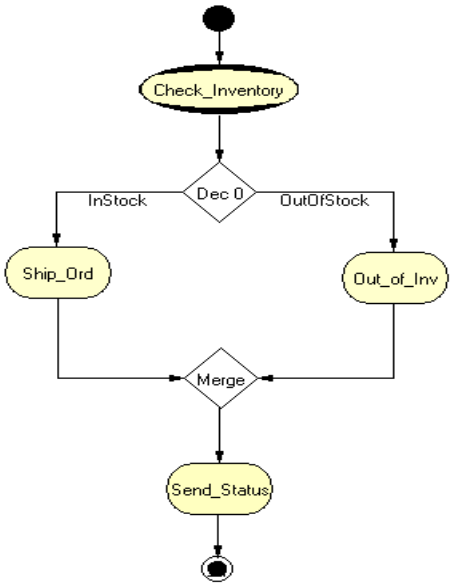
The following is a summary of the procedure for modifying the ProcessOrder BP in the e*Insight GUI.

1 Delete the Check_Inv activity.

2 Add the Check_Inventory Remote Sub-Process.

3 Make the connections between the remote sub-process, decision gate and activity as shown in Figure 75.

Figure 75 ProcessOrder BP with Remote Sub-Process



4 Configure the Partner Information.

A From the **Options** menu, select **Define Information for Partners**.

B Enter **UUID**.
This is a unique name that identifies your e*Insight database to other Trading Partners. For example, **SystemA**.

C Enter the URN.
This corresponds to the URN setup in the SOAP client. For example, **urn:ProcessOrder**.

D Enter the URL.
This is the location of the SOAP client. By default this is **http://localhost:8080/soap/servlet/rpcrouter**.

E Click **OK**.

5 Configure the **Check_Inventory** remote sub-process properties.

A Enter the relevant information for the remote e*Insight database including URL, URN, user name and password. Use the following values:

URL	http://SystemB:8080/soap/servlet/rpcrouter
URN	urn:CheckInventory
User Name	Anonymous
Password	

Note: *To use an Anonymous login you must create a user called Anonymous on the remote system. Although e*Insight 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 27.

Table 27 Sub-Process attribute mapping

Sub-Process Attributes	Business Process Attributes	Direction
Item_Number	Item_Number	Input
Order_Quantity	Order_Quantity	Input
In_Stock	In_Stock	Output
Order_Status	Order_Status	Output

6 Enable the business process version.

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

11.17 Configure the Integration Schema for ProcessOrder

The schema created in **"e*Insight Implementation (eISchema)" on page 413** does not need to be modified for this example.

11.18 Run and Test the e*Insight scenario

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

In-Stock Processing

Use the following procedure to test the functionality of the example for an item that is in stock.

To test the Remote Sub-Process scenario

1. Start Tomcat on both machines.
2. Start the control broker for ProcessOrder, and check all components are running.
3. Start the control broker for CheckInventory, and check all components are running.
4. Rename the data file.

To monitor a remote business process

1. Select the remote business process, and then select the down arrow.
2. If prompted, enter a user name and password provided by the partner.

Chapter 12

Active and Passive Modes

This chapter discusses the difference between implementing an actively controlled and passively controlled activity. This chapter also describes a business process that is first created with all activities using active mode. Then it is updated so one activity uses passive mode.

12.1 Overview

The difference between running an activity in active and passive mode is how the activity is started. In active mode e*Insight sends a message to e*Gate to start the e*Gate component that executes that step in the business process. e*Insight requires a message from e*Gate to determine that the activity has completed processing. If you choose this option, you can manually repair and restart failed activities.

In passive mode, e*Insight does not send a message to start the e*Gate component that executes that step in the business process, e*Gate must be configured to do this. e*Insight requires a message from e*Gate to determine that the activity has completed processing. If you choose this option, you cannot manually repair or restart failed activities.

12.1.1 Case Study

The case study discussed in this chapter illustrates a simplified implementation of order processing. In this case, e*Insight receives an incoming order as a delimited text file. Once e*Insight has received the order the customer is billed and then the order is shipped. See Figure 78 for the Business Process diagram.

12.1.2 Case Study - Active Control Mode

Figure 76 shows the components involved in the business process implementation when both activities are running in an active mode. Below is a description of how the data flows between these components for an item that is shipped successfully.

Figure 76 e*Insight Data Flow Diagram

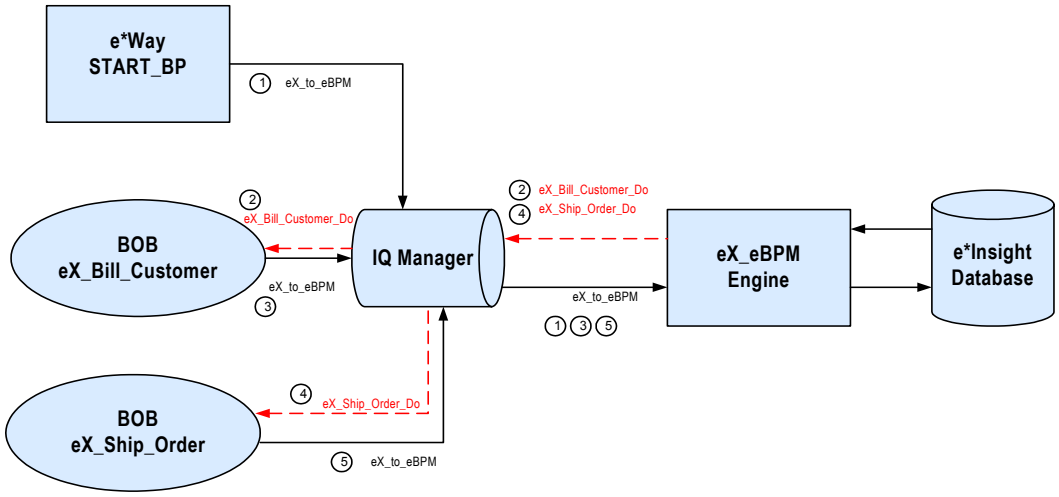


Figure 76 data flow description

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

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

③ When the **Bill_Customer** activity is finished, the **eX_Bill_Customer** BOB publishes a "Done" Event using the **eX_to_eBPM** Event Type.

④ The e*Insight engine publishes a "Do" Event (**eX_Ship_Order_Do**) corresponding to the **Ship_Order** activity in the business process. The **eX_Ship_Order** BOB retrieves this Event and uses the information it contains to ship the order to the customer.

⑤ When the **Ship_Order** activity is finished, the **eX_Ship_Order** BOB publishes a "Done" Event indicating that the order has been shipped.

12.1.3 Case Study - Passive Control Mode

We are now going to change the configuration of the Ship_Order activity to use passive control, that is e*Gate controls the Activity rather than e*Insight. The e*Gate configuration is modified so the Bill_Customer collaboration publishes eX_Ship_Order_Do in addition to returning a "Done" message to the e*Insight engine.

Figure 77 e*Insight Data Flow Diagram

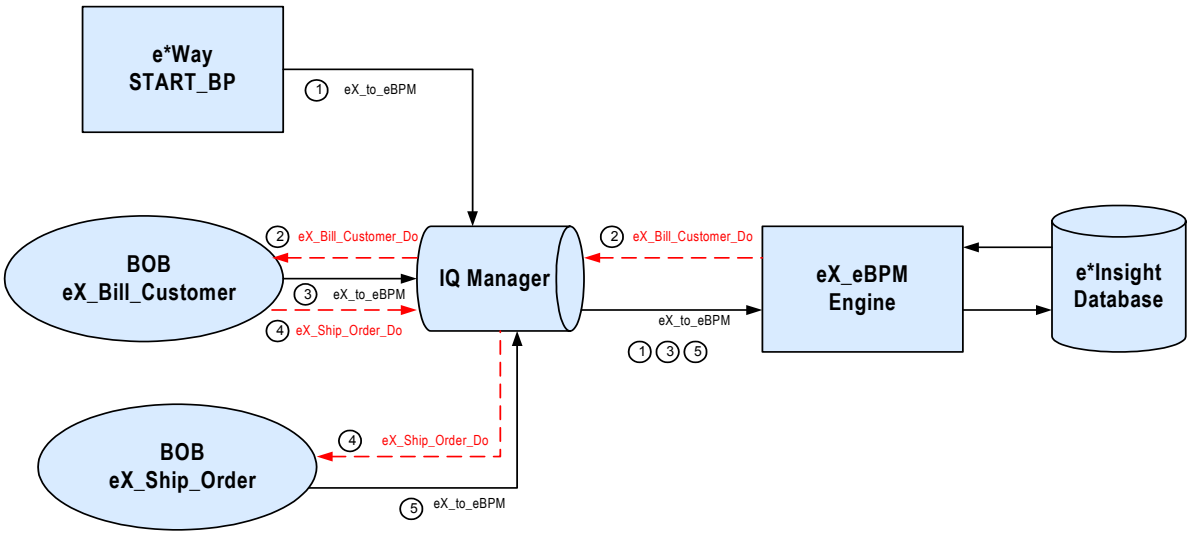


Figure 76 data flow description

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

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

③ When the **Bill_Customer** activity is finished, the **eX_Bill_Customer** BOB publishes a "Done" Event using the **eX_to_eBPM** Event Type.

④ The **Bill_Customer** activity also publishes a "Do" Event (**eX_Ship_Order_Do**) corresponding to the **Ship_Order** activity in the business process. The **eX_Ship_Order** BOB retrieves this Event and uses the information it contains to ship the order to the customer.

⑤ When the **Ship_Order** activity is finished, the **eX_Ship_Order** BOB publishes a "Done" Event indicating that the order has been shipped.

The major steps in the implementation are:

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

This chapter shows how to use these steps to set it up. Since the configuration for steps 2 to 4 is different for eIJSchema and eISchema, there is a separate section for each schema type.

Follow the steps below to configure the schema using the active mode for all activities:

- Step 1: **"Create the Order BP in e*Insight" on page 163**
- Step 2, 3, and 4:
  - **"Configure the Integration Schema (eIJSchema)" on page 164** or
  - **"Configure the Integration Schema (eISchema)" on page 173**
- Step 5: **"Run and Test the e*Insight scenario" on page 178**

Once the schema is working in active mode, then follow the steps below to modify one activity to use passive mode using eIJSchema:

A **"Modify the Order BP in e*Insight (eIJSchema)" on page 179**

B **"Modify User-defined e*Gate Components (eIJSchema)" on page 179**

C **"Run and Test the e*Insight scenario" on page 182**

Use the following steps if you are using eISchema:

A **"Modify the Order BP in e*Insight (eISchema)" on page 183**

B **"Modify User-defined e*Gate Components (eISchema)" on page 183**

C **"Run and Test the e*Insight scenario" on page 184**

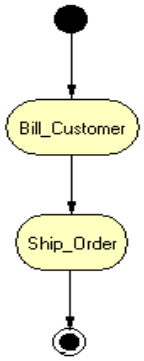
12.2 Create the Order BP in e*Insight

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

1. Add the Activities.
2. Make the connections between the Activities.
3. Add all the global attributes.
4. Assign global attributes to Activities.
5. Configure the properties for the activities.

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

Figure 78 Order Process



The case study is implemented using both active and passive control for the Ship_Order Activity.

The following Attributes should be configured for both the active and passive examples.

Table 28 BP Global Attributes

Attribute	Type	Data Direction	Default Value
Customer_Name	String	Input	
Item_Number	String	Input	
Order_Quantity	String	Input	
Order_Status	String	Internal	Received

Table 29 Activity Attributes

Activity	Attribute(s)	Input/Output
Bill_Customer	Customer_Name	Input
	Order_Status	Output

Table 29 Activity Attributes

Activity	Attribute(s)	Input/Output
Ship_Order	Cust_Name	Input
	Item_Number	Input
	Order_Status	Output

12.3 Configure the Integration Schema (eIJSchema)

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

To create a copy of eIJSchema

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

Integration Schema Activity Components Summary

Use the information in Table 33 to configure the e*Gate schema that supports the example.

Table 30 Integration Schema Activity Components

Name	Type	Participating Host	Active/Passive	Manual Restart	TimeOut
eX_Bill_Customer	BOB	localhost	Active	Yes	Not used
eX_Ship_Order	BOB	localhost	Active	Yes	Not used

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

Creating the eX_Bill_Customer BOB

The Bill_Customer translation implements the logic associated with billing the customer. In this simple example, this component sets the "Order_Status" attribute value.

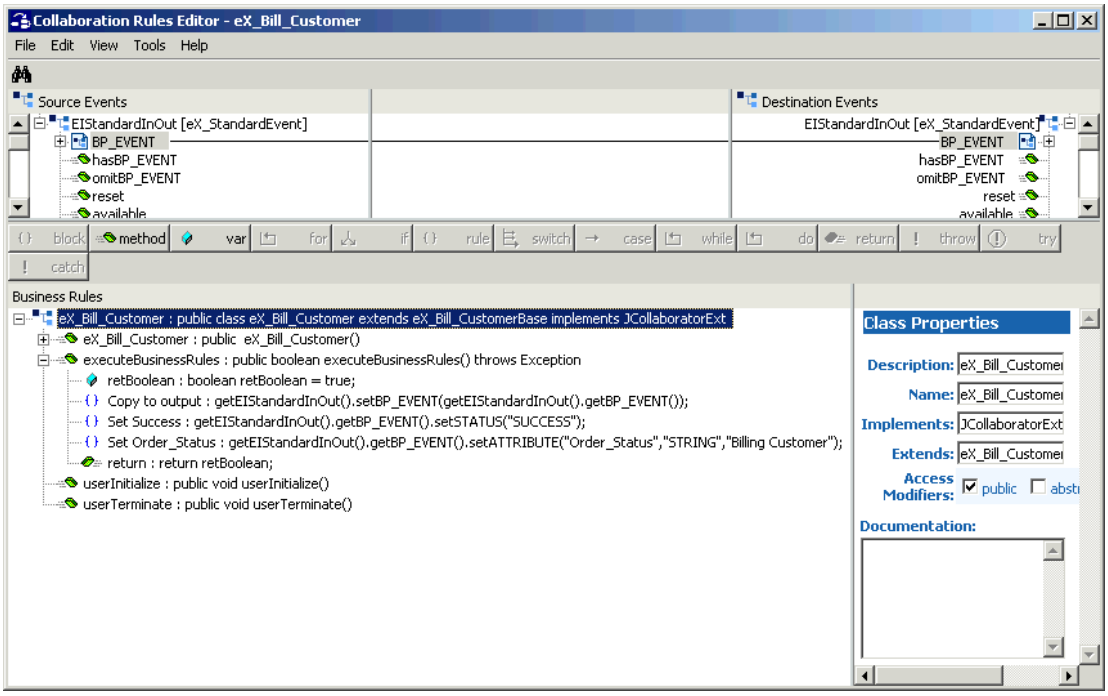
To configure the Bill_Customer activity

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

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

   Figure 79 shows the eX_Bill_Customer CRS used in the example.

Figure 79 eX_Bill_Customer.xpr CRS



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

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

Creating the eX_Ship_Order BOB

The Ship_Order translation implements the logic associated with shipping to the customer. In this simple example, this component sets the "Order_Status" attribute value.

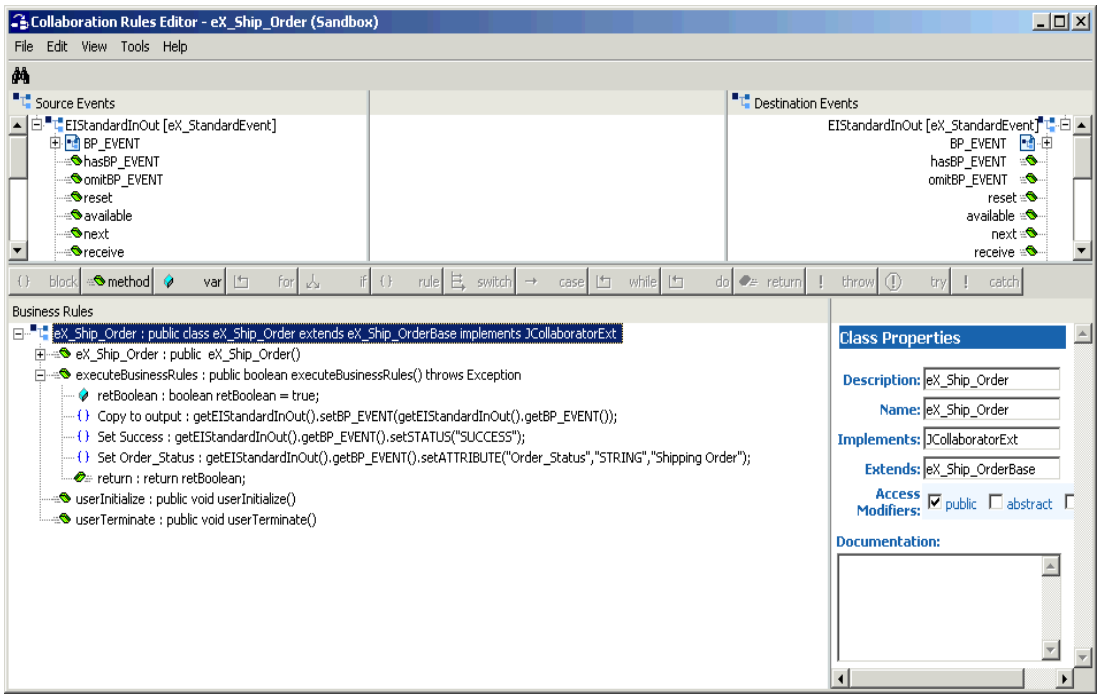
To configure the Ship_Order activity

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

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

   Figure 79 shows the eX_Ship_Order CRS used in the example.

Figure 80 eX_Ship_Order.xpr CRS



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

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

12.4 Configure the e*Insight Engine (eIJSchema)

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

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

Edit the eIcp_eInsightEngine Connection Configuration File

Most of the parameter settings in the eIcp_eInsightEngine connection's configuration file should not be changed. **"Using XA" on page 32** discusses the parameters that may need to be changed depending on the implementation. Use the e*Way Editor and the information in **"Using XA" on page 32** to make the required changes for the example.

12.4.1 Configure the JMS Connection

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

To configure the JMS connection

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

12.5 Configure User-defined e*Gate Components (eIJSchema)

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

The Order example uses a file e*Way to start the business process and BOBs to run all the other activities.

Configuration Order for the User-defined Components

1 Add and configure the **START_BP** e*Way.

2 Configure the Collaborations for the activity components running as BOBs.

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

12.5.1 Configure the START_BP e*Way

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

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

Table 31 shows the steps to configure the **START_BP** e*Way.

Table 31 Configuration steps for the START_BP e*Way

	Step	Section
1	Add the e*Way and create the e*Way configuration file	**"Step 1: Create the START_BP e*Way" on page 168**
2	Create the Input ETD	**"Step 2: Create the Input ETD" on page 169**
3	Create the START_BP Collaboration Rules script (CRS)	**"Step 3: Create the START_BP Collaboration" on page 170**
4	Configure the Collaboration in the GUI	**"Step 4: Configure the Collaboration in the GUI" on page 172**

Step 1: Create the START_BP e*Way

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

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

Table 32 Start e*Way Parameters

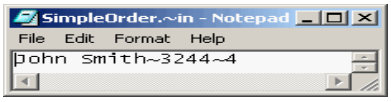
Screen	Parameter	Setting
General Settings	(All)	(Default)
Outbound (send) settings	(All)	(Default)
Poller (inbound) settings	PollDirectory	<eGate>\client\data\Order
	(All others)	(Default)
Performance Testing	(All)	(Default)

Step 2: Create the Input ETD

The input ETD is based on the format of the input data. The Order example uses a delimited text file (**SimpleOrder.~in**) that contains the data needed to process the order.

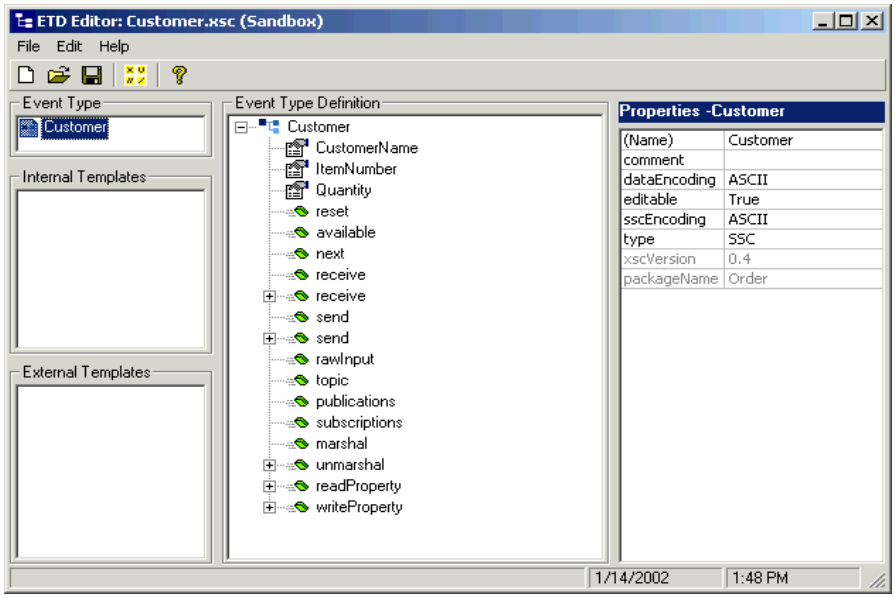
The input data file used in this example is shown in Figure 81. Place this data file at the directory location **c:\eGate\client\data\Order**.

Figure 81 Input Text File (SimpleOrder.~in)



Using the ETD Editor and the input data as a guide, create an ETD like the one shown in Figure 87. Set the global delimiter to a ~ character. For more information on using the ETD Editor see the ETD Editor's online help.

Figure 82 Input ETD: Customer.xsc (Java)



Step 3: Create the START_BP Collaboration

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

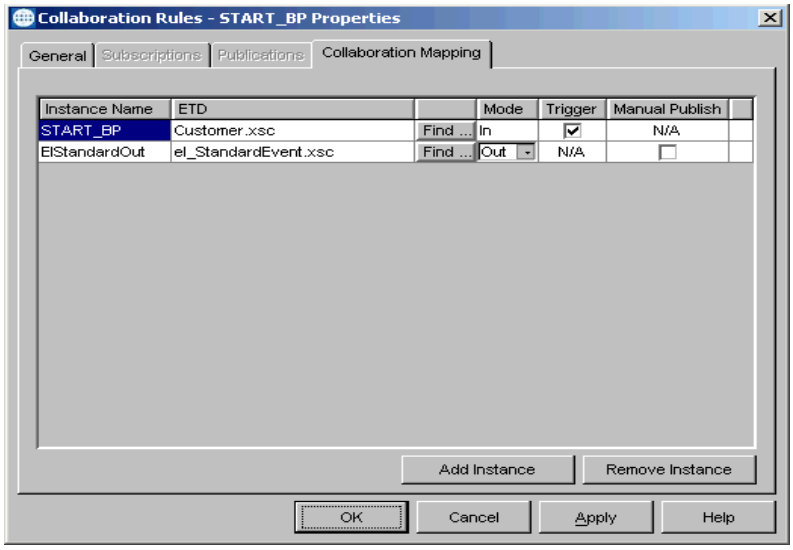
- Put the data into e*Insight ETD (**eI_StandardEvent.xsc)** format.
- Populate the Event with the information the e*Insight engine needs to start a BPI.

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

1 Create a Collaboration Rule, **START_BP**, that uses the Java service.

2 Configure the Collaboration Mapping tab, as shown in Figure 83.

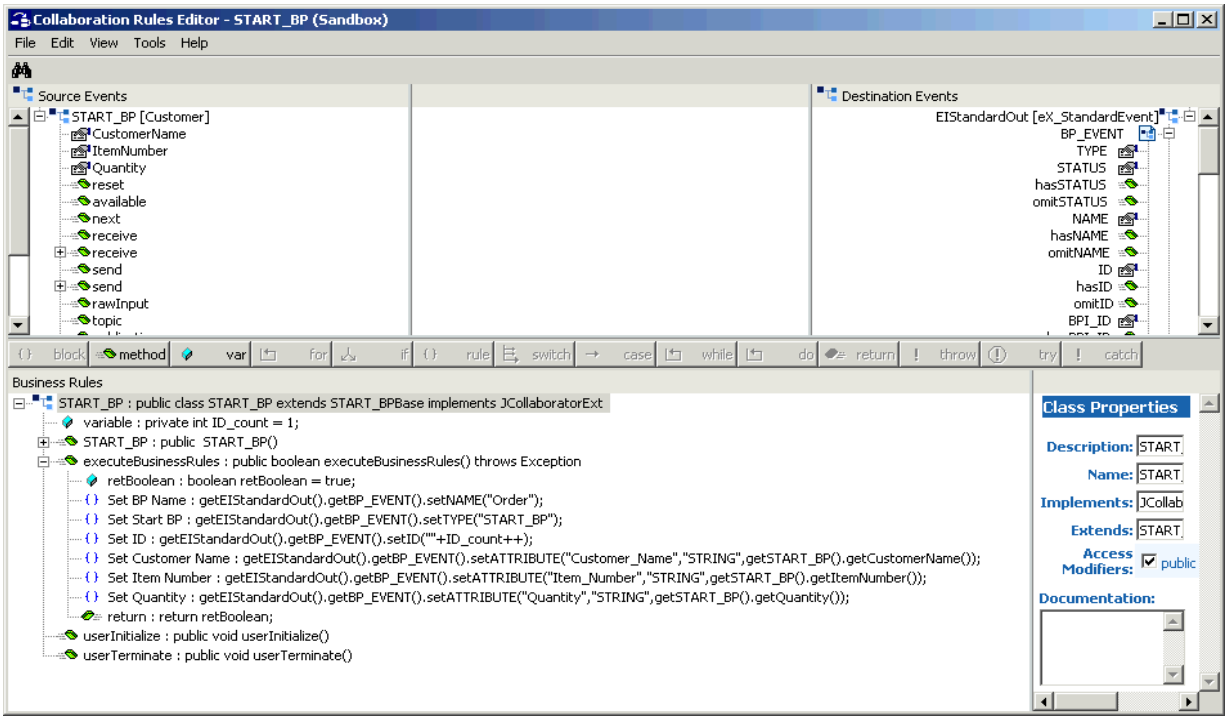
Figure 83 Start_BP Properties, Collaboration Mapping Tab



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

4 Click **New** to create a new CRS, as show in Figure 84.

Figure 84 START_BP CRS

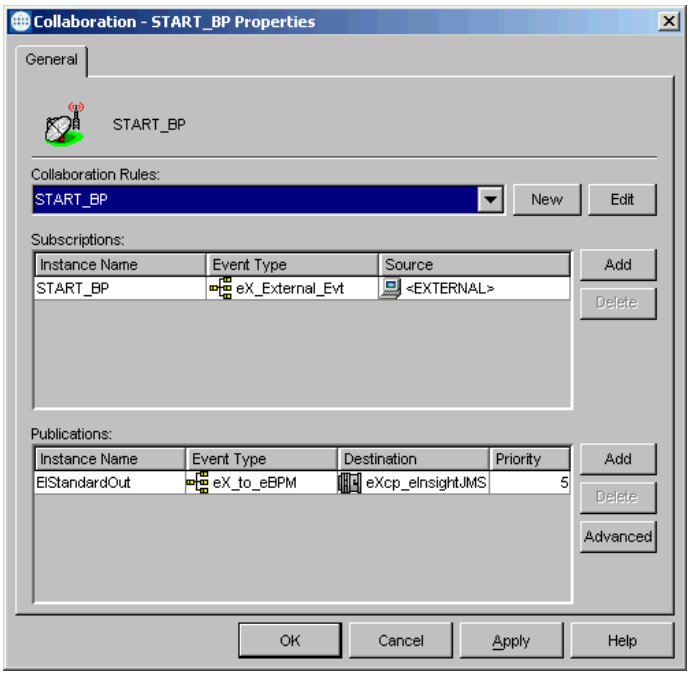


Step 4: Configure the Collaboration in the GUI

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 Schema Designer.

1 Create a Collaboration for the **Start_BP** e*Way configured as shown in Figure 85.

Figure 85 Start_BP Collaboration



12.6 Configure the Integration Schema (eISchema)

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

Important: *Before you begin to make changes to the e*Gate Registry, make a copy of the e*Insight schema. See* **"Copy the e*Insight Schema" on page 391** *for instructions on how to do this.*

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

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

Integration Schema Activity Components Summary

Use the information in Table 33 to configure the e*Gate schema that supports the example.

Table 33 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*.

12.7 Configure the e*Insight Engine (eISchema)

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

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

Edit the eX_eBPM Engine's Configuration File

Most of the parameter settings in the eX_eBPM engine's configuration file should not be changed. Use the e*Way Editor and the information in **"Configuring the e*Insight Engine" on page 369** to make the required changes for the Order example.

12.8 Configure User-defined e*Gate Components (eISchema)

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

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

Configuration Order for the User-defined Components

1. Add and configure the **START_BP** e*Way.
2. Configure the Collaborations for the activity components running as BOBs.

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

12.8.1 Configure the START_BP e*Way

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

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

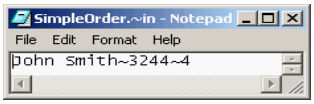
1. Create the Input ETD
2. Create the **START_BP** Collaboration Rules script (CRS)
3. Add the e*Way and create the e*Way configuration file
4. Configure the Collaboration in the GUI

Step 1: Create the Input ETD

The input ETD is based on the format of the input data. The Order example uses a delimited text file (**SimpleOrder.~in**) that contains the data needed to process the order.

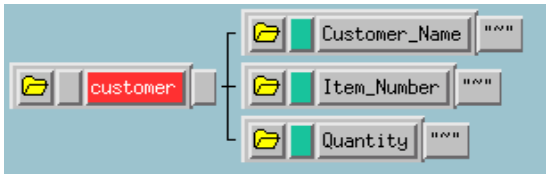
The input data file used in this example is shown in Figure 86. Place this data file at the directory location **c:\eGate\client\data\Order**.

Figure 86 Input Text File (SimpleOrder.~in)



Using the ETD Editor and the input data as a guide, create an ETD like the one shown in Figure 87. For more information on using the ETD Editor see the ETD Editor's online help.

Figure 87 Input ETD: Customer.ssc



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

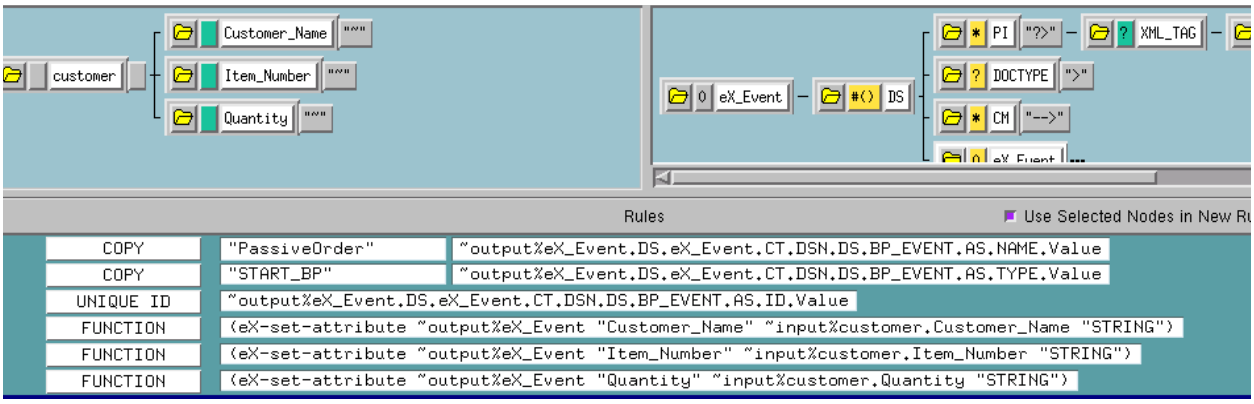
The Collaboration that sends the Event that starts the BPI must do two things:

- 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 88 shows the **START_BP** CRS used in the Order example:

Figure 88 START_BP CRS



Step 3: Add the e*Way and Create the e*Way Configuration File

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

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

Table 34 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 Schema Designer.

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.

12.8.2 Configure the Activity BOBs

You must complete these two tasks in order to configure the Activity BOBs:

- Set up the activity Collaborations in the e*Gate GUI.
- Create the corresponding CRS used by the activity Collaborations.

Note: *Any time you create e*Ways, you must configure those e*Ways before e*Insight can communicate with external systems or components.*

Create the Activity BOB CRSs

You must create the CRSs associated with the activity components. What these programs do varies depending on the type of component and the business logic they must implement. Nevertheless, they all must take the information provided to them by the e*Insight engine, process it, and return the appropriate response to the e*Insight engine. This includes setting the values of any output attributes and the BP_EVENT status node to "SUCCESS" or "FAILURE" depending on whether the activity completes successfully or not.

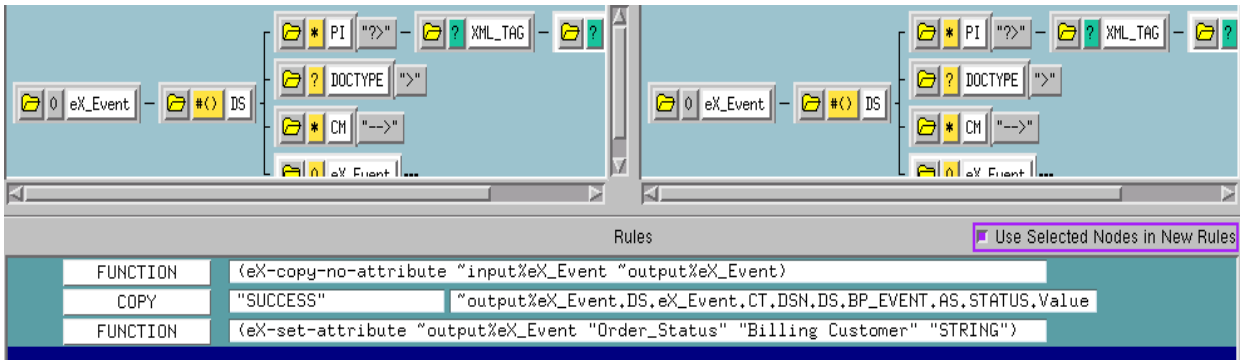
The CRSs associated with the BOBs used in this example are described in the following sections along with a screen capture showing the actual Monk Collaboration Rules Script used.

Bill Customer CRS

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

Figure 89, shows the **Billing_Customer.tsc** CRS used in the example.

Figure 89 Billing_Customer.tsc CRS

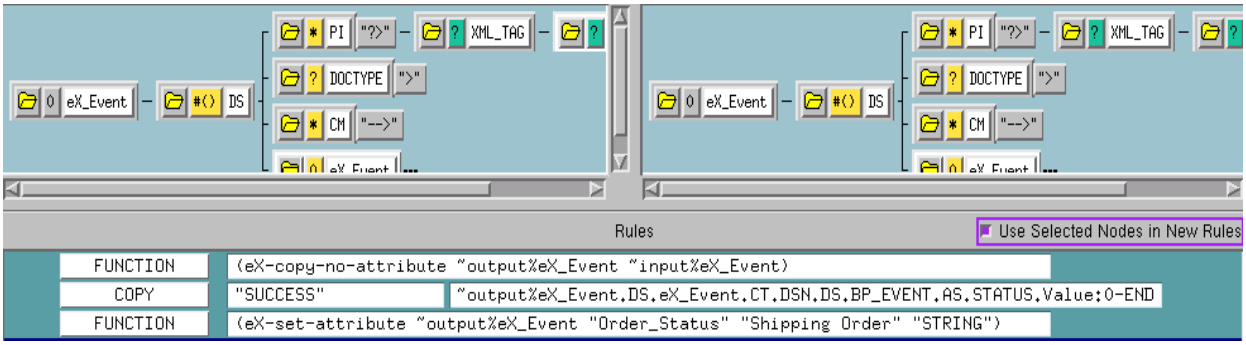


Ship_Order CRS

The Ship_Order translation simulates the activity of sending out an item that is in stock. It sets the value of the Order_Status attribute to a short message indicating that the order has been sent and returns “SUCCESS” to the e*Insight engine.

Figure 90 shows the Ship_Order CRS used in the example.

Figure 90 Ship_Order.tsc CRS



Configure the Activity BOB Collaborations in the Schema Designer

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.

12.9 Run and Test the e*Insight scenario

Once the schema has been set up in e*Gate you can run the scenario. Use the following procedure to test the functionality of the example.

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

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

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

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

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

Substitute the appropriate username and password for your installation.

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

5 Navigate to the location for the input data file, **SimpleOrder.~in**, shown in **Figure 86 on page 175** (**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.

12.10 Case Study - Passive Control Mode

We are now going to change the configuration of the Ship_Order activity to use passive control, that is e*Gate controls the Activity rather than e*Insight. The e*Gate configuration is modified so the Bill_Customer collaboration publishes eX_Ship_Order_Do in addition to returning a "Done" message to the e*Insight engine. The Ship_Order CRS is also modified to ensure that the "Done" message contains the correct information.

12.11 Passive Control Mode (eIJSchema)

12.11.1 Modify the Order BP in e*Insight (eIJSchema)

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

1. Save the business process as a new version.
2. Change the Ship_Order activity properties to use passive control.

12.11.2 Modify User-defined e*Gate Components (eIJSchema)

Configuration Order for the User-defined Components

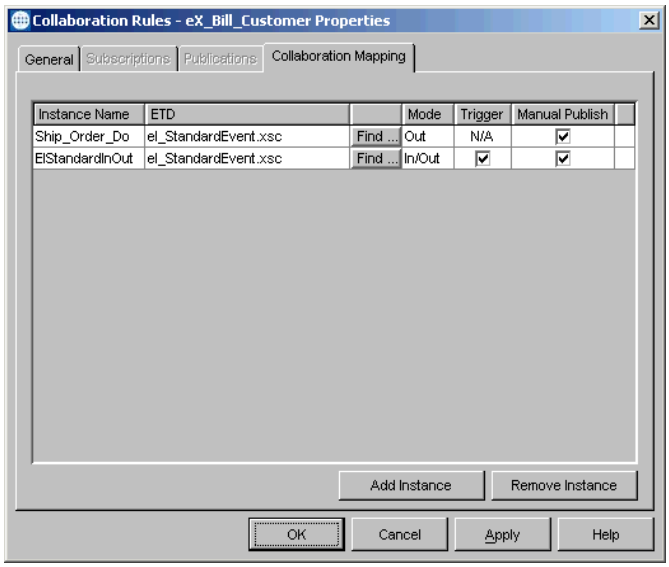
1. Modify the Bill_Customer CRS.
2. Modify the Ship_Order CRS.
3. Modify the Bill_Customer Collaboration Rule and Collaboration.

Configure the Bill_Customer Collaboration Rule and Collaboration

1. Update the Bill_Customer Collaboration Rule with the following Collaboration Mapping.

Note: *Both instances are now configured to publish the event manually.*

Figure 91 eX_Bill_Customer Collaboration Mapping



2 Update the Bill_Customer Collaboration to additionally publish a **Ship_Order_Do** instance, Event Type **eX_Ship_Order_Do**, to the destination **eIcr_eInsightJMS**.

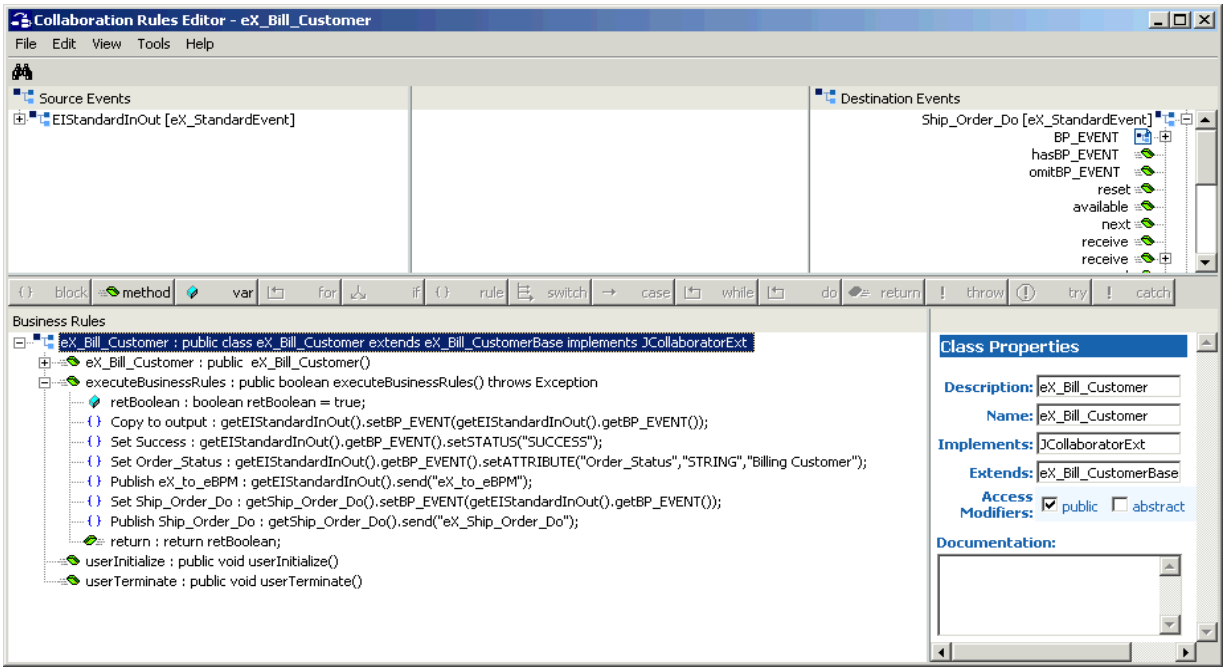
Bill Customer CRS (eIJSchema)

The CRS is already defined to set the value of the Order_Status attribute to "Billing Customer" and send a "SUCCESS" Event back to the e*Insight, indicating that the activity has completed successfully.

The script is modified to manually publish eX_to_eBPM, and to additionally publish **eX_Ship_Order_Do**.

Figure 92, shows the **Billing_Customer.xpr** CRS used in the example.

Figure 92 Bill_Customer.xpr CRS



Ship_Order CRS

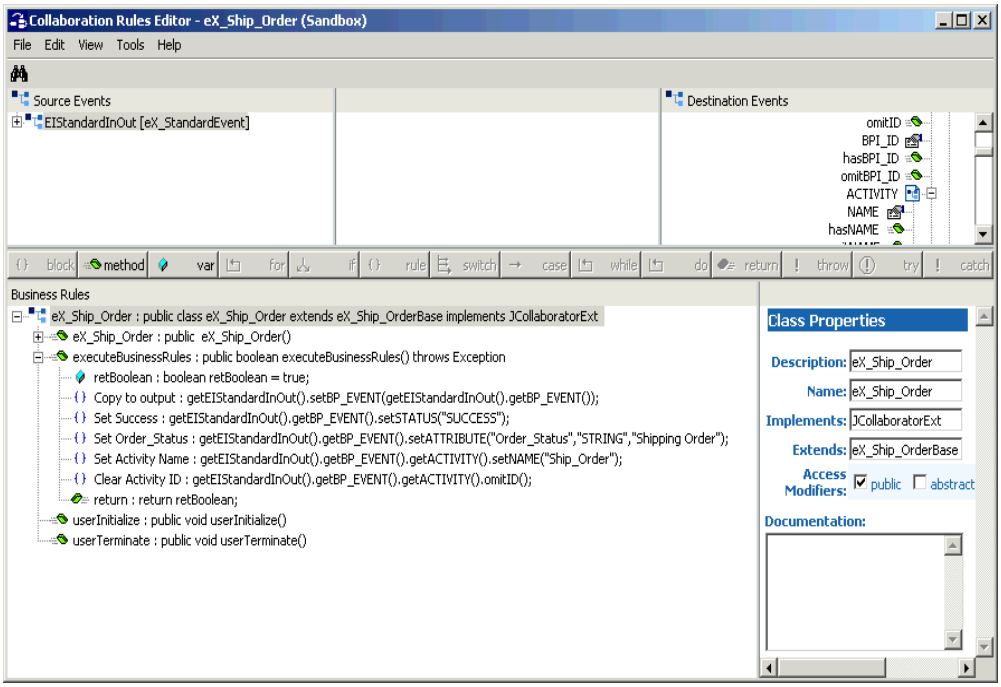
The Ship_Order translation simulates the activity of sending out an item that is in stock. It sets the value of the Order_Status attribute to a short message indicating that the order has been sent and returns "SUCCESS" to the e*Insight engine.

In the active example, the event is sent from the e*Insight engine with information pertaining to the Ship_Order activity. In the passive example, the event was originally created by the e*Insight engine for the Bill_Customer activity, and the eX_Bill_Customer Collaboration is configured to send a "Do" event. This "Do" event contains information pertaining to the Bill_Customer activity, rather than the Ship_Order activity, so the following values need to be updated:

- Activity Name — should be set to "Ship_Order"
- Activity ID — should be deleted

Figure 90 shows the Ship_Order CRS used in the example.

Figure 93 Ship_Order.xpr CRS



12.11.3 Run and Test the e*Insight scenario

Run the Schema again as described in **"Run and Test the e*Insight scenario" on page 178**.

12.12 Passive Control Mode (eISchema)

This section describes the steps for updating a schema based on eISchema (Classic). For information on eIJSchema, see **"Passive Control Mode (eIJSchema)" on page 179**.

12.12.1 Modify the Order BP in e*Insight (eISchema)

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

1. Save the business process as a new version.
2. Change the Ship_Order activity properties to use passive control.

12.12.2 Modify User-defined e*Gate Components (eISchema)

Configuration Order for the User-defined Components

1. Modify the Bill_Customer CRS.
2. Modify the Ship_Order CRS.
3. Modify the Bill_Customer Collaboration Rule and Collaboration.

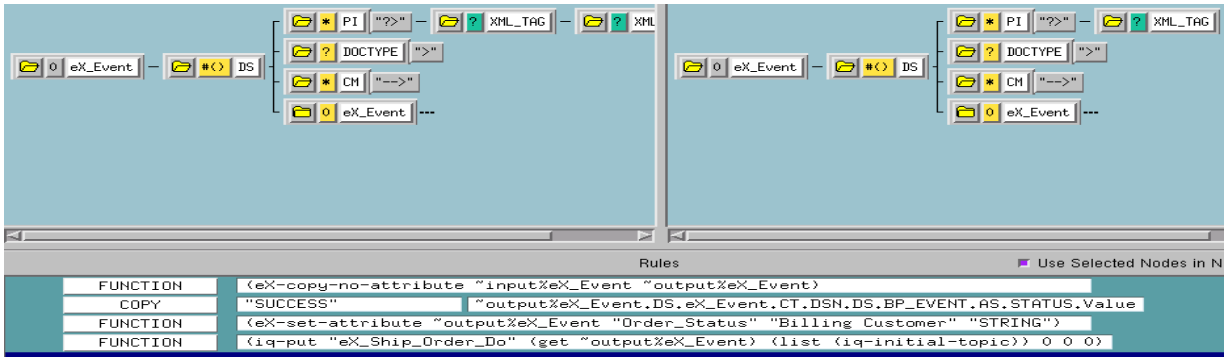
Bill Customer CRS

The CRS is already defined to set the value of the Order_Status attribute to "Billing Customer" and send a "SUCCESS" Event back to the e*Insight, indicating that the activity has completed successfully.

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

Figure 92, shows the **Billing_Customer.tsc** CRS used in the example.

Figure 94 Billing_Customer.tsc CRS



Ship_Order CRS

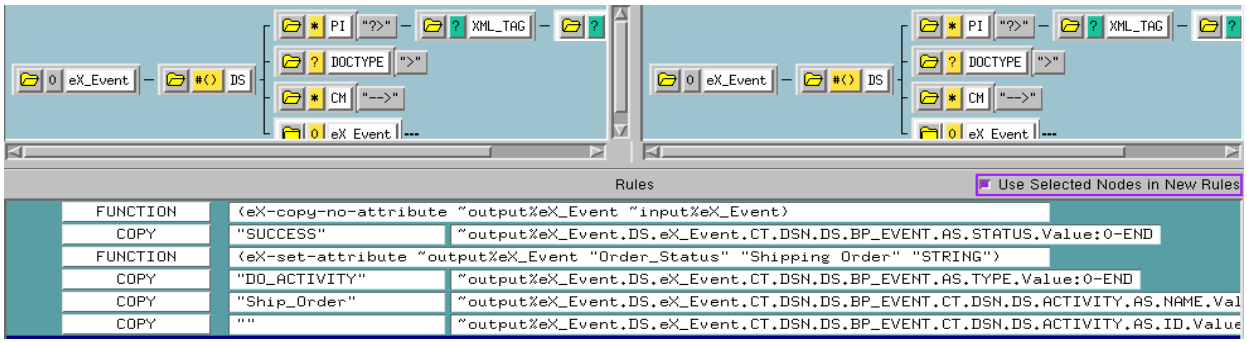
The Ship_Order translation simulates the activity of sending out an item that is in stock. It sets the value of the Order_Status attribute to a short message indicating that the order has been sent and returns "SUCCESS" to the e*Insight engine.

In the active example, the event is sent from the e*Insight engine with information pertaining to the Ship_Order activity. In the passive example, the event was originally created by the e*Insight engine for the Bill_Customer activity, and the eX_Bill_Customer Collaboration is configured to send a “Do” event. This “Do” event contains information pertaining to the Bill_Customer activity, rather than the Ship_Order activity, so the following values need to be updated:

- Activity Name — should be set to “Ship_Order”
- Activity ID — should be deleted

Figure 90 shows the Ship_Order CRS used in the example.

Figure 95 Ship_Order.tsc CRS



Configure the Bill_Customer Collaboration Rule and Collaboration

1. Update the Bill_Customer Collaboration Rule to additionally publish **eX_Ship_Order_Do**. This should not be selected as the default publication.
2. Update the Bill_Customer Collaboration to additionally publish **eX_Ship_Order_Do** to the **eX_eBPM** IQ.

12.12.3 Run and Test the e*Insight scenario

Run the Schema again as described in **“Run and Test the e*Insight scenario” on page 178**.

Chapter 13

e*Insight Performance

The purpose of this chapter is to describe methods that can be used to improve performance. The chapter is divided into three sections; the first describes the performance enhancements that can be made to a schema based on eIJSchema (Java). The second describes the performance enhancements that can be made to a schema based on eISchema (Classic). The last section describes performance enhancements that are not schema specific.

13.1 Performance Improvements Using eIJSchema

The purpose of this section is to describe methods that can be used to improve performance of the eIJSchema.

- Instance caching
- Using multiple engines
- Using instance caching with multiple engines
- Using binary XML
- Configuring e*Insight to ignore e*Xchange ETD
- Changing the Event Type "get" Interval

13.1.1 Instance Caching

Instance Caching is the most efficient way to process Business Process Instances. Using instance caching keeps a cache of the instance information throughout the life span of the Business Process Instance. If instance caching is not used the instance information is retrieved from the database instead.

To configure the engine to use instance caching

1. In the **e*Way Connections** folder, open the engine's e*Way connector properties.
2. Click **Edit** to open the engine's configuration file.
3. In the **eBPM Setting** section, set **Instance Caching** to **YES**.

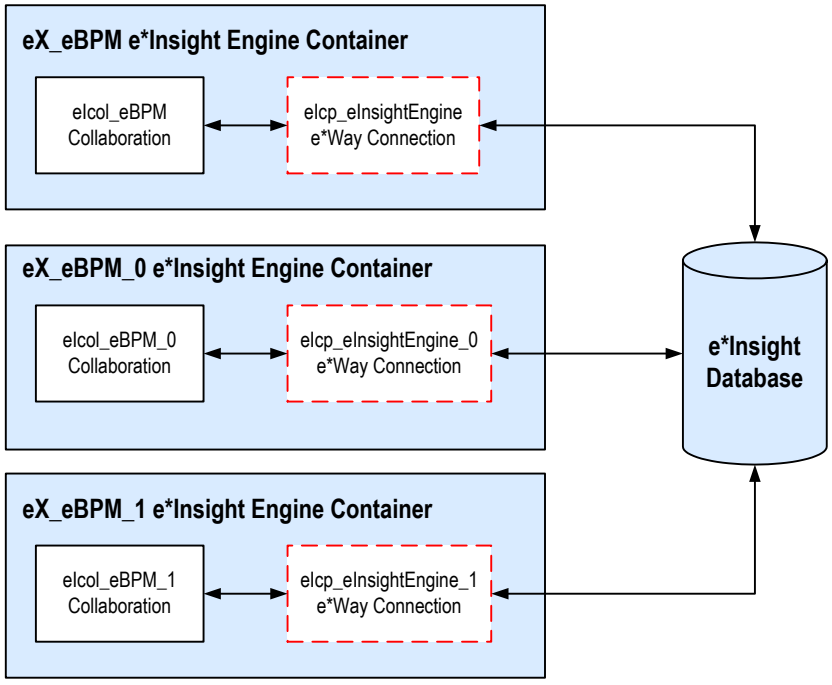
13.1.2 Using Multiple e*Insight Engines

An e*Insight engine is comprised of a specially configured Collaboration (eIcol_eBPM) and the e*Insight e*Way Connection (eIcp_eInsightEngine). The e*Insight engine runs within a Multi-Mode e*Way (eX_eBPM), which is referred to as the e*Insight Engine container. You can use multiple e*Insight engines to increase performance. This section describes how to add and configure these. You can use instance caching to further improve performance. See **"e*Insight Engine Affinity (eIJSchema)" on page 189** for more information.

When you add e*Insight engines, you can either add them to an existing e*Insight engine container or create additional e*Insight engine containers. **Figure 96** shows a scenario where three e*Insight engine containers are used, each with a single engine.

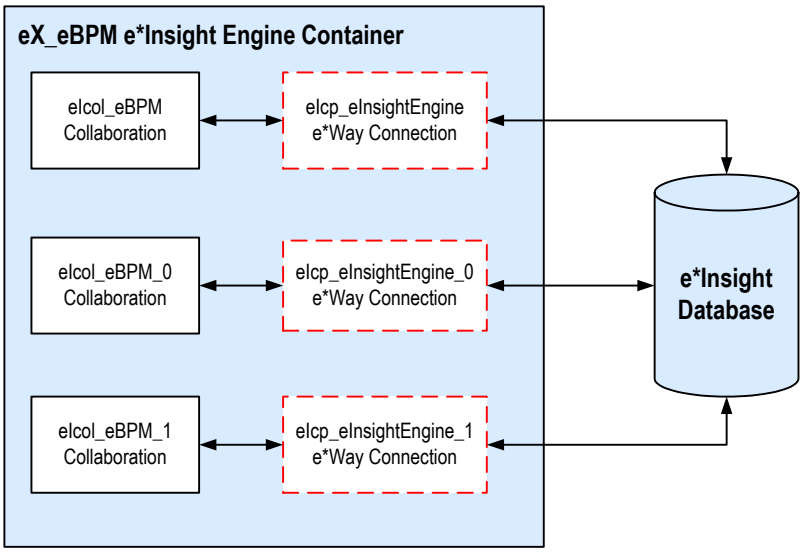
Important: *A single e*Insight e*Way connection cannot be used by multiple Collaborations.*

Figure 96 Multiple Engine Containers with a Single Engine
Needs changing!!



Alternatively, you can have multiple e*Insight engines in a single e*Insight engine container. See Figure 97.

Figure 97 A Single Engine Container with Multiple Engines



You can either create a new engine from scratch, or copy the engine container, Collaboration, Collaboration Rules, and engine connection.

To create a new e*Insight engine container

1. Add a new e*Insight engine in the participating host.
2. In the engine properties, **Executable file** section, click **Find**.
3. Browse for **stceway.exe** and click **Select**.
4. Clear the engine's configuration file.

Note: *You can create a new e*Insight engine container by copying the* ***eX_eBPM*** *e*Insight engine that is provided in the eIJSchema.*

To add and configure an e*Insight e*Way Connection

1. Select the **e*Way Connections** folder and create a new e*Way Connection.
2. Open the Connection properties and set the e*Way Connection Type to e*Insight Engine.
3. Create and configure a new Configuration File. For information on the settings, see **"Using XA" on page 32**.

To configure an e*Insight engine container for a new e*Way Connection

1. Create a new e*Way Connection. See **"To add and configure an e*Insight e*Way Connection" on page 187**.
2. Add a Collaboration Rule and configure it as shown in Figure 98.

Figure 98 e*Insight Engine General Tab

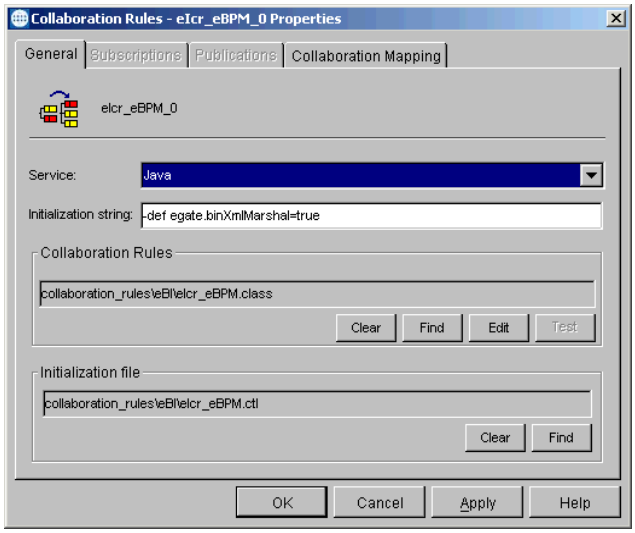
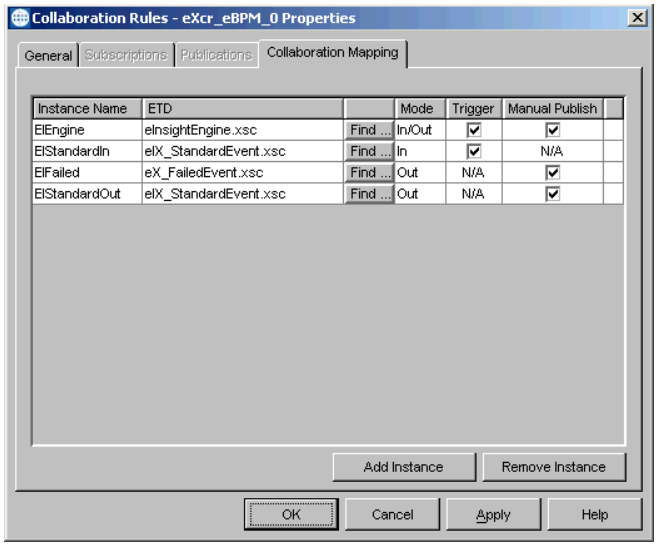


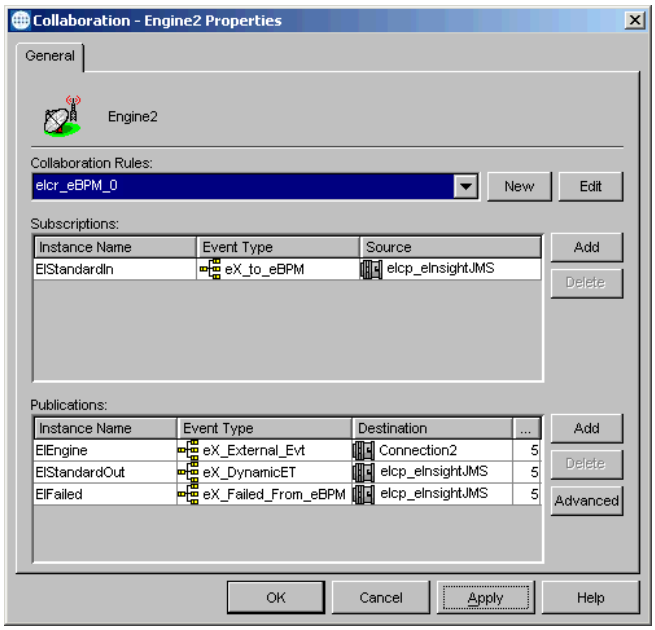
Figure 99 e*Insight Engine Collaboration Mapping Tab



Note: *You can create this Collaboration Rule by copying the* ***eIcr_eBPM*** *Collaboration Rule that is provided in the eIJSchema.*

3 Add a Collaboration to the e*Insight engine container that uses the Collaboration Rule created in step 2. Configure the Collaboration as shown Figure 100. You should configure the Collaboration to publish to the e*Way Connection created in step 1.

Figure 100 e*Insight Engine Collaboration



13.1.3 e*Insight Engine Affinity (eIJSchema)

e*Insight Engine Affinity allows e*Insight engines in a multi-engine environment to cache information about particular Business Process Instances as they flow through the e*Gate schema using Instance Caching. Since the engines hold the instance information in a cache, it is essential that an individual instance is always processed by the same engine or Collaboration. This is achieved by setting a JMS property in e*Gate.

Using Engine Affinity can possibly improve the overall message throughput but if an engine is shut down for some reason, the instances associated with that engine do not finish being processed until the engine is manually restarted using the e*Gate Monitor.

13.1.4 Using Engine Affinity with e*Gate — Active Mode

The JMS Message Server uses message selectors to filter out certain messages from a specific queue before sending them to the JMS e*Way connection. The message selector ensures that an engine only receives Business Process Instances that it originally processed, or START_BP messages .

To configure multiple engines to use e*Insight Engine Affinity

1 Create multiple engines. See **"Using Multiple e*Insight Engines" on page 186**.

2 Edit the configuration file for each e*Insight engine connection. In the **eBPM Settings**, set the **Instance Caching** parameter to **Yes**.

Note: *The engines can refer to the same connection configuration file.*

3 Configure the Engine Affinity JMS properties in your Activity Collaborations, if you are using different source and destination instances for eI_StandardEvent. See **"Configuring the Engine Affinity JMS Properties"**.

Configuring the Engine Affinity JMS Properties

You need to set up the Engine Affinity JMS properties in your Activity Collaborations, if you are using different source and destination instances for eI_StandardEvent. You can set up the properties using the prepareReplyToSender method in the eI_StandardEvent ETD.

Create a rule before you return from executeBusinessRules() by dragging the root node of the inbound eI_StandardEvent into the parameter for the prepareReplyToSender() method of the outbound eI_StandardEvent. This generates code such as:

```
getEIStandardOut.prepareReplyToSender(getEIStandardIn());
```

Note: *If you are using one instance (such as EIStandardInOut), you do not need to use prepareReplyToSender() since the Engine Affinity JMS Properties already exist.*

For backwards compatibility, if the first Business Process Activity is passive, the e*Insight engine still echoes back the incoming START_BP message for synchronization. To disable this feature, add another System Property definition, **ei.pasvNoEcho=true**, in the Initialization string of the e*Insight Engine Collaboration Rules.

13.1.5 Using Engine Affinity with e*Gate — Passive Mode

When you define every activity in e*Insight as Passive, then the e*Insight engine is simply receives the status from the processing of business process activities. In this situation it is essential that:

- the START_BP Event is received by the engine before any status Events are received.
- the status Events are processed by the same engine that processes the START_BP Event

This is achieved by using the Collaboration that sends the START_BP to the e*Insight engine to randomly select and assign an e*Insight engine to each business process instance. The details of the engine selected are also sent in the Event to the module processing the first, and subsequent, activities.

The JMS Message Server uses message selectors to filter out certain messages from a specific queue before sending them to the JMS e*Way connection. The message selector ensures that an engine only receives Business Process Instances that are assigned to it.

To configure multiple engines to use e*Insight Engine Affinity — Passive Mode

1. Create multiple engines. See **"Using Multiple e*Insight Engines" on page 186**.
2. Edit the configuration file for each e*Insight engine connection. In the **eBPM Settings**, set the **Instance Caching** parameter to **Yes**.

Note: *The engines can refer to the same connection configuration file.*

3. Configure the Collaboration that sends the START_BP Event to the e*Insight engine to call the **setPassiveStart()** method in the **userInitialize()** section of the Collaboration Rules script.

13.1.6 Using e*Xchange with e*Insight (eIJSchema)

You can use an Activity Specific ETD that supports e*Xchange or you can manually configure **eIX_StandardEvent.xsc**.

The Event Type Definition used in the engine's Collaboration Rules Script, **eIX_StandardEvent.xsc** contains two sections, **BP_EVENT**, containing information for e*Insight and **TP_EVENT**, containing information for e*Xchange. If you are not using the e*Xchange section of the ETD, then you can configure the engine ignore this section, reducing processing.

To configure the engine to ignore e*Xchange section of the ETD

1. Select the e*Way Connections folder.
2. Open the properties of the e*Insight engine e*Way Connector.
3. Click **Edit**.
4. Go to the **eBPM Settings** section, and find **Using e*Xchange with e*Insight**.
5. Click **No**.
6. Close the configuration editor and save the changes.

13.1.7 Using Binary XML (eIJSchema)

By default, the e*Insight engine generates a binary XML message. This data format reduces parsing and so can increase performance. The following initialization string in the Collaboration Rule properties determines that a binary XML message is created:

```
-def egate.binXmlMarshal=true
```

Note: *An Event Type Definition can be used to define the structure of XML data. This Event Type Definition must be created from a .xsd (XML Schema Definition) or .dtd (Document Type Definition) file. In order to create these Event Type Definitions you must install the XML ETD Builders with e*Gate.*

Important: *Monk is not able to interpret this data format so you should only configure Java Collaborations to subscribe to this Event.*

13.1.8 Subscribing to Event Types

The method of subscribing for multiple Event Types from a JMS e*Way Connection entails waiting 'n' milliseconds (configured in the Event Type "get" interval of the e*Way Connection Properties dialog, see **"Event Type "get" Interval — JMS Server" on page 193**) for the first Event Type and if none is found, waiting for 1 millisecond for the next Event Type and if none again, the JMS Server waits 'n' milliseconds for the second Event Type and so on. This method works well if the distribution of the different Event Types is relatively even, but there are likely to be more "Do" Events than there are "Undo" Events for e*Insight. Therefore, the engines spend a lot of time waiting for a Event Type that is not there.

You can avoid unnecessary wait times due to the "Do" and "Undo" Event Types, by:

- Subscribing to a single "Go" Event Type which retrieves both "Do" and "Undo" Events
- Having a separate Collaboration to subscribe to the "Do" and "Undo" Event Types
- Removing unnecessary subscriptions from Collaborations

The options are described in detail below.

Subscribing to a Single "Go" Event

The e*Insight engine can be configured to publish an Event Type **eI_<Activity Name>_Go**, rather than **eX_<Activity Name>_Do** or **eI_<Activity Name>_Undo**. This allows you to subscribe to a single Event Type and then check the type of Event within your Collaboration Rule script.

To configure the engine to publish the single Event Type, set the initialization string in the e*Insight Engine Collaboration Rules dialog to:

```
-def egate.multiDef=;egate.binXmlMarshal=true;ei.oneETPerAct=true
```

This new Java System Property definition, **ei.oneETPerAct=true**, directs the engine to publish only one ET, **eI_<Activity Name>_Go**, for both "Do" and "Undo" Events.

Note: *You must use the egate.multiDef parameter to define more than one System Property definition in the initialization string. The first character after the equal (=) sign is the delimiter to use for separating different System Property key=value pairs (for example, the semi-colon (;) is used above). The Schema Designer does not allow the use of commas (,) since the e*Gate Registry uses this delimiter internally. Special characters can be specified by using standard escape codes (such as\t, \n) or Unicode escape codes (such as \u003d for '=').*

Configuring a Separate Collaboration for Do and Undo Events

To remove the requirement for a single Collaboration to subscribe to both "Do" and "Undo" Events, separate the "Do" and "Undo" logic in your Collaboration Rules script.

Then create a "Do" and an "Undo" Collaboration, each subscribing to a single Event Type.

Removing Unnecessary Subscriptions

Subscribing to multiple Event Types is less efficient than subscribing to a single Event Type. If you have not implemented "undo" logic in your business process, then remove the subscriptions to the eX_<Activity_Name>_Undo Event Types.

13.1.9 Event Type "get" Interval — JMS Server

The e*Insight engine uses the "get" interval to determine how long the JMS Server waits for an Event of a particular Event Type to arrive.

Ideally, a Collaboration should only subscribe to a single Event Type, but if that is not possible then the "get" interval can impact performance. The default interval is 10,000 milliseconds which should be suitable if the distribution of the different Event Types is even. If the distribution of the different Event Types is not even, the you might want to reduce the "get" interval. Consider an example where you are subscribing to Event Types A and B, and Event Type B rarely (or never) arrives. Using the default "get" interval, you wait 10,000 milliseconds after every Event Type A is processed. Reducing the "get" interval to a value less than 100 can dramatically increase performance.

The **Event Type "get" Interval** is set in the e*Way Connection properties for the JMS Connection (eIcp_eInsightJMS).

13.1.10 Event Type "get" Interval — e*Insight Engine

The e*Insight engine uses the interval to determine how often to poll the control table. The control table contains entries for batch tasks, such as manual restarts, and also contains entries for instances using the User Activity or Authorization Activity. Depending on your implementation of e*Insight, it may be desirable to reduce the polling frequency to increase performance.

The **Event Type "get" Interval** is set in the e*Way Connection properties for the e*Insight Engine Connection (eIcp_eInsightEngine).

13.1.11 Review JVM Settings

You may be able to improve performance by changing the memory allocated for the Java Virtual Machine. This is set in the e*Insight engine configuration file, **JVM Settings**, **Maximum Heap Size**. If this is left set to zero (0), the preferred value for the maximum heap size of the Java VM is used (this is set by the Operation System configuration).

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

13.2 Performance Improvements Using eISchema

The purpose of this section is to describe methods that can be used to improve performance of the eIJSchema.

- Instance caching
- Using multiple engines
- Using instance caching with multiple engines
- Setting the Exchange Data Interval

13.2.1 Instance Caching

Instance Caching is the most efficient way to process Business Process Instances. Using instance caching keeps a cache of the instance information throughout the life span of the Business Process Instance. If instance caching is not used the instance information is retrieved from the database instead. This allows more flexibility and fault tolerance at the cost of performance.

To configure the engine to use instance caching

1. Open the engine's properties.
2. Click **Edit** to open the engine's configuration file.
3. In the **eBPM Setting** section, set **Instance Caching** to **YES**.

13.2.2 Using Multiple e*Insight Engines (eISchema)

You can use multiple e*Insight engines to increase performance. This section describes how to add and configure additional engines. You can use instance caching to further improve performance. See **"e*Insight Engine Affinity (eISchema)" on page 195** 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.*

13.2.3 e*Insight Engine Affinity (eISchema)

e*Insight Engine Affinity allows e*Insight engines in a multi-engine e*Gate schema to cache information about particular Business Process Instances as they flow through the e*Gate schema using Instance Caching. Using Engine Affinity can possibly improve the overall message throughput but if an engine is shutdown for some reason, the instances associated with that engine do not finish being processed until the engine is manually restarted using the e*Gate Monitor.

e*Insight uses the attribute "eX_eBPMServer" to hold the name of the engine that publishes the "Do" Event. This value is referenced when publishing a "Done" Event back to the engine and the value used in the published Event Type name. For example, if the engine name is **EngineA** then an Event Type **eX_to_EngineA** is published. **EngineA** is configured to subscribe to Event Type **eX_to_EngineA** to ensure that the Event is processed by the same engine.

Note: *Since the Event Type names are determined by the engine name, if you rename the engine you must rename the Event Types.*

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 101.

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:

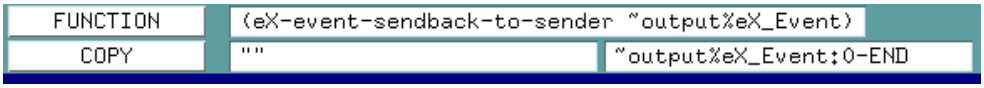
eX-event-sendback-to-sender <root-path>

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 101.

Note: *Make sure that the Monk file **eX-event-sendback-to-sender.monk**, containing this function, is loaded before calling it in a Collaboration Rules Script. You can do this by putting it in the root of the **monk_library** directory, or loading it explicitly in your CRS.*

Finally, you must suppress the default output. This is achieved by overwriting the destination Event Type Definition with an empty string. Use the copy function as shown in Figure 101.

Figure 101 Monk Collaboration Rule Script for Engine Affinity



13.2.4 Exchange Data Interval (eISchema)

The e*Insight engine uses the interval to determine how often to poll the control table. The control table contains entries for batch tasks, such as manual restarts, and also contains entries for instances using the User Activity or Authorization Activity. Depending on your implementation of e*Insight, it may be desirable to reduce the polling frequency to increase performance.

The Exchange Data Interval is set in the e*Way Configuration file, **Communication Setup** section.

13.2.5 Review JVM Settings

You may be able to improve performance by changing the memory allocated for the Java Virtual Machine. This is set in the e*Insight engine configuration file, **JVM Settings**, **Maximum Heap Size**. If this is left set to zero (0), the preferred value for the maximum heap size of the Java VM is used.

13.3 General e*Insight Performance Tips

This section describes some ways of improving performance that are not specific to the type of e*Insight schema that you are running.

- Review the *SeeBeyond Business Integration Suite Deployment Guide*.
- Use a third party tool to determine if hardware is the limitation.
- Enhance database access by one of the following:
  - Use RAID/stripe disks with a multi-controller.
  - Install the database and e*Insight engines on the same machine.
  - Tune the database.
- Use a Model Specific database — The Model Specific database uses a different structure for storing the attribute values, which increases performances. If you are using the Model Specific database, you can modify the database to your specific data requirements by controlling the size allocated to each attribute value. You need to create the necessary database tables before running your e*Insight schema. Every business process version uses its own set of tables.

See the *e*Insight Business Process Manager User's Guide* for information on creating the Model Specific database.

- Do not automatically reload models — The Auto Model Reload engine configuration parameter determines if the engine dynamically loads an enabled Business Process Version if the enabled/disabled status of Business Process Version changes. If the value is set to YES then Business Process Versions that are enabled or disabled while the engine is running are immediately recognized. However, setting this value to YES may degrade performance.
- Do not preload unnecessary Business Processes — The Business Processes to Preload engine configuration parameter allows you to load all or a subset of all the business processes stored in the e*Insight database. The default setting is ALL. Naming specific business processes to preload may improve performance if you have a large number of business processes defined.
- Do not include input only attributes in the "Done" Event — To simplify your Collaboration Rules script, you may decide to copy everything from the source ETD to the destination ETD, especially when using Java. This copies all the input attributes to the destination ETD and, unless they are removed, they are included in the "Done" Event. This requires additional processing by the engine, as it tries to write the attribute to the database but is refused permission.
- Depending on your configuration and the volume of your message traffic, there may be a performance benefit if you use multiple JMS Servers in your schema. The message traffic can also be distributed to different JMS Servers. All publishers/subscribers for a particular topic should be attached to the same JMS Server to simplify the schema.

Chapter 14

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.

14.1 Log Files

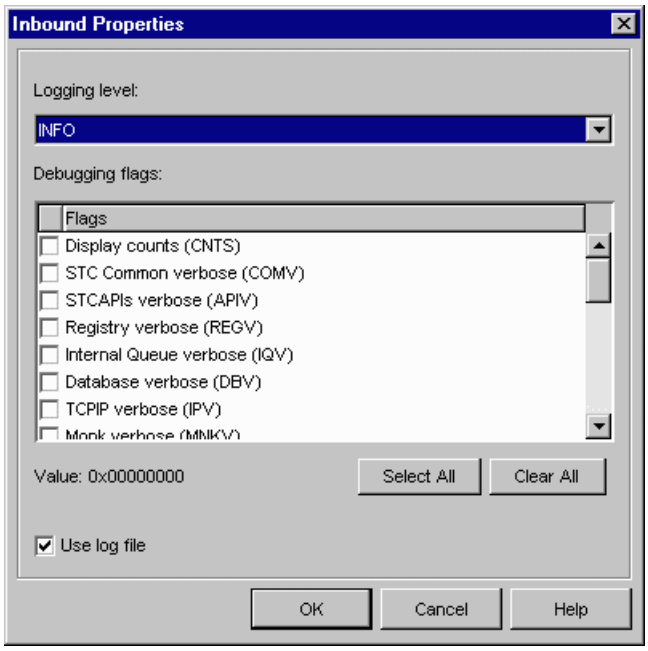
All log files are stored in the **\eGate\client\logs** directory on the Participating Host running the elements that generate the log entries. Logs are named after the component that creates them; for example, the **eX_eBPM** engine creates a log file called **eX_eBPM.log**.

14.1.1 Generating Log Files

To configure a component to generate a log file:

1. In the e*Gate Schema Designer 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 find in a log file is, "Unable to load module configuration." This message means that you have created an e*Gate component but not assigned both an executable file and a configuration file to it.

When you first start to debug your e*Insight schema, you should apply minimal flags so it is easier to find useful messages. You can start with the following flags with a DEBUG logging level selected:

- e*Way (EWY)
- Collaboration (COL) — for eIJSchema
- Monk (MNK) — for eISchema

Note: *A specific debugging flag does not appear in the flags list for the e*Insight engine, but if the logging level is set to anything other than None, then messages of type* ***EBPM*** *appear in the log.Setting the e*Insight engine logging level to* ***Trace*** *generates a very detailed log of the engine's activity.*

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

Reading Log Files

Refer to the following information to decipher your log files.

Table 35 Log File Definitions

Level	Symbol	Trace Name	e*Gate Equivalent
0	F	Trace.FATAL	TRACE_EVENT_FATAL
1	E	Trace.ERROR	TRACE_EVENT_LOGERROR
2	W	Trace.WARNING	TRACE_EVENT_WARNING
3	I	Trace.INFO	TRACE_EVENT_INFORMATION
4	D	Trace.DEBUG	TRACE_EVENT_DEBUG
5	T	Trace.TRACE	TRACE_EVENT_TRACE

Trace.FATAL

This is the message that must be reported otherwise the e*Way engine will not start or execute; For example, **Failed to start an e*Way**. This must be fixed in order to successfully run the e*Insight process.

Trace.ERROR

This is the message that reports error messages; For example: **Failed to establish a connection to database**. This must be fixed in order to successfully run the eInsight process. Any exception thrown will be a trace error.

Trace.WARNING

This is the message that reports the warning to the user that certain configurations are not setup up properly and may cause runtime problems later on; For example: Attribute not initialized or defined, or user does not have access privilege to a business process model.

Trace.INFO

This is the message that displays useful information such as business process names, enabled BPIs or BPIStack.

Trace.DEBUG

This is the message that displays the workflow logic of the business process model; For example: Displaying the flow logic of the workflow process. The **BusinessProcessInstanceMgr**, instance access, or workflow logic should use this trace.

Trace.TRACE

This is lowest level of tracing. This is more for the quality assurance or developer level to resolve low level issues. For example: Updating or inserting a SQL statement on a database attribute table. DBObject or database related calls should use this trace.

The e*Insight trace definitions are defined in the com.stc.bpms.util.Trace class. The e*Gate trace event definitions are defined in the com.stc.common.collabService.Egate class.

14.2 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 36 and Table 37 show a number of common problems and suggested actions you should take. The tables also shows where to look for the error message and the logging required to display the error.

Table 36 Common errors (eIJSchema)

Error Message	Debug Flag	Problem	Resolution
JMSException: Could not connect to host: [hostname], port: 24053	COL	eIcp_eInsightJMS e*Way connection is configured incorrectly	In the eIcp_eInsightJMS connection configuration file, **Message Service** section, check the value for **Server Name** and **Host Name**.
Cannot connect to e*Insight Backend DataBase: I/O exception: Invalid number format for port number	EWY	JDBC URL incorrect	In the eIcp_eInsightEngine configuration file, **eBPM Settings** section, check the value defined for **JDBC URL String**.
Cannot connect to e*Insight Backend DataBase: Invalid Oracle URL specified	EWY		
Cannot connect to e*Insight Backend DataBase: I/O exception: Connection refused	EWY	Unable to connect to the database	In the eIcp_eInsightEngine configuration file, **eBPM Settings** section, check the value defined for **JDBC URL String**.
Cannot connect to e*Insight Backend DataBase: ORA-01017: invalid username/ password; logon denied	EWY	Incorrect user name or password	In the eIcp_eInsightEngine configuration file, **eBPM Settings** section, check the values defined for **Database User Name** and **Encrypted Password**.

Table 36 Common errors (eIJSchema)

Error Message	Debug Flag	Problem	Resolution
startActivity() failed: Definition not found for :*business process*	EWY	Incorrect business process name	Check that the business process name defined in the CRS matches the business process name defined in e*Insight exactly. Also, check that a business process version has been enabled for this business process. Note: Although the error message appears in the eX_eBPM log, the problem is in the component that sent the message to the e*Insight engine.
startActivity() failed: Invalid BP_EVENT type:*STARTBP*	EWY	Incorrect BP_EVENT type	Check that the BP_EVENT type defined in the CRS is correct. Note: Although the error appears in the eX_eBPM log, the problem is in the component that sent the message to the e*Insight engine.
stcewjx: Exception getMessage(): processOutgoing(): eBPM cannot process XML data: ORA-01407: cannot update ("EX_ADMIN"."BUSINESS_PROCESS_INSTANCE"."BPI_NM") to NULL	EWY	No ID set	Check that the ID is defined in the CRS. Note: Although the error appears in the eX_eBPM log, the problem is in the component that sent the message to the e*Insight engine.

Table 37 Common errors (eISchema)

Error Message	Debug Flag	Problem	Resolution
eX_eBPM (Fatal): ewjx: the "JNI DLL" (<jvm path>) specified is not a Java 2 version	EBPM	jvm configuration incorrect	In the eX_eBPM configuration file, Java VM Configuration section, check the value defined for JNI DLL.
eX_eBPM (Warning): Cannot connect to eBPM Backend DataBase: I/O 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 variable <eX-set-attribute> has not been defined.	MNK	eX-eBPM-utils.monk has not been loaded	Copy eX-eBPM-utils.monk to <egate>\client\monk_library.
>>>>MONKEXCEPT:0069: throw: eBPM: Cannot process event	MNK	Incorrect business process name	Check that the business process name defined in the CRS matches the business process name defined in e*Insight exactly. Also, check that a business process version has been enabled for this business process. Note: Although the error message appears in the eX_eBPM log, the problem is in the component that sent the message to the e*Insight engine.
ewjx: Exception getMessage(): processOutgoing(): eBPM cannot process XML data: ERROR: Unable to load business process. bpoId not found for :fred	EWY		

Table 37 Common errors (eISchema)

Error Message	Debug Flag	Problem	Resolution
>>>>MONKEXCEPT:0069: throw: eBPM: Cannot process event	MNK	Incorrect BP_EVENT type	Check that the BP_EVENT type defined in the CRS is correct. Note: Although the error appears in the eX_eBPM log, the problem is in the component that sent the message to the e*Insight engine.
ewjx: Exception getMessage(): processOutgoing(): eBPM cannot process XML data: Invalid BP_EVENT type:START_BD	EWY		
>>>>MONKEXCEPT:0069: throw: eBPM: Cannot process event	MNK	No ID set	Check that the ID is defined in the CRS. Note: Although the error appears in the eX_eBPM log, the problem is in the component that sent the message to the e*Insight engine.
ewjx: Exception getMessage(): processOutgoing(): eBPM cannot process XML data: ORA-01407: cannot update ("EX_ADMIN"."BUSINESS_PROCESS_INSTANCE"."BPI_NM") to NULL	EWY		
ERROR - CONTINUING: BP attribute not found:Cust_Addres in createBPI() bpiId:1016	EBPM	Attribute does not exist in e*Insight	Check that the attribute name matches the name defined in the CRS. Note: Although the error appears in the eX_eBPM log, the problem is in the component that sent the message to the e*Insight engine.
(get (eX-get-attribute ~input%eX_Event "Item_Number")) >>>>MONKEXCEPT:0009: get: argument 1 must be a valid path.	MNK	Attribute does not exist in the message sent to an e*Way or BOB for an activity	Check that the attribute is defined as an input attribute for the activity. Also, check that the attribute name matches the name defined in the CRS. Note: This error appears in the log for the e*Way or BOB for the activity.

14.3 General Troubleshooting Tips

14.3.1 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.

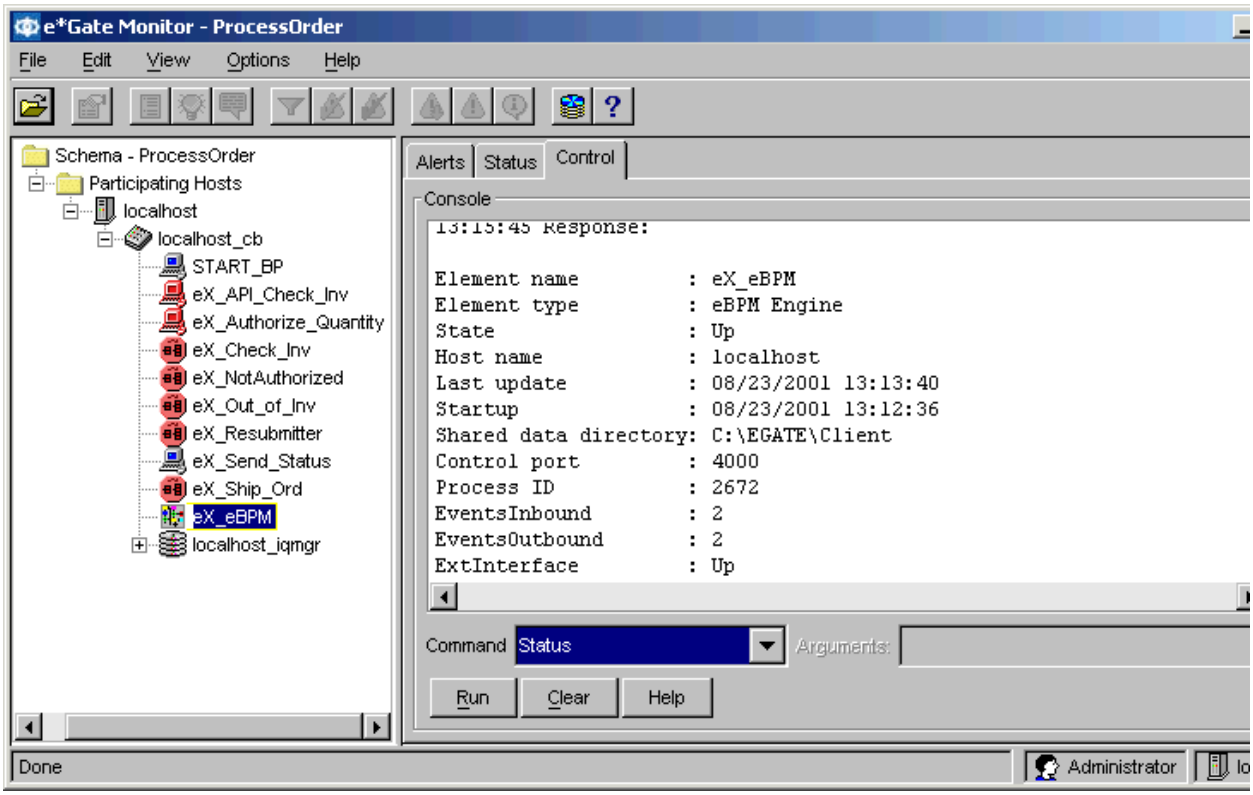
Use the e*Gate Enterprise Monitor GUI to check whether a message has been processed by a component. The number of inbound and outbound messages for a component is displayed by the status command.

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



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".*

14.3.2 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 **20010823134448016 9** (the BPI name in the GUI), and there are a number of global attributes defined (namely, **Cust_Address**, **Cust_Name**, **Cust_email**, **Item_Description**, **Item_Number**, and **Order_Quantity**).

```
13:44:48.219 IQV  D 2488 (iqput.cxx:114): sending to iq manager Data Follows
(bytes 581):
      3C 65 58 5F 45 76 65 6E 74 3E 0A 3C 42 50 5F 45   |  <eX_Event>.<BP_E
      56 45 4E 54 0A 54 59 50 45 3D 22 53 54 41 52 54   |  VENT.TYPE="START
      5F 42 50 22 0A 49 44 3D 22 32 30 30 31 30 38 32   |  _BP".ID="2001082
      33 31 33 34 34 34 38 30 31 36 39 22 0A 4E 41 4D   |  31344480169".NAM
      45 3D 22 50 72 6F 63 65 73 73 4F 72 64 65 72 22   |  E="ProcessOrder"
      3E 0A 3C 41 54 54 52 49 42 55 54 45 0A 56 41 4C   |  >.<ATTRIBUTE.VAL
      55 45 3D 22 34 30 34 20 45 2E 20 48 75 6E 74 69   |  UE="404 E. Hunti
      6E 67 74 6F 6E 20 44 72 2E 22 0A 54 59 50 45 3D   |  ngton Dr.".TYPE=
      22 53 54 52 49 4E 47 22 0A 4E 41 4D 45 3D 22 43   |  "STRING".NAME="C
      75 73 74 5F 41 64 64 72 65 73 73 22 3E 3C 2F 41   |  ust_Address"></A
      54 54 52 49 42 55 54 45 3E 0A 3C 41 54 54 52 49   |  TTRIBUTE>.<ATTRI
```

```
42 55 54 45 0A 56 41 4C 55 45 3D 22 4A 6F 68 6E  |  BUTE.VALUE="John
20 53 6D 69 74 68 22 0A 54 59 50 45 3D 22 53 54  |   Smith".TYPE="ST
52 49 4E 47 22 0A 4E 41 4D 45 3D 22 43 75 73 74  |  RING".NAME="Cust
5F 4E 61 6D 65 22 3E 3C 2F 41 54 54 52 49 42 55  |  _Name"></ATTRIBU
54 45 3E 0A 3C 41 54 54 52 49 42 55 54 45 0A 56  |  TE>.<ATTRIBUTE.V
41 4C 55 45 3D 22 6A 73 6D 69 74 68 40 73 65 65  |  ALUE="jsmith@see
62 65 79 6F 6E 64 2E 63 6F 6D 22 0A 54 59 50 45  |  beyond.com".TYPE
3D 22 53 54 52 49 4E 47 22 0A 4E 41 4D 45 3D 22  |  ="STRING".NAME="
43 75 73 74 5F 65 6D 61 69 6C 22 3E 3C 2F 41 54  |  Cust_email"></AT
54 52 49 42 55 54 45 3E 0A 3C 41 54 54 52 49 42  |  TRIBUTE>.<ATTRIB
55 54 45 0A 56 41 4C 55 45 3D 22 4D 69 6C 6C 65  |  UTE.VALUE="Mille
6E 6E 69 75 6D 20 50 65 74 20 52 6F 63 6B 22 0A  |  nnium Pet Rock".
54 59 50 45 3D 22 53 54 52 49 4E 47 22 0A 4E 41  |  TYPE="STRING".NA
4D 45 3D 22 49 74 65 6D 5F 44 65 73 63 72 69 70  |  ME="Item_Descrip
74 69 6F 6E 22 3E 3C 2F 41 54 54 52 49 42 55 54  |  tion"></ATTRIBUT
45 3E 0A 3C 41 54 54 52 49 42 55 54 45 0A 56 41  |  E>.<ATTRIBUTE.VA
4C 55 45 3D 22 33 33 33 33 33 22 0A 54 59 50 45  |  LUE="33333".TYPE
3D 22 53 54 52 49 4E 47 22 0A 4E 41 4D 45 3D 22  |  ="STRING".NAME="
49 74 65 6D 5F 4E 75 6D 62 65 72 22 3E 3C 2F 41  |  Item_Number"></A
54 54 52 49 42 55 54 45 3E 0A 3C 41 54 54 52 49  |  TTRIBUTE>.<ATTRI
42 55 54 45 0A 56 41 4C 55 45 3D 22 31 22 0A 54  |  BUTE.VALUE="1".T
59 50 45 3D 22 4E 55 4D 42 45 52 22 0A 4E 41 4D  |  YPE="NUMBER".NAM
45 3D 22 4F 72 64 65 72 5F 51 75 61 6E 74 69 74  |  E="Order_Quantit
79 22 3E 3C 2F 41 54 54 52 49 42 55 54 45 3E 3C  |  y"></ATTRIBUTE><
2F 42 50 5F 45 56 45 4E 54 3E 3C 2F 65 58 5F 45  |  /BP_EVENT></eX_E
76 65 6E 74 3E                                   |  vent>
```

The sample below shows an example of a "Do" message. You should be able to determine from the message that the business process name is **ProcessOrder,** the ID is **200108231344480169,** and the activity name is **Check_Inv**. There are two global attributes defined as input attributes for this activity, **Item_Number** and **Order_Quantity**.

```
14:42:27.716 MNKV D 2228 (monk_extension.cxx:745): Output topic:
eX_Check_Inv_Do
14:42:27.726 MNKV D 2228 (monk_extension.cxx:763): Msg body Data Follows
(bytes 389):
      3C 65 58 5F 45 76 65 6E 74 3E 0A 3C 42 50 5F 45  |  <eX_Event>.<BP_E
      56 45 4E 54 0A 42 50 49 5F 49 44 3D 22 31 36 31  |  VENT.BPI_ID="161
      36 2E 30 2E 32 30 32 3A 22 0A 54 59 50 45 3D 22  |  6.0.202:".TYPE="
      44 4F 5F 41 43 54 49 56 49 54 59 22 0A 49 44 3D  |  DO_ACTIVITY".ID=
      22 32 30 30 31 30 38 32 30 31 34 33 38 30 34 30  |  "200108231344480
      36 34 36 22 0A 4E 41 4D 45 3D 22 50 72 6F 63 65  |  169".NAME="Proce
      73 73 4F 72 64 65 72 22 3E 0A 3C 41 43 54 49 56  |  ssOrder">.<ACTIV
      49 54 59 20 4E 41 4D 45 3D 22 43 68 65 63 6B 5F  |  ITY NAME="Check_
      49 6E 76 22 20 49 44 3D 22 31 36 31 36 2E 35 36  |  Inv" ID="1616.56
      39 22 20 2F 3E 0A 3C 41 54 54 52 49 42 55 54 45  |  9" />.<ATTRIBUTE
      20 4E 41 4D 45 3D 22 49 74 65 6D 5F 4E 75 6D 62  |   NAME="Item_Numb
      65 72 22 20 54 59 50 45 3D 22 53 54 52 49 4E 47  |  er" TYPE="STRING
      22 20 56 41 4C 55 45 3D 22 39 39 39 39 39 22 20  |  " VALUE="99999"
      2F 3E 0A 3C 41 54 54 52 49 42 55 54 45 20 4E 41  |  />.<ATTRIBUTE NA
      4D 45 3D 22 4F 72 64 65 72 5F 51 75 61 6E 74 69  |  ME="Order_Quanti
      74 79 22 20 54 59 50 45 3D 22 4E 75 6D 62 65 72  |  ty" TYPE="Number
      22 20 56 41 4C 55 45 3D 22 31 2E 30 22 20 2F 3E  |  " VALUE="1.0" />
      0A 3C 41 54 54 52 49 42 55 54 45 0A 56 41 4C 55  |  .<ATTRIBUTE.VALU
      45 3D 22 65 42 50 4D 22 0A 4C 4F 43 41 54 49 4F  |  E="eBPM".LOCATIO
      4E 3D 22 45 4D 42 45 44 44 45 44 22 0A 54 59 50  |  N="EMBEDDED".TYP
      45 3D 22 54 52 41 4E 53 49 45 4E 54 22 0A 4E 41  |  E="TRANSIENT".NA
      4D 45 3D 22 65 58 5F 65 42 50 4D 53 65 72 76 65  |  ME="eX_eBPMServe
      72 22 3E 3C 2F 41 54 54 52 49 42 55 54 45 3E 3C  |  r"></ATTRIBUTE><
      2F 42 50 5F 45 56 45 4E 54 3E 3C 2F 65 58 5F 45  |  /BP_EVENT></eX_E
      76 65 6E 74 3E                                   |  vent>
```

Chapter 15

Java Helper Methods

A number of Java methods have been added to make it easier to set information in the e*Insight Event (ETD) and to get information from it. These methods are contained in classes:

- **"ACTIVITY Class" on page 210**
- **"ATTRIBUTE Class" on page 230**
- **"BP_EVENT Class" on page 248**
- **"eX_StandardEvent Class" on page 284**

15.1 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:

```
<!--Business 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 on page 211
clearATTRIBUTE on page 212
countATTRIBUTE on page 213
getATTRIBUTE_VALUE on page 214
getATTRIBUTE on page 215
getID on page 216
getNAME on page 217
hasID on page 218
hasNAME on page 219
marshal on page 220
omitID on page 221
omitNAME on page 222
removeATTRIBUTE on page 223
setATTRIBUTE on page 224
setID on page 226
setNAME on page 227
toString on page 228
unmarshal on page 229

addATTRIBUTE

Syntax

```
void addATTRIBUTE(ATTRIBUTE value)
void addATTRIBUTE(int index, ATTRIBUTE value)
```

Description

addATTRIBUTE inserts a new local Attribute into this ACTIVITY object.

Parameters

Name	Type	Description
index	integer	The offset to the list at which insertion occurs (zero-based).
value	ATTRIBUTE	The local Attribute.

Return Values

None.

Throws

None.

clearATTRIBUTE

Syntax

```
void clearATTRIBUTE()
```

Description

clearATTRIBUTE removes all the local Attributes from this ACTIVITY object.

Parameters

None.

Return Values

None.

Throws

None.

Example

```
clearATTRIBUTE();
```

countATTRIBUTE

Syntax

```
int countATTRIBUTE()
```

Description

countATTRIBUTE retrieves the number of local Attributes currently existing in this Activity of the Business Process object.

Parameters

None.

Return Values

integer
Returns the number of global Attributes.

Throws

None.

Example

```
countATTRIBUTE();
   => 5
```

getATTRIBUTE_VALUE

Syntax

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

Description

getATTRIBUTE_VALUE retrieves the value of a specific local Attribute by name.

Parameters

Name	Type	Description
name	java.lang.String	The name of the local Attribute.

Return Values

String

Returns the value of the local Attribute. Can be null if the Attribute of that name doesn't exist.

Throws

None.

Example

```
getInstance().getBP_EVENT().getACTIVITY.getATTRIBUTE_VALUE("In_Stock"
);
   => "yes"
```

getATTRIBUTE

Syntax

```
ATTRIBUTE[] getATTRIBUTE()
ATTRIBUTE getATTRIBUTE(int i)
ATTRIBUTE getATTRIBUTE(java.lang.String name)
```

Description

getATTRIBUTE retrieves local Attributes. A specific Attribute can be retrieved by name or by index. Alternatively, it can be used to retrieve all the local Attributes of an Activity of the Business Process as an array.

Parameters

Name	Type	Description
i	integer	The list index of the Attribute to be retrieved (zero-based).
name	string	The name of the local Attribute.

Return Values

Returns one of the following values:

ATTRIBUTE[]
Returns the array of local Attributes.

ATTRIBUTE
Returns the requested Attribute.

Throws

None.

getID

Syntax

```
java.lang.String getID()
```

Description

getID retrieves the internal unique identifier created by e*Insight for this Activity in the Business Process.

Parameters

None.

Return Values

java.lang.String

Returns the unique internal identifier created for this Activity.

Throws

None.

Example

```
getID();
   => "12345"
```

getNAME

Syntax

```
java.lang.String getNAME()
```

Description

getNAME retrieves the case-sensitive name of this Business Process Activity.

Parameters

None.

Return Values

java.lang.String
Returns the case-sensitive name of this ACTIVITY object.

Throws

None.

Example

```
getInstance().getBP_EVENT().getACTIVITY().getNAME();
   => "Ship_Order"
```

hasID

Syntax

```
boolean hasID()
```

Description

hasID checks if there is an unique identifier for this ACTIVITY object.

Parameters

None.

Return Values

boolean
Returns true if there exists an unique ID; otherwise returns false.

Throws

None.

Example

```
hasID();
   => true
```

hasNAME

Syntax

```
boolean hasNAME()
```

Description

hasNAME checks if there exists a name for this ACTIVITY object.

Parameters

None.

Return Values

boolean
Returns true if name exists for this ACTIVITY object; otherwise returns false.

Throws

None.

Example

```
hasNAME();
   => true
```

marshal

Syntax

```
void marshal(org.xml.sax.ContentHandler handler,
org.xml.sax.ErrorHandler errorHandler)
```

Description

marshal gathers the data contained within this ETD object and formulates it back into a serialized XML message.

Parameters

Name	Type	Description
handler	org.xml.sax.ContentHandler	The handler that converts content within to XML.
errorHandler	org.xml.sax.ErrorHandler	The handler to address errors during conversion.

Return Values

None.

Throws

None.

omitID

Syntax

```
void omitID()
```

Description

omitID removes the unique identifier definition for this ACTIVITY object.

Parameters

None.

Return Values

None.

Throws

None.

Example

```
omitID();
```

omitNAME

Syntax

```
void omitNAME()
```

Description

omitNAME removes the name definition from this ACTIVITY object.

Parameters

None

Return Values

None.

Throws

None.

Example

```
omitNAME();
```

removeATTRIBUTE

Syntax

```
void removeATTRIBUTE(java.lang.String name)
void removeATTRIBUTE(int index)
```

Description

removeATTRIBUTE removes a specific local attribute from this ACTIVITY object.

Parameters

Name	Type	Description
name	java.lang.String	The name of the local attribute.
index	int	The index to the list of local attributes (zero-based).

Return Values

None.

Throws

None.

Example

```
removeATTRIBUTE(1);
```

setATTRIBUTE

Syntax

```
void setATTRIBUTE(ATTRIBUTE[] val)
void setATTRIBUTE(int i, ATTRIBUTE val)
void setATTRIBUTE(java.lang.String name java.lang.String value)
void setATTRIBUTE(java.lang.String name, java.lang.String type,
java.lang.String value)
void setATTRIBUTE(java.lang.String name, java.lang.String type,
java.lang.String encoding,java.lang.String value)
void setATTRIBUTE(java.lang.String name, java.lang.String type,
java.lang.String value, java.lang.String encoding,java.lang.String
location)
```

Description

setATTRIBUTE can be used to set all the local Attributes of an Activity of the Business Process, set a local Attribute of an Activity of the Business Process, or set a specific local Attribute by name.

Parameters

Name	Type	Description
val	ATTRIBUTE	The Attribute object.
val	ATTRIBUTE[]	The array of local Attributes.
i	int	The list index of the Attribute to be retrieved (zero-based).
name	java.lang.String	The name of the global Attribute.
value	java.lang.String	The value of the global Attribute.
type	java.lang.String	The type of Attribute. Examples: ▪ "BIN" - Interpreted as binary, however, must be suitably encoded for XML. ▪ "XML" - Interpreted as XML, however, must be Base64 encoded for XML. ▪ "STRING" - Interpreted as a string (default). ▪ "TRANSIENT" - Interpreted as a transient. The e*Insight engine does not process the value but simply return it as-is. ▪ "NUMBER" - Interpreted as a decimal number, however, must be given as a string. ▪ "BOOLEAN" - Interpreted as a boolean, such as "true" and "false".

Name	Type	Description
encoding	java.lang.String	The encoding used for the value. Examples: ▪ "base64' - Standard MIME Base64 encoding. ▪ Null if plain text.
location	java.lang.String	The location of the actual data associated with the Attribute value. Examples: ▪ "FILE" - Attribute value specifies a file where actual data exists. ▪ "DB" - Attribute value is a reference to a row in a database table. ▪ "URL" - Attribute value specifies a URL of where actual data exists. ▪ "EMBEDDED" - Attribute value is the actual data (default). ▪ "AUTO" - Attribute value is actual data but storage in e*Insight is automatically determined.

Return Values

None.

Throws

None.

Example

```
getInstance().getBP_EVENT().getACTIVITY.setATTRIBUTE("In_Stock","STRI
NG","yes");
```

setID

Syntax

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

Description

setID sets the internal unique identifier created by e*Insight for this Activity in the Business Process.

Parameters

Name	Type	Description
val	string	The unique internal identifier created for this Activity.

Return Values

None.

Throws

None.

Example

```
setID("12345");
```

setNAME

Syntax

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

Description

setNAME sets the case-sensitive name of this Business Process Activity.

Parameters

Name	Type	Description
val	string	The case-sensitive name of this ACTIVITY object.

Return Values

None.

Throws

None.

Example

```
setNAME("Ship_Order");
```

toString

Syntax

```
java.lang.String toString()
```

Description

toString converts this ETD object to a printable String form.

Parameters

None.

Return Values

java.lang.String
Returns the XML message represent by this ETD object.

Throws

None.

Example

```
toSTRING();
```

unmarshal

Syntax

```
void unmarshal(org.xml.sax.InputSource inputSource,
com.stc.jcsre.sml.SAXLexer lexer)
```

Description

unmarshal takes a serialized (marshalled) form of the ACTIVITY XML Event and distributes (unmarshals) the data into this ETD object.

Parameters

Name	Type	Description
inputSource	org.xml.sax.InputSource	The input source for the serialized data.
lexer	com.stc.jcsre.sml.SAXLexer	The SAX lexer (parser) to distribute the data.

Return Values

None.

Throws

org.xml.sax.SAXException - thrown when the data cannot be parsed

com.stc.jcsre.UnmarshalException - throw when the data cannot be unmarshalled

15.2 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:

```
<!--Business 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 on page 231
getLOCATION on page 232
getNAME on page 233
getTYPE on page 234
getVALUE on page 235
hasENCODING on page 236
hasLOCATION on page 237
marshal on page 238
omitENCODING on page 239
omitLOCATION on page 240
setENCODING on page 241
setLOCATION on page 242
setNAME on page 243
setTYPE on page 244
setVALUE on page 245
toString on page 246
unmarshal on page 247

getENCODING

Syntax

```
java.lang.String getENCODING()
```

Description

getENCODING retrieves the encoding algorithm for the data contained in the Business Process or Activity Attribute. Currently, only the **base64** algorithm is supported. If not defined, clear-text is assumed.

Parameters

None.

Return Values

java.lang.String
Returns the encoding algorithm used on the data.

Throws

None.

Example

```
getENCODING();
   => "base64"
```

getLOCATION

Syntax

```
java.lang.String getLOCATION()
```

Description

getLOCATION retrieves the location type of where the data for an Attribute is actually stored. In cases where the data is too long to be stored in standard database column, it can be stored in another table where the column can be defined as a **LONG RAW** for example, or it may be stored in a file on some file system. In such cases, a reference to the actual data location is stored as the data for the Attribute.

Parameters

None.

Return Values

java.lang.String

Returns the location type for the Attribute data. This is one of the following values:

"FILE"	Attribute data is the name of a file where actual data is stored.
"DB"	Attribute data is a reference such as **ROWID** to a row in a table.
"URL"	Attribute data is the URL to where the actual data is stored.
"EMBEDDED"	Attribute data is the actual data (this is the default).
"AUTO"	The actual data storage location is automatically determined by the e*Insight engine.

Throws

None.

Example

```
getLOCATION();
   => "EMBEDDED"
```

getNAME

Syntax

```
java.lang.String getNAME()
```

Description

getNAME retrieves the name of the Business Process or Activity Attribute.

Parameters

None.

Return Values

java.lang.String
Returns the name of the Attribute.

Throws

None.

Example

```
getInstance().getBP_EVENT().getNAME();
   => "ProcessOrder"
```

getTYPE

Syntax

```
java.lang.String getTYPE()
```

Description

getTYPE Retrieves the type of data stored in the Attribute.

Parameters

None.

Return Values

java.lang.String

Returns the type of data stored as one of the following values:

"BIN"	Attribute data is binary in nature. However, must be safely encoded for XML.
"XML"	Attribute data represents a XML message. However, must be base64 encoded.
"STRING"	Attribute data appears as clear-text string (this is the default).
"TRANSIENT"	Attribute data is not persisted in the e*Insight database; is passed-through.
"NUMBER"	Attribute data represents a decimal numeric string.
"BOOLEAN"	Attribute data represents a boolean string such as "true" or "false".

Throws

None.

Example

```
getInstance().getBP_EVENT().getTYPE();
   => "STRING"
```

getVALUE

Syntax

```
java.lang.String getVALUE()
```

Description

getVALUE retrieves the value of the Business Process or Activity Attribute.

Parameters

None.

Return Values

java.lang.String
Returns the value of the Attribute.

Throws

None.

Example

```
getVALUE();
   => "yes"
```

hasENCODING

Syntax

```
boolean hasENCODNG()
```

Description

hasENCODING checks if the encoding algorithm is defined for this ATTRIBUTE object.

Parameters

None.

Return Values

boolean
Returns true if the encoding algorithm exists; otherwise returns false.

Throws

None.

Example

```
hasENCODING();
   => true
```

hasLOCATION

Syntax

```
boolean hasLOCATION()
```

Description

hasLOCATION checks if the location is defined for this ATTRIBUTE object.

Parameters

None.

Return Values

boolean
Returns true if location exists, otherwise returns false.

Throws

None.

Example

```
hasLOCATION();
   => true
```

marshal

Syntax

```
void marshal(org.xml.sax.ContentHandler handler,
org.xml.sax.ErrorHandler errorHandler)
```

Description

marshal gathers the data contained within this ETD object and formulates it back into a serialized XML message.

Parameters

Name	Type	Description
handler	org.xml.sax.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: ▪ "FILE" - Attribute data is the name of a file where actual data is stored. ▪ "DB" - Attribute data is a reference such as "ROWID" to a row in a table. ▪ "URL" - Attribute data is the URL to where the actual data is stored. ▪ "EMBEDDED" - Attribute data is the actual data (this is the default). ▪ "AUTO" - The actual data storage location is automatically determined by the e*Insight engine.

Return Values

None.

Throws

None.

Example

```
setLOCATION("EMBEDDED");
```

setNAME

Syntax

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

Description

setNAME sets the name of the Business Process or Activity Attribute.

Parameters

Name	Type	Description
val	java.lang.String	The name of the Attribute.

Return Values

None.

Throws

None.

Example

```
setNAME("In_Stock");
```

setTYPE

Syntax

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

Description

setTYPE sets the type of data stored in the Attribute.

Parameters

Name	Type	Description
val	java.lang.String	The type of data stored. This can take one of the following values: ▪ "BIN" - Attribute data is binary in nature. However, must be safely encoded for XML. ▪ "XML" - Attribute data represents a XML message. However, must be base64 encoded. ▪ "STRING"- Attribute data appears as clear-text string (this is the default). ▪ "TRANSIENT" - Attribute data is not persisted in the e*Insight database; is passed-thru. ▪ "NUMBER" - Attribute data represents a decimal numeric string. ▪ "BOOLEAN" - Attribute data represents a boolean string such as "true" or "false".

Return Values

None.

Throws

None.

Example

```
setTYPE("STRING");
```

setVALUE

Syntax

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

Description

setVALUE sets the value of the Business Process or Activity Attribute.

Parameters

Name	Type	Description
val	java.lang.String	The value of the Attribute.

Return Values

None.

Throws

None.

Example

```
setVALUE("yes");
```

toString

Syntax

```
java.lang.String toString()
```

Description

toString Converts this ETD object to a printable String form.

Parameters

None.

Return Values

java.lang.String
Returns the XML message represent by this ETD object.

Throws

None.

Example

```
toSTRING();
```

unmarshal

Syntax

```
void unmarshal(org.xml.sax.InputSource inputSource,
com.stc.jcsre.sml.SAXLexer lexer)
```

Description

unmarshal takes a serialized (marshalled) form of the e*Insight XML Event and distributes (unmarshals) the data into this ETD object.

Parameters

Name	Type	Description
inputSource	org.xml.sax.InputSource	The input source for the serialized data.
lexer	com.stc.jcsre.xml.SAXLexer	The SAX Lexer (parser) to distribute the data.

Return Values

None.

Throws

org.xml.sax.SAXException, when the data cannot be parsed.

com.stc.jcsre.UnmarshalException, when the data cannot be unmarshalled.

15.3 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:

```
<!--Business 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:

addATTRIBUTE on page 250
clearATTRIBUTE on page 251
countATTRIBUTE on page 252
getACTIVITY on page 253
getATTRIBUTE_VALUE on page 254
getATTRIBUTE on page 255
getBPI_ID on page 256
getID on page 257
getNAME on page 258
getSTATUS on page 259
getTYPE on page 260
hasACTIVITY on page 261
omitACTIVITY on page 267
omitBPI_ID on page 268
omitID on page 269
omitNAME on page 270
omitSTATUS on page 271
removeATTRIBUTE on page 272
setACTIVITY on page 273
setATTRIBUTE on page 274
setBPI_ID on page 276
setEventInfo on page 277
setID on page 278
setNAME on page 279

hasBPI_ID on page 262
hasID on page 263
hasNAME on page 264
hasSTATUS on page 265
marshal on page 266
setSTATUS on page 280
setTYPE on page 281
toString on page 282
unmarshal on page 283

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.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

omitACTIVITY

Syntax

```
void omitACTIVITY()
```

Description

omitACTIVITY removes the Activity object definition from this Business Process object.

Parameters

None.

Return Values

None.

Throws

None.

Example

```
omitACTIVITY();
```

omitBPI_ID

Syntax

```
void omitBPI_ID()
```

Description

omitBPI_ID removes the internal Business Process ID definition from the Business Process object.

Parameters

None.

Return Values

None.

Throws

None.

Example

```
omitBPI_ID();
```

omitID

Syntax

```
void omitID()
```

Description

omitID removes the user-assigned unique ID definition from the Business Process object.

Parameters

None.

Return Values

None.

Throws

None.

Example

```
omitID();
```

omitNAME

Syntax

```
void omitNAME()
```

Description

omitNAME removes the name definition from the Business Process object.

Parameters

None

Return Values

None.

Throws

None.

Example

```
omitNAME();
```

omitSTATUS

Syntax

```
void omitSTATUS()
```

Description

omitSTATUS removes the status definition from the Business Process object.

Parameters

None.

Return Values

None.

Throws

None.

Example

```
omitSTATUS();
```

removeATTRIBUTE

Syntax

```
void removeATTRIBUTE(java.lang.String name)
void removeATTRIBUTE(int index)
```

Description

removeATTRIBUTE removes a specific global attribute from the Business Process object.

Parameters

Name	Type	Description
name	java.lang.String	The name of the local attribute.
index	int	The index to the list of global attributes (zero-based).

Return Values

None.

Throws

None.

Example

```
removeATTRIBUTE(1);
```

setACTIVITY

Syntax

```
void setACTIVITY(ACTIVITY val)
```

Description

setACTIVITY sets the current Activity of a Business Process.

Parameters

Name	Type	Description
val	ACTIVITY	The current Activity object.

Return Values

None.

Throws

None.

setATTRIBUTE

Syntax

```
void setATTRIBUTE(ATTRIBUTE[] val)
void setATTRIBUTE(int i, ATTRIBUTE val)
void setATTRIBUTE(java.lang.String name java.lang.String value)
void setATTRIBUTE(java.lang.String name, java.lang.String type,
java.lang.String value)
void setATTRIBUTE(java.lang.String name, java.lang.String type,
java.lang.String encoding,java.lang.String value)
void setATTRIBUTE(java.lang.String name, java.lang.String type,
java.lang.String value, java.lang.String encoding,java.lang.String
location)
```

Description

setATTRIBUTE sets a global Attribute of the Business Process.

Parameters

Name	Type	Description
val	ATTRIBUTE[]	The Attribute object.
i	int	The list index of the Attribute to be retrieved (zero-based).
name	java.lang.String	The name of the global Attribute.
value	java.lang.String	The value of the global Attribute.
type	java.lang.String	The type of Attribute. Examples: ▪ "BIN" - Interpreted as binary, however, must be suitably encoded for XML. ▪ "XML" - Interpreted as XML, however, must be Base64 encoded for XML. ▪ "STRING" - Interpreted as a string (default). ▪ "TRANSIENT" - Interpreted as a transient. The e*Insight engine does not process the value but simply return it as-is. ▪ "NUMBER" - Interpreted as a decimal number, however, must be given as a string. ▪ "BOOLEAN" - Interpreted as a boolean, such as "true" and "false".
encoding	java.lang.String	The encoding used for the value. Examples: ▪ "base64' - Standard MIME Base64 encoding. ▪ Null if plain text.

Name	Type	Description
location	java.lang.String	The location of the actual data associated with the Attribute value. Examples: ▪ "FILE" - Attribute value specifies a file where actual data exists. ▪ "DB" - Attribute value is a reference to a row in a database table. ▪ "URL" - Attribute value specifies a URL of where actual data exists. ▪ "EMBEDDED" - Attribute value is the actual data (default). ▪ "AUTO" - Attribute value is actual data but storage in e*Insight is automatically determined.

Return Values

None.

Throws

None.

Example

```
setATTRIBUTE("In_Stock","STRING","yes");
```

setBPI_ID

Syntax

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

Description

setBPI_ID sets the internal Business Process ID used by the e*Insight engine. It must be returned as-is in a "Done" event back to the e*Insight engine when the active mode is enabled.

Parameters

Name	Type	Description
val	java.lang.String	The internal Business Process ID.

Return Values

None.

Throws

None.

Example

```
setBPI_ID("605.0.21");
```

setEventInfo

Syntax

```
void setEventInfo(java.lang.String type, java.lang.String status,
java.lang.String id, java.lang.String name)
void setEventInfo(java.lang.String type, java.lang.String status,
java.lang.String id, java.lang.String name, java.lang.String bpi_id)
```

Description

setEventInfo sets the Type, Status, ID and Name information for the Business Process object.

Parameters

Name	Type	Description
type	java.lang.String	The Type of command represented by this Business Process object.
status	java.lang.String	The Status of the current Activity.
id	java.lang.String	The user-assigned unique ID for this Business Process instance.
name	java.lang.String	The Name of the Business Process.
bpi_id	java.lang.String	The internal Business Process Instance ID.

Return Values

None.

Throws

None.

Example

```
setEventInfo("DO_ACTIVITY","SUCCESS","605","ProcessOrder","605.0.21")
;
```

setID

Syntax

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

Description

setID sets the Business Process user-assigned unique ID (relative to an ERP, for example).

Parameters

Name	Type	Description
val	java.lang.String	The Business Process user-assigned unique ID.

Return Values

None.

Throws

None.

Example

```
setID("12345");
```

setNAME

Syntax

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

Description

setNAME sets the case-sensitive name of the Business Process.

Parameters

Name	Type	Description
val	java.lang.String	The name of the Business Process.

Return Values

None.

Throws

None.

Example

```
setNAME("ProcessOrder");
```

setSTATUS

Syntax

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

Description

setSTATUS sets the status of the current Activity in a Business Process.

Parameters

Name	Type	Description
val	java.lang.String	The status of the current Activity in a Business Process. Examples: ▪ "SUCCESS" - Indicates that the current Activity has successfully completed. ▪ "FAILURE" - Indicates that the current Activity has not successfully completed.

Return Values

None.

Throws

None.

Example

```
setSTATUS("SUCCESS");
```

setTYPE

Syntax

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

Description

setTYPE sets the type of command represented by the Business Process object.

Parameters

Name	Type	Description
val	java.lang.String	The command represented by the Business Process object. Examples: ▪ "START_BP" - Instructs the e*Insight engine to start a Business Process Instance. ▪ "DO_ACTIVITY" - Indicates a "Do" Event for the current Activity. ▪ "UNDO_ACTIVITY" - Indicates an "Undo" Event for the current Activity. ▪ "UNDO_BPI" - Indicates an "Undo" for the entire Business Process Instance. ▪ "RESTART_ACTIVITY" - Indicates restarting the current Activity after it has paused. ▪ "SKIP_ACTIVITY" - Indicates the current Activity should be skipped. ▪ "RELOAD_BP" - Indicates the Business Process definition should be reloaded. ▪ "AUTHORIZE" - Indicates that the current Activity should be authorized. ▪ "DONT_AUTHORIZE" - Indicates that the current Activity should not be authorized.

Return Values

None.

Throws

None.

Example

```
setTYPE("DO_ACTIVITY");
```

toString

Syntax

```
java.lang.String toString()
```

Description

toString converts this ETD object to a printable String form.

Parameters

None.

Return Values

java.lang.String
Returns the XML message to represent by this ETD object.

Throws

None.

Example

```
toSTRING();
```

unmarshal

Syntax

```
void unmarshal(org.xml.sax.InputSource inputSource,
com.stc.jcsre.sml.SAXLexer lexer)
```

Description

unmarshal takes a serialized (marshalled) form of the e*Insight XML Event and distributes (unmarshals) the data into this ETD object.

Parameters

Name	Type	Description
inputSource	org.xml.sax.InputSource	The input source for the serialized data.
lexer	com.stc.jcsre.xml.SAXLexer	The SAX Lexer (parser) to distribute the data.

Return Values

None.

Throws

org.xml.sax.SAXException, when the data cannot be parsed.

com.stc.jcsre.UnmarshalException, when the data cannot be unmarshalled.

15.4 eX_StandardEvent Class

public class eX_StandardEvent

extends com.stc.jcsre.SMLETDImpl

implements com.stc.jcsre.ETD

eX_StandardEvent class is an ETD class used to represent the standard XML message that is used to interchange information with the e*Insight Business Process Manager engine. The DTD is:

```
<!ELEMENT eX_Event (BP_EVENT?, TP_EVENT?)>
<!--Business 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 286
getBP_EVENT on page 287
getTP_EVENT on page 288
hasBP_EVENT on page 289
hasTP_EVENT on page 290
marshal on page 291
omitBP_EVENT on page 292
omitTP_EVENT on page 293
setBP_EVENT on page 294
setTP_EVENT on page 295
to_eBPMConvert on page 296
toString on page 297
unmarshal on page 298

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 16

e*Insight User Activity API Methods

This chapter provides information on the e*Insight User Activity API methods. 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 or from a business process, from the e*Insight database. The e*Insight engine uses the returned value of the attributes to continue the business process.

16.0.1 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 sessions, 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*Insight 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*Insight Administrator.*

The User Activity methods are contained in:

- **"Imessage Interface" on page 301**
- **"UserActivityMessage Class" on page 324**
- **"IClient Interface" on page 325**
- **"EbpmMonitor Class" on page 363**

16.0.2 Defining the Classpath

In order to use the User Activity API methods, you must define the following files in your classpath:

activation.jar	DGutil.jar	stcjcs.jar
antlrall.jar	eIX_StandardEvent.jar	workflow.jar
classes12.zip	jconn2.jar	xerces.jar
DGbase.jar	js.jar	xml.jar
DGsqlserver.jar	mail.jar	
DGsybase.jar	soap.jar	

These files are all located in **<eInsight>\Integrator**.

16.1 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”
```

Note: *This method returns a string parameter, but the original source is an integer so the method converts from one data type to another.*

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 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

```
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 type of Attribute. Examples: ▪ "BIN" - Interpreted as binary, however, must be suitably encoded for XML. ▪ "XML" - Interpreted as XML, however, must be Base64 encoded for XML. ▪ "STRING" - Interpreted as a string (default). ▪ "TRANSIENT" - Interpreted as a transient. The e*Insight engine does not process the value but simply return it as-is. ▪ "NUMBER" - Interpreted as a decimal number, however, must be given as a string. ▪ "BOOLEAN" - Interpreted as a boolean, such as "true" and "false".
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 type of Attribute. Examples: ▪ "BIN" - Interpreted as binary, however, must be suitably encoded for XML. ▪ "XML" - Interpreted as XML, however, must be Base64 encoded for XML. ▪ "STRING" - Interpreted as a string (default). ▪ "TRANSIENT" - Interpreted as a transient. The e*Insight engine does not process the value but simply return it as-is. ▪ "NUMBER" - Interpreted as a decimal number, however, must be given as a string. ▪ "BOOLEAN" - Interpreted as a boolean, such as "true" and "false".

Return Values

None.

Throws

None.

Example

```
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. Possible values include: ▪ START_BP ▪ DO_ACTIVITY ▪ UNDO_ACTIVITY ▪ UNDO_BPI ▪ RESTART_ACTIVITY ▪ SKIP_ACTIVITY ▪ RELOAD_BP ▪ AUTHORIZE ▪ DONT_AUTHORIZE

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. Possible values include: ▪ SUCCESS ▪ FAILURE

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>
```

16.2 UserActivityMessage Class

Implements IMessage.

See **"Imessage Interface" on page 301**.

16.3 IClient Interface

These methods are described in detail on the following pages:

getActivityGlobalAttributeNames on page 328
getActivityInstanceEndTime on page 329
getActivityInstanceStartTime on page 330
getActivityInstanceStatus on page 331
getActivityNames on page 332
getAssignedBPIIdByState on page 333
getAuthorizationActivityNames on page 334
getBPIStack on page 335
getBusinessModelInstancesIds on page 336
getBusinessModelInstanceName on page 337
getBusinessModelInstanceStatus on page 338
getBusinessModelName on page 339
getEnabledBusinessModelId on page 340
getEnabledBusinessModelsIds on page 341
getGlobalAttributeDefaultValue on page 342
getGlobalAttributeDirection on page 343
getGlobalAttributeNames on page 344
getGlobalAttributeType on page 345
getGlobalAttributeValue on page 346
getLocalAttributeNames on page 347
getLocalAttributeType on page 348
getLocalAttributeValue on page 349
getMessageStatus on page 350
getUser on page 351
getUserActivityNames on page 352
getUUID on page 353
initialize on page 354
refreshCachedMemory on page 355
releaseActivityInstance on page 356
resetUser on page 358
sendMessage on page 359
setGlobalAttributeValue on page 360
setLocalAttributeValue on page 361
setUser on page 362

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 activity name.

Return Values

java.lang.String

Returns the userId who is using this activity 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 as one of the following values:

aborted	undofailed
done	running
failed	waiting
rollingBack	neverInvoked
rolledback	timedout
rollbackFailed	expired
undoing	pending
undone	

Throws

java.lang.Exception

Example

```
getActivityInstanceStatus("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
businessModelID)
```

Description

getBusinessModelName gets the business model name.

Parameters

Name	Type	Description
businessModelID	java.lang.String	The business model ID (bpoid).

Return Values

java.lang.String

Returns the business model name.

Throws

java.lang.Exception

Example

```
getBusinessModelName(“127“);
    => “ProcessOrder”
```

Note: *This method returns a string parameter, but the original source is an integer so the method converts from one data type to another.*

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. For example: dbURL=jdbc:oracle:thin:@hostname:1521:dbname userID=ex_admin password=xxxxx driverName=oracle.jdbc.driver.OracleDriver DBServerType=Oracle

Return Values

None.

Throws

java.lang.Exception

Example

```
Properties p = new Properties();
p.setProperty("dbURL","jdbc:oracle:thin:@localhost:1521:eXchange");
p.setProperty("userID","ex_admin");
p.setProperty("password","ex_admin");
p.setProperty("driverName","oracle.jdbc.driver.OracleDriver");
p.setProperty("DBServerType","Oracle");
initialize( p );
```

refreshCachedMemory

Syntax

```
void refreshCachedMemory()
```

Description

refreshCachedMemory refreshes the cached memory used by this client.

Parameters

None.

Return Values

None.

Throws

java.lang.Exception

Example

```
refreshCachedMemory();
```

releaseActivityInstance

Syntax

```
void releaseActivityInstance(java.lang.String businessModelId,
java.lang.String businessModelInstanceId, java.lang.String
activityName)
```

Description

releaseResources releases the usage on the activity instance by this current userId.

Parameters

Name	Type	Description
businessModelId	java.lang.String	The business model id.
businessModelInstanceId	java.lang.String	The business model instance id.
activityName	java.lang.String	The activity 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");
```

16.4 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 325, for details of methods contained in this interface.

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

e*Insight Schema Components (eISchema)

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 you must perform, the Collaborations it uses, and what is contained in the Events it processes.

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

The e*Insight Schema is the e*Gate schema that implements a particular e*Insight installation. The starting point for a working e*Gate Schema for e*Insight is the e*Gate schema called **eISchema** created when you install the e*Gate Schema for e*Insight from the installation CD. This schema contains a number of pre-configured and partially pre-configured e*Gate components used by e*Insight.

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

16.4.2 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.

16.5 e*Insight Schema Components Overview

Table 38 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.
- **More Information**—A cross reference to the section that describes this component in detail.

Additional Components

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

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

The above components are provided to enable you to upgrade your schema to use the engine provided with eIJSchema. For more information, see **"e*Insight Schema Components (eIJSchema)" on page 28**. You can delete these components if you do not want to use them.

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

Table 38 e*Insight Schema Component Types

Component	Description	In Default eISchema?	More Information
eX_eBPM Engine	This is a specially configured Java e*Way Connector that runs business processes.	Yes	16.6.1 on page 368
eI_Resubmitter BOB	A placeholder component used in the e*Insight Event failure handling process.	Yes	16.6.2 on page 373
Start BP Component	Either an e*Way or BOB that sends the Event that starts a business process instance.	No	16.6.4 on page 375
eX_*Activity* e*Way	Implements an e*Insight activity that connects to an external system.	No	16.6.5 on page 377
eX_*Activity* BOB	Implements an e*Insight activity that does not connect to an external system.	No	16.6.6 on page 380

16.5.1 e*Insight Schema Component Relationships Diagram

Figure 105 on the next page illustrates the relationships among the e*Insight schema components. Not every e*Insight implementation uses all of these components. Some of the components shown are not provided as part of the e*Gate Schema for e*Insight installation. These components are shown in light blue and must be added to the base e*Insight schema, as needed.

Figure 104 e*Insight Overview Legend

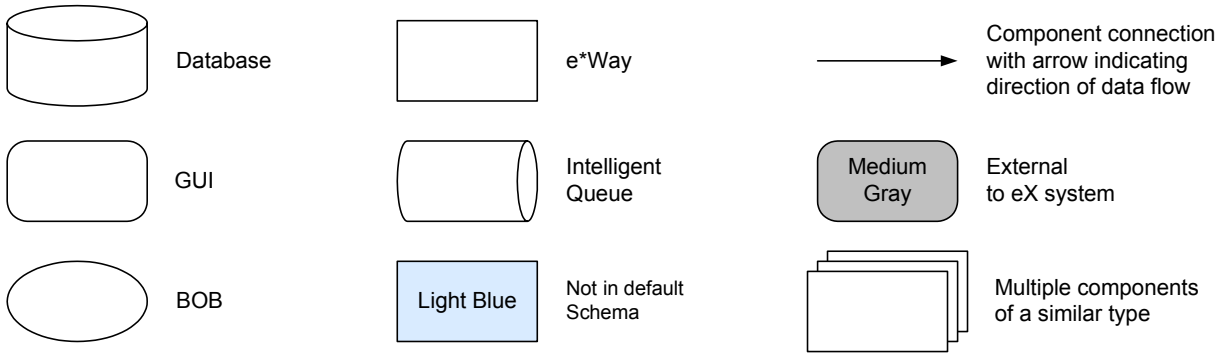
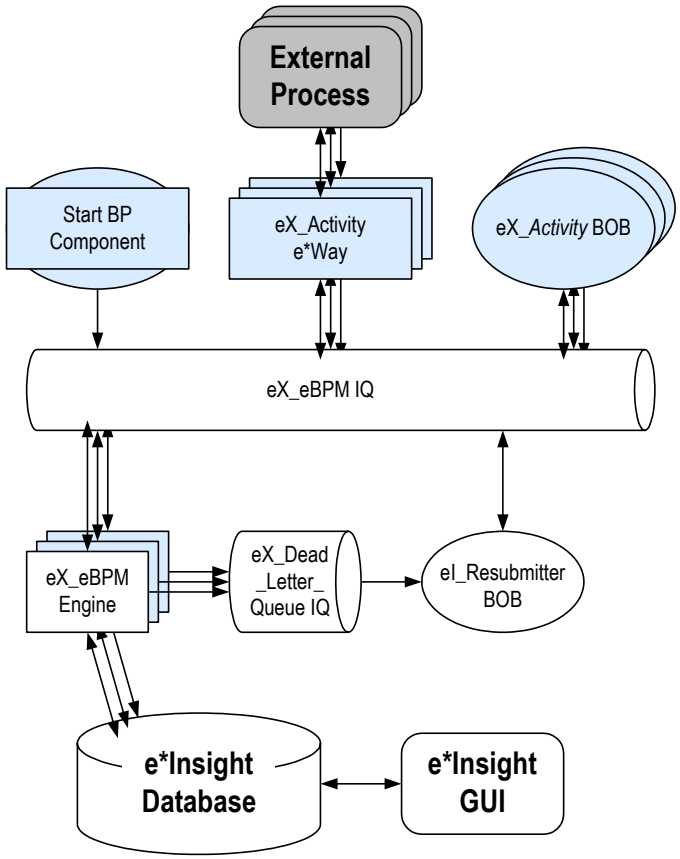


Figure 105 e*Insight Components



16.6 e*Insight Business Process Manager Components

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

Components That Run Business Processes

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

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

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

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

Components that Start Business Processes

The component that starts a business process is:

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

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

Components that Implement Business Process Activities

The components that implement business process activities are:

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

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

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

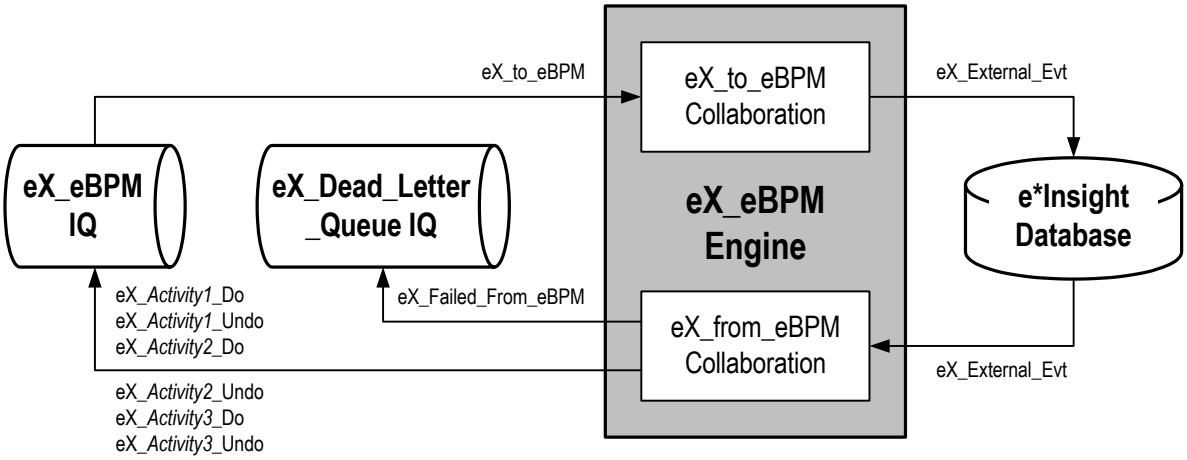
16.6.1 e*Insight Engine

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

For more information on using multiple engines, see **"Using Multiple e*Insight Engines (eISchema)" on page 194**.

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 106.

Figure 106 eX_eBPM Engine Detail



Configuring the e*Insight Engine

The e*Insight engine requires only minimal configuration on the part of the user. Table 39 lists those parameters in the engine's configuration file that the user can change.

Table 39 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 machine. ***Note:*** *In a UNIX environment the name and location of this file is different; for example,* ***/usr/Solaris_JDK_1.2.2_05a/jre/sparc/libjvm.sol***
	Enable Custom Error Handling	The default value is "YES". Change the value to "NO" if you do not want custom error handling enabled. See the **eI_Resubmitter** BOB below for more information.
	(Others)	(Default)

Table 39 e*Insight Engine Configuration Settings (Continued)

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

Table 39 e*Insight Engine Configuration Settings (Continued)

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

Table 39 e*Insight Engine Configuration Settings (Continued)

Screen	Parameter	Setting
	Maximum Instance Cache Size	Enter the maximum number of Business Process Instances that the e*Insight engine caches in memory. When the maximum size is reached, the Engine first removes the Least Recently Used (LRU) Instance from the cache. Entering a value of -1 means that there is no limit to the number of Instances kept in memory. The value entered for this parameter effects the total amount of memory used by the engine. Limit the number of instances if you start getting out of memory messages when running the engine. Note: A value of zero (0) should NOT be used.
	Instance Caching	Instance Caching is the most efficient way to process Business Process Instances.Setting this value to YES keeps a cache of the instance information throughout the life span of the Business Process Instance. Setting this value to NO retrieves the information from the database instead. This allows more flexibility and fault tolerance at the cost of performance. To improve performance it is recommended to set this parameter to YES and use multiple e*Insight engines. To use both instance caching and multiple engines it is necessary to ensure that a single instance is always processed by the same engine. This is achieved by using engine affinity. For information using Instance Caching with multiple engines see **"e*Insight Engine Affinity (eISchema)" on page 195**.
	Business Processes to Preload	This parameter allows you to load all or a subset of all the business processes stored in the e*Insight database. The default is ALL.
	Use Default Locale Encoding	Setting this value to YES specifies that all data received by and sent from this e*Insight Engine is encoded in the same language as the default Locale setting of this Participating Host. Setting this value to NO specifies that data is being passed to and from this e*Insight Engine in an UTF-8 encoding format.

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 379** 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 380** 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.

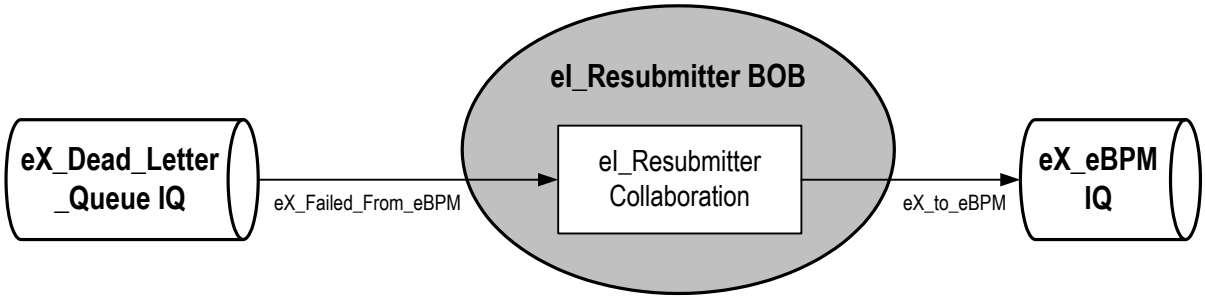
Published Event Type: eX_External_Evt

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

16.6.2 eI_Resubmitter BOB

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

Figure 107 eI_Resubmitter BOB Detail



Configuring the eI_Resubmitter BOB

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

eI_Resubmitter Collaboration

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

Subscribed Event Type: eX_Failed_From_eBPM

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

Published Event Type: eX_to_eBPM

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

16.6.3 Failed Event Handling by the e*Insight Engine

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

Error Types

Connection errors

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

Data errors

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

Error Handling

Normal Event Failure Handling

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

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

Special Event Failure Handling

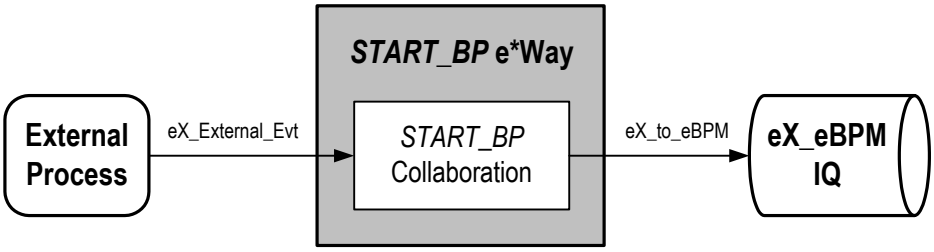
When custom error handling is enabled in the e*Insight engine's configuration file (as it is by default) Events that fail to process due to data errors are handled in a special way. Events that fail to process due to communication errors are not affected by custom error handling. When the e*Insight engine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

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

16.6.4 START_BP Component

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

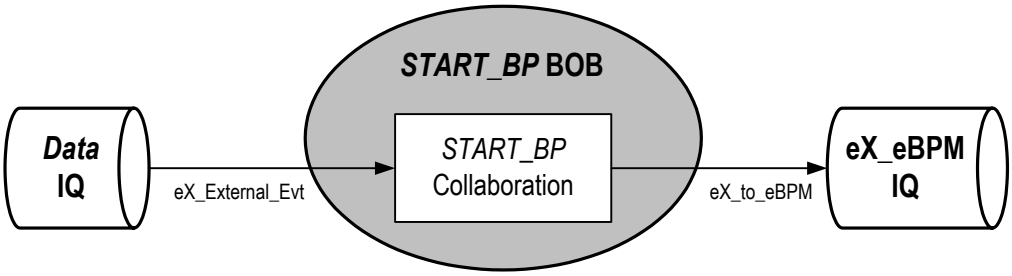
Figure 108 START_BP e*Way Detail



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 109.

Figure 109 START_BP BOB Detail



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

Configuring the START_BP Component

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

START_BP Collaboration

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

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

Start BP Nodes in the e*Insight Standard Event

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

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

Converting Input Data to e*Insight Format

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

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

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

See **"Starting a Business Process (eIJSchema)" on page 393** 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.

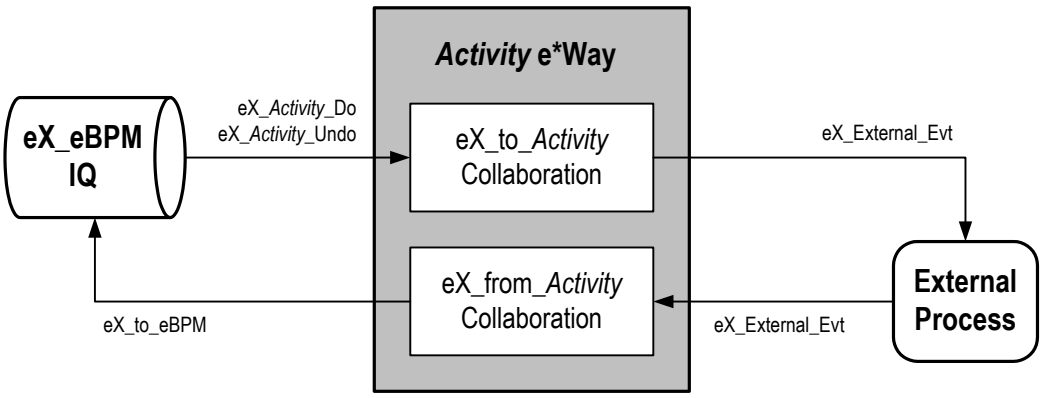
Subscribed Event Type: eX_External_Evt

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.

16.6.5 Activity e*Way

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

Figure 110 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 your preference.

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

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

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

Configuring the eX_to_*Activity* e*Way

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

eX_to_*Activity* Collaboration

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

Important: *In addition to passing the attribute data it receives from the e*Insight engine to the external system, the **eX_to_Activity** Collaboration must preserve the e*Insight Business Process Instance tracking information contained in the **eX_Activity_Do** or **eX_Activity_Undo** Events. This information is used to send the return or "Done" Event back to the e*Insight engine, when the activity completes. See* **Sending the "Done" Event Back to e*Insight (eIJSchema)** *on page 395* *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. You may 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 59**.

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 379** for an explanation of "undoing" an activity). This Event Type is in standard e*Insight format, and contains the current values of any global variables designated as "Input" by the activity in the appropriate location in the **eX_Standard_Event.ssc** ETD. Also, the **eX_*Activity*_Undo** Event contains any local variables set by the Collaboration executing the "Do" logic associated with this activity.

Published Event Type: eX_External_Evt

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

eX_from_*Activity* Collaboration

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

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

See **Sending the "Done" Event Back to e*Insight (eIJSchema)** on page 395 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

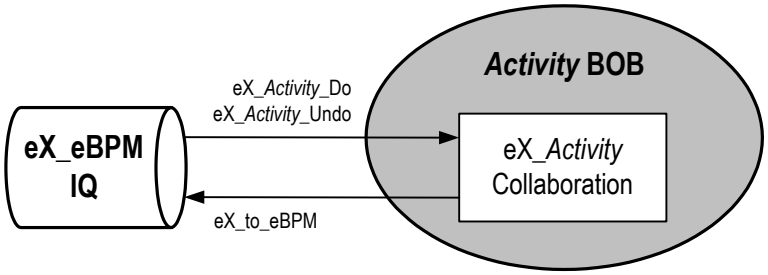
This is the "Done" Event that carries the data from a completed activity back to the e*Insight engine.

16.6.6 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 111 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 379** 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 380** 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 380** for more information.

Appendix B

The e*Insight ETD for Monk

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 eX_Standard_Event.ssc Monk Event Type Definition is structured.

16.6.7 ETD Structure

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

Table 40 lists these *facilitator* nodes.

Table 40 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.

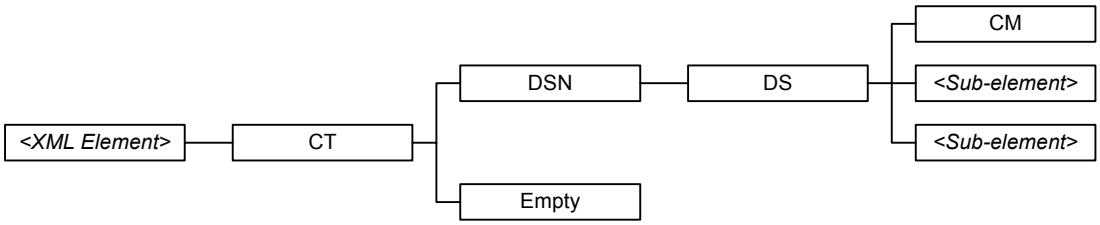
The facilitator nodes always occur in a set order and define the structure of the XML message. In the e*Insight ETD, the facilitator nodes define three types of branches:

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

16.6.8 XML Element with Sub-elements

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

Figure 112 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:

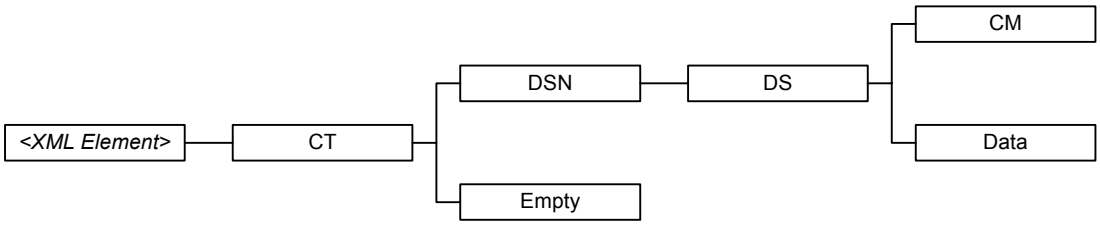
- **CM**, which holds XML comments for the element
- ***<sub-element>***, the name of a sub-element of the parent element

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.

16.6.9 XML Element without sub-elements

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

Figure 113 XML Element without sub-elements

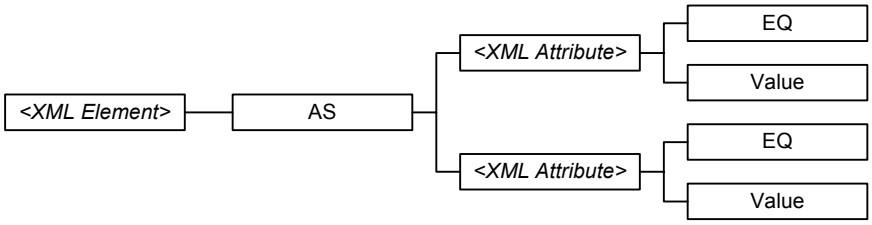


Notice that the only difference between this diagram and the previous diagram is a **Data** child node in place of the ***<sub-element>*** child nodes above. The **Data** node contains the actual data for the XML element that is defined. When creating Collaboration Rules scripts, you must map the XML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

16.6.10 XML Attribute

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

Figure 114 XML Attribute

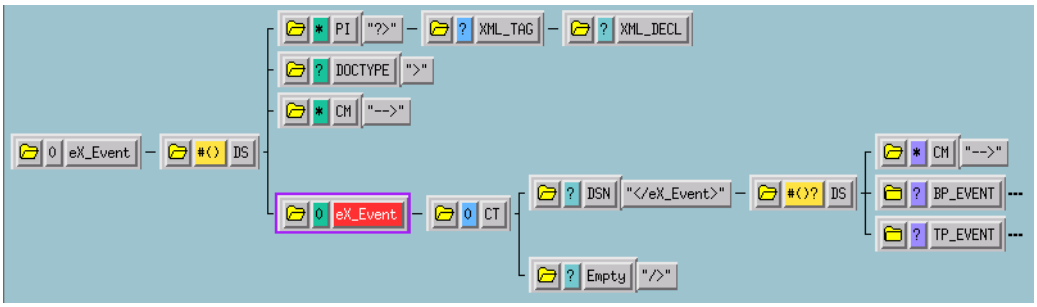


In this case, the XML element contains one child node, **AS**, which identifies the branch as XML attributes of the parent element. The **AS** node contains the ***<XML Attribute>*** nodes as child nodes. Each ***<XML Attribute>*** node has two child nodes: **EQ** to represent the equal sign (=) in the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

16.6.11 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 **"The e*Insight ETD for Monk" on page 382**.

Figure 115 The e*Insight ETD



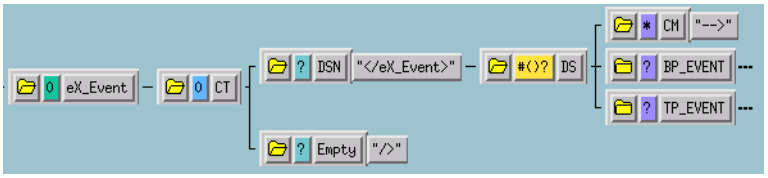
All data pertinent to e*Insight is contained in the XML element **eX_Event**. **eX_Event** contains two distinct "trees": **BP_EVENT** and **TP_EVENT**. **BP_EVENT** contains all of the information pertaining to e*Insight. **TP_EVENT** contains all of the information pertaining to e*Xchange. Both **BP_EVENT** 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 116 shows the ETD structure for this element.

Figure 116 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**.

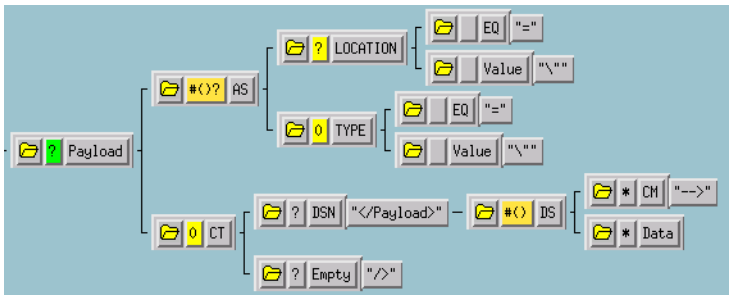
The **DS** node always contains a **CM** child node to hold XML comments. In this example, the eX_Event element does not hold data directly, but contains two sub-elements—**BP_EVENT** and **TP_EVENT**—which have similar facilitator node branches associated with them.

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 117 shows the ETD structure for these attributes.

Figure 117 XML Attribute **Type**



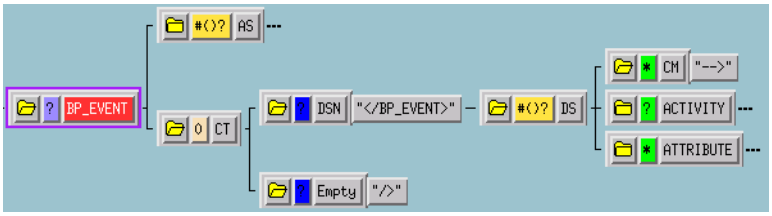
16.7 Using eX_Standard_Event.ssc

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

16.7.1 BP_EVENT

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

Figure 118 BP_EVENT



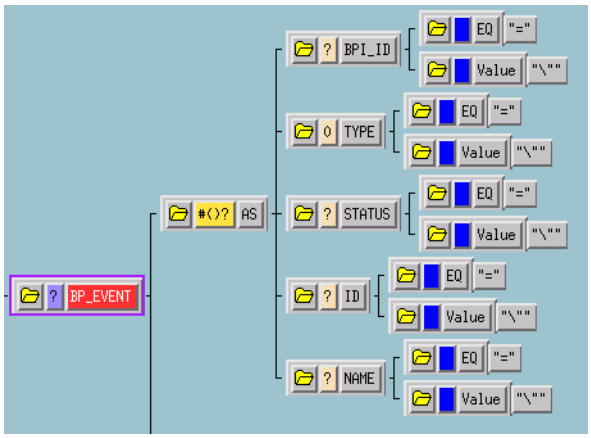
Three of the nodes shown in Figure 118 are collapsed (followed by three dashes) indicating there are additional nodes underneath these nodes. Each of these nodes contains a different type of information pertinent to e*Insight.

- **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 119.

Figure 119 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.

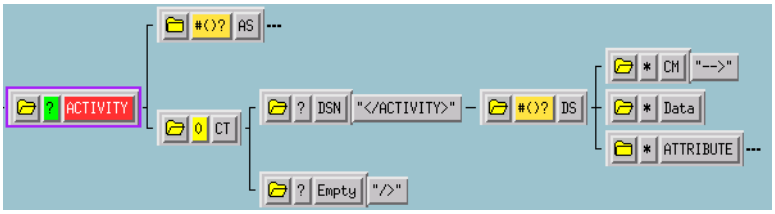
NAME

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

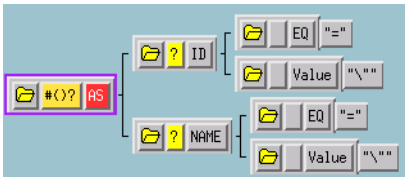
Figure 120 BP_EVENT.CT.DSN.DS.ACTIVITY



ACTIVITY.AS Nodes

This location in the e*Insight ETD contains ID information about the current activity in two nodes as shown in Figure 121.

Figure 121 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

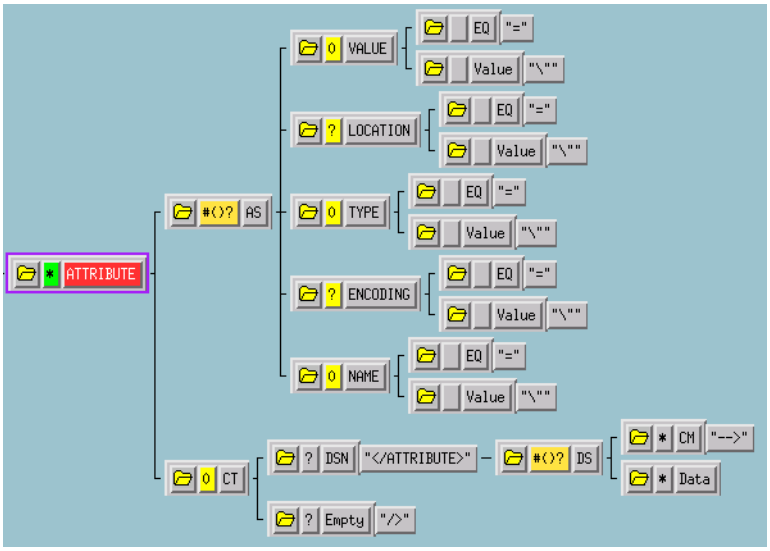
This repeating node structure contains the local attribute information defined for the current activity. The structure itself is exactly the same as the global attribute node

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 122:

Figure 122 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.

Appendix C

Common Configuration Tasks

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

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

16.8 Copy the e*Insight Schema

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

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

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

Using the e*Insight GUI

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

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

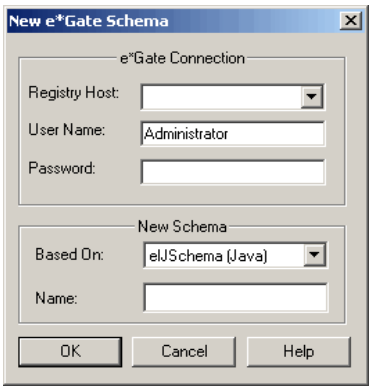
To create a new e*Insight Schema using the e*Insight GUI

1 Open the e*Insight GUI.

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

The New e*Gate Schema dialog appears. See Figure 123.

Figure 123 New e*Gate Schema Dialog



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

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

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

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

7 Click **OK**.

Copying the Schema from the Registry Host

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

To create a copy of the e*Insight Schema

1 Open the eISchema in the e*Gate Schema Designer.

A Start the e*Gate Enterprise Manager.

B Log in to eIJSchema or eISchema.

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

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

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

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

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

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

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

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

E Click **Open**.

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

Installing from the CD

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

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

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

16.9.1 Starting a Business Process (eIJSchema)

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

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

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

The business process name, unique ID, and "START_BP" event type are set by setting a value in the relevant node. The three nodes required to start the BPI populated in the **eI_StandardEvent.xsc** ETD are described in Table 41.

Table 41 Configuring a start message

eI_StandardEvent	How to populate
BP_EVENT.NAME	Must be filled with the exact name of the BP as it appears in the e*Insight GUI.

Table 41 Configuring a start message

eI_StandardEvent	How to populate
BP_EVENT.ID	Must be filled with a unique ID (for example, a timestamp).
BP_EVENT.TYPE	Must be filled with the string “START_BP”.

For information on setting global attribute values, see **“Setting Attributes (eIJSchema)” on page 394**.

16.9.2 Setting Attributes (eIJSchema)

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

If you are using an Activity Specific ETD then you can use the attribute nodes within the Global and Local nodes to set attribute values. The underlying Java code is a set method that has been created for that particular attribute. For example, if you have a global attribute named **Bonus**, then the method is **setBonus**. The method requires the attribute value as the only parameter. The data type of that parameter depends on the attribute type specified in e*Insight.

```
EIActivity1Out().getGlobalAttributes().setBonus(9000);
EIActivity1Out().getGlobalAttributes().setComments("Generous bonus");
EIActivity1Out().getGlobalAttributes().setEligible(true);
EIActivity1Out().getLocalAttributes().setBonus(9000);
```

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

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

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

```
getEIActivty1Out().getEI_StandardEvent().getBP_EVENT().setAttribute("
Bonus","NUMBER","9000");

getEIActivty1Out().getEI_StandardEvent().getBP_EVENT().getActivity().
setAttribute("Bonus","NUMBER","9000");
```

Note: *An Event Type Definition can be used to define the structure of XML data. This Event Type Definition must be created from a .xsd (XML Schema Definition) or .dtd (Document Type Definition) file. In order to create these Event Type Definitions you must install the XML ETD Builders with e*Gate.*

16.9.3 Getting Attributes (eIJSchema)

You may need to 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.

If you are using an Activity Specific ETD then you can use the attribute nodes within the Global and Local nodes on the inbound ETD, to retrieve attribute values. The underlying Java code is a get method that has been created for that particular attribute. For example, if you have a global attribute named **Bonus**, then the method is **getBonus**. The method requires no parameters. The data type of the return value depends on the attribute type specified in e*Insight.

```
EIActivity1Out().getGlobalAttributes().getBonus();
EIActivity1Out().getLocalAttributes().getBonus();
```

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

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

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

Note: *When using the Activity Specific ETD, the User Activity and Authorization Activity do not require additional coding to produce a "Done" Event to be sent back to the e*Insight engine. This is automatically configured for you.*

The "Done" Event must have the following nodes in **eI_StandardEvent** (Java) ETD populated. Table 42 describes the required nodes when an activity is set to Active control and Table 43 describes the required nodes when an activity is set to Passive control.

Table 42 Done Event in Active control mode\

Java Node Location	How to populate
BP_EVENT.BPI_ID	Copy from source to destination.
BP_EVENT.ID	Copy from source to destination.
BP_EVENT.NAME	Copy from source to destination.
BP_EVENT.TYPE	Copy from source to destination.
BP_EVENT.STATUS	Copy either the string "SUCCESS" or "FAILURE", depending on whether or not the activity completed successfully.

Java Node Location	How to populate
BP_EVENT.ACTIVITY.ID	Copy from source to destination. (Not absolutely required, but recommended to speed processing.)
BP_EVENT.ACTIVITY.NAME	Copy from source to destination.

Note: *With the exception of the eventStatus node, you can copy the EventSpecifics source node to the EventSpecifics destination node instead of copying each field individually. You must set the eventStatus node individually.*

Table 43 Done Event in Passive control mode

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

In addition, the "Done" Event must carry with it the values for any attributes (global or local) specified as "Output" or "Input/Output". These can be set as shown in **"Setting Attributes (eIJSchema)" on page 394**.

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

16.10.1 Starting a Business Process (eISchema)

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

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

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

The business process name, unique ID, and "START_BP" event type are set by setting a value in the relevant node. The three nodes required to start the BPI populated in the **eX_Standard_Event.ssc** (Monk), or **eI_StandardEvent.xsc** (Java) ETDs are described in Table 41.

Table 44 Configuring a start message (eISchema)

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

For information on setting global attribute values, see **"Setting Attributes (eIJSchema)" on page 394**.

16.10.2 Setting Attributes (eISchema)

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

Setting Attributes in a Monk Collaboration

The Monk Collaboration must do one of the following:

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

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

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

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

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

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

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

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

- replace **<*root-path*>** with **~input%eX_Event** or **~output%eX_Event**
- replace **<*attribute*>** with the exact name of the attribute as it appears in the e*Insight GUI
- replace **<*value*>** with the attribute value
- replace **<*type*>** with one of the strings “STRING”, “XML”, or “BIN” as appropriate

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

Setting Attributes in a Java Collaboration

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

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

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

16.10.3 Getting Attributes (eISchema)

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

Getting Attributes in a Monk Collaboration

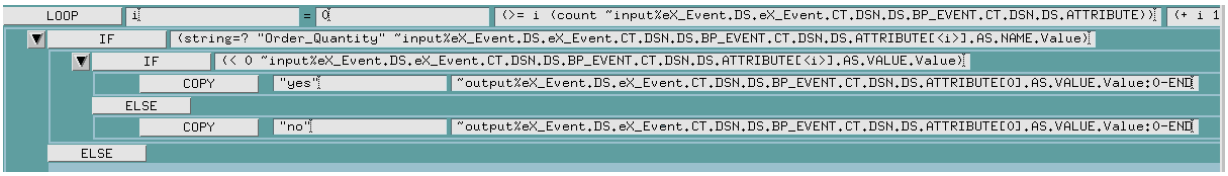
This Collaboration must do one of the following:

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

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

To get the value from the correct node, your Collaboration must systematically search through all of the attribute nodes until it finds the one containing the value for the required attribute. Figure 124 shows an example of a LOOP rule that does this.

Figure 124 Get Attribute Loop



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

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

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

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

The "Done" Event must have the following nodes in either the **eX_Standard_Event.ssc** (Monk), or **eI_Standard_Event** (Java) ETD populated. Table 42 describes the required nodes when an activity is set to Active control and Table 43 describes the required nodes when an activity is set to Passive control.

Table 45 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 46 Done Event in Passive control mode

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

In addition, the "Done" Event must carry with it the values for any attributes (global or local) specified as "Output" or "Input/Output". These can be set as shown in **"Setting Attributes (eIJSchema)" on page 394**.

Appendix D

e*Insight Authorization Activity Implementation (eISchema)

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

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

This case study is a continuation of the previous example. See **"e*Insight Implementation (eISchema)" on page 413** for the initial configuration instructions.

16.11 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.

16.11.1 Case Study: Order Processing

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

Figure 125, 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 414**.

Figure 125 e*Insight Data Flow Diagram

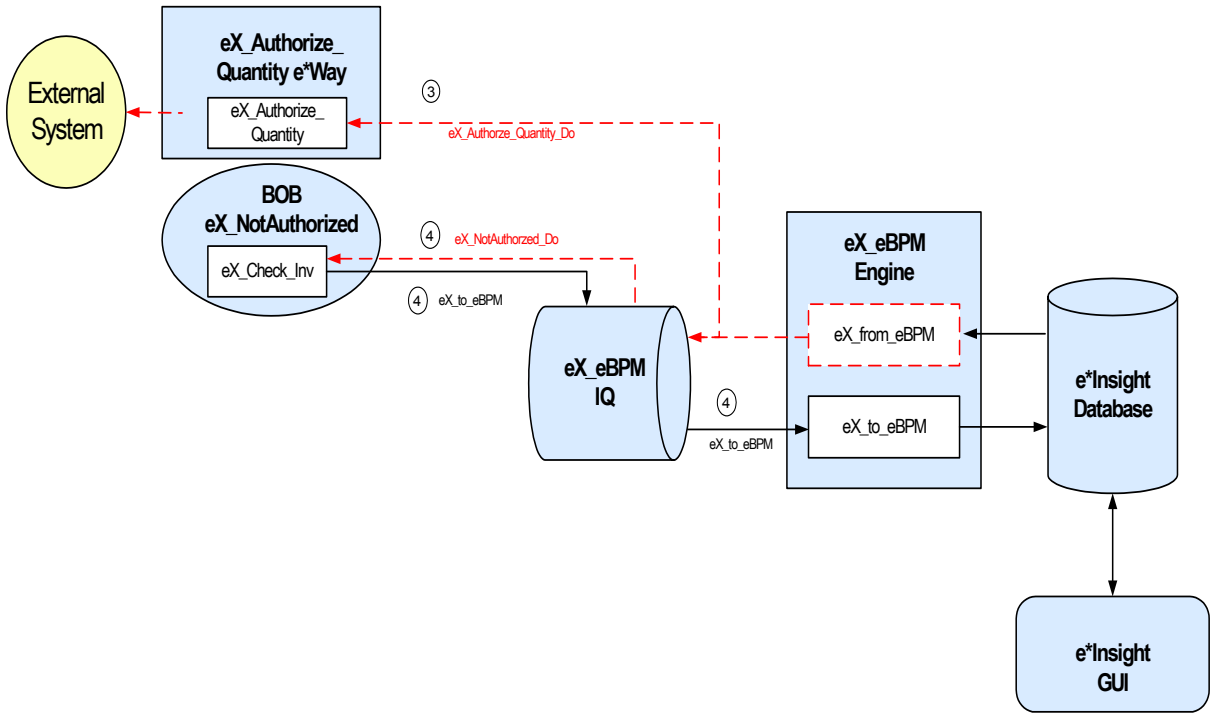


Figure 125 data flow description

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

Let's assume the items are in stock. The BPI moves to the next activity based on the result of the second decision gate. If the quantity is less than 100, the next activity is **Ship_Ord**; otherwise the next activity is **eX_Authorize_Quantity**.

Let's assume the quantity is greater than or equal to 100. The e*Insight engine publishes a "Do" Event (**eX_Authorize_Quantity_Do**) corresponding to the **Authorize_Quantity** authorization activity in the business process. The **eX_Authorize_Quantity** e*Way retrieves this Event from the **eX_eBPM** IQ and uses the information it contains to retrieve the BPI ID and alert the relevant person that this instance requires authorization.

When the **eX_Authorize_Quantity** activity has either been authorized or rejected, the e*Insight engine moves forward to the next activity in the business process. This is **Not_Authorized** if the quantity was not authorized; otherwise the next activity is **Ship_Order**.

④ Let's assume the quantity is not authorized. The e*Insight engine publishes a "Do" Event (**eX_Not_Authorized_Do**) corresponding to the **Not_Authorized** activity in the business process. The **eX_Not_Authorized** BOB retrieves this Event from the **eX_eBPM** IQ and updates the status. When the **Not_Authorized** activity is finished, the **eX_Not_Authorized** BOB publishes a "Done" Event using the **eX_to_eBPM** Event Type to the **eX_eBPM** IQ. The e*Insight engine retrieves the "Done" Event from the IQ, updates the BPI to reflect the status, and then moves forward to the **Send_Status** activity.

Let's assume that either the quantity is authorized, or the item was in stock, but the quantity was less than 100. The e*Insight engine processes the e*Insight script corresponding to the **Ship_Ord** activity in the business process, and then moves forward to the **Send_Status** activity.

Important Considerations

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

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

16.12 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.

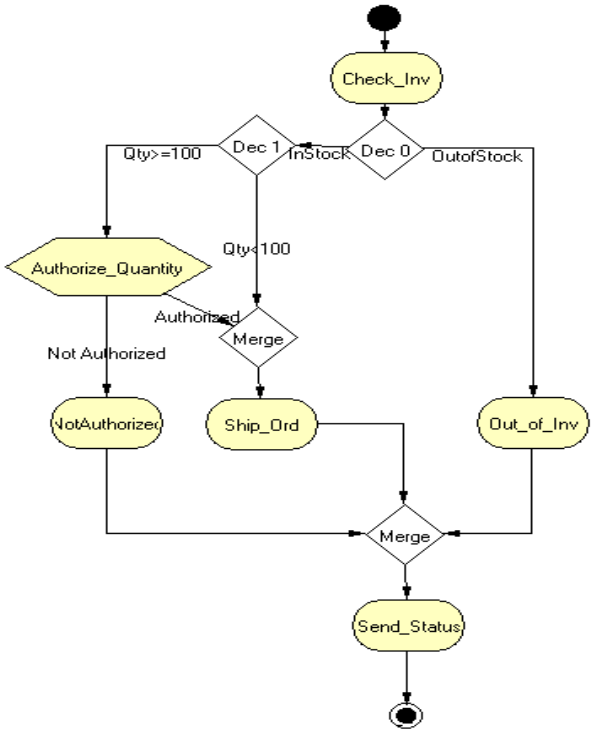
Note: *The Authorization Activity has two fixed Local Attributes—assignedTo (the user to whom the Authorization process is assigned) and performedBy (automatically assigned at run time as the user logged in to the e*Insight GUI). These two values must match in order for the user to Authorize, Reject, 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 126 and the following tables to create the BP in e*Insight.

Figure 126 ProcessOrder Business Process Model



16.13 Step 2: Configure the Integration Schema

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

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

In this example, you use an additional BOB for the **NotAuthorized** Activity, and an additional e*Way for the **Authorize_Quantity** Authorization Activity. The **Send_Status** activity must be associated with an e*Way because it interfaces with an external component.

Integration Schema Activity Components Summary

Use the information in Table 47 to configure the e*Gate schema that supports this example.

Table 47 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*.

16.14 Step 3: Configure User-defined e*Gate Components

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

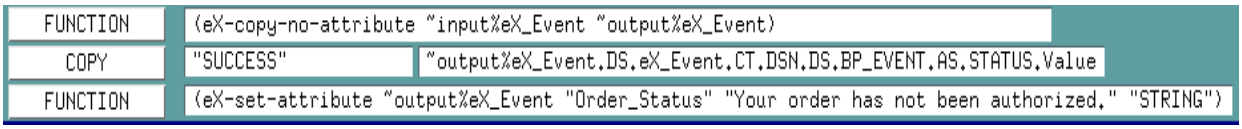
Configure the Activity BOB CRS in the Schema Designer

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 127, on the following page, shows the **Not_Authorized.tsc** CRS used in this example. The source and destination ETDs are eX_Standard_Event.ssc.

Figure 127 Not_Authorized.tsc CRS



Configure the Activity BOB Collaborations in the Schema Designer

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.

16.14.1 Configure the Authorize_Quantity e*Way

The last component that must be configured in the ProcessOrder example is the **Authorize_Quantity** e*Way.

This e*Way must create a file containing the text of e-mail message that can be sent to advise that an order requires authorization that can be sent via e-mail (simulated; no actual mail is sent)

This e*Way simulates sending an e-mail order status message simply by writing a short status message to a text file giving the Business Process Instance ID. An Event is not returned to the e*Insight engine as it is expecting authorization via the e*Insight GUI in this example.

Follow these steps to configure the Send_Status e*Way:

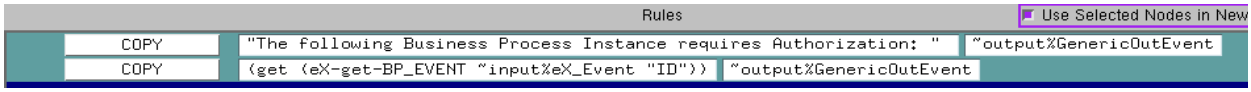
1. Create the **Authorize_Quantity.tsc** CRS.
2. Find the executable and create the e*Way configuration file.
3. Configure the Collaboration in the GUI.

Step 2: Create the Authorize_Quantity.tsc CRS

The Authorize_Quantity translation copies a fixed text message and the Business Process Instance ID into the outgoing message. No message is sent back to the e*Insight engine.

The Authorize_Quantity.tsc CRS is shown in Figure 128. The source ETD is eX_Standard_Event.ssc and the destination ETD is GenericOutEvent.ssc

Figure 128 Authorize_Quantity.tsc CRS



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 48 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>**.

16.15 Step 5: Run and Test the e*Insight scenario

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

16.15.1 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.*

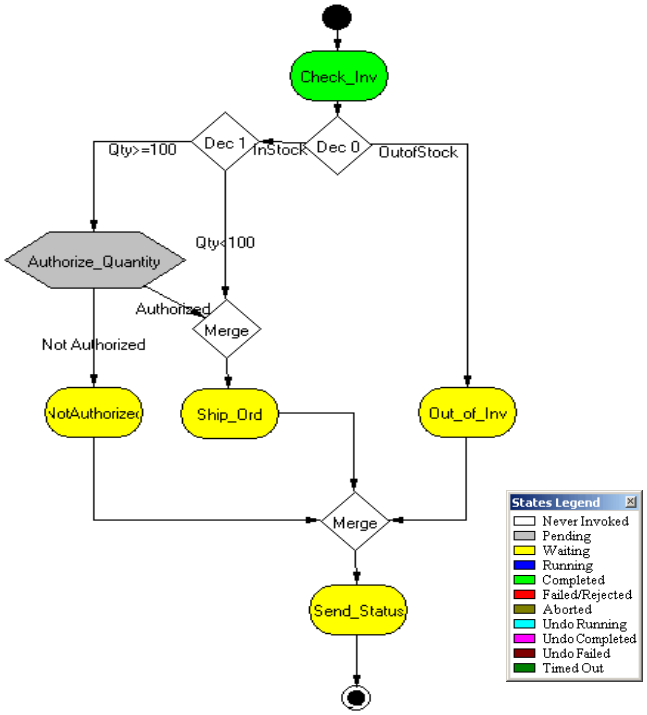
If everything is working correctly, an output file (**output#.dat**) as shown in Figure 129 appears in the directory indicating that an order requires authorization.

Figure 129 In Stock Output File

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 130 Authorize_Quantity Pending BPI Diagram



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

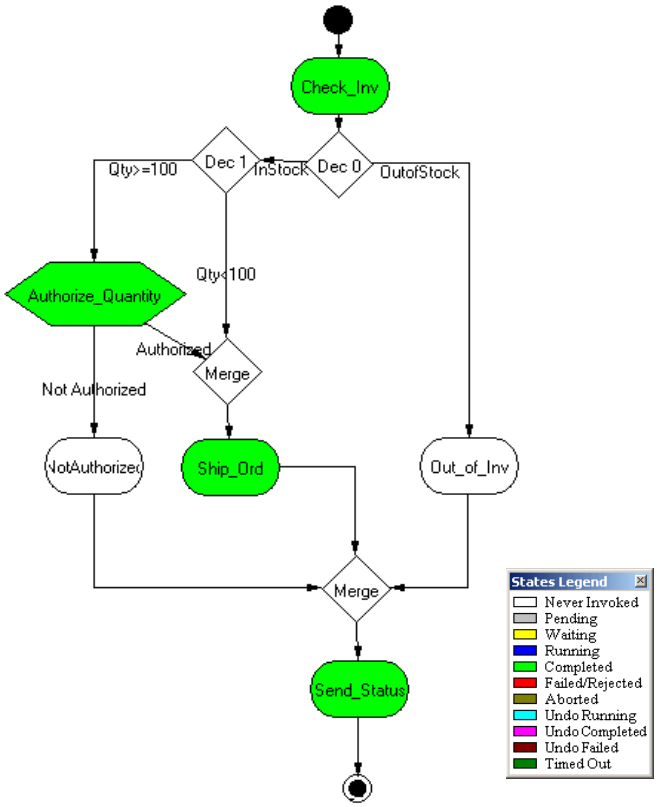
The **Authorization Activity Properties - Monitor Mode: (Authorize_Quantity)** is displayed.

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

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

10 The Business Process then completes using the route shown in Figure 131.

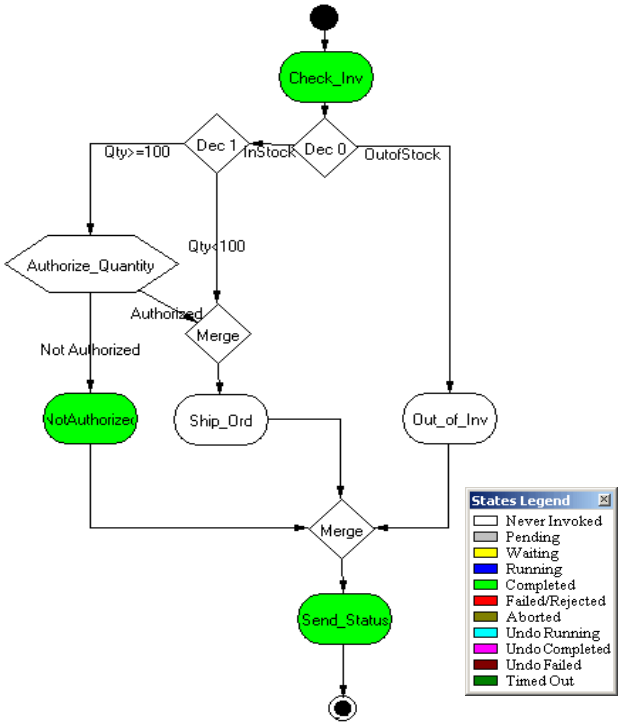
Figure 131 Authorization_Quantity - Authorized



Not Authorized Processing

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

Figure 132 Authorization_Quantity - Not Authorized



Appendix E

e*Insight Implementation (eISchema)

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

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

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

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

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

16.16 Overview

The major tasks in the implementation are shown in Table 49.

Table 49 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 417**
2	Use the e*Insight GUI to configure the e*Gate schema that supports e*Insight	**"Configure the Integration Schema" on page 420**
3	Configure the e*Insight Engine	**"Configure the e*Insight Engine" on page 427**
4	Add and configure the user-defined e*Gate components	**"Configure User-defined e*Gate Components" on page 427**
5	Run and test the scenario	**"Run and Test the e*Insight scenario" on page 441**

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

16.16.1 Case Study: Order Processing

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

Figure 134 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 133 e*Insight Data Flow Diagram

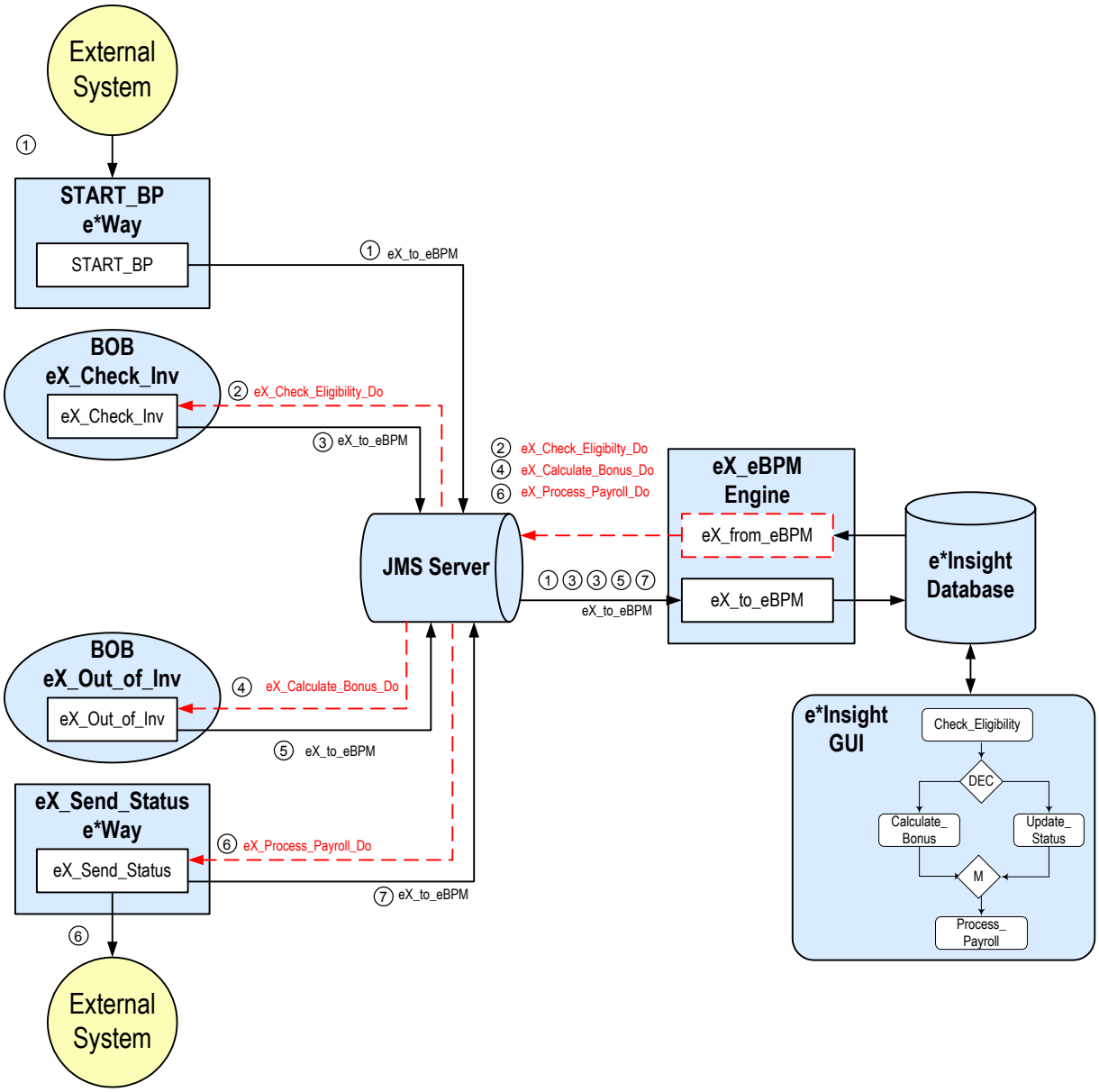
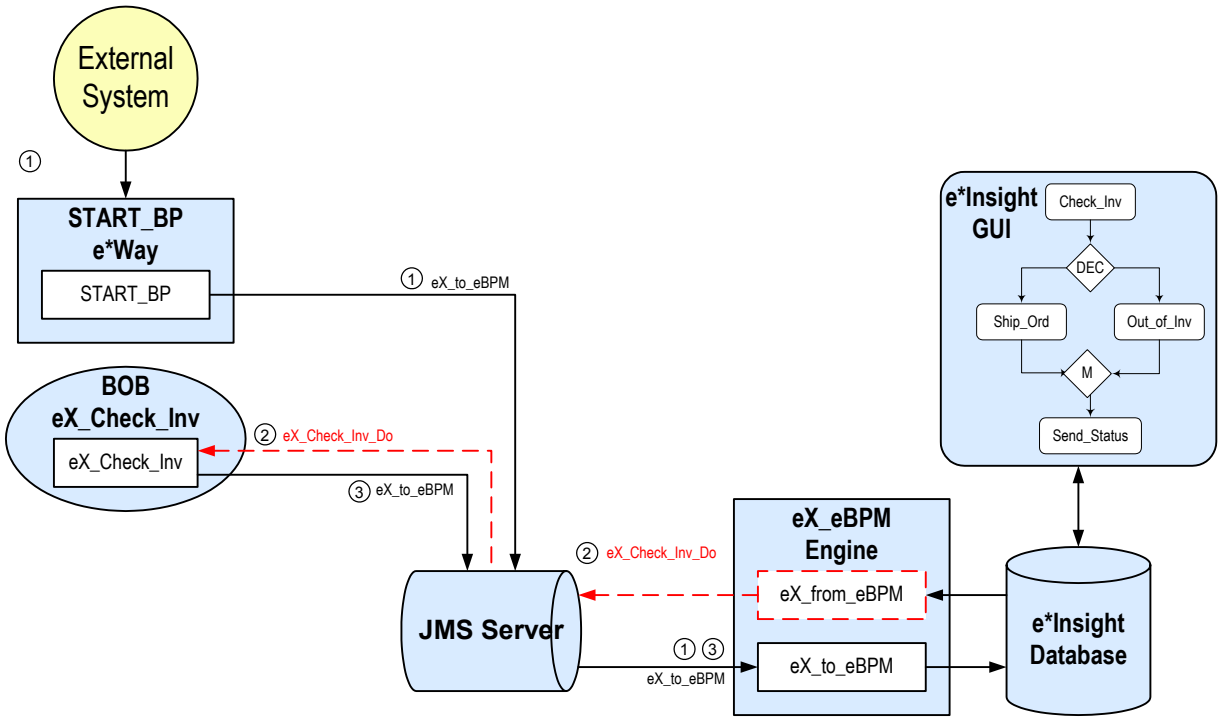


Figure 134 e*Insight Data Flow Diagram (Part 1)



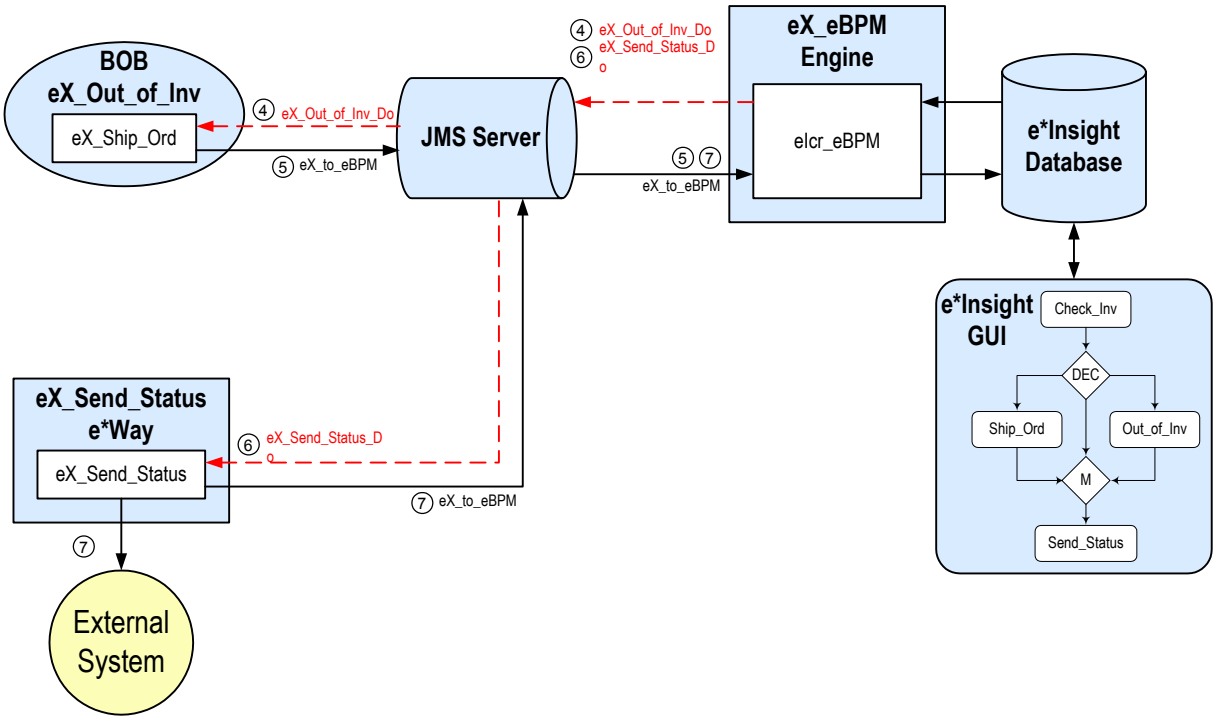
① The user-defined **START_BP** e*Way picks up the text file containing the order information from a shared location on the network, uses the order information to create the e*Insight Event that causes the e*Insight engine to start a business process instance, and publishes it using the **eX_to_eBPM**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

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

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

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

Figure 135 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.

16.17 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 136 and the following tables to create the BP in e*Insight.

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

Figure 136 ProcessOrder Business Process Model

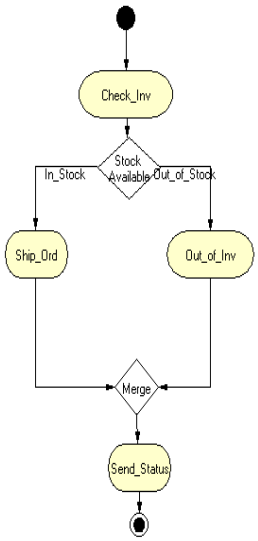


Table 50 BP Global Attributes

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 51 Activity Attributes

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

Table 52 Decision Gates

Feeding Activity	Expression
Check_Inv	In_Stock==true

16.17.1 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 420**.

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

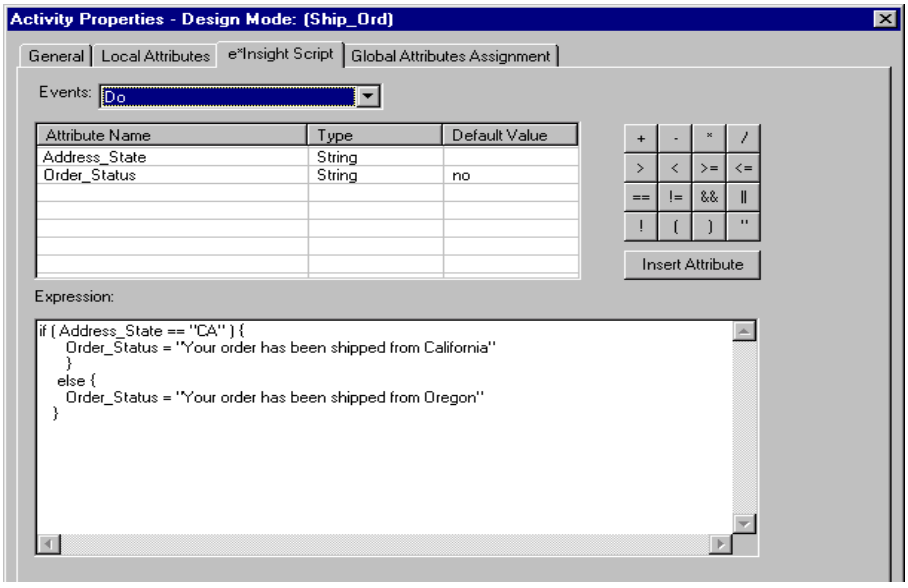
Configuring the e*Insight Script for Ship_Ord

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

To configure the e*Insight Script for Ship_Ord

1. From the **Ship_Ord** properties, **Activity Performed by** area, select **e*Insight Script**.
2. Select the **e*Insight Script** tab.
3. Configure the script as shown in Figure 137.

Figure 137 Ship_Order e*Insight Script Tab



16.18 Configure the Integration Schema

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

To create a copy of eIJSchema

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

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

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

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

Integration Schema Activity Components Summary

The information in Table 53 shows a summary of the e*Gate components that support this example.

Table 53 Integration Schema Activity Components

Name	Type	Participating Host	Configuration Instructions
eX_Check_Inv	BOB	localhost	**"Creating the eX_Check_Inv BOB" on page 421**
eX_Out_of_Inv	BOB	localhost	**"Creating the eX_Out_of_Inv BOB" on page 424**
eX_Send_Status	e*Way	localhost	**"Send_Status e*Way Configuration" on page 426**

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

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

Creating the eX_Check_Inv BOB

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

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

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

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

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

To configure the Check_Inv activity using Monk

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

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

   Figure 138, on the following page, shows the **eX_Check_Inv.tsc** CRS used in this example.

Figure 138 eX_Check_Inv.tsc CRS (Monk)

7 Validate and save the CRS.

8 Close the editor.

9 In the **Check_Inv** activity properties, click **Configure e*Gate Schema**.

You may be required to log into e*Gate.

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

11 Close the **Check_Inv** Activity properties.

To configure the Check_Inv activity using Java

1 In the e*Insight GUI, check that the Default Editor is Java.

2 Open the **Check_Inv** activity properties.

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

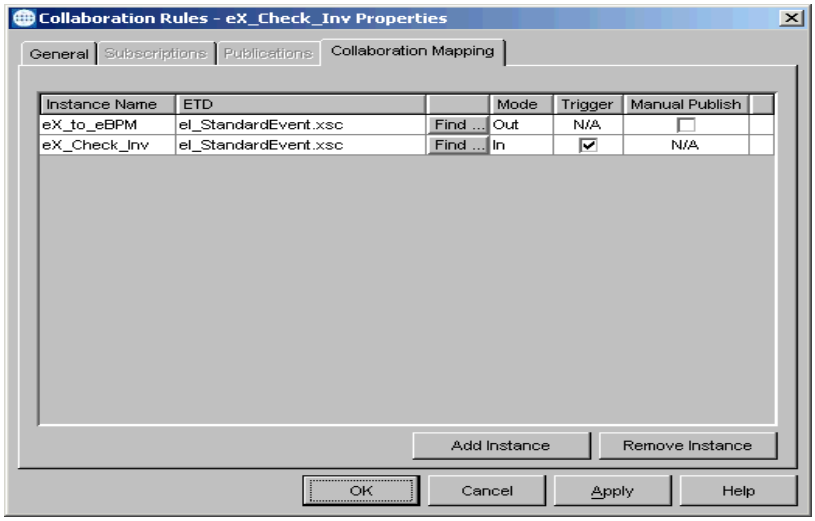
4 Click **New**.

The Define Collaboration dialog appears.

5 Select the **Define Mapping** tab.

6 Configure the instances as shown in Figure 139.

Figure 139 Define Mapping for eX_Check_Inv

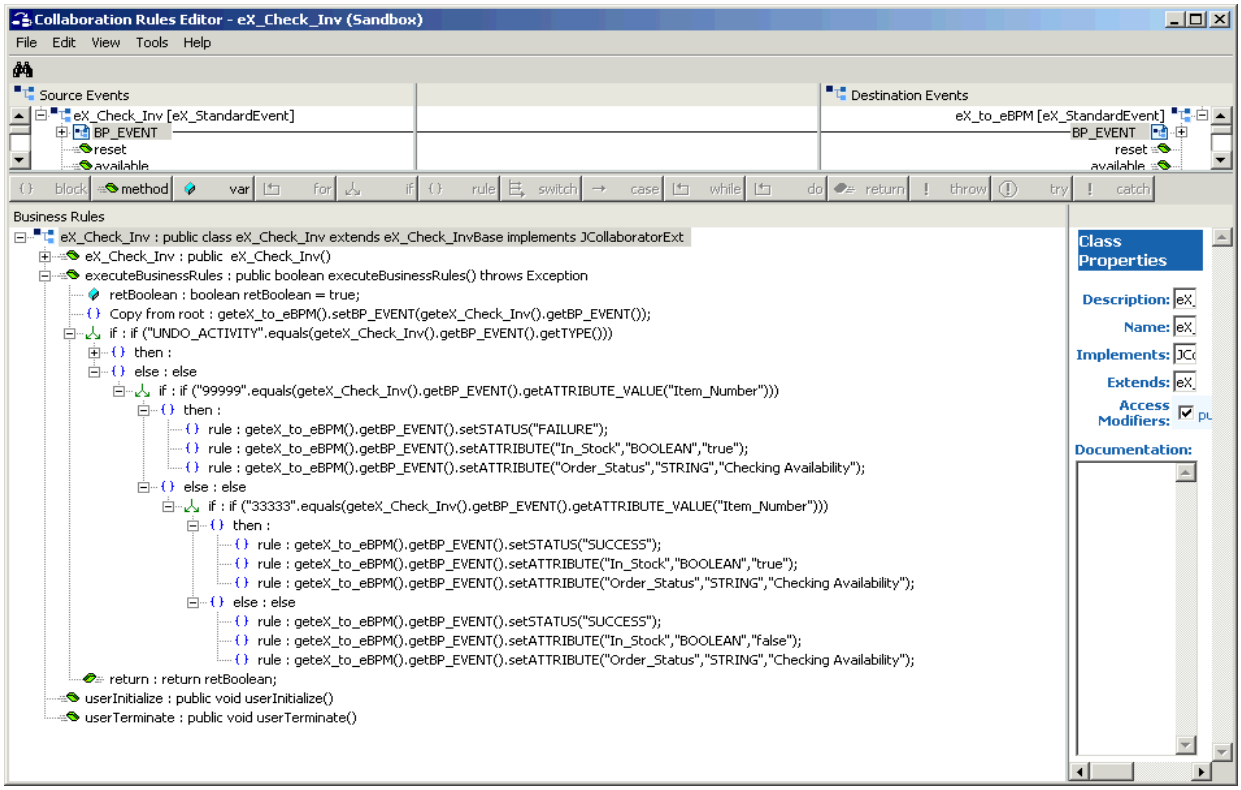


7 Click **OK**.

8 Create eX_Check_Inv.xsc.

Figure 140, on the following page, shows the **eX_Check_Inv.xsc** CRS used in this example.

Figure 140 eX_Check_Inv.xsc CRS (Java)



9 Compile and save the CRS.

10 Close the editor.

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

You may be required to log into e*Gate.

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

13 Close the **Check_Inv** Activity properties.

Creating the eX_Out_of_Inv BOB

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

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

To configure the Out_of_Inv activity using Monk

1 In the e*Insight GUI, check that the Default Editor is Monk.

2 Open the **Out_of_Inv** activity properties.

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

4 Click **New**.

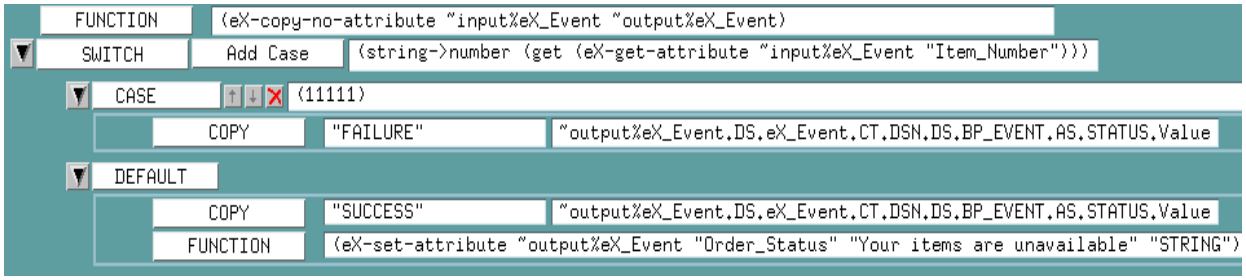
The **Define Collaboration** dialog appears.

5 Click **OK**.

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

Figure 141 shows the eX_Out_of_Inv CRS used in this example.

Figure 141 eX_Out_of_Inv.tsc CRS (Monk)



7 Validate and save the CRS.

8 Close the editor.

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

You may be required to log into e*Gate.

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

11 Close the **Out_of_Inv** Activity properties.

To configure the Out_of_Inv activity using Java

1 In the e*Insight GUI, check that the Default Editor is Java.

2 Open the **Out_of_Inv** activity properties.

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

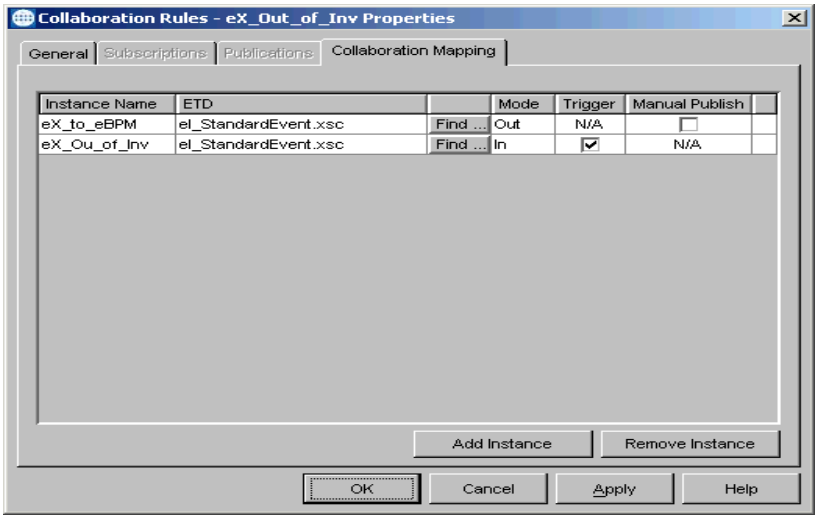
4 Click **New**.

The **Define Collaboration** dialog appears.

5 Select the **Define Mapping** tab.

6 Configure instances as shown in Figure 142.

Figure 142 Define Mapping for eX_Out_of_Inv (Java)

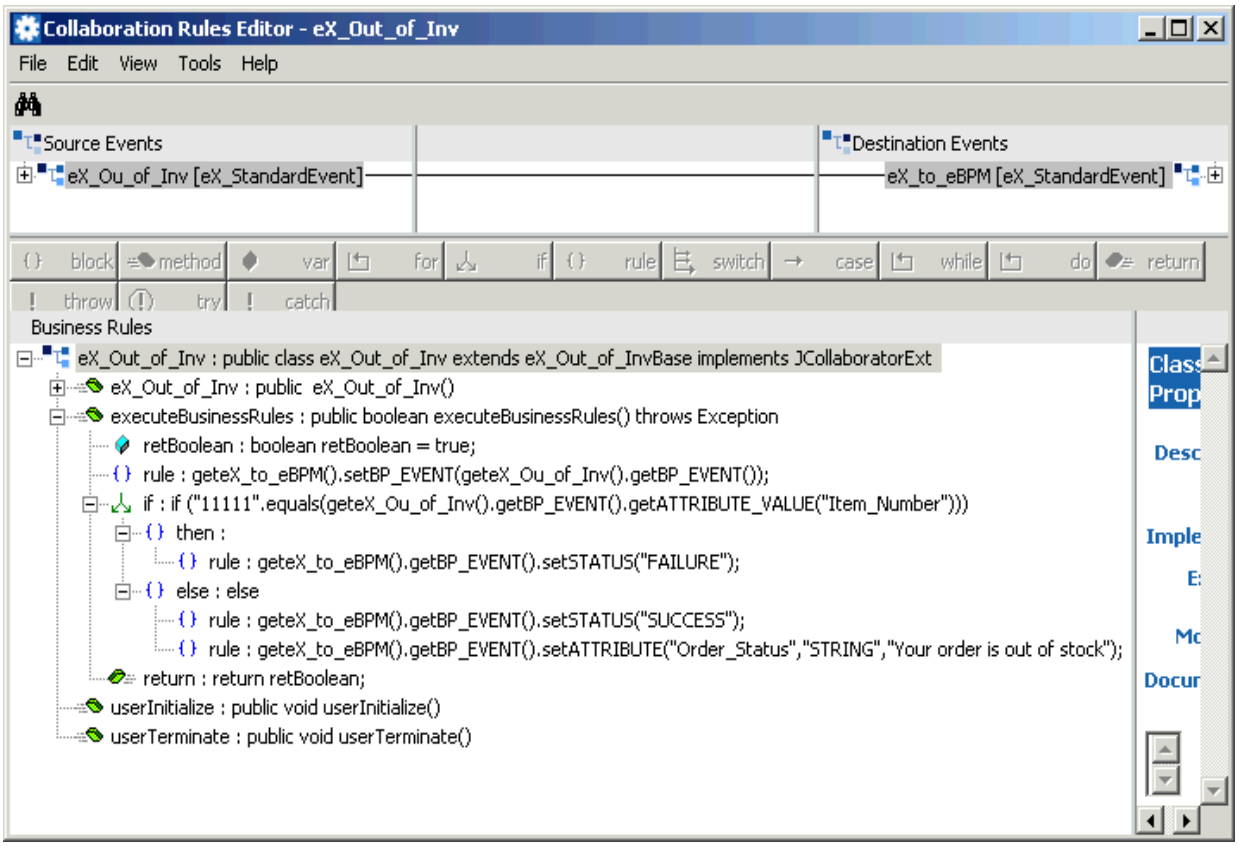


7 Click **OK**.

8 Create eX_Out_of_Inv.tsc.

Figure 143 shows the eX_Out_of_Inv CRS used in this example.

Figure 143 eX_Out_of_Inv.xsc CRS (Java)



9 Compile and save the CRS.

10 Close the editor.

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

You may be required to log into e*Gate.

12 Click OK, to close the information dialog.

13 Close the Out_of_Inv Activity properties.

Send_Status e*Way Configuration

The Send_Status Collaboration is configured using e*Gate, see **"Configure the Send_Status e*Way" on page 435**. 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.

16.19 Configure the e*Insight Engine

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

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

Edit the eX_eBPM Configuration File

Most of the parameter settings in the eX_eBPM engine's configuration file should not be changed. Table 54 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 369** to make the required changes for the ProcessOrder example.

16.20 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 54 shows the configuration order for the user-defined components.

Table 54 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 428**
2	Configure the Send_Status e*Way	**"Configure the Send_Status e*Way" on page 435**

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.*

16.20.1 Configure the START_BP e*Way

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

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

Table 55 shows the steps to configure the **START_BP** e*Way.

Table 55 Configuration steps for the START_BP e*Way

	Step	Section	
		Monk	Java
1	Add the e*Way and create the e*Way configuration file	**"Step 1: Create the START_BP e*Way using Monk" on page 428**	**"Step 1: Create the START_BP e*Way using Java" on page 431**
2	Create the Input ETD	**"Step 2: Create the Input ETD using Monk" on page 429**	**"Step 2: Create the Input ETD using Java" on page 431**
3	Create the START_BP Collaboration Rules script (CRS)	**"Step 3: Create the START_BP CRS using Monk" on page 429**	**"Step 3: Create the START_BP Collaboration using Java" on page 432**
4	Configure the Collaboration in the GUI	**"Step 4: Configure the START_BP Collaboration in the GUI using Monk" on page 430**	**"Step 4: Configure the Collaboration in the GUI using Java" on page 434**

Step 1: Create the START_BP e*Way using Monk

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

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

Table 56 Start e*Way Parameters

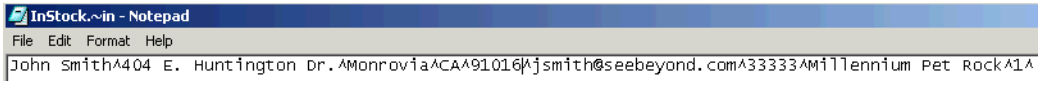
Screen	Parameter	Setting
General Settings	(All)	(Default)
Outbound (send) settings	(All)	(Default)
Poller (inbound) settings	PollDirectory	<eGate>\client\data\ProcessOrder
	(All others)	(Default)
Performance Testing	(All)	(Default)

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.

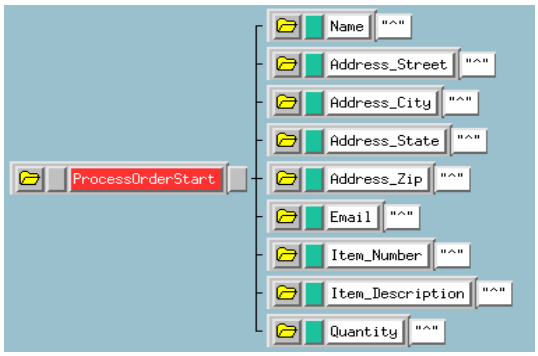
The input data file used in this example is shown in Figure 144. Place this data file at the directory location **<eGate>\client\data\ProcessOrder**.

Figure 144 Input Text File (InStock.~in)



Using the ETD Editor and the input data as a guide, create an ETD like the one shown in Figure 145. For more information on using the ETD Editor see the ETD Editor's online help.

Figure 145 Input ETD: ProcessOrderStart.ssc (Monk)



Step 3: Create the START_BP CRS using Monk

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

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

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

Figure 146, shows the **START_BP** CRS used in the ProcessOrder example:

Figure 146 START_BP CRS (Monk)

```
COPY          "ProcessOrder"   ^output%eX_Event.DS.eX_Event.CT.DSN.DS.BP_EVENT.AS.NAME.Value
COPY          "START_BP"       ^output%eX_Event.DS.eX_Event.CT.DSN.DS.BP_EVENT.AS.TYPE.Value
UNIQUE ID     ^output%eX_Event.DS.eX_Event.CT.DSN.DS.BP_EVENT.AS.ID.Value
FUNCTION      (eX-set-attribute ^output%eX_Event "Address_Street" ^input%ProcessOrderStart.Address_Street "STRING")
FUNCTION      (eX-set-attribute ^output%eX_Event "Address_City" ^input%ProcessOrderStart.Address_City "STRING")
FUNCTION      (eX-set-attribute ^output%eX_Event "Address_State" ^input%ProcessOrderStart.Address_State "STRING")
FUNCTION      (eX-set-attribute ^output%eX_Event "Address_Zip" ^input%ProcessOrderStart.Address_Zip "STRING")
FUNCTION      (eX-set-attribute ^output%eX_Event "Cust_Name" ^input%ProcessOrderStart.Name "STRING")
FUNCTION      (eX-set-attribute ^output%eX_Event "Cust_email" ^input%ProcessOrderStart.Email "STRING")
FUNCTION      (eX-set-attribute ^output%eX_Event "Item_Description" ^input%ProcessOrderStart.Item_Description "STRING")
FUNCTION      (eX-set-attribute ^output%eX_Event "Item_Number" ^input%ProcessOrderStart.Item_Number "STRING")
FUNCTION      (eX-set-attribute ^output%eX_Event "Order_Quantity" ^input%ProcessOrderStart.Quantity "NUMBER")
```

Step 4: Configure the START_BP Collaboration in the GUI using Monk

In addition to creating the configuration file for the e*Way and the CRS used by the Collaboration, you must also configure the **START_BP** e*Way's Collaboration in the Schema Designer.

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 Schema Designer and the following table to add the **START_BP** e*Way and create its configuration file.

Table 57 Start e*Way Parameters

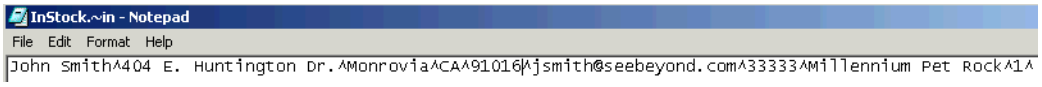
Screen	Parameter	Setting
General Settings	(All)	(Default)
Outbound (send) settings	(All)	(Default)
Poller (inbound) settings	PollDirectory	<eGate>\client\data\ProcessOrder
	(All others)	(Default)
Performance Testing	(All)	(Default)

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.

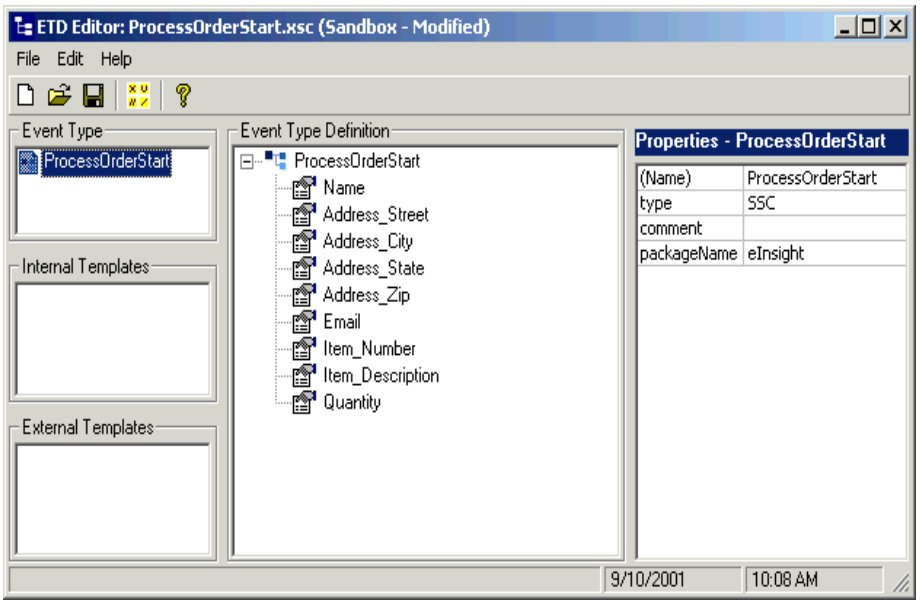
The input data file used in this example is shown in Figure 147. Place this data file at the directory location **c:\eGate\client\data\ProcessOrder**.

Figure 147 Input Text File (InStock.fin)



Using the ETD Editor and the input data as a guide, create an ETD like the one shown in Figure 148. Set the global delimiter to a ^ character.For more information on using the ETD Editor see the ETD Editor's online help.

Figure 148 Input ETD: ProcessOrderStart.xsc (Java)



Step 3: Create the START_BP Collaboration using Java

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

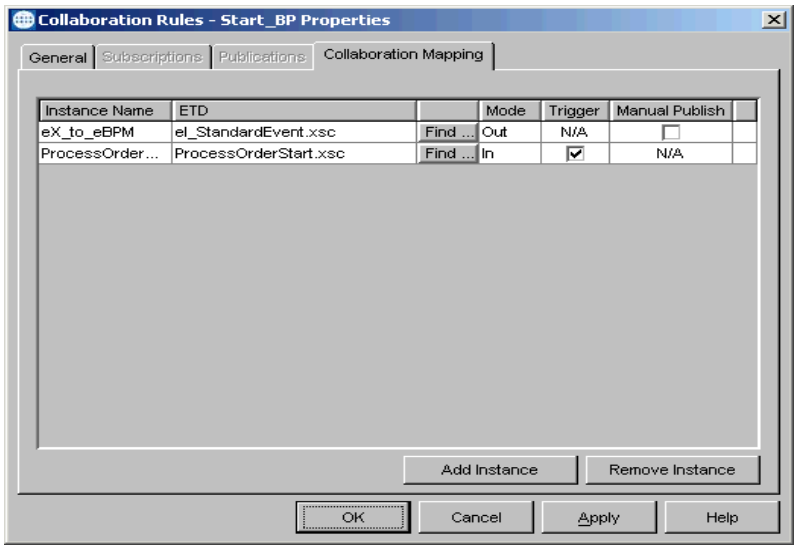
- Put the data into e*Insight ETD (**eI_Standard_Event.xsc)** format.
- Populate the Event with the information the e*Insight engine needs to start a BPI.

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

1 Create a Collaboration Rule, **START_BP**, that uses the Java service.

2 Configure the Collaboration Mapping tab, as shown in Figure 149.

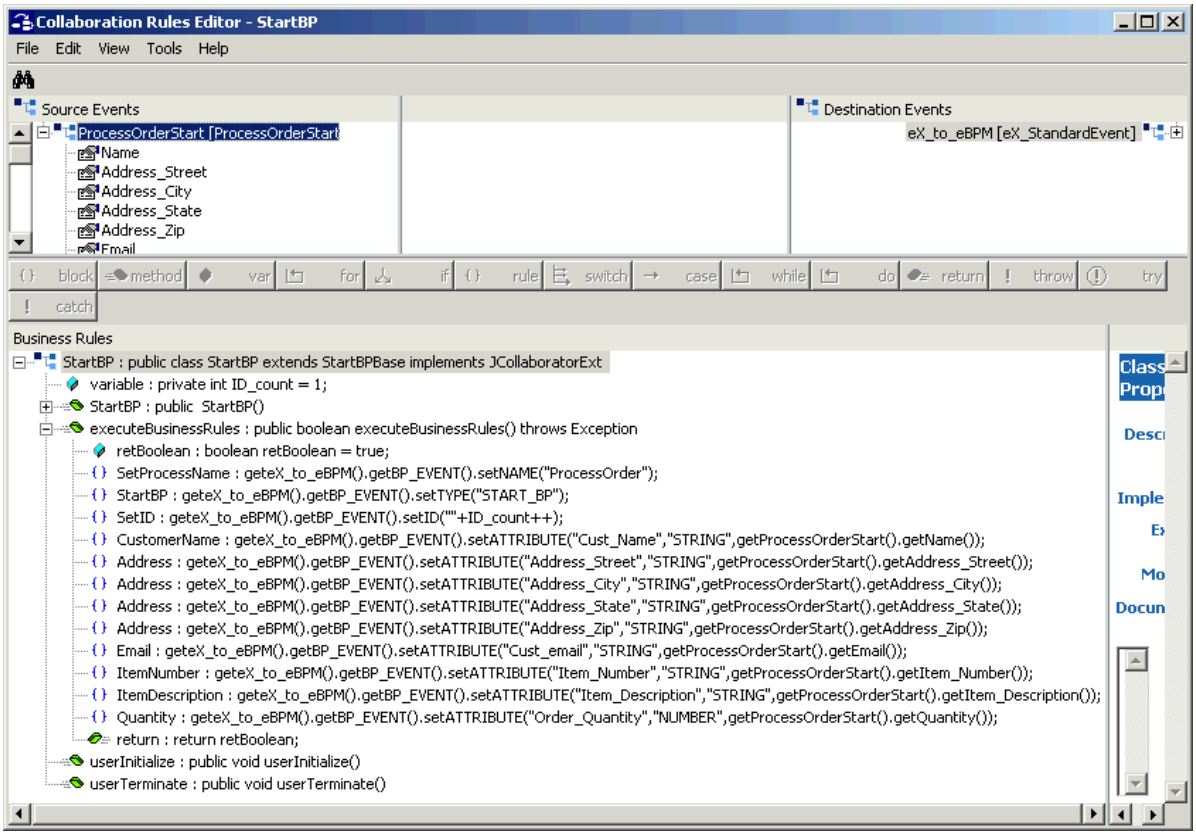
Figure 149 Start_BP Properties, Collaboration Mapping Tab (Java)



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

4 Click **New** to create a new CRS, as show in Figure 150.

Figure 150 START_BP CRS (Java)

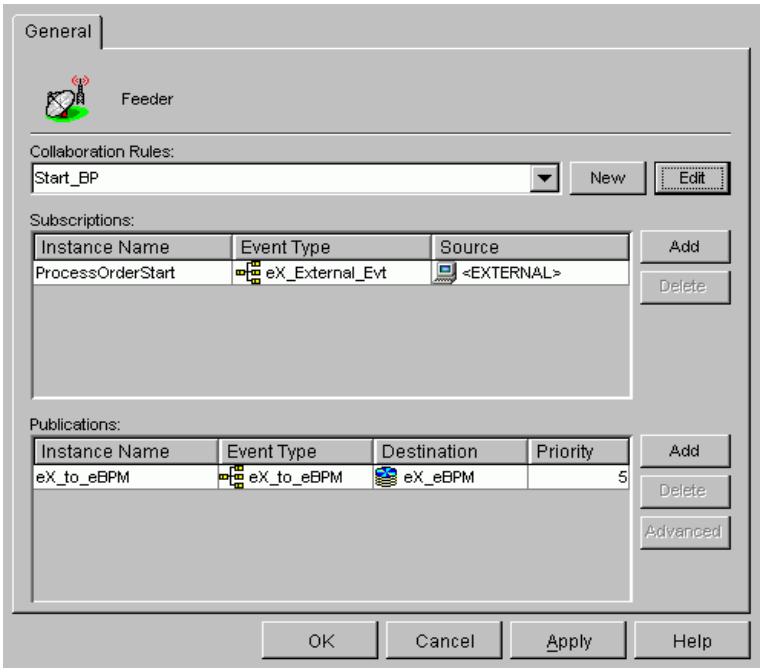


Step 4: Configure the Collaboration in the GUI using Java

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 Schema Designer.

1 Create a Collaboration for the **START_BP** e*Way configured as shown in Figure 151.

Figure 151 START_BP Collaboration



16.20.2 Configure the Send_Status e*Way

The last component that must be configured in the ProcessOrder example is the **Send_Status** e*Way.

This e*Way must accomplish two tasks:

- Create a file containing the text of e-mail message that can be sent to an order-status message via e-mail (simulated; no actual mail is sent)
- Return "SUCCESS" to the e*Insight engine

This e*Way simulates sending an e-mail order status message by writing the customer's e-mail address and a short status message to a text file. When this is successful, an Event is returned to the e*Insight engine with the status node set to "SUCCESS".

Table 58 shows the steps to configure the Send_Status e*Way.

Table 58 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 435**	**"Step 1: Configure the e*Way using Java" on page 438**
2	Create the Output ETD	**"Step 2: Create the Output ETD using Monk" on page 436**	**"Step 2: Create the Output ETD: SendStatus.xsc using Java" on page 438**
3	Create the eX_Send_Status.tsc CRS	**"Step 3: Create the eX_Send_Status.tsc CRS using Monk" on page 436**	**"Step 3: Create the Send_Status Collaboration Rule using Java" on page 439**
4	Configure the Collaboration in the GUI	**"Step 4: Configure the Collaboration using Monk" on page 437**	**"Step 4: Configure the Collaboration using Java" on page 440**

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 59 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 152.

Figure 152 root.ssc ETD

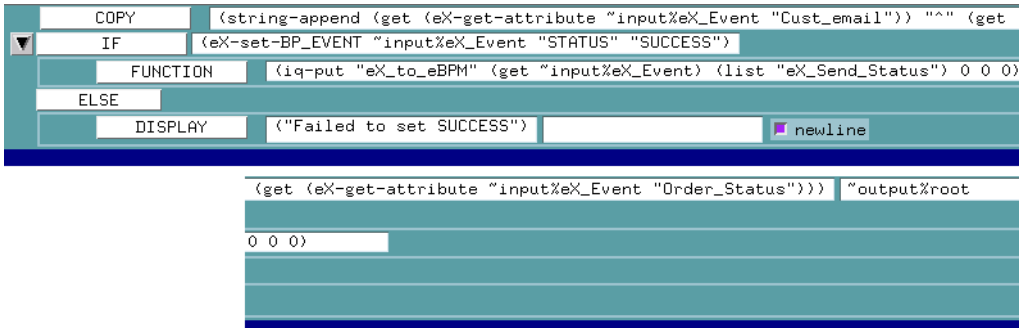
Step 3: Create the eX_Send_Status.tsc CRS using Monk

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

Figure 153 eX_Send_Status.tsc CRS (Monk)



Step 4: Configure the Collaboration using Monk

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

To configure the collaboration

1. Highlight the **eX_Send_Status** e*Way.
2. Delete the two Collaborations **eX_to_Send_Status** and **eX_from_Send_Status**.
3. Add a Collaboration named **eX_Send_Status**.
4. Highlight the **Collaboration Rules** folder.
5. Delete the two Collaboration Rules **eX_to_Send_Status** and **eX_from_Send_Status**.
6. Add a Collaboration Rule named **eX_Send_Status**.
7. Edit the Collaboration Rule.
8. In the **Collaboration Rules Properties** dialog box, select the **Monk** service.
9. Find the CRS **eX_Send_Status.tsc** and associate it with the Collaboration Rule.
10. On the **Subscriptions** tab, move the **eX_Send_Status_Do** and **eX_Send_Status_Undo** Event Types to the **Selected Input Event Types** box.
11. On the **Publications** tab, move the **eX_External_Evt** and the **eX_to_eBPM** Event Types to the **Selected Output Event Types** box.

    Verify that the **eX_External_Evt** Event Type is marked as the default.
12. Click **OK** to close the **Collaboration Rules Properties** dialog box.
13. Highlight the **eX_Send_Status** e*Way and edit the **eX_Send_Status** Collaboration you associated with it in step 3.
14. In the **Collaboration Properties** dialog box, select the **eX_Send_Status** Collaboration Rule.
15. Under **Subscriptions** add the **eX_Send_Status_Do** and **eX_Send_Status_Undo** Event Types from the **eX_from_eBPM** source.
16. Under **Publications** add the Event Type **eX_External_Evt** with destination **<EXTERNAL>** and Event Type **eX_to_eBPM** with destination **eX_eBPM** IQ.

Step 1: Configure the e*Way using Java

First find the executable, then create the configuration file.

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

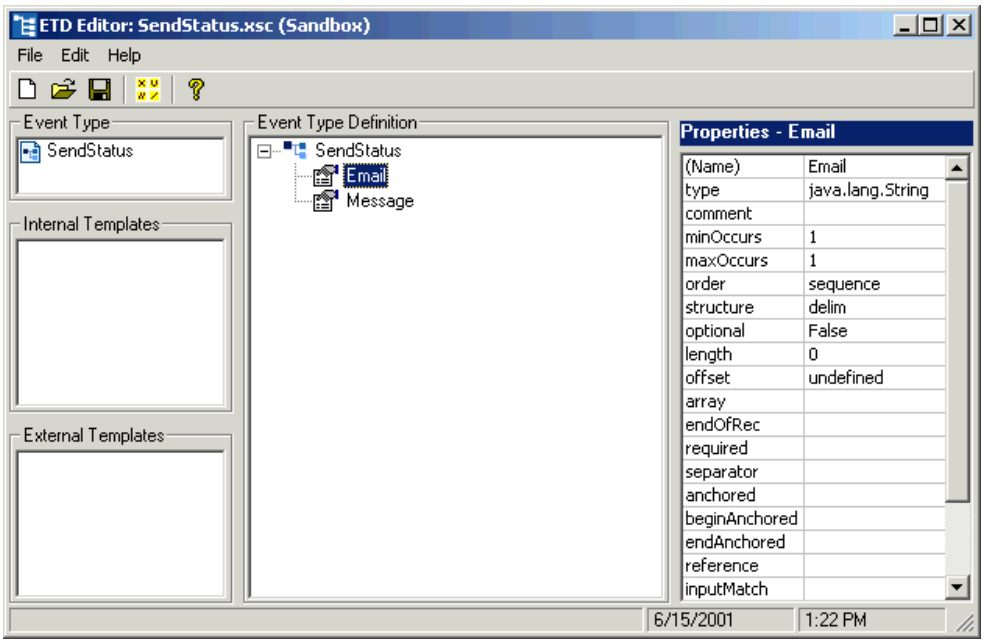
Table 60 Send_Status e*Way Parameters

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 154.

Figure 154 SendStatus.xsc ETD (Java)



Step 3: Create the Send_Status Collaboration Rule using Java

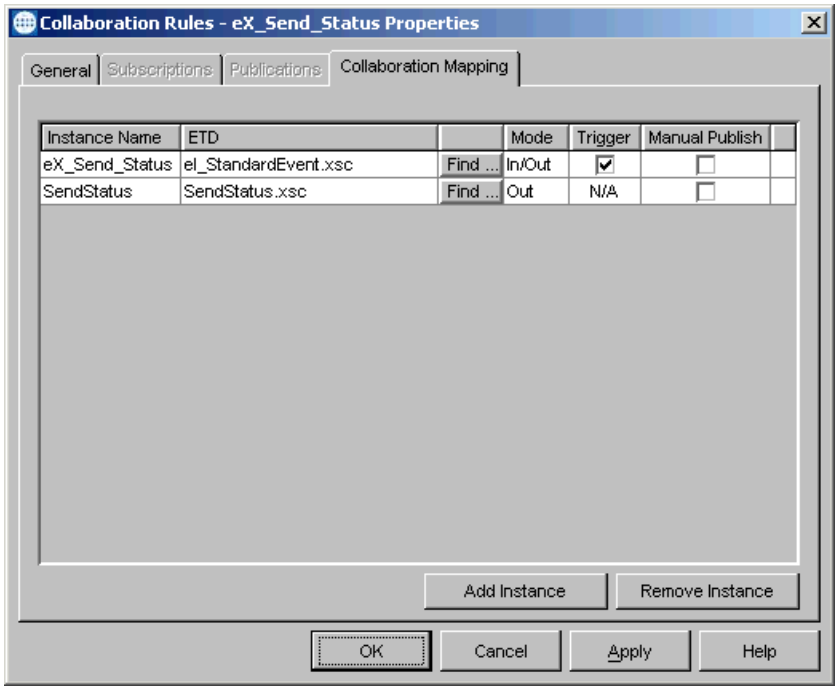
This CRS must accomplish three things:

- put the output data into a readable format that can be written to a file
- set the BP status node to "SUCCESS"
- send the "Done" Event back to the e*Insight engine

To Configure the eX_Send_Status Collaboration Rule

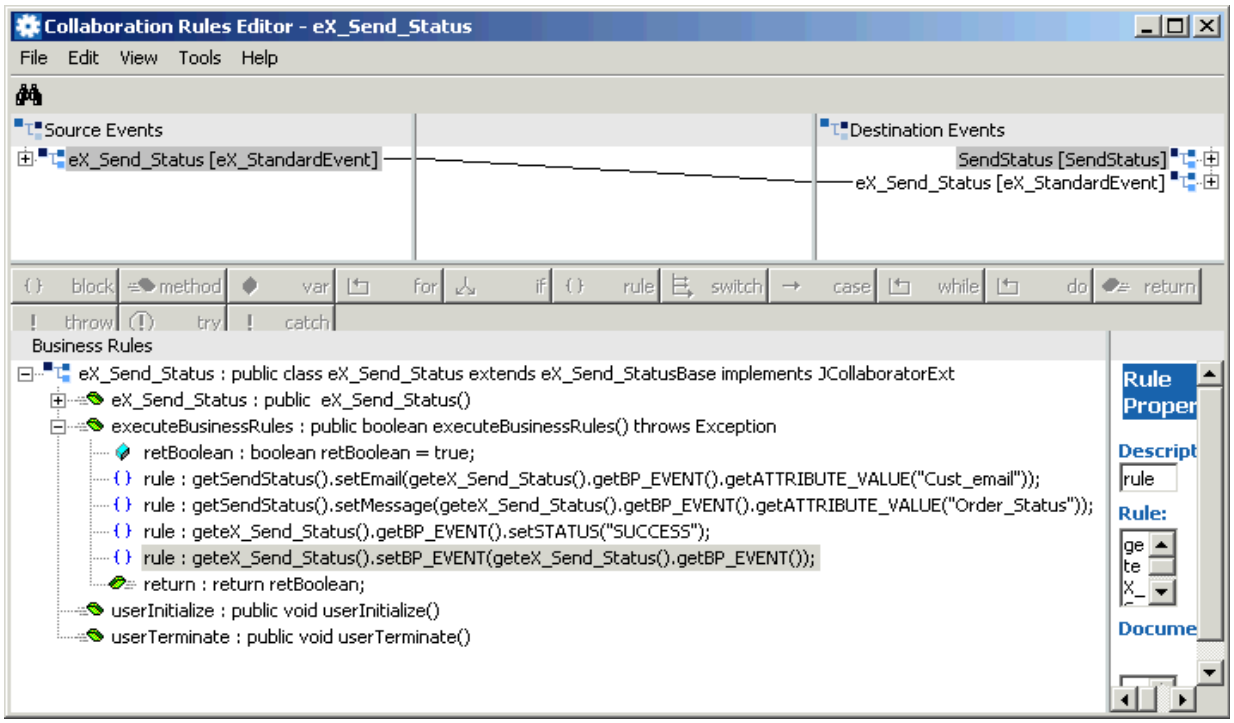
1. From the eX_Send_Status Collaboration Rule General tab, select the Java Collaboration Service.
2. From the eX_Send_Status Collaboration Rule Collaboration Mapping tab, create two new instances as shown in Figure 155.

Figure 155 eX_Send_Status CR Properties, Collaboration Mapping tab (Java)



3. Click **Apply**, and click the **General Tab**.
4. Click **New** to create a new CRS, as show in Figure 156.

Figure 156 eX_Send_Status.xsc CRS (Java)



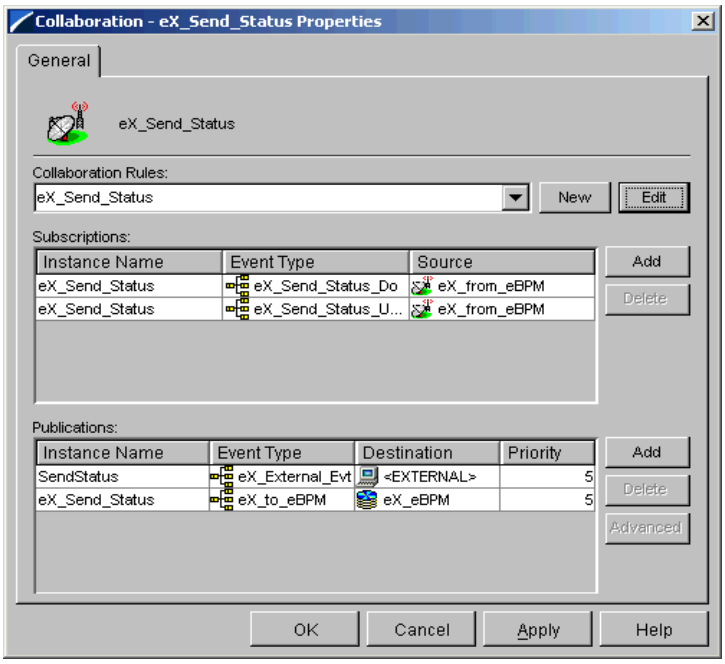
Step 4: Configure the Collaboration using Java

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

In the Enterprise Manager:

1. Highlight the **eX_Send_Status** e*Way.
2. Delete the two Collaborations **eX_to_Send_Status** and **eX_from_Send_Status**.
3. Add a Collaboration named **eX_Send_Status**.
4. Configure the Collaboration, as shown in Figure 157.

Figure 157 eX_Send_Status Collaboration (Java)



16.21 Run and Test the e*Insight scenario

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

16.21.1 Testing the Standard Business Logic

The following procedure tests the standard business logic of the e*Insight ProcessOrder case study example. That logic is as follows: a check is made to see whether or not the item ordered is available. If it is in stock the Ship_Order activity is invoked and a message is generated that can be sent to the customer indicating that his order has been shipped to him. If the item is unavailable, then the Out_of_Inv activity is invoked which creates a message informing the customer that his item is unavailable.

The test is made by sending in data with different item numbers and verifying the correct processing. Input data with an item number of 33333 is interpreted as being in stock and any other number except for the three special numbers (11111, 22222, and 99999) is interpreted as being out of stock.

In-Stock Processing

Use the following procedure to test the functionality of the example for an item that is in stock.

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

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

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

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

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

Substitute the appropriate username and password for your installation.

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

5 Navigate to the location for the input data file, **InStock.~in**, shown in **Figure 144 on page 429** (**c:\eGate\client\data\ProcessOrder**) and change the extension to "**.fin**".

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

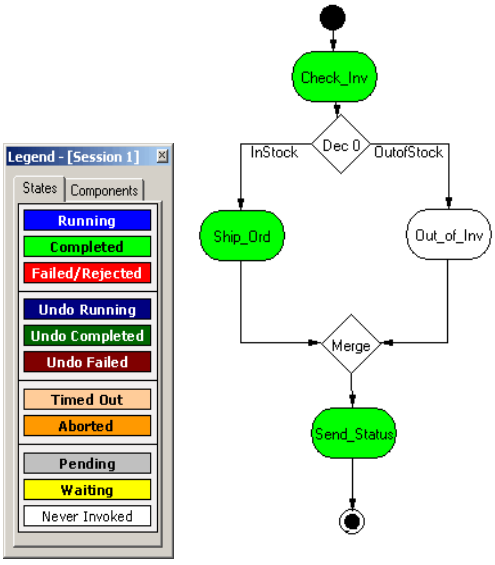
If everything is working correctly, an output file (**ProcessOrder_output#.dat**) as appears in the directory indicating successful completion of the BPI.

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

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

In the ProcessOrder example, an activity that stays blue for more than couple minutes indicates a problem, and the e*Gate component associated with that activity should be investigated for the cause of the problem. Figure 158 illustrates how the successfully completed BPI appears in the e*Insight GUI.

Figure 158 In Stock Completed BPI Diagram

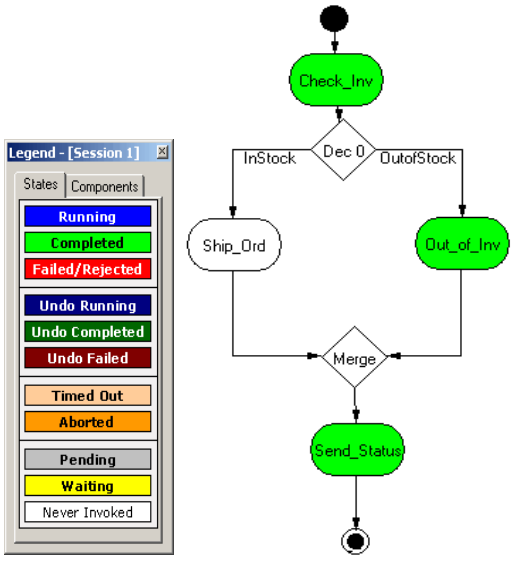


Out-of-Stock Processing

Testing the functionality for out of stock processing uses exactly the same procedure as that for in stock processing except that different input data is submitted.

- Verify that sending in the data with an item number of 44444 causes the business process to take the "FALSE" branch of the decision gate and create the diagram shown in Figure 159.

Figure 159 Out of Stock Completed BPI Diagram



16.21.2 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 **Manual Restart** is selected 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.

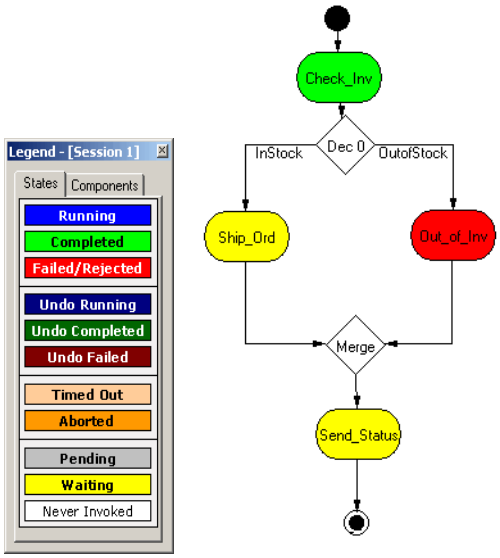
1. Perform steps 1 through 4 outlined in **"In-Stock Processing" on page 441**.
2. Verify that **Manual Restart** is selected for the activities in the business process.

   If **Manual Restart** is not selected, you must delete the BPIs for the business process, or save the business process as a new version, before you can mark it. Refer to the *e*Insight Business Process Manager User's Guide* for information on how to do this.
3. Navigate to the location (**c:\eGate\client\data\ProcessOrder\ManualUndo.~in**) and create an input data file with an item number of 11111 and change the extension to "**.fin**".

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

4. Switch to the e*Insight GUI and, while in monitor mode, select the most recent business process instance. Observe the path that the data has taken, as shown in Figure 160 on the next page.

Figure 160 Manual Undo—Failed BPI Diagram



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

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

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

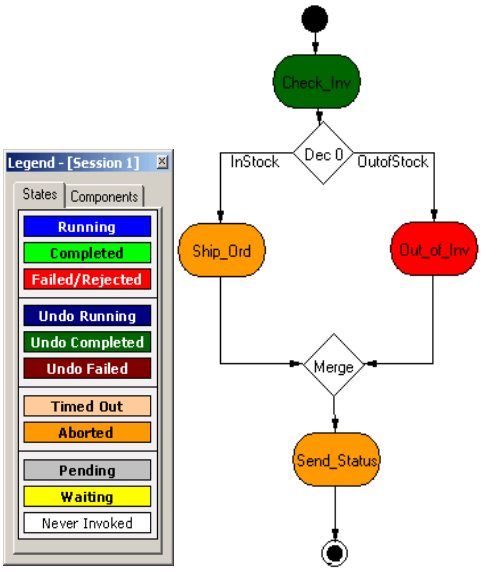
6 Select the **Business Process Attributes** tab.

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

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

The **Check_Inv** activity should now appear dark green indicating that the activity has been successfully undone.

Figure 161 Manual Undo Completed BPI Diagram



16.21.3 Demonstrating Business Process Restart Functionality

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

Repairing a String Attribute

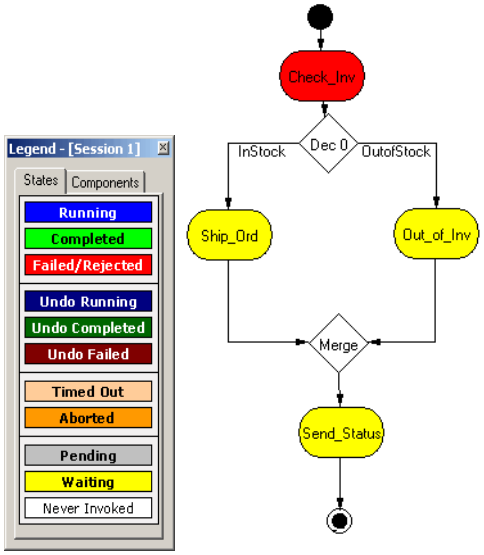
Attributes can be of various types; Boolean, number, string, and XML. The following example shows the procedure to repair an attribute of type string. For information on repairing an attribute with type XML, see the *e*Insight Business Process Manager User's Guide*.

1. Perform steps 1 through 4 outlined in **"In-Stock Processing" on page 441**.
2. Verify that **Manual Restart** is selected for the activities in the business process.
3. Navigate to the location (**c:\eGate\client\data\ProcessOrder\AttributeRepair.~in**) and create an input data file with an item number of 99999, and change the extension to "**.fin**".

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

4. Switch to the e*Insight GUI and, while in monitor mode, select the most recent business process instance from the **List** tab. Select the **Diagram** tab to observe the path that the data has taken.

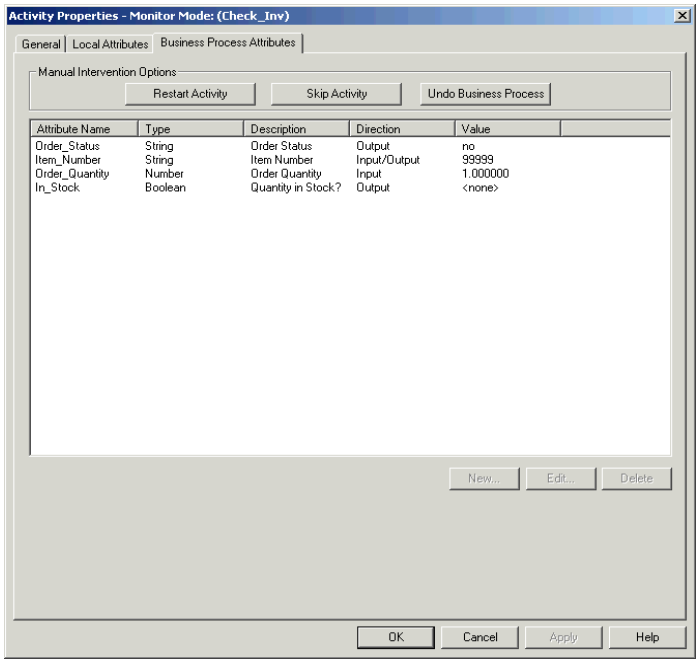
Figure 162 Attribute Repair—Failed BPI Diagram



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

5 Double-click the **Check_Inv** activity, then click the **Business Process Attributes** tab.

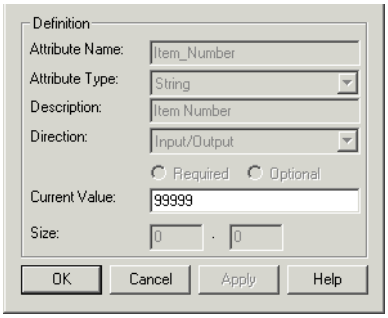
Figure 163 Attribute Repair—Business Process Attributes tab



6 Highlight the **Item_Number** attribute line, and then click **Edit**.

The **Edit Business Process Attribute** dialog box is displayed, as shown in Figure 164.

Figure 164 Attribute Repair—Edit BP Attribute



7 In the **Edit Business Process Attribute** dialog box, change the value of the attribute in the **Current Value:** box from 99999 to 44444, and then click **OK**.

The value 99999 was supplied to the e*Insight engine by the input file and it is this value that causes the **Check_Inv** Collaboration to return "FAILURE" to the e*Insight engine.

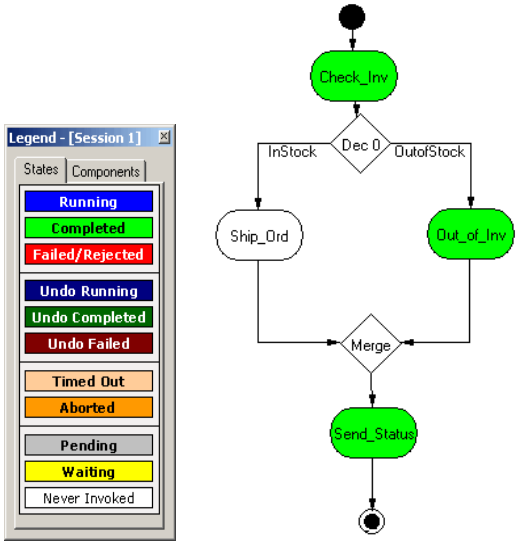
8 Click **OK** to close the **Edit Business Process Attribute** dialog box.

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

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

The completed BPI diagram is displayed, as shown in Figure 165 on the next page.

Figure 165 Attribute Repair Completed BPI Diagram



The **Check_Inv**, **Out_of_Inv**, and **Send_Status** activities now appear green indicating that the BPI has been restarted and has now completed successfully.

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

Appendix F

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.*

F.1 XML Structure

```
<!--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?)>
<!--Business 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)>
```

Appendix G

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 451**.

16.22 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 on page 452
eX-count-attribute on page 453
eX-set-attribute on page 454
eX-set-BP_EVENT on page 455
eX-get-BP_EVENT on page 456
eX-get-Activity on page 457
eX-set-Activity on page 458
eX-string-set-attribute on page 459
eX-xml-set-attribute on page 460
eX-bin-set-attribute on page 461
eX-count-local-attribute on page 462
eX-get-local-attribute on page 463
eX-set-local-attribute on page 464
eX-copy-no-attribute on page 465
eX-set-all-BP_EVENT on page 466
eX-get-all-attribute on page 467
eX-get-all-local-attribute on page 468

eX-get-attribute

Syntax

```
(eX-get-attribute root-path attribute)
```

Description

eX-get-attribute finds the path to the value of the attribute specified in the e*Insight Event named in the root-path.

Parameters

Name	Type	Description
root-path	path	Either ~input%eX_Event or ~output%eX_Event
attribute	string	The name of the attribute as it appears in the e*Insight GUI.

Return Values

Returns one of the following values:

Boolean

Returns **#f** (false) if the attribute value is not found

path

Returns the path to the attribute in the e*Insight Event. Use **get** to return the actual value of the attribute.

Throws

None.

Example

For an Event where the value of Is_Valid_account? is "yes":

```
(get (eX-get-attribute ~input%eX_Event "Is_Valid_account?"))
=> yes
```

eX-count-attribute

Syntax

```
(eX-count-attribute root-path)
```

Description

eX-count-attribute searches the Event specified for attributes, and counts the number of attributes found.

Parameters

Name	Type	Description
root-path	path	Either ~input%eX_Event or ~output%eX_Event

Return Values

integer
Returns 0 to n depending on the number of attributes found.

Throws

None.

Example

For an Event containing three attributes:

```
(eX-count-attribute ~input%eX_Event)

=> 3
```

eX-set-attribute

Syntax

```
(eX-set-attribute root-path attribute value type)
```

Description

If the attribute exists in the Event specified in root-path, **eX-set-attribute** is reset to the new value, otherwise a new entry for the specified attribute is created at the appropriate location in the Event.

Parameters

Name	Type	Description
root-path	path	Either ~input%eX_Event or ~output%eX_Event
attribute	string	The name of the attribute as it appears in the e*Insight GUI.
value	string, XML or BIN	The value to which you want to set the attribute.
type	string	The type of Attribute. Examples: ▪ "BIN" - Interpreted as binary, however, must be suitably encoded for XML. ▪ "XML" - Interpreted as XML, however, must be Base64 encoded for XML. ▪ "STRING" - Interpreted as a string (default). ▪ "TRANSIENT" - Interpreted as a transient. The e*Insight engine does not process the value but simply return it as-is. ▪ "NUMBER" - Interpreted as a decimal number, however, must be given as a string. ▪ "BOOLEAN" - Interpreted as a boolean, such as "true" and "false".

Return Values

None.

Throws

None.

Example

```
(eX-set-attribute ~input%eX_Event "Is_Valid_account?" "no" "STRING")
=> sets the value of Is_Valid_account to "no".
```

eX-set-BP_EVENT

Syntax

```
(eX-set-BP_EVENT root-path event-type value)
```

Description

eX-set-BP_EVENT sets the value of the event type node in the e*Insight Event.

Parameters

Name	Type	Description
root-path	path	Either ~input%eX_Event or ~output%eX_Event
event-type	string	STATUS, ID, NAME, or TYPE
value	string	The value for the business process Event. For event-type "STATUS" value must be either "SUCCESS" or "FAILURE". For event-type "TYPE" value must be DO_ACTIVITY, START_BP, or UNDO_ACTIVITY.

Return Values

Boolean

Returns **#t** (true) except when an invalid parameter is passed, then **#f** (false) is returned.

Throws

None.

Example

```
(eX-set-BP_EVENT ~input%eX_Event "STATUS" "SUCCESS")

=> sets the status of the activity to SUCCESS
```

eX-get-BP_EVENT

Syntax

```
(eX-get-BP_EVENT root-path event-type)
```

Description

eX-get-BP_EVENT finds the path to the value for the event-type in the e*Insight Event named in the root-path.

Parameters

Name	Type	Description
root-path	path	Either ~input%eX_Event or ~output%eX_Event
event-type	string	STATUS, ID, NAME or TYPE

Return Values

Returns one of the following values:

Boolean
Returns **#f** (false) if no data is found.

path
Returns the path to the value in the e*Insight Event. Use **get** to return the actual value.

Throws

None.

Example

For an Event with an ID of 11111:

```
(get (eX-get-BP_EVENT ~input%eX_Event "ID"))
=> 11111
```

eX-get-Activity

Syntax

```
(eX-get-Activity root-path event-type)
```

Description

eX-get-Activity searches the e*Insight Event specified in for the name or ID of the current activity.

Parameters

Name	Type	Description
root-path	path	Either ~input%eX_Event or ~output%eX_Event
event-type	string	Either ID, NAME

Return Values

Returns one of the following values:

Boolean
Returns **#f** (false) if the requested value is not found.

path
Returns the path to the name of the current activity as found in the e*Insight Event. Use **get** to return the actual value.

Throws

None.

Example

For an Event with an activity name of "Check_Credit":

```
(get (eX-get-Activity ~input%eX_Event "NAME"))
=> Check_Credit
```

eX-set-Activity

Syntax

```
(eX-set-Activity root-path event-type data)
```

Description

eX-set-Activity sets the value of either the current activity name or ID.

Parameters

Name	Type	Description
root-path	path	Either ~input%eX_Event or ~output%eX_Event
event-type	string	Either ID, NAME
data	string	The value of the activity ID or NAME.

Return Values

None.

Throws

None.

Example

```
(eX-set-Activity ~input%eX_Event "ID" "12345")
=> sets the activity ID to "12345"
```

eX-string-set-attribute

Syntax

```
(eX-string-set-attribute root-path attribute value)
```

Description

eX-string-set-attribute automatically calls **eX-set-attribute** with the last argument as "STRING".

Parameters

Name	Type	Description
root-path	path	Either ~input%eX_Event or ~output%eX_Event
attribute	string	The name of the attribute as it appears in the e*Insight GUI.
value	string	The value to which you want to set the attribute.

Return Values

None.

Throws

None.

Example

```
(eX-sting-set-attribute ~input%eX_Event "Is_Valid_account?" "no")

=> sets the value of Is_Valid_account to "no".
```

eX-xml-set-attribute

Syntax

```
(eX-xml-set-attribute root-path attribute value)
```

Description

eX-xml-set-attribute automatically calls **eX-set-attribute** with last argument as "XML".

Parameters

Name	Type	Description
root-path	path	Either ~input%eX_Event or ~output%eX_Event
attribute	string	The name of the attribute as it appears in the e*Insight GUI.
value	XML	The base 64 encoded XML value to which you want to set the attribute.

Return Values

None.

Throws

None.

Example

```
(eX-xml-set-attribute ~input%eX_Event "Cust_name" (raw->base64
"<a>Bryce Ferney</a>)")

=> sets Cust_name to "PGE+QnJ5Y2UgRmVybmV5PC8+"
```

eX-bin-set-attribute

Syntax

```
(eX-bin-set-attribute root-path attribute value)
```

Description

eX-bin-set-attribute automatically calls **eX-set-attribute** with last argument as "BIN".

Parameters

Name	Type	Description
root-path	path	Either ~input%eX_Event or ~output%eX_Event
attribute	string	The name of the attribute as it appears in the e*Insight GUI.
value	string	The base 64 encoded binary value to which you want to set the attribute.

Return Values

None.

Throws

None.

Example

```
(eX-sting-set-attribute ~input%eX_Event "Cust_name" "<base64 encoded
binary data>")

=> sets Cust_name to the specified value
```

eX-count-local-attribute

Syntax

```
(eX-count-local-attribute root-path)
```

Description

eX-count-local-attribute counts the number of local attributes a specific e*Insight Event contains.

Parameters

Name	Type	Description
root-path	path	Either ~input%eX_Event or ~output%eX_Event

Return Values

integer
Returns 0 to n, depending on the number of attributes found.

Throws

None.

Example

For an Event containing three local attributes:

```
(eX-count-local-attribute ~input%eX_Event)

=> 3
```

eX-get-local-attribute

Syntax

```
(eX-get-local-attribute root-path attr-name)
```

Description

eX-get-local-attribute finds the path to the specified local attribute in the e*Insight Event named in the root-node.

Parameters

Name	Type	Description
root-path	path	Either ~input%eX_Event or ~output%eX_Event
attr-name	string	The name of the local attribute as it appears in the e*Insight GUI.

Return Values

Returns one of the following values:

Boolean
Returns **#f** (false) if the attribute value is not found.

path

Returns the path to the value of the local attribute. Use **get** to return the actual value.

Throws

None.

Example

For an Event where the value of the local attribute "Debit_Amount" is "500":

```
(get (eX-get-local-attribute ~input%eX_Event "Debit_Amount"))
=> 500
```

eX-set-local-attribute

Syntax

```
(eX-set-local-attribute root-path attr-name attr-value attr-type)
```

Description

If the local attribute exists in the Event specified in root-path, **eX-set-local-attribute** is reset to the new value; otherwise a new entry for the specified attribute is created at the appropriate location in the Event.

Parameters

Name	Type	Description
root-path	path	Either ~input%eX_Event or ~output%eX_Event
attr-name	string	The name of the attribute as it appears in the e*Insight GUI.
attr-value	string, XML or BIN	The value to which you want to set the attribute.
attr-type	string	The type of Attribute. Examples: ▪ "BIN" - Interpreted as binary, however, must be suitably encoded for XML. ▪ "XML" - Interpreted as XML, however, must be Base64 encoded for XML. ▪ "STRING" - Interpreted as a string (default). ▪ "TRANSIENT" - Interpreted as a transient. The e*Insight engine does not process the value but simply return it as-is. ▪ "NUMBER" - Interpreted as a decimal number, however, must be given as a string. ▪ "BOOLEAN" - Interpreted as a boolean, such as "true" and "false".

Return Values

None.

Throws

None.

Example

```
(eX-set-local-attribute ~input%eX_Event "Debit_Amount" "500"
"STRING")

=> sets the value of the local attribute "Debit_Amount" to 500.
```

eX-copy-no-attribute

Syntax

```
(eX-copy-no-attribute source-root-node dest-root-node)
```

Description

eX-copy-no-attribute copies all of the business process tracking information in the source e*Insight Event to the destination e*Insight Event. No attribute information is copied except for the machine-defined attribute **eX_eBPMServer**, which is used to return the e*Insight Event to the proper e*Insight engine.

The **eX_*Activity*_Do** (or Undo) Event published by the e*Insight engine contains tracking information (such as the business process instance ID, name of the activity, and so on) that must be included in the "Done" Event that is sent back to the e*Insight engine when the activity is finished.

eX-copy-no-attribute provides a convenient way for an activity Collaboration to copy the e*Insight tracking information from source to destination, without copying the input attribute information that does not belong in the "Done" Event. See **"Sending the "Done" Event Back to e*Insight (eIJSchema)" on page 395** 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"
```

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.

Decision Gate
Decision Gates control the logical flow of data-based decisions in the business process model. A Decision Gate outputs specific information when specified input conditions are met.

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 Business Integration Suite product suite that facilitates the automation of the business process flow of business 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 business protocols and your validation Collaborations.

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*Insight Administrator
An application within the Business Integration Solutions suite of products that you use to establish user security for e*Insight Business Process Manager (e*Insight).

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*Insight that contain the parameters of all the components that control, route, and transform data as it moves through e*Insight.

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

A

Activity BOB
  configuring **176**
  creating the Collaboration Rules scripts **176**
addATTRIBUTE **211**, **250**, **251**
APIs
  eX-bin-set-attribute **461**
  eX-copy-no-attribute **226**, **465**
  eX-count-attribute **212**, **453**
  eX-count-local-attribute **222**, **462**
  eX-get-Activity **216**, **457**
  eX-get-all-attribute **467**
  eX-get-all-local-attribute **468**
  eX-get-attribute **211**, **452**
  eX-get-BP_EVENT **215**, **456**
  eX-get-local-attribute **463**
  eX-set-Activity **217**, **458**
  eX-set-all-BP_EVENT **227**, **466**
  eX-set-attribute **213**, **454**
  eX-set-BP_EVENT **214**, **455**
  eX-set-local-attribute **224**, **464**
  eX-string-set-attribute **218**, **459**
  eX-xml-set-attribute **219**, **460**
  getBusinessModelInstanceId **306**
  getBusinessModelName **307**
attributes
  getting **395**, **398**
  global **58**
  input **59**
  input/output **59**
  local **59**
  output **59**
  setting **394**, **397**
  using local attributes to implement undo logic **59**
authenticate **326**

B

Basic **24**
BOB
  Activity BOB **54**, **380**
BP_EVENT **62**, **386**
business logic, testing **89**, **100**, **108**, **113**, **409**, **441**
business processes
  attributes, about **58**
  restarting **94**, **446**
  starting **393**, **396**

C

case study (order processing) **69**, **97**, **106**, **111**, **159**, **402**, **414**
checkoutActivityInstance **327**
checkUserPrivileges **363**
clearATTRIBUTE **212**
clearMessage **301**
Collaboration
  START_BP Collaboration **49**, **376**
Collaborations
  configuring **85**, **172**, **176**, **430**, **434**
  Do and Undo logic in **52**, **379**
  eX_Activity **50**, **54**, **381**
  eX_from_eBPM **46**, **373**
  eX_Resubmitter **48**, **374**
  eX_to_Activity **52**, **379**
  eX_to_eBPM **373**
configuring
  e*Insight engine **32**, **167**, **173**, **369**
  editing the eX_eBPM engine's configuration file **79**, **149**, **167**, **173**, **427**
  eX_Resubmitter BOB **47**, **374**
  eX_to_Activity e*Way **52**, **378**
  Send_Status e*Way **85**, **407**, **435**
  START_BP e*Way **80**, **168**, **174**, **428**
  starting a business process **393**, **396**
  user-defined e*Gate components **167**, **174**, **179**, **183**, **406**
Configuring the START_BP Component **49**, **376**
conventions, writing in document **16**
Copy the eISchema **25**
countATTRIBUTE **252**

D

demonstrating business process restart **94**, **446**
demonstrating business process undo **92**, **444**
Do and Undo logic in an Activity collaboration **52**, **379**
Driver Type **38**

E

e*Gate Schema for e*Insight **28**, **364**
e*Insight
  schema components **28**, **185**, **364**
  sending the "Done" Event back **395**, **399**
e*Insight Authorization Activity Implementation

97–102, 402–412
e*Insight Business Process Manager **21**
e*Insight engine
  configuring **167, 173**
e*Insight ETD, understanding **57–67**
e*Insight Helper Monk Functions **451–468**
e*Insight Implementation **68–96, 413–448**
e*Insight Java Helper Methods **209–298**
e*Insight Schema components overview **28, 364**
e*Insight Sub-Process Implementation **116–127, 128–141**
e*Insight User Activity Implementation **104–115**
e*Insight User Activity Methods **299–363**
e*Ways
  Send_Status, configuring **85, 407, 435**
  START_BP, configuring **80, 168, 174, 428**
error handling **48, 375**
  connection failure handling **48**
  data failure handling **48**
  normal event failure handling **375**
  special event failure handling **375**
error type **48, 374**
  connection **48, 374**
  data **48, 374**
ETD Monk
  structure **382**
eX **284**
eX-bin-set-attribute **461**
eX-copy-no-attribute **226, 465**
eX-count-attribute **212, 453**
eX-count-local-attribute **222, 462**
eX-get-Activity **216, 457**
eX-get-all-attribute **467**
eX-get-all-local-attribute **468**
eX-get-attribute **211, 452**
eX-get-BP_EVENT **215, 456**
eX-get-local-attribute **463**
eXSchema **364**
eXSchema, copying **25**
eX-set-Activity **217, 458**
eX-set-all-BP_EVENT **227, 466**
eX-set-attribute **213, 454**
eX-set-BP_EVENT **214, 455**
eX-set-local-attribute **224, 464**
eX-string-set-attribute **218, 459**
eX-xml-set-attribute **219, 460**

F

from **286**
from_eBPMConvert **286**

G

getACTIVITY **253**
getActivityAttributesCount **302**
getActivityAttributeValue **303**
getActivityGlobalAttributeNames **328**
getActivityInstanceEndTime **329**
getActivityInstanceStartTime **330**
getActivityInstanceStatus **331**
getActivityName **304, 332**
getAssignedBPIIdByState **333**
getATTRIBUTE **215, 255**
getATTRIBUTE_VALUE **214, 254**
getAuthorizationActivityNames **334**
getBP_EVENT **287**
getBPI_ID **256**
getBPIStack **335**
getBusinessModelId **305**
getBusinessModelInstanceId **306**
getBusinessModelInstanceName **337**
getBusinessModelInstancesIds **336**
getBusinessModelInstanceStatus **338**
getBusinessModelName **307, 319, 339**
getEnabledBusinessModelId **340**
getEnabledBusinessModelsIds **341**
getENCODING **231**
getGlobalAttributeCount **308**
getGlobalAttributeDefaultValue **342**
getGlobalAttributeDirection **343**
getGlobalAttributeNames **344**
getGlobalAttributeType **309, 345**
getGlobalAttributeValue **310, 346**
getID **216, 257**
getLocalAttributeNames **347**
getLocalAttributeType **348**
getLocalAttributeValue **349**
getLOCATION **232**
getMessageStatus **350**
getMsgType **311**
getNAME **217, 233, 258**
getSTATUS **259**
getTP_EVENT **288**
getTYPE **234, 260**
getUser **351**
getUserActivityNames **352**
getUUID **353**
getVALUE **235**
global attributes **58**

H

hasACTIVITY **261**
hasBP_EVENT **289**
hasBPI_ID **262**

hasENCODING 236
hasID 218, 263
hasLOCATION 237
hasNAME 219, 264
hasSTATUS 265
hasTP_EVENT 290

I

implementation
  basic information 22, 24
  configuring the e*Gate components 26
  configuring the e*Insight schema based on the business process 25
  copying the eXSchema 25
  creating a business process 25
  e*Insight 68–96, 413–448
  e*Insight Authorization Activity 97–102, 402–412
  e*Insight User Activity 104–115
  Remote Sub-Process 142–158
  road map 24
  Sub-Process 116–127, 128–141
  testing and tuning the system 26
initialize 354
input attributes 59
input/output attributes 59
Introducing e*Insight Business Process Manager (e*Insight) 21

J

Java APIs
  addATTRIBUTE 211, 250, 251
  authenticate 326
  checkoutActivityInstance 327
  checkUserPrivileges 363
  clearATTRIBUTE 212
  clearMessage 301
  countATTRIBUTE 252
  from_eBPMConvert 286
  getACTIVITY 253
  getActivityAttributesCount 302
  getActivityAttributeValue 303
  getActivityGlobalAttributeNames 328
  getActivityInstanceEndTime 329
  getActivityInstanceStartTime 330
  getActivityInstanceStatus 331
  getActivityName 304, 332
  getAssignedBPIIdByState 333
  getATTRIBUTE 215, 255
  getATTRIBUTE_VALUE 214, 254
  getAuthorizationActivityNames 334
  getBP_EVENT 287
  getBPI_ID 256
  getBPIStack 335
  getBusinessModelId 305
  getBusinessModelInstanceName 337
  getBusinessModelInstancesIds 336
  getBusinessModelInstanceStatus 338
  getBusinessModelName 319, 339
  getEnabledBusinessModelId 340
  getEnabledBusinessModelsIds 341
  getENCODING 231
  getGlobalAttributeCount 308
  getGlobalAttributeDefaultValue 342
  getGlobalAttributeDirection 343
  getGlobalAttributeNames 344
  getGlobalAttributeType 309, 345
  getGlobalAttributeValue 310, 346
  getID 216, 257
  getLocalAttributeNames 347
  getLocalAttributeType 348
  getLocalAttributeValue 349
  getLOCATION 232
  getMessageStatus 350
  getMsgType 311
  getNAME 217, 233, 258
  getSTATUS 259
  getTP_EVENT 288
  getTYPE 234, 260
  getUser 351
  getUserActivityNames 352
  getUUID 353
  getVALUE 235
  hasACTIVITY 261
  hasBP_EVENT 289
  hasBPI_ID 262
  hasENCODING 236
  hasID 218, 263
  hasLOCATION 237
  hasNAME 219, 264
  hasSTATUS 265
  hasTP_EVENT 290
  initialize 354
  marshal 220, 238, 266, 291
  omitACTIVITY 267
  omitBP_EVENT 292
  omitBPI_ID 268
  omitENCODING 239
  omitID 221, 269
  omitLOCATION 240
  omitNAME 222, 270
  omitSTATUS 271
  omitTP_EVENT 293
  refreshCachedMemory 355
  releaseActivityInstance 356
  releaseResources 357

removeActivity **312**
removeATTRIBUTE **223**, **272**
removeGlobalAttribute **313**
resetUser **358**
sendMessage **359**
setACTIVITY **273**
setActivityAttributeValue **314**
setActivityName **315**
setATTRIBUTE **224**, **274**
setBP_EVENT **294**
setBPI_ID **276**
setBPIStack **316**
setBusinessModelId **318**
setBusinessModelInstanceId **317**
setENCODING **241**
setEventInfo **277**
setGlobalAttributeValue **360**
setID **226**, **278**
setLocalAttributeValue **361**
setLOCATION **242**
setMsgType **321**
setNAME **227**, **243**, **279**
setSTATUS **280**
setStatus **322**
setTP_EVENT **295**
setTYPE **244**, **281**
setUser **362**
setVALUE **245**
to_eBPMConvert **296**
toString **228**, **246**, **282**, **297**
toXML **323**
unmarshal **229**, **247**, **283**, **298**
Java APIssetGlobalAttributeValue **320**

L

local attributes **59**
using to implement undo logic **59**

M

marshal **220**, **238**, **266**, **291**
Monk functions see functions

O

omitACTIVITY **267**
omitBP_EVENT **292**
omitBPI_ID **268**
omitENCODING **239**
omitID **221**, **269**
omitLOCATION **240**
omitNAME **222**, **270**
omitSTATUS **271**
omitTP_EVENT **293**
order processing case study **69**, **97**, **106**, **111**, **159**, **402**, **414**
output attributes **59**
Overview **110**

R

refreshCachedMemory **355**
releaseActivityInstance **356**
releaseResources **357**
Remote Sub-Process Implementation **142–158**
removeActivity **312**
removeATTRIBUTE **223**, **272**
removeGlobalAttribute **313**
resetUser **358**
restart, demonstrating **94**, **446**

S

schema, copying **25**
SeeBeyond eBusiness Integration Suite **??–22**
sending the "Done" Event back to e*Insight **395**, **399**
sendMessage **359**
setACTIVITY **273**
setActivityAttributeValue **314**
setActivityName **315**
setATTRIBUTE **224**, **274**
setBP_EVENT **294**
setBPI_ID **276**
setBPIStack **316**
setBusinessModelId **318**
setBusinessModelInstanceId **317**
setENCODING **241**
setEventInfo **277**
setGlobalAttributeValue **320**, **360**
setI **226**
setID **278**
setLocalAttributeValue **361**
setLOCATION **242**
setMsgType **321**
setNAME **227**, **243**, **279**
setSTATUS **280**
setStatus **322**
setting attributes **394**, **397**
setTP_EVENT **295**
setTYPE **244**, **281**
setUser **362**
setVALUE **245**
START_BP component, configuring **49**, **376**
START_BP e*Way, configuring **80**, **168**, **174**, **428**
starting a business process **393**, **396**

T

testing the standard business logic **89**, **100**, **108**, **113**, **409**, **441**
to_eBPMConvert **296**
toString **228**, **246**, **282**, **297**
toXML **323**

U

understanding the e*Insight ETD **57–67**
Undo, demonstrating **92**, **444**
unmarshal **229**, **247**, **283**, **298**
user-defined e*Gate components, configuring **167**, **174**, **179**, **183**, **406**

X

XML
- element with sub-elements **383**
- element without sub-elements **383**
- ETD structure for an XML attribute **384**
- structure for the e*Xchange Event **449–450**