

**SeeBeyond™ eBusiness Integration Suite**

# e\*Insight Business Process Manager Implementation Guide

*Release 4.5.3*



The information contained in this document is subject to change and is updated periodically to reflect changes to the applicable software. Although every effort has been made to ensure the accuracy of this document, SeeBeyond Technology Corporation (SeeBeyond) assumes no responsibility for any errors that may appear herein. The software described in this document is furnished under a License Agreement and may be used or copied only in accordance with the terms of such License Agreement. Printing, copying, or reproducing this document in any fashion is prohibited except in accordance with the License Agreement. The contents of this document are designated as being confidential and proprietary; are considered to be trade secrets of SeeBeyond; and may be used only in accordance with the License Agreement, as protected and enforceable by law. SeeBeyond assumes no responsibility for the use or reliability of its software on platforms that are not supported by SeeBeyond.

e\*Gate, e\*Insight, e\*Way, e\*Xchange, e\*Xpressway, eBI, iBridge, Intelligent Bridge, IQ, SeeBeyond, and the SeeBeyond logo are trademarks and service marks of SeeBeyond Technology Corporation. All other brands or product names are trademarks of their respective companies.

© 1999–2002 by SeeBeyond Technology Corporation. All Rights Reserved. This work is protected as an unpublished work under the copyright laws.

**This work is confidential and proprietary information of SeeBeyond and must be maintained in strict confidence.**

Version 20020903130619.

# Contents

---

## Chapter 1

<b>Introduction</b>	<b>18</b>
Document Purpose and Scope	18
Intended Audience	18
Writing Conventions	19
Supporting Documents	20
SeeBeyond Web Site	20

---

## Chapter 2

<b>Introduction to the SeeBeyond eBI Suite</b>	<b>21</b>
SeeBeyond eBusiness Integration Suite	21
SeeBeyond eBusiness Integration Suite Components	22
e*Gate Integrator Components	23
Introducing e*Insight Business Process Manager (e*Insight)	24
Building an eApplication	25
Basic Information	25

---

## Chapter 3

<b>Implementation Overview</b>	<b>26</b>
Basic Information	26
Implementation Road Map	26
Step 1: Create a Business Process	27
Step 2: Copy the e*Insight Schema	27
Step 3: Configure the e*Insight Schema Based on the Business Process	27
Step 4: Configure the e*Gate Components	28
Step 5: Test and Tune the System	28
The e*Insight Schema	28
The eIJSchema (Java)	28
The eISchema (Classic)	29

Chapter 4

<b>e*Insight Schema Components (eIJSchema)</b>	<b>30</b>
<b>e*Insight Schema Components Overview</b>	<b>30</b>
Schema Component Types	30
e*Insight Schema Component Relationships	31
e*Insight Business Process Manager Components	32
Start a Business Processes	32
Run a Business Processes	32
Implement Business Process Activities	33
e*Insight Engine	33
Configuring the e*Insight Engine	33
<b>Using XA</b>	<b>34</b>
How XA Protocol Works	34
<b>Database Requirements for XA Compliance</b>	<b>34</b>
Oracle	34
Sybase	35
Microsoft SQL Server	35
<b>e*Way Connection Configuration Changes</b>	<b>35</b>
ORACLE DataSource Section	35
Parameter: XA Compliant	35
Parameter: Driver Type	36
Parameter: ServerName	36
Parameter: PortNumber	36
Parameter: DatabaseName	36
Parameter: Row Prefetch Size	36
Parameter: Execute Batch Size	36
Parameter: Attribute Value Pair Separator	36
Parameter: Attribute Value Pairs	36
Parameter: Standard Class	37
Parameter: XA Class	37
Sybase DataSource Section	37
Parameter: ServerName	37
Parameter: PortNumber	37
Parameter: DatabaseName	37
Parameter: Attribute Value Pairs	37
Microsoft SQL Server DataSource Section	37
Parameter: PortNumber	37
eBPM Settings Section	37
Parameter: JDBC URL String	38
<b>Performance Improvements</b>	<b>38</b>
eBPM Tuning	38
Parameter: Commit Frequency	38
Parameter: Maximum Commit Frequency	38
Parameter: Receive Wait Timeout	39
Parameter: Stability Threshold	39
Parameter: Show Change Audit	39
<b>Connection Management</b>	<b>39</b>
Connector Section Changes	40

Parameter: Connection Inactivity Timeout	40
Parameter: Connection Establishment Mode	40
Parameter: Connection Verification Interval	40
<b>Manual Database XA Recovery</b>	<b>40</b>
Oracle Database	41
<b>Engine Connection Configuration</b>	<b>41</b>
elcr_eBPM Collaboration	50
el_Resubmitter BOB	51
Configuring the el_Resubmitter BOB	52
el_Resubmitter Collaboration	52
Failed Event Handling by the e*Insight Engine	52
Error Types	52
Error Handling	52
START_BP Component	53
Configuring the START_BP Component	53
START_BP Collaboration	53
Multi-Mode Activity e*Way	54
eX_Activity Collaboration	54
Single-Mode Activity e*Way	55
Configuring the eX_to_Activity e*Way	56
eX_to_Activity Collaboration	56
eX_from_Activity Collaboration	57
Activity BOB	58
eX_Activity Collaboration	58
<b>Using Monk in elJSchema</b>	<b>59</b>
<b>Updating an elSchema to use the elJSchema engine</b>	<b>59</b>

---

## Chapter 5

<b>e*Insight Schema Components (elSchema)</b>	<b>61</b>
The Purpose of the e*Gate Schema for e*Insight	61
e*Insight Components	61
<b>e*Insight Schema Components Overview</b>	<b>61</b>
Additional Components	62
e*Insight Schema Component Relationships Diagram	63
<b>e*Insight Business Process Manager Components</b>	<b>64</b>
Components That Run Business Processes	64
Components that Start Business Processes	65
Components that Implement Business Process Activities	65
e*Insight Engine	65
Configuring the e*Insight Engine	66
eX_from_eBPM Collaboration	70
eX_to_eBPM Collaboration	70
el_Resubmitter BOB	70
Configuring the el_Resubmitter BOB	71
el_Resubmitter Collaboration	71
Failed Event Handling by the e*Insight Engine	71
Error Types	71
Error Handling	72

START_BP Component	72
Configuring the START_BP Component	73
START_BP Collaboration	73
Activity e*Way	74
Configuring the eX_to_Activity e*Way	75
eX_to_Activity Collaboration	76
eX_from_Activity Collaboration	77
Activity BOB	77
eX_Activity Collaboration	78

---

## Chapter 6

# Understanding the e\*Insight ETD 79

<b>Selecting an ETD</b>	<b>79</b>
About Business Process Attributes	80
Global Attributes	80
Local Attributes	81
<b>Activity Specific ETDs</b>	<b>82</b>
GlobalAttributes	82
LocalAttributes	82
EventSpecifics	82
<b>eI_StandardEvent</b>	<b>84</b>
BP_EVENT	84
BP_EVENT Element	85
BP_EVENT.ACTIVITY Nodes	86
ACTIVITY Element	86
ACTIVITY.ATTRIBUTE Element	87
<b>BP_EVENT.ATTRIBUTE Nodes</b>	<b>87</b>
<b>e*Insight ETD for Monk—eX_Standard_Event.ssc</b>	<b>89</b>
ETD Structure	89
XML Element with Sub-elements	89
XML Element without sub-elements	90
XML Attribute	90
Element Overview	91
Example: XML Element with Sub-elements	91
Example: XML Element with Attributes	92
<b>Using eX_Standard_Event.ssc</b>	<b>93</b>
BP_EVENT	93
BP_EVENT.AS Nodes	93
BP_EVENT.CT.DSN.DS.ACTIVITY Nodes	95
ACTIVITY.AS Nodes	95
ACTIVITY.CT.DSN.DS.ATTRIBUTE Nodes	95
BP_EVENT.CT.DSN.DS.ATTRIBUTE.AS Nodes	96

---

 Chapter 7

<b>Common Configuration Tasks</b>	<b>98</b>
<b>Copy the e*Insight Schema</b>	<b>98</b>
Using the e*Insight GUI	98
Copying the Schema from the Registry Host	99
Installing from the CD	100
<b>Sending Messages to the e*Insight Engine (eIJSchema)</b>	<b>100</b>
Starting a Business Process (eIJSchema)	100
Setting Attributes (eIJSchema)	101
Getting Attributes (eIJSchema)	102
Sending the “Done” Event Back to e*Insight (eIJSchema)	102
<b>Sending Messages to the e*Insight Engine (eISchema)</b>	<b>103</b>
Starting a Business Process (eISchema)	103
Setting Attributes (eISchema)	104
Setting Attributes in a Monk Collaboration	104
Setting Attributes in a Java Collaboration	105
Getting Attributes (eISchema)	105
Getting Attributes in a Monk Collaboration	105
Getting Attributes in a Java Collaboration	106
Sending the “Done” Event Back to e*Insight (eISchema)	106

---

## Chapter 8

<b>e*Insight Implementation (eIJSchema)</b>	<b>109</b>
<b>Overview</b>	<b>109</b>
Case Study: Payroll Processing	110
<b>Create the Payroll BP in e*Insight</b>	<b>113</b>
Creating the processes performing the Activities	115
Configuring the e*Insight Script for Update_Status	115
<b>Configure the Integration Schema (e*Insight)</b>	<b>116</b>
Integration Schema Activity Components Summary	116
Creating the eX_Check_Eligibility Multi-Mode e*Way	117
Creating the eX_Calculate_Bonus BOB	119
Process_Payroll e*Way Configuration	120
<b>Configure the Integration Schema (e*Gate)</b>	<b>120</b>
Configure the e*Insight Engine	120
Edit the elcp_eInsightEngine Connection Configuration File	120
Configure the JMS Connection	120
<b>Configure User-defined e*Gate Components</b>	<b>121</b>
Configuration Order for the User-defined Components	121
Configure the START_BP e*Way	121
Step 1: Create the START_BP e*Way	122
Step 2: Create the Input ETD	122
Step 3: Create the START_BP Collaboration	123
Step 4: Configure the Collaboration in the GUI	126

<b>Configure the Process_Payroll e*Way</b>	<b>127</b>
Step 1: Create the Output ETD:PayrollProcess.xsc using Java	127
Step 3: Create the Process_Payroll Collaboration Rule	128
Step 4: Configure the Collaboration	129
Step 1: Configure the e*Way	131
<b>Run and Test the e*Insight Scenario</b>	<b>131</b>
Testing the Standard Business Logic	131
Payroll Processing	131
Not Eligible Processing	133
<b>Demonstrating Business Process Undo Functionality</b>	<b>135</b>
Manual Undo	136
<b>Demonstrating Business Process Restart Functionality</b>	<b>137</b>
Repairing a String Attribute	137

## Chapter 9

### **e\*Insight Authorization Activity Implementation (eIJSchema) 140**

<b>Overview</b>	<b>140</b>
Case Study: Payroll Processing	140
<b>Step 1: Update the Payroll BP in e*Insight</b>	<b>142</b>
Creating the processes performing the Activities	143
Configuring the e*Insight Script for Bonus_Refused	143
<b>Step 2: Run and Test the e*Insight scenario</b>	<b>143</b>
Testing the Standard Business Logic	144
Authorized Processing	144
Not Authorized Processing	146

## Chapter 10

### **e\*Insight Implementation (eISchema) 147**

<b>Overview</b>	<b>147</b>
Case Study: Order Processing	148
<b>Create the ProcessOrder BP in e*Insight</b>	<b>151</b>
Creating the processes performing the Activities	153
Configuring the e*Insight Script for Ship_Ord	153
<b>Configure the Integration Schema</b>	<b>154</b>
Integration Schema Activity Components Summary	154
Creating the eX_Check_Inv BOB	155
Creating the eX_Out_of_Inv BOB	158
Send_Status e*Way Configuration	161
<b>Configure the e*Insight Engine</b>	<b>162</b>
Edit the eX_eBPM Configuration File	162
<b>Configure User-defined e*Gate Components</b>	<b>162</b>
Configuration Order for the User-defined Components	162
<b>Configure the START_BP e*Way</b>	<b>163</b>
Step 1: Create the START_BP e*Way using Monk	163



Step 2: Create the Input ETD using Monk	164
Step 3: Create the START_BP CRS using Monk	164
Step 4: Configure the START_BP Collaboration in the GUI using Monk	165
Step 1: Create the START_BP e*Way using Java	166
Step 2: Create the Input ETD using Java	166
Step 3: Create the START_BP Collaboration using Java	167
Step 4: Configure the Collaboration in the GUI using Java	169
<b>Configure the Send_Status e*Way</b>	<b>170</b>
Step 1: Configure the eX_Send_Status e*Way using Monk	170
Step 2: Create the Output ETD using Monk	171
Step 3: Create the eX_Send_Status.tsc CRS using Monk	171
Step 4: Configure the Collaboration using Monk	172
Step 1: Configure the e*Way using Java	173
Step 2: Create the Output ETD: SendStatus.xsc using Java	173
Step 3: Create the Send_Status Collaboration Rule using Java	174
Step 4: Configure the Collaboration using Java	175
<b>Run and Test the e*Insight scenario</b>	<b>176</b>
Testing the Standard Business Logic	176
In-Stock Processing	176
Out-of-Stock Processing	178
Demonstrating Business Process Undo Functionality	179
Manual Undo	179
Demonstrating Business Process Restart Functionality	181
Repairing a String Attribute	181

---

## Chapter 11

### e\*Insight Authorization Activity Implementation (eISchema) 184

<b>Overview</b>	<b>184</b>
Case Study: Order Processing	184
<b>Step 1: Create the ProcessOrder BP in e*Insight</b>	<b>187</b>
<b>Step 2: Configure the Integration Schema</b>	<b>188</b>
Integration Schema Activity Components Summary	188
<b>Step 3: Configure User-defined e*Gate Components</b>	<b>188</b>
Configure the Activity BOB CRS in the Enterprise Manager GUI	188
Configure the Activity BOB Collaborations in the Enterprise Manager GUI	189
<b>Configure the Authorize_Quantity e*Way</b>	<b>189</b>
Step 2: Create the Authorize_Quantity.tsc CRS	189
Step 3: Configure the e*Way	190
Step 4: Configure the Collaboration	190
<b>Step 5: Run and Test the e*Insight scenario</b>	<b>191</b>
Testing the Standard Business Logic	191
Authorized Processing	191
Not Authorized Processing	193

---

**Chapter 12**

<b>e*Insight User Activity Implementation</b>	<b>195</b>
<b>Overview of the User Activity</b>	<b>195</b>
User Activity Security	195
Deployment of the User Activity	196
<b>Overview of the Payroll BP</b>	<b>196</b>
<b>Overview</b>	<b>197</b>
Case Study: Payroll Processing with User Activity	197
<b>Step 1: Update the Payroll BP in e*Insight</b>	<b>197</b>
<b>Step 2: Configure the Integration Schema</b>	<b>198</b>
<b>Step 3: Run and Test the e*Insight scenario</b>	<b>199</b>
Testing the User Activity	199
<b>Overview of the ProcessOrder BP</b>	<b>201</b>
<b>Overview</b>	<b>202</b>
Case Study: Order Processing with User Activity	202
<b>Step 1: Update the ProcessOrder BP in e*Insight</b>	<b>202</b>
<b>Step 2: Configure the Integration Schema</b>	<b>203</b>
<b>Step 3: Run and Test the e*Insight scenario</b>	<b>204</b>
Testing the User Activity	204

---

**Chapter 13**

<b>e*Insight Sub-Process Implementation (eIJSchema)</b>	<b>207</b>
<b>Overview of the Sub-Process Example</b>	<b>207</b>
<b>Create the CalculateBonus BP in e*Insight</b>	<b>208</b>
<b>Configure the Integration Schema for CalculateBonus</b>	<b>209</b>
<b>Modify the Payroll BP in e*Insight</b>	<b>210</b>
<b>Configure the Integration Schema for Payroll</b>	<b>211</b>
<b>Run and Test the e*Insight scenario</b>	<b>211</b>
<b>Overview of the Dynamic Sub-Process Example</b>	<b>212</b>
<b>Create the accounts BP in e*Insight</b>	<b>213</b>
<b>Configure the Integration Schema for accounts</b>	<b>213</b>
Creating the CRS in e*Gate	214
<b>Create the marketing BP in e*Insight</b>	<b>215</b>
<b>Configure the Integration Schema for marketing</b>	<b>215</b>
Creating the CRS for eX_Calculate_Bonus_marketing in e*Gate	215
<b>Modify the Payroll BP in e*Insight</b>	<b>217</b>

Configure the Integration Schema for Payroll	217
Run and Test the e*Insight scenario	218

---

## Chapter 14

<b>e*Insight Sub-Process Implementation (eISchema)</b>	<b>219</b>
Overview of the Sub-Process Example	219
Create the CheckInventory BP in e*Insight	220
Configure the Integration Schema for CheckInventory	221
Modify the ProcessOrder BP in e*Insight	222
Configure the Integration Schema for ProcessOrder	223
Run and Test the e*Insight scenario	223
Overview of the Dynamic Sub-Process Example	224
Create the CA BP in e*Insight	225
Configure the Integration Schema for CA	225
Creating the CRS in e*Gate	227
Create the OR BP in e*Insight	228
Configure the Integration Schema for OR	228
Creating the CRS in e*Gate	230
Modify the ProcessOrder BP in e*Insight	231
Configure the Integration Schema for ProcessOrder	232
Run and Test the e*Insight scenario	232

---

## Chapter 15

<b>e*Insight Remote Sub-Process Implementation</b>	<b>233</b>
Overview	233
Overview of the Remote Sub-Process	233
Installation and Configuration of Tomcat	234
Installing Tomcat	234
Configuring Tomcat	235
Deploying the SOAP Service	235
Installation of Tomcat and e*Insight on Different Hosts	236
Overview of the Remote Sub-Process Example (eIJSchema)	237
Install and configure Tomcat	237
Create the CalculateBonus BP in e*Insight	238
Configure the Integration Schema for CalculateBonus	239

Create the CalculateBonus Schema	239
Configure the CalculateBonus Schema	240
Create the Calculate_Bonus activity BOB	240
Edit the elcp_eInsightEngine Connection Configuration File	240
Configure the JMS Connection	240
<b>Modify the Payroll BP in e*Insight</b>	<b>241</b>
<b>Configure the Integration Schema for Payroll</b>	<b>242</b>
<b>Run and Test the e*Insight scenario</b>	<b>243</b>
<b>Overview of the Remote Sub-Process Example (elSchema)</b>	<b>243</b>
<b>Install and configure Tomcat</b>	<b>244</b>
<b>Create the CheckInventory BP in e*Insight</b>	<b>244</b>
<b>Configure the Integration Schema for CheckInventory</b>	<b>245</b>
Create the CheckInventory Schema	245
Configure the e*Insight engine	246
Create the Check_Inv activity BOB	246
<b>Modify the ProcessOrder BP in e*Insight</b>	<b>247</b>
<b>Configure the Integration Schema for ProcessOrder</b>	<b>248</b>
<b>Run and Test the e*Insight scenario</b>	<b>248</b>
In-Stock Processing	248

---

## Chapter 16

<b>Active and Passive Modes</b>	<b>250</b>
<b>Overview</b>	<b>250</b>
Case Study	250
Case Study - Active Control Mode	251
Case Study - Passive Control Mode	252
<b>Create the Order BP in e*Insight</b>	<b>254</b>
<b>Configure the Integration Schema (elJSchema)</b>	<b>255</b>
Integration Schema Activity Components Summary	255
Creating the eX_Bill_Customer BOB	256
Creating the eX_Ship_Order BOB	257
<b>Configure the e*Insight Engine (elJSchema)</b>	<b>258</b>
Edit the elcp_eInsightEngine Connection Configuration File	258
Configure the JMS Connection	258
<b>Configure User-defined e*Gate Components (elJSchema)</b>	<b>258</b>
Configuration Order for the User-defined Components	259
<b>Configure the START_BP e*Way</b>	<b>259</b>
Step 1: Create the START_BP e*Way	259
Step 2: Create the Input ETD	260
Step 3: Create the START_BP Collaboration	261
Step 4: Configure the Collaboration in the GUI	263
<b>Configure the Integration Schema (elSchema)</b>	<b>264</b>

Integration Schema Activity Components Summary	264
<b>Configure the e*Insight Engine (eISchema)</b>	<b>264</b>
Edit the eX_eBPM Engine's Configuration File	264
<b>Configure User-defined e*Gate Components (eISchema)</b>	<b>265</b>
Configuration Order for the User-defined Components	265
<b>Configure the START_BP e*Way</b>	<b>265</b>
Step 1: Create the Input ETD	265
Step 2: Create the START_BP Collaboration Rules Script (CRS)	266
Step 3: Add the e*Way and Create the e*Way Configuration File	266
Step 4: Configure the Collaboration in the GUI	267
<b>Configure the Activity BOBs</b>	<b>267</b>
Create the Activity BOB CRSs	267
Configure the Activity BOB Collaborations in the Enterprise Manager GUI	268
<b>Run and Test the e*Insight scenario</b>	<b>269</b>
<b>Case Study - Passive Control Mode</b>	<b>270</b>
<b>Passive Control Mode (eIJSchema)</b>	<b>270</b>
Modify the Order BP in e*Insight (eIJSchema)	270
Modify User-defined e*Gate Components (eIJSchema)	270
Configuration Order for the User-defined Components	270
Configure the Bill_Customer Collaboration Rule and Collaboration	270
Run and Test the e*Insight scenario	273
<b>Passive Control Mode (eISchema)</b>	<b>274</b>
Modify the Order BP in e*Insight (eISchema)	274
Modify User-defined e*Gate Components (eISchema)	274
Configuration Order for the User-defined Components	274
Configure the Bill_Customer Collaboration Rule and Collaboration	275
Run and Test the e*Insight scenario	275

---

## Chapter 17

<b>e*Insight Performance</b>	<b>276</b>
<b>Performance Improvements Using eIJSchema</b>	<b>276</b>
Instance Caching	276
Using Multiple e*Insight Engines	277
e*Insight Engine Affinity (eIJSchema)	280
Using Engine Affinity with e*Gate 4.5.2 or 4.5.3 — Active Mode	280
Configuring the Engine Affinity JMS Properties	281
Using Engine Affinity with e*Gate 4.5.1 — Active Mode	281
Using Engine Affinity with e*Gate 4.5.2 or 4.5.3 — Passive Mode	282
Using Engine Affinity with e*Gate 4.5.1 — Passive Mode	283
Using e*Xchange with e*Insight (eIJSchema)	284
Using Binary XML (eIJSchema)	284
Subscribing to Event Types	285
Subscribing to a Single "Go" Event	285
Configuring a Separate Collaboration for Do and Undo Events	285
Removing Unnecessary Subscriptions	286
Event Type "get" Interval — JMS Server	286
Event Type "get" Interval — e*Insight Engine	286

Review JVM Settings	286
<b>Performance Improvements Using eISchema</b>	<b>287</b>
Instance Caching	287
Using Multiple e*Insight Engines (eISchema)	287
e*Insight Engine Affinity (eISchema)	288
Manually Publishing Events using eX-event-sendback-to-sender	289
Exchange Data Interval (eISchema)	290
Review JVM Settings	290
<b>General e*Insight Performance Tips</b>	<b>290</b>

## Chapter 18

<b>Troubleshooting</b>	<b>292</b>
<b>Log Files</b>	<b>292</b>
Generating Log Files	292
Reading Log Files	293
<b>Common Problems</b>	<b>295</b>
<b>General Troubleshooting Tips</b>	<b>299</b>
Locating the problem	299
Viewing the Message Content	300

## Chapter 19

<b>e*Insight Helper Monk Functions</b>	<b>302</b>
<b>e*Insight Helper Monk Functions</b>	<b>303</b>
eX-get-attribute	304
eX-count-attribute	305
eX-set-attribute	306
eX-set-BP_EVENT	307
eX-get-BP_EVENT	308
eX-get-Activity	309
eX-set-Activity	310
eX-string-set-attribute	311
eX-xml-set-attribute	312
eX-bin-set-attribute	313
eX-count-local-attribute	314
eX-get-local-attribute	315
eX-set-local-attribute	316
eX-copy-no-attribute	317
eX-set-all-BP_EVENT	318
eX-get-all-attribute	319
eX-get-all-local-attribute	320

## Chapter 20

<b>Java Helper Methods</b>	<b>321</b>
<b>ACTIVITY Class</b>	<b>322</b>
addATTRIBUTE	323

clearATTRIBUTE	324
countATTRIBUTE	325
getAttribute_VALUE	326
getAttribute	327
getID	328
getName	329
hasID	330
hasNAME	331
marshal	332
omitID	333
omitNAME	334
removeATTRIBUTE	335
setAttribute	336
setID	338
setName	339
toString	340
unmarshal	341
<b>ATTRIBUTE Class</b>	<b>342</b>
getENCODING	343
getLocation	344
getName	345
getType	346
getValue	347
hasENCODING	348
hasLOCATION	349
marshal	350
omitENCODING	351
omitLOCATION	352
setENCODING	353
setLOCATION	354
setName	355
setType	356
setValue	357
toString	358
unmarshal	359
<b>BP_EVENT Class</b>	<b>360</b>
addATTRIBUTE	362
clearATTRIBUTE	363
countATTRIBUTE	364
getACTIVITY	365
getAttribute_VALUE	366
getAttribute	367
getBPI_ID	368
getID	369
getName	370
getStatus	371
getType	372
hasACTIVITY	373
hasBPI_ID	374
hasID	375
hasNAME	376
hasSTATUS	377
marshal	378
omitACTIVITY	379
omitBPI_ID	380
omitID	381
omitNAME	382
omitSTATUS	383
removeATTRIBUTE	384
setACTIVITY	385
setAttribute	386
setBPI_ID	388
setEventInfo	389

setID	390
setNAME	391
setStatus	392
setType	393
toString	394
unmarshal	395
<b>eX_StandardEvent Class</b>	<b>396</b>
from_eBPMConvert	398
getBP_EVENT	399
getTP_EVENT	400
hasBP_EVENT	401
hasTP_EVENT	402
marshal	403
omitBP_EVENT	404
omitTP_EVENT	405
setBP_EVENT	406
setTP_EVENT	407
to_eBPMConvert	408
toString	409
unmarshal	410

---

## Chapter 21

<b>e*Insight User Activity API Methods</b>	<b>411</b>
User Activity Security	411
Defining the Classpath	412
<b>Imessage Interface</b>	<b>413</b>
clearMessage	413
getActivityAttributesCount	414
getActivityAttributeValue	415
getActivityName	416
getBusinessModelId	417
getBusinessModelInstanceId	418
getBusinessModelName	419
getGlobalAttributeCount	420
getGlobalAttributeType	421
getGlobalAttributeValue	422
getMsgType	423
removeActivity	424
removeGlobalAttribute	425
setActivityAttributeValue	426
setActivityName	427
setBPISStack	428
setBusinessModelInstanceId	429
setBusinessModelId	430
setBusinessModelName	431
setGlobalAttributeValue	432
setMsgType	433
setStatus	434
toXML	435
<b>UserActivityMessage Class</b>	<b>436</b>
<b>IClient Interface</b>	<b>437</b>
authenticate	438
checkoutActivityInstance	439
getActivityGlobalAttributeNames	440
getActivityInstanceEndTime	441
getActivityInstanceStartTime	442
getActivityInstanceStatus	443



## Contents

getActivityNames	444
getAssignedBPIDByState	445
getAuthorizationActivityNames	446
getBPIShared	447
getBusinessModelInstanceIds	448
getBusinessModelInstanceName	449
getBusinessModelInstanceStatus	450
getBusinessModelName	451
getEnabledBusinessModelId	452
getEnabledBusinessModelIds	453
getGlobalAttributeDefaultValue	454
getGlobalAttributeDirection	455
getGlobalAttributeNames	456
getGlobalAttributeType	457
getGlobalAttributeValue	458
getLocalAttributeNames	459
getLocalAttributeType	460
getLocalAttributeValue	461
getMessageStatus	462
getUser	463
getUserActivityNames	464
getUUID	465
initialize	466
refreshCachedMemory	467
releaseActivityInstance	468
releaseResources	469
resetUser	470
sendMessage	471
setGlobalAttributeValue	472
setLocalAttributeValue	473
setUser	474
<b>EbpmMonitor Class</b>	<b>475</b>
checkUserPrivileges	475

---

## Appendix A

<b>XML Structure for the e*Insight Event</b>	<b>476</b>
XML Structure	476
<b>Glossary</b>	<b>478</b>
<b>Index</b>	<b>481</b>

# Introduction

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

---

## 1.1 Document Purpose and Scope

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

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

This guide gives 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 NT and UNIX operations and administration, and should be thoroughly familiar with Windows-style GUI operations.

Since most of the work in an e\*Insight implementation involves setting up the e\*Gate components that send data into and out of the e\*Insight system, 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.

For information on these and related topics, see **“Supporting Documents” on page 20**.

### Command Line

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

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

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

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

### Code and Samples

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

```
Configuration for BOB_Promotion
```

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

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

### Notes and Cautions

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

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

### User Input

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

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

### File Names and Paths

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

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

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

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

## Parameter, Function, and Command Names

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

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

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

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

---

## 1.4 Supporting Documents

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

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

---

## 1.5 SeeBeyond Web Site

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

<http://www.SeeBeyond.com>

# Introduction to the SeeBeyond eBI Suite

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

---

## 2.1 SeeBeyond eBusiness Integration Suite

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

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

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

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

This suite also provides:

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

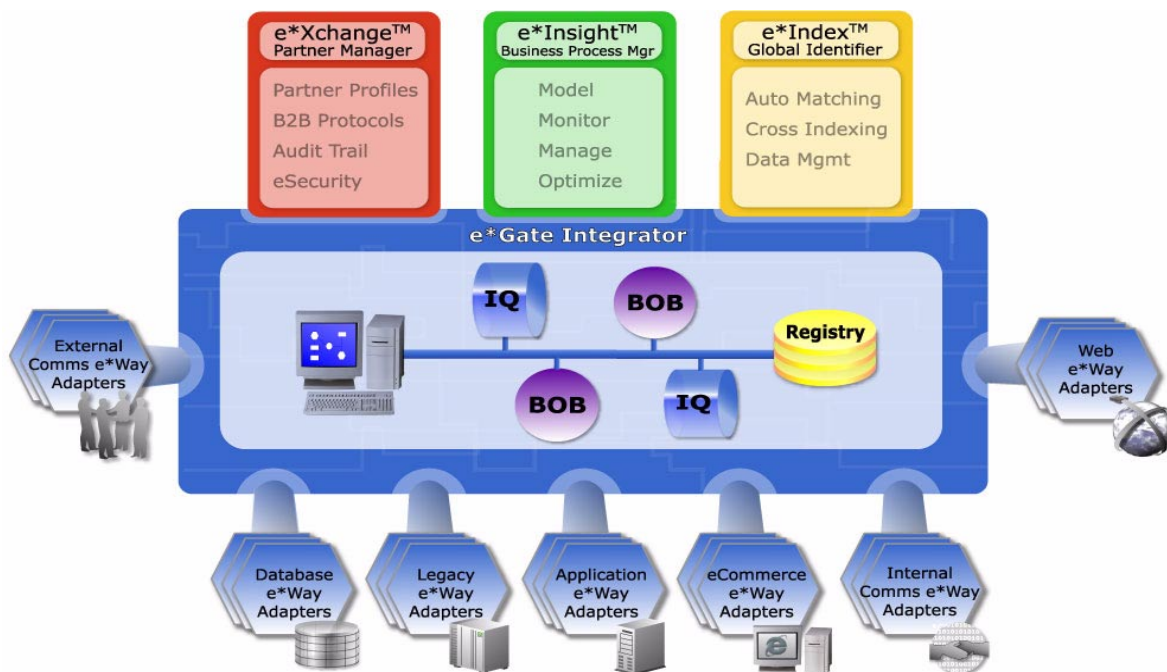
## 2.1.1 SeeBeyond eBusiness Integration Suite Components

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

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

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

**Figure 1** SeeBeyond eBusiness Integration Suite



### e\*Insight Business Process Manager

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

## e\*Xchange Partner Manager

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

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

## e\*Index Global Identifier

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

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

## e\*Gate Integrator Components

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

## e\*Way Intelligent Adapters

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

## Intelligent Queues

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

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

## Business Object Brokers

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

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

---

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

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

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

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

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

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

e\*Insight provides the capability to analyze the performance of a business process model on a historical basis, so that trends can be determined and possible bottlenecks identified. The analyst can create charts on the performance of the business process model against an array of system attributes (such as "duration" and "state"), and user-



defined attributes (for example, “order amount” or “PO source”). Charting the data in this way makes it easy to discern areas where the model needs re-design.

---

## 2.3 Building an eApplication

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

Implementing e\*Insight involves the following steps:

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

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

- 2 Obtain a working knowledge of e\*Insight.

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

- 3 Create an implementation plan.

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

---

## 2.4 Basic Information

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

# 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.

---

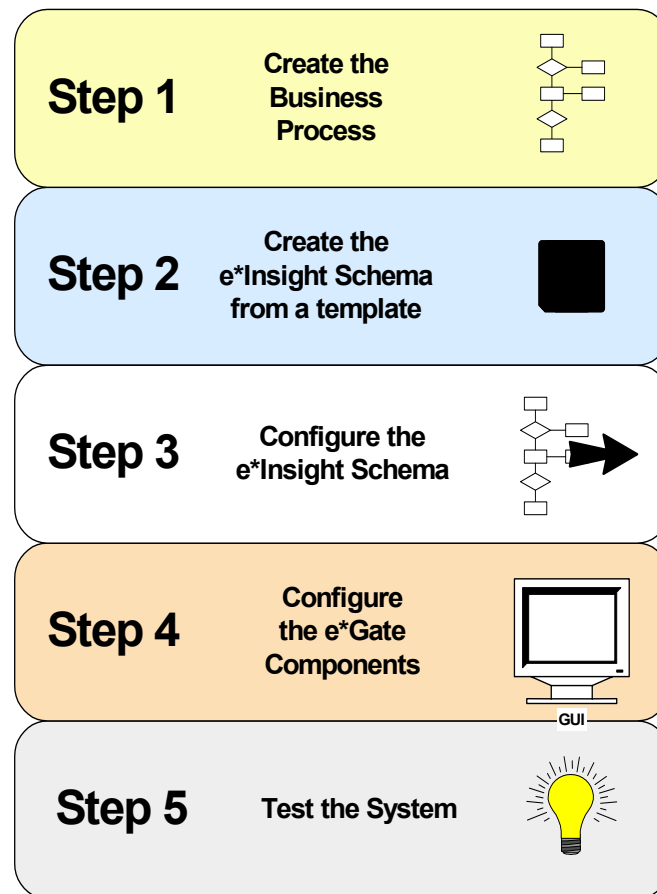
## 3.2 Implementation Road Map

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: Copy the e\*Insight Schema

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

For information on creating a copy of the e\*Insight schema, see [“Copy the e\\*Insight Schema” on page 98](#).

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

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

## Step 4: Configure the e\*Gate Components

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

- configure the e\*Ways that send data into and out of the e\*Insight system
- make all user-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 30](#)
  - ♦ [“Common Configuration Tasks” on page 98](#)
  - ♦ [“e\\*Insight Performance” on page 276](#)

- ♦ “Troubleshooting” on page 292
- ♦ “Java Helper Methods” on page 321
- Sample Implementations
  - ♦ “e\*Insight Implementation (eIJSchema)” on page 109
  - ♦ “e\*Insight Authorization Activity Implementation (eIJSchema)” on page 140
  - ♦ “e\*Insight User Activity Implementation” on page 195
  - ♦ “e\*Insight Sub-Process Implementation (eIJSchema)” on page 207
  - ♦ “e\*Insight Remote Sub-Process Implementation” on page 233
  - ♦ “Active and Passive Modes” on page 250

### 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 61
  - ♦ “e\*Insight ETD for Monk—eX\_Standard\_Event.ssc” on page 89
  - ♦ “Common Configuration Tasks” on page 98
  - ♦ “e\*Insight Performance” on page 276
  - ♦ “Troubleshooting” on page 292
  - ♦ “e\*Insight Helper Monk Functions” on page 302
- Sample Implementations
  - ♦ “e\*Insight Authorization Activity Implementation (eISchema)” on page 184
  - ♦ “e\*Insight User Activity Implementation” on page 195
  - ♦ “e\*Insight Sub-Process Implementation (eISchema)” on page 219
  - ♦ “e\*Insight Remote Sub-Process Implementation” on page 233
  - ♦ “Active and Passive Modes” on page 250

# 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.

### 4.1.1 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

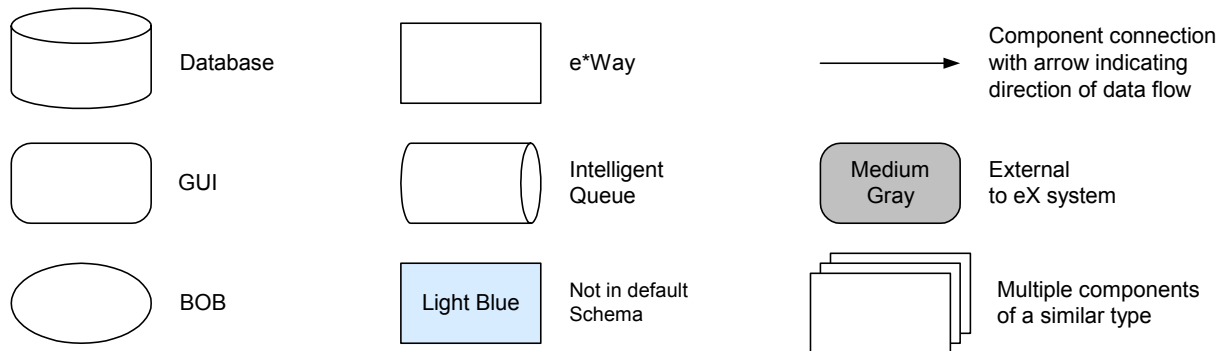
Component	Description	In Default eISchema?	More Information
e*Insight Engine	This is a specially configured Multi-Mode e*Way that runs business processes, using the e*Insight e*Way Connection.	Yes	<a href="#">4.1.4 on page 33</a>
eI_Resubmitter	A placeholder component used in the e*Insight Event failure handling process.	Yes	<a href="#">4.8.1 on page 51</a>
Start BP Component	Either an e*Way or BOB that sends the Event that starts a business process instance.	No	<a href="#">4.8.3 on page 53</a>
eX_Activity e*Way	Implements an e*Insight activity that connects to an external system.	No	<a href="#">4.8.5 on page 55</a>
eX_Activity BOB	Implements an e*Insight activity that does not connect to an external system.	No	<a href="#">§ on page 58</a>

### 4.1.2 e\*Insight Schema Component Relationships

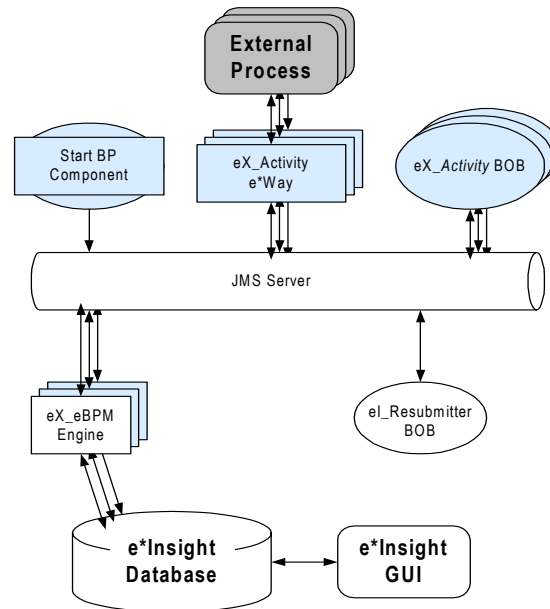
e\*Insight implementations do not *always* use all components available. Some components that you may need, are not provided as part of the e\*Gate schema and must be added to the base e\*Insight schema. [Figure 4](#) illustrates the relationships between e\*Insight schema components.

*Note:* [Figure 3](#) is the legend for the diagram in [Figure 4](#).

**Figure 3** e\*Insight Overview Legend



**Figure 4** e\*Insight Components



### 4.1.3 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 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 277](#).



- The **eI\_Resubmitter** e\*Way

The **eI\_Resubmitter** e\*Way is used in e\*Insight Event failure handling.

## Implement Business Process Activities

The component that implements business process activities is:

- **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 .

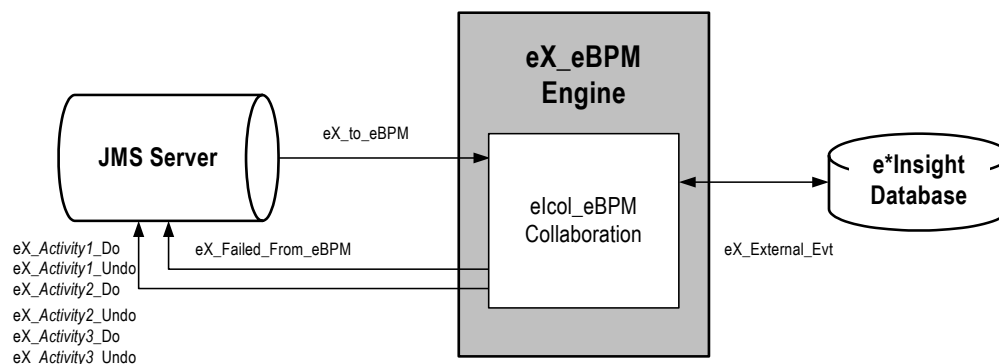
An **eX\_Activity** e\*Way can connect to an external system if required by the business activity. You must supply the programming to carry out the business logic of the activity and return an activity completed message (the “Done” Event) to the e\*Insight engine.

*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.*

### 4.1.4 e\*Insight Engine

An e\*Insight engine is comprised of a specially configured Collaboration (**eIcol\_eBPM**) and the e\*Insight e\*Way Connection (**eIcp\_eInsightEngine**). The e\*Insight engine runs within a Multi-Mode e\*Way (**eX\_eBPM**), which is referred to as the e\*Insight Engine container. An e\*Insight engine allows an e\*Way to communicate with the e\*Insight database as shown in [Figure 5](#).

**Figure 5** The e\*Insight Engine (Java)



## Configuring the e\*Insight Engine

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

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

---

## 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.

---

## 4.3 Database Requirements for XA Compliance

### 4.3.1 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

### 4.3.2 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*.

### 4.3.3 Microsoft SQL Server

To use XA with Microsoft SQL Server, you must:

- 1 Install Merant DataDirect 2.2 JDBC Driver for SQL Server.
- 2 Copy the sqljdbc.dll from the SQLServer\JTA directory to the SQLServer\bin directory.
- 3 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*.

---

## 4.4 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.

### 4.4.1 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.

## 4.4.2 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.

## 4.4.3 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.

## 4.4.4 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.

---

## 4.5 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.

### 4.5.1 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. If running e\*Gate 4.5.1 or earlier version, this option is not available and is ignored.

#### 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 Commit Frequency constant. If running e\*Gate 4.5.1 or earlier, this option is not available.

### 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. If running e\*Gate 4.5.2 or earlier, this option is not available.

### 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. If running e\*Gate 4.5.1 or earlier, this option is not available.

### 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. If running e\*gate 4.5.1 or earlier, this option is not available.

---

## 4.6 Connection Management

The e\*Insight Engine e\*Way Connection is now also Connection Management enabled, and can be controlled by the Connection Manager (available with e\*Gate 4.5.2 and up). Some of the benefits this brings include:

- Automatic Alerts sent to the e\*Gate Monitor should the Database connections go down.
- Continual verification of Database connection status on a periodic basis.

## 4.6.1 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. If running e\*Gate 4.5.1 or earlier, this option is not available.

### 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. If running e\*gate 4.5.1 or earlier, this option is not available.

### 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 e\*Gate Monitor indicating the Down status.

If the connection comes from a previous connection error, a compensating Alert is sent to the e\*Gate Monitor to indicate the connection is Up again. If no value is specified, 60000 ms is used. If running e\*gate 4.5.1 or earlier, this option is not available.

---

## 4.7 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.

### 4.7.1 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$;
```

---

## 4.8 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. <i>Note: If XA Compliancy is enabled, then the JDBC URL String parameter in the eBPM Settings section is ignored.</i>
	DriverType	Specifies the driver type. Select one of the following: <ul style="list-style-type: none"> <li>▪ oci8 – uses Type 2 OCI Transports</li> <li>▪ thin (default) – uses Oracle JDBC Pure Java Drivers.</li> </ul>
	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. <i>Changing this value may improve performance. You should follow the guidance of a Database Administrator when changing this value.</i>
	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. <i>Changing this value may improve performance. You should follow the guidance of a Database Administrator when changing this value.</i>
	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 <b>Attribute-Value Pair Separator</b> . The default separator is ^. If the Oracle native JDBC OCI Driver Type is used, the Attribute-Value pairs that should be defined include: <ul style="list-style-type: none"> <li>▪ NetworkProtocol — Default value is <b>tcp</b>. Can be set to all possible protocols that Net8 supports.</li> <li>▪ TNSEntryName — TNS entry name. This assumes that the Oracle Client is installed and TNS_ADMIN environmental variable is set properly.</li> </ul>
	Standard Class	Specifies the name of the Java class in the JDBC driver that implements the <b>ConnectionPoolDataSource</b> interface.
	XA Class	Specifies the name of the Java class in the JDBC driver that implements the <b>XADataSource</b> 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 Compliant is enabled. However, setting this value to YES may degrade performance. <i>Note: If XA Compliant is enabled, then the JDBC URL String parameter in the eBPM Settings section is ignored.</i>
	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 <b>Attribute-Value Pair Separator</b> . 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 <b>ConnectionPoolDataSource</b> interface.
	XA Class	Specifies the name of the Java class in the JDBC driver that implements the <b>XADataSource</b> 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. <i>Note: If XA Compliancy is enabled, then the JDBC URL String parameter in the eBPM Settings section is ignored.</i>
	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 <b>Attribute-Value Pair Separator</b> . 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 <b>ConnectionPoolDataSource</b> 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 <b>XADataSource</b> interface.
DB2 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. <i>Note: If XA Compliancy is enabled, then the JDBC URL String parameter in the eBPM Settings section is ignored.</i>
	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.
	LocationName	Specifies the name of the DB2 location that you want to access. This is used on OS/390.
	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.
	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 <b>Attribute-Value Pair Separator</b> . The default separator is ^. For Merant DB2 JDBC drivers, it is recommended to define the "SelectMethod^cursor" and "WithHoldCursor^false" Attribute-Value pairs if XA-Compliance is selected.
	Standard Class	Specifies the name of the Java class in the JDBC driver that implements the <b>ConnectionPoolDataSource</b> interface.
	XA Class	Specifies the name of the Java class in the JDBC driver that implements the <b>XADataSource</b> interface.

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

Screen	Parameter	Setting
eBPM Settings	Database Type	<p>Specifies the type of e*Insight database. Select one of the following:</p> <ul style="list-style-type: none"> <li>▪ <b>DB2</b> when using DB2 Universal Database (7.1 and 7.2 on Windows or UNIX; 6.1 on OS/390).</li> <li>▪ <b>Oracle</b> when using Oracle 8i (8.1.6 and 8.1.7) or 9i.</li> <li>▪ <b>SQL Server</b> when using Microsoft SQL Server 7.0 or 2000.</li> <li>▪ <b>Sybase</b> when using Sybase 11.9, 12 or 12.5</li> </ul>
	JDBC URL String	<p>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 &lt;NONE&gt;.</p> <p>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. <i>(Refer to the relevant driver documentation for more details on configuring your system).</i></p>

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

Screen	Parameter	Setting
eBPM Settings	JDBC URL String	<p>An Oracle database connection will use the string: <b>jdbc:oracle:thin:@machine_name:port:db</b> where</p> <ul style="list-style-type: none"> <li>▪ <b>thin</b> is the type of Oracle client interface. See your e*Insight database administrator for more information.</li> <li>▪ <b>machine_name</b> 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.</li> <li>▪ <b>port</b> is correct port for communicating with the e*Insight database (1521 is the default).</li> <li>▪ <b>db</b> is the service name used to communicate with e*Insight Oracle database from the local machine.</li> </ul> <p>If you are using XML data and the Model Specific database option, then you should use the OCI8 driver. See the e*Insight Business Process Manager Installation Guide for information on installing this driver. You will use this string to connect: <b>jdbc:oracle:oci8:@db</b> where</p> <ul style="list-style-type: none"> <li>▪ <b>oci8</b> is the type of oracle client interface.</li> <li>▪ <b>db</b> is the service name used to communicate with e*Insight Oracle database from the local machine.</li> </ul> <p>A SQL Server database connection uses the string: <b>jdbc:SeeBeyond:sqlserver://&lt;server&gt;:&lt;port#&gt;;DatabaseName=&lt;dbname&gt;;embedded=true;SelectMethod=cursor</b></p> <ul style="list-style-type: none"> <li>▪ <b>server</b> 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.</li> <li>▪ <b>dbname</b> is the name of the e*Insight SQL Server database.</li> <li>▪ <b>port</b> is correct port for communicating with the e*Insight database.</li> </ul> <p>A Sybase database connection uses the string: <b>jdbc:sybase:Tds:&lt;server&gt;:&lt;port&gt;</b></p> <ul style="list-style-type: none"> <li>▪ <b>server</b> 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.</li> <li>▪ <b>port</b> is correct port for communicating with the e*Insight database.</li> </ul>

**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, <b>oracle.jdbc.driver.OracleDriver</b> can be used with an Oracle database, and <b>sun.jdbc.odbc.JdbcOdbcDriver</b> 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 <b>ei_admin</b> ) 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 <b>ei_admin</b> .
	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. <i>Note: This feature severely impacts performance.</i>
	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. <i>Note: A value of zero (0) should NOT be used.</i>

**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 <a href="#">“e*Insight Engine Affinity (eIJSchema)” on page 280</a> .
	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 <a href="#">“e*Insight Engine Affinity (eIJSchema)” on page 280</a> .
	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: <ul style="list-style-type: none"> <li>▪ <b>Automatic</b> — the connection is automatically established when the Collaboration is started and the connection is kept alive as necessary.</li> <li>▪ <b>OnDemand</b> — 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.</li> <li>▪ <b>Autonomous</b> — the connection is maintained without the aid of the Connection Manager.</li> </ul>
	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. <i>This parameter is not supported with e*Gate version 4.5.1 or below.</i>
	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 e*Gate Monitor indicating the connection has been restored. If now value is specified, 6000 ms is used. <i>This parameter is not supported with e*Gate version 4.5.1 or below.</i>

## elcr\_eBPM Collaboration

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

### Subscribed Event Type

- **eX\_External\_Evt**—This Event carries the data retrieved from the e\*Insight database to the e\*Insight engine.

- **eX\_to\_eBPM**—This Event Type carries all Events intended for e\*Insight. These include Start BP Events, “Done” Events, and other Events that must be processed by the e\*Insight engine. The corresponding ETD is **eI\_StandardEvent.xsc**. This is the only Event Type to which the e\*Insight engine subscribes; all data sent to the e\*Insight engine must use this Event Type.

*Note: The eI\_StandardEvent.xsc does not contain a section for the e\*Xchange Partner Manager information. If your implementation also uses e\*Xchange Partner Manager then you need to use eIX\_StandardEvent.xsc as this contains the Trading Partner Event (TP\_EVENT) information.*

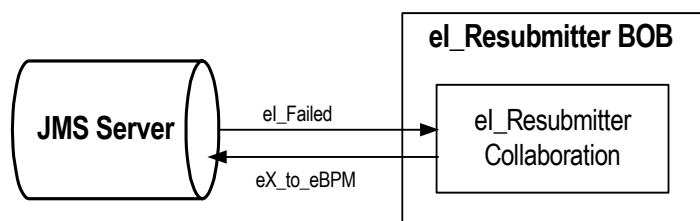
### Published Event Types

- **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 57](#) 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 57](#) 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.

## 4.8.1 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.

## 4.8.2 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

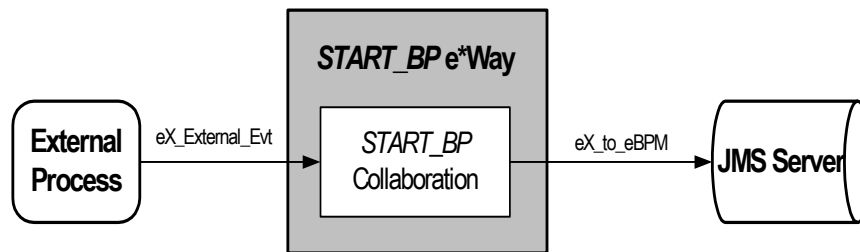
Events that fail to process due to data errors are not retried, but a notation is made in the e\*Insight log file and the Event itself is published to a special location. This method allows the e\*Insight engine to move on to other processing and not spend time attempting to resend failed Events.

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.

### 4.8.3 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 100 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.

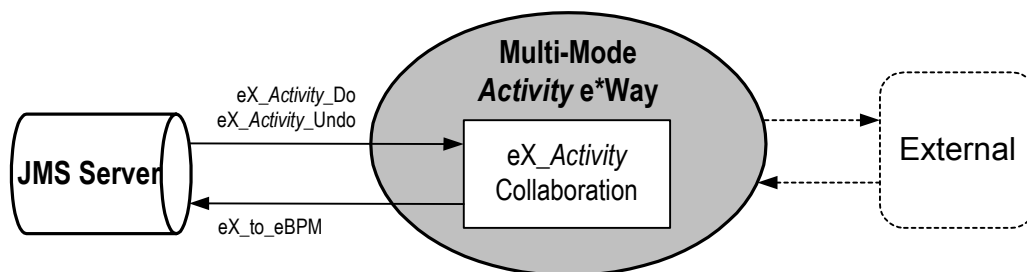
#### Published Event Type: eX\_to\_eBPM

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

### 4.8.4 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 57 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 57 for more information.

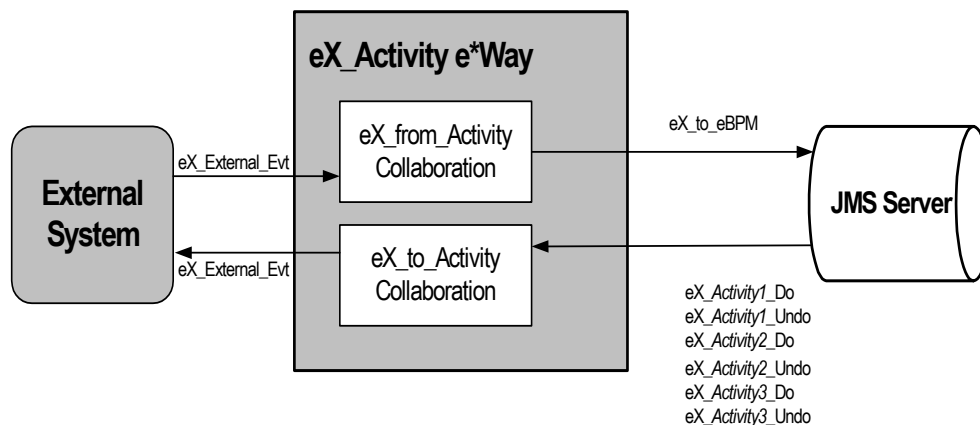
**Published Event Type: eX\_to\_eBPM**

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 58 for more information.

### 4.8.5 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 102 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 81](#).

#### 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 56](#) 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 102 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.

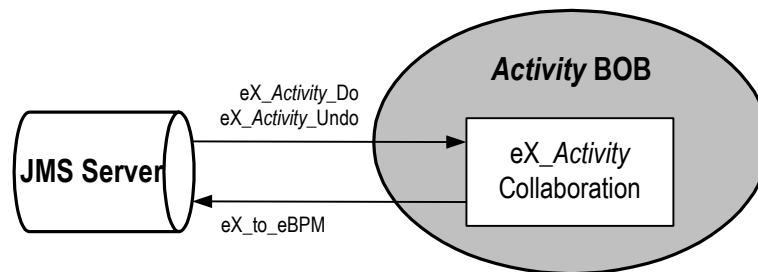
#### Published Event Type: eX\_to\_eBPM

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

### 4.8.6 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 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 57](#) 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 57](#) 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 58](#) for more information.

---

## 4.9 Using Monk in eIJSchema

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

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

---

## 4.10 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.

# 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.

## 5.0.1 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.

## 5.0.2 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.

---

## 5.1 e\*Insight Schema Components Overview

Table 4 lists all of the component types used by e\*Insight. It lists the components that are provided as part of the e\*Insight schema (eISchema) installation, and also the

components that the user adds in the implementation process. The meaning of the column headings is as follows.

- **Component**—The e\*Gate logical name for the component. *Italics* indicates that the name varies by association or is user-defined.
- **Description**—A brief description of what the component does in e\*Insight.
- **In Default eISchema**—Whether or not this component is provided as part of the schema installation of e\*Insight.
- **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 30. You can delete these components if you do not want to use them.

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

**Table 4** e\*Insight Schema Component Types

Component	Description	In Default elSchema?	More Information
eX_eBPM Engine	This is a specially configured Java e*Way Connector that runs business processes.	Yes	<a href="#">5.2.1 on page 65</a>
el_Resubmitter BOB	A placeholder component used in the e*Insight Event failure handling process.	Yes	<a href="#">5.2.2 on page 70</a>
Start BP Component	Either an e*Way or BOB that sends the Event that starts a business process instance.	No	<a href="#">5.2.4 on page 72</a>
eX_Activity e*Way	Implements an e*Insight activity that connects to an external system.	No	<a href="#">5.2.5 on page 74</a>
eX_Activity BOB	Implements an e*Insight activity that does not connect to an external system.	No	<a href="#">5.2.6 on page 77</a>

### 5.1.1 e\*Insight Schema Component Relationships Diagram

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

**Figure 11** e\*Insight Overview Legend

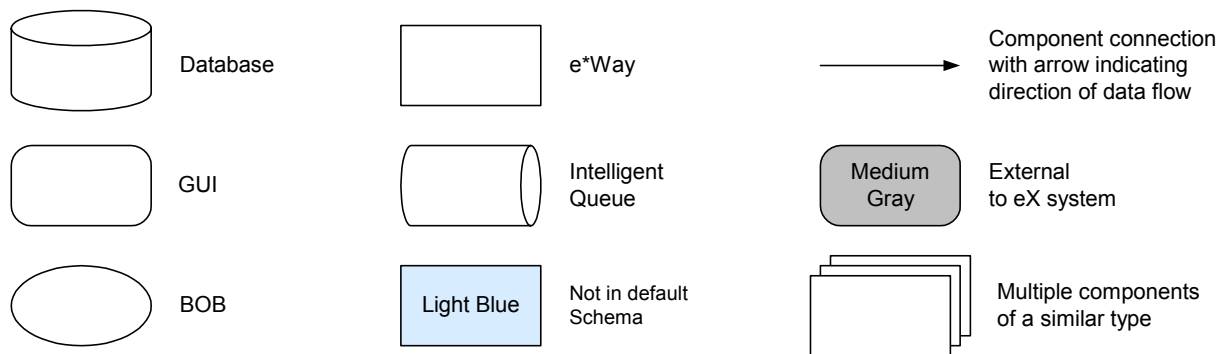
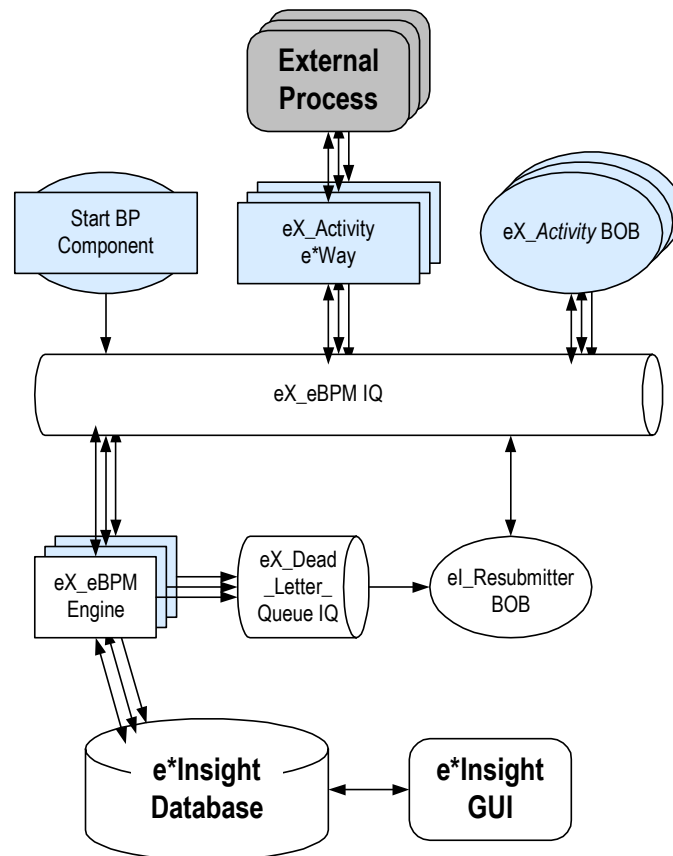


Figure 12 e\*Insight Components



## 5.2 e\*Insight Business Process Manager Components

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

### Components That Run Business Processes

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

- One or more e\*Insight engines
- The **eI\_Resubmitter** BOB

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

The **eI\_Resubmitter** BOB is used in e\*Insight Event failure handling.



## Components that Start Business Processes

The component that starts a business process is:

- The *START\_BP* component (the exact name can be chosen by the user).

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

## Components that Implement Business Process Activities

The components that implement business process activities are:

- *eX\_Activity* e\*Ways
- *eX\_Activity* BOBs

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

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

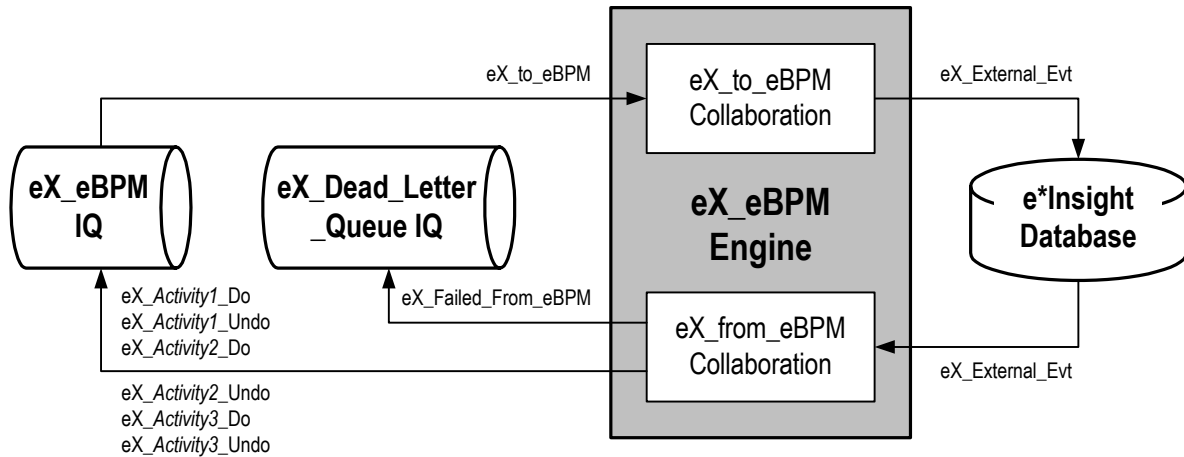
### 5.2.1 e\*Insight Engine

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 287](#).

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

Figure 13 eX\_eBPM Engine Detail



## Configuring the e\*Insight Engine

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

Table 5 e\*Insight Engine Configuration Settings

Screen	Parameter	Setting
General Settings	(All)	(Default)
Communication Setup	(All)	(Default)
Java VM Configuration	JNI DLL	This is the path to the java virtual machine used by the e*Insight engine. If necessary, replace the default value, <b>C:\eGate\Client\JRE1.3\bin\hotspotjvm.dll</b> , with the fully qualified path to the JNI DLL file on the Participating Host that is running the e*Insight engine. The default location given above is where the public installation of java 1.2.2 places this file on a Windows NT machine. <b>Note:</b> In a UNIX environment the name and location of this file is different; for example, <b>/usr/Solaris_JDK_1.2.2_05a/jre/sparc/libjvm.sol</b>
	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 <b>el_Resubmitter</b> BOB below for more information.
	(Others)	(Default)

**Table 5** e\*Insight Engine Configuration Settings (Continued)

Screen	Parameter	Setting
eBPM Settings	JDBC URL String	<p>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).</p> <p>An Oracle database connection might use the string: <b>jdbc:oracle:thin:@machine_name:port:db</b> where</p> <ul style="list-style-type: none"> <li>▪ <b>thin</b> is the type of oracle client interface. See your e*Insight database administrator for more information.</li> <li>▪ <b>machine_name</b> 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.</li> <li>▪ <b>port</b> is correct port for communicating with the e*Insight database (1521 is the default).</li> <li>▪ <b>db</b> is the service name used to communicate with e*Insight Oracle database from the local machine.</li> </ul> <p>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: <b>jdbc:oracle:oci8:@db</b> where</p> <ul style="list-style-type: none"> <li>▪ <b>oci8</b> is the type of oracle client interface.</li> <li>▪ <b>db</b> is the service name used to communicate with e*Insight Oracle database from the local machine.</li> </ul> <p>A SQL Server database connection might use the string: <b>jdbc:SeeBeyond:sqlserver://&lt;server&gt;:&lt;port#&gt;;DatabaseName=&lt;dbname&gt;;embedded=true</b></p> <ul style="list-style-type: none"> <li>▪ <b>server</b> 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.</li> <li>▪ <b>dbname</b> is the name of the e*Insight SQL Server database.</li> <li>▪ <b>port</b> is correct port for communicating with the e*Insight database.</li> </ul>

**Table 5** e\*Insight Engine Configuration Settings (Continued)

Screen	Parameter	Setting
eBPM Settings	JDBC URL String	A Sybase database connection might use the string: <b>jdbc:sybase:Tds:&lt;server&gt;:&lt;port&gt;</b> <ul style="list-style-type: none"> <li>▪ <b>server</b> 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.</li> <li>▪ <b>port</b> is correct port for communicating with the e*Insight database.</li> </ul>
	Database Type	Specifies the type of e*Insight database. Select one of the following: <ul style="list-style-type: none"> <li>▪ <b>Oracle</b> when using an Oracle 8i (8.1.5 and above)</li> <li>▪ <b>SQL Server</b> when using SQL Server 7.0 or SQL Server 2000</li> <li>▪ <b>Sybase</b> when using Sybase 11.9</li> </ul>
	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 <b>ex_admin</b> ) 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 <b>ex_admin</b> .
	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, <b>oracle.jdbc.driver.OracleDriver</b> can be used with an ORACLE database, and <b>sun.jdbc.odbc.JdbcOdbcDriver</b> can be used with a SQL Server database or a Sybase database.
	Maximum Business Process Cache Size	This is the number of business processes that the e*Insight engine can hold in memory at one time. If the cache is full and another business process needs to be loaded, the least recently used (LRU) business process in the cache is replaced with the new business process. The default is 1024 business processes. The size of the business processes does not matter.  Entering the special value of zero (0) implies that caching of Business Process definitions is NOT desired, and thus the e*Insight engine ALWAYS reloads the Business Process definition from the database for EVERY Activity event of a Business Process Instance. Note, this feature severely impacts performance.

**Table 5** e\*Insight Engine Configuration Settings (Continued)

Screen	Parameter	Setting
	Maximum Instance Cache Size	<p>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.</p> <p>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.</p> <p>Note: A value of zero (0) should NOT be used.</p>
	Instance Caching	<p>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.</p> <p>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 <a href="#">“e*Insight Engine Affinity (eISchema)” on page 288</a>.</p>
	Business Processes to Preload	<p>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.</p>
	Use Default Locale Encoding	<p>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.</p> <p>Setting this value to NO specifies that data is being passed to and from this e*Insight Engine in an UTF-8 encoding format.</p>

## 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 76](#) 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 77](#) for more information.
- **eX\_Failed\_From\_eBPM**—This Event contains the failed Event along with error information.

## eX\_to\_eBPM Collaboration

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

### Subscribed Event Type: eX\_to\_eBPM

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

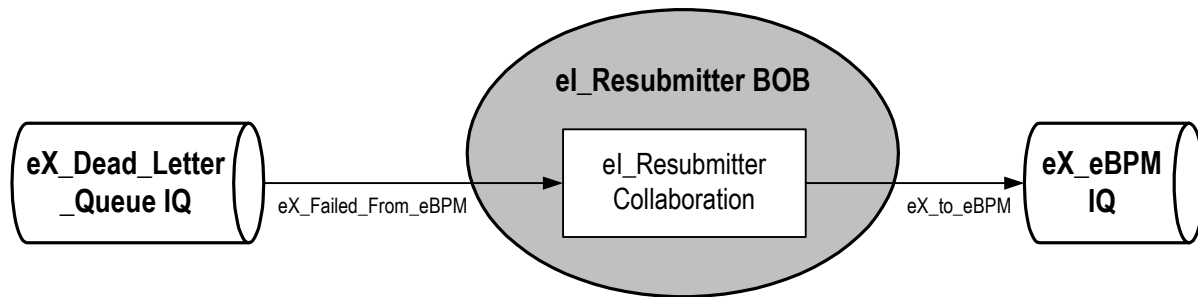
### Published Event Type: eX\_External\_Evt

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

## 5.2.2 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 14 el\_Resubmitter BOB Detail



## Configuring the el\_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.

## 5.2.3 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.

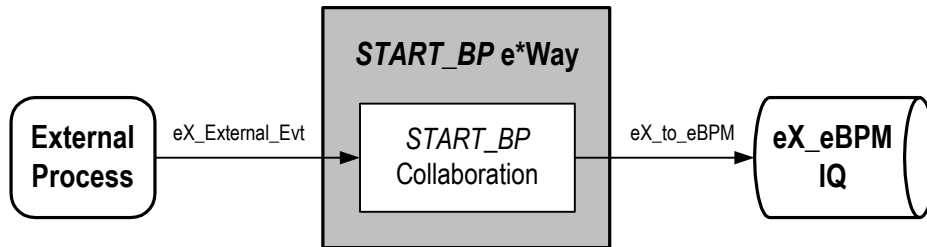
The e\*Insight engine publishes the failed Event to the **eX\_Dead\_Letter\_Queue** IQ under the **eX\_Failed\_From\_eBPM** Event Type. The **eI\_Resubmitter** BOB subscribes to this Event Type and you can use it to repair the Event and republish it to the **eX\_eBPM** IQ. The **eX\_Failed\_Event.ssc** ETD associated with the **eX\_Failed\_From\_eBPM** Event Type has two major node structures. One contains error information, and the other contains the failed Event. The failed Event is placed in the CDATA leaf node that can contain XML data.

### 5.2.4 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.



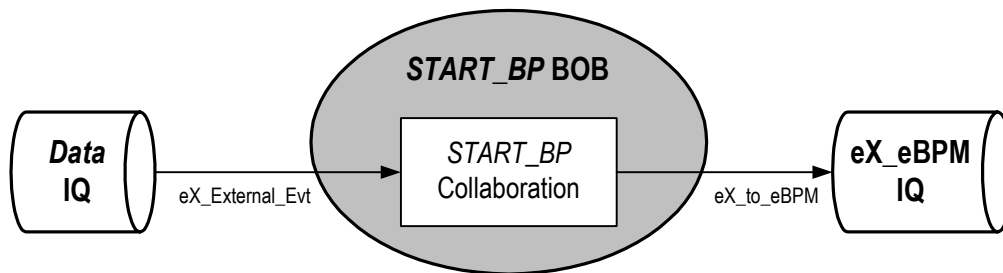
**Figure 15** START\_BP e\*Way Detail



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

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

**Figure 16** START\_BP BOB Detail



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

## Configuring the START\_BP Component

Configuring the START\_BP Component depends on the type of component it is. For example, a BOB has no configuration file, and an e\*Way's configuration file is different depending on the type of e\*Way that is used. See the appropriate e\*Way User Guide for information on configuring a specific e\*Way. In addition, 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 100 for more information on how to start a BPI.

### Published Event Type: eX\_to\_eBPM

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

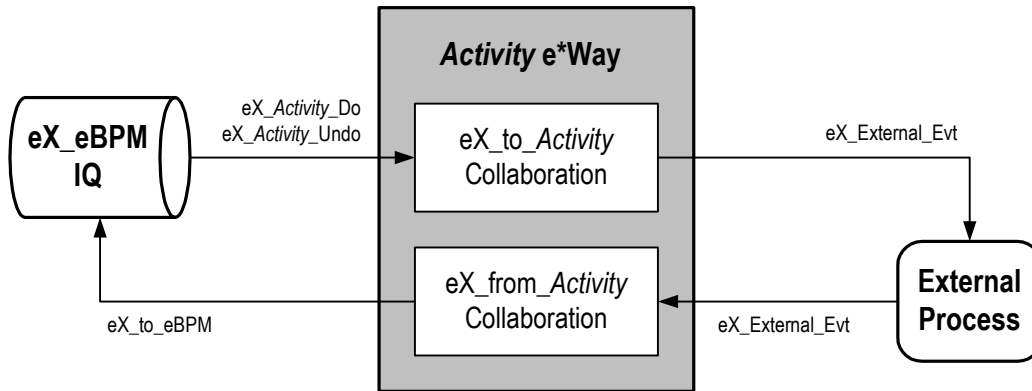
### Subscribed Event Type: eX\_External\_Evt

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

## 5.2.5 Activity e\*Way

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

**Figure 17** Activity e\*Way Detail



When you use the e\*Insight GUI to configure the e\*Gate schema supporting the e\*Insight implementation, each activity in the business process becomes either a pair of Collaborations in an e\*Way or a single Collaboration in a BOB. The choice to use an e\*Way or a BOB to hold the activity Collaboration or Collaborations depends on 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 102 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 81](#).

### 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 76](#) 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 102 for more information.

### Subscribed Event Type: eX\_External\_Evt

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

### Published Event Type: eX\_to\_eBPM

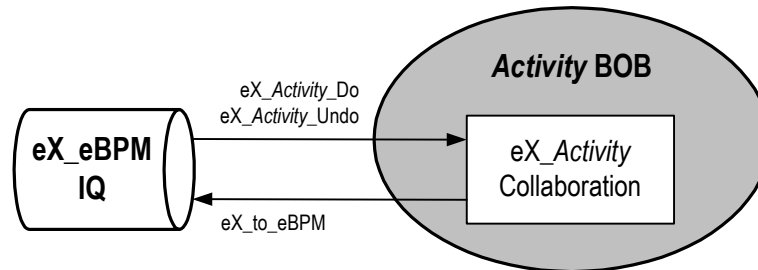
This is the “Done” Event that carries the data from a completed activity back to the e\*Insight engine.

## 5.2.6 Activity BOB

The Activity BOB implements an e\*Insight activity that does not require a connection to a system outside of e\*Gate. A BOB only needs one Collaboration to process the data and return it to the **eX\_eBPM** IQ. The CRS associated with the BOB’s Collaboration Rule carries out the business logic of the activity to which it corresponds. This Rule

could be a Monk script, a Java program, or any other script or application supported by the Collaboration Service under which the CRS runs.

**Figure 18** Activity BOB Detail



## eX\_Activity Collaboration

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

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

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

### Subscribed Event Types:

- **eX\_Activity\_Do**—This Event causes the subscribing Collaboration to execute the “Do” logic of the activity with the same name in the business process. It is the same Event Type as that subscribed to by the **eX\_to\_Activity Collaboration**. See [“Subscribed Event Type: eX\\_Activity\\_Do” on page 76](#) 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 77](#) 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 77](#) for more information.

# 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 rules 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.

---

## 6.1 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 is based on the `eI_StandardEvent.xsc`. You may still select the `eI_StandardEvent.xsc` for Java or the `eX_Standard_Event.ssc` for Monk.

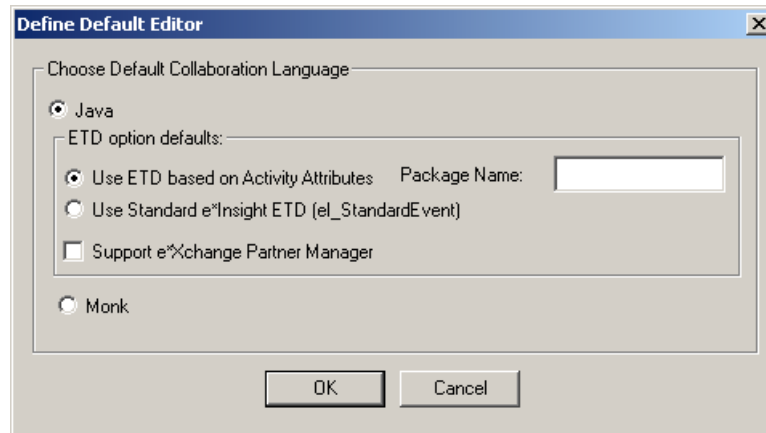
The Activity Specific ETD and the `eI_StandardEvent.xsc` do not contain a section for e\*Exchange Partner Manager information. When you install the e\*Gate Schema for e\*Insight, `eI_StandardEvent.xsc` is also created. This ETD has the Trading Partner Event (TP\_EVENT) location that is used for e\*Exchange Partner Manager. You should use this ETD if your implementation requires both e\*Insight and e\*Exchange.

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

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

If you choose to use `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 19](#).

Figure 19 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\*Exchange Partner Manager** option if you plan to use e\*Exchange Partner Manager in this implementation.

## 6.1.1 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 el\_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 “Do” 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.

## 6.2 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. It is recommended that you copy EventSpecifics 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 on using eI\_StandardEvent, see [“eI\\_StandardEvent” on page 84](#).

### 6.2.1 GlobalAttributes

Every global attribute that is assigned to the activity appears as a node under the GlobalAttributes node. Figure 20 shows an example where three global attributes are assigned to an activity, GlobalBoolean, GlobalString, and GlobalNumber.

**Figure 20** Activity Specific ETD — GlobalAttributes



### 6.2.2 LocalAttributes

Every local attribute that is assigned to the activity appears as a node under the LocalAttributes node. Figure 21 shows an example where three global attributes are assigned to an activity, LocalB, LocalS, and LocalN.

**Figure 21** Activity Specific ETD — LocalAttributes

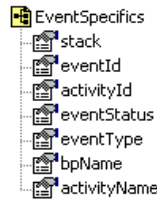


### 6.2.3 EventSpecifics

This section contains general information about the activity and business process in the seven nodes as shown in [Figure 22](#). 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 22** Activity Specific ETD – EventSpecifics



### stack

This node is used to contain the stack.

### 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.

### 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 number to speed up processing.

### 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 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”	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.

### bpName

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

### activityName

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.

**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.

---

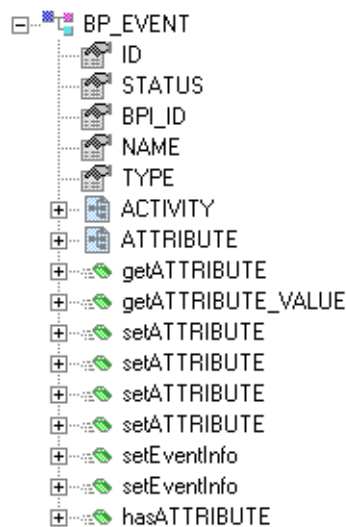
## 6.3 eI\_StandardEvent

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.

### 6.3.1 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 23.

**Figure 23** BP\_EVENT



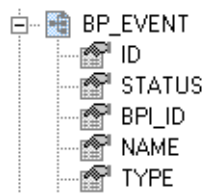
The information is held in three different locations within the Event Type Definition. Each of these nodes contains a different type of information pertinent to e\*Insight.

- **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 35.

**Figure 24** 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.

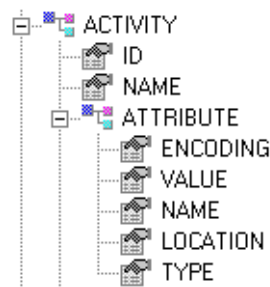
Value	Purpose
"START_BP"	Indicates to the e*Insight engine that this Event starts a BPI.
"DO_ACTIVITY"	Indicates that this is a "Do" Event for the current activity.
"UNDO_ACTIVITY"	Indicates that this is an "Undo" Event for the current activity.

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 25 shows the location of these nodes in the e\*Insight ETD.

Figure 25 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 26.

Figure 26 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.

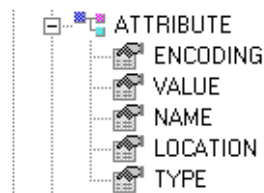
## ACTIVITY.ATTRIBUTE Element

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 27.

Figure 27 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 <b>VALUE</b> field.
"DB"	Indicates that the value for the attribute can be found in the e*Insight database at the location specified in the <b>VALUE</b> field.
"URL"	Indicates that the value for the attribute can be found at the URL location specified in the <b>VALUE</b> field.
"EMBEDDED"	Indicates that the value for the attribute is contained in the current e*Insight Event in the <b>VALUE</b> 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 <b>VALUE</b> 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 <b>VALUE</b> field is XML data that has been encoded using the scheme described in the <b>ENCODING</b> field. Currently only base 64 encoding is supported.
"STRING"	Indicates that the data in the <b>VALUE</b> field is clear ASCII data, with no characters that could be interpreted as XML tags.
"TRANSIENT"	Indicates that the data in the <b>VALUE</b> 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 <b>VALUE</b> 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 <b>VALUE</b> field is interpreted as boolean.



## 6.4 e\*Insight ETD for Monk—eX\_Standard\_Event.ssc

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

### 6.4.1 ETD Structure

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

Table 6 lists these *facilitator* nodes.

**Table 6** Facilitator Nodes in the ETD

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

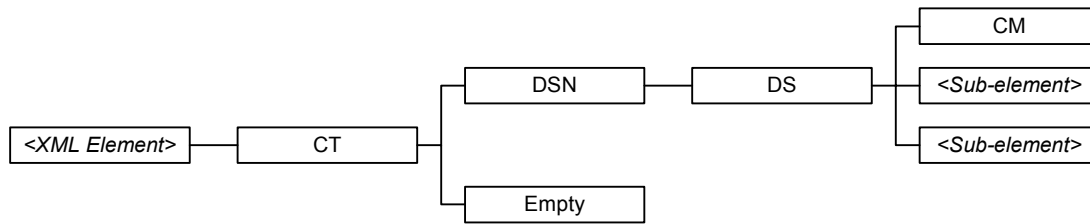
The facilitator nodes always occur in a set order and define the structure of the XML message. In the e\*Insight ETD, the facilitator nodes define three types of branches:

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

### 6.4.2 XML Element with Sub-elements

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

**Figure 28** XML Element with Sub-elements



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

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

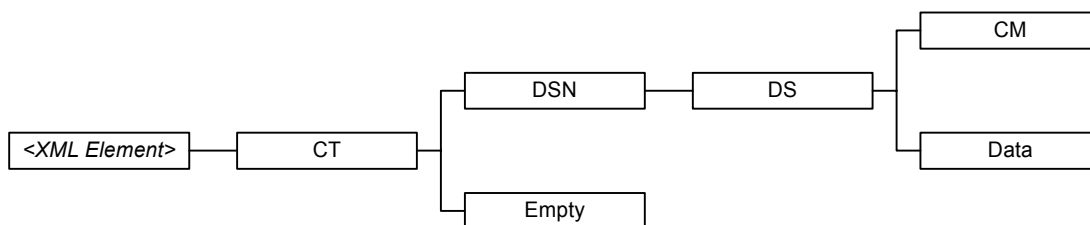
- **CM**, which holds XML comments for the element
- `<sub-element>`, the name of a sub-element of the parent element

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

### 6.4.3 XML Element without sub-elements

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

**Figure 29** XML Element without sub-elements

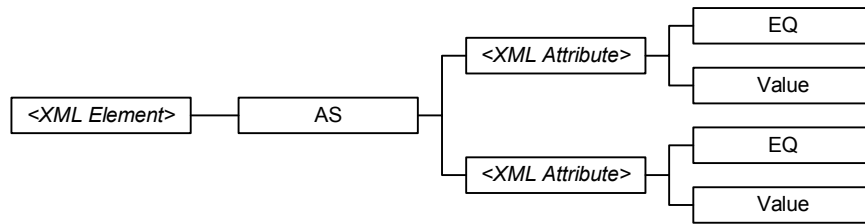


Notice that the only difference between this diagram and the previous diagram is a **Data** child node in place of the `<sub-element>` child nodes above. The **Data** node contains the actual data for the XML element that is defined. When creating Collaboration Rules scripts, you must map the XML element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

### 6.4.4 XML Attribute

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

Figure 30 XML Attribute

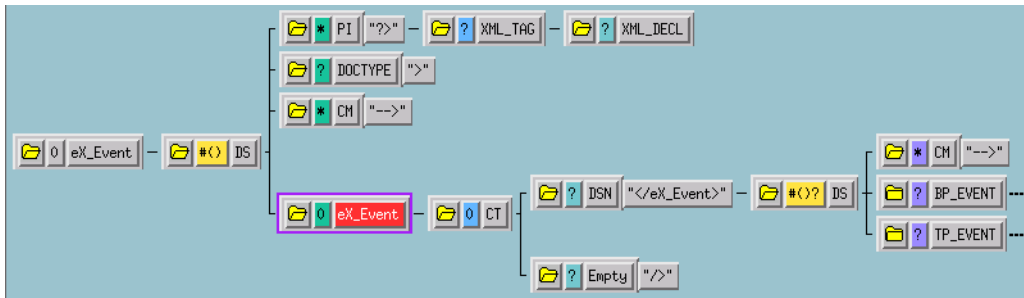


In this case, the XML element contains one child node, **AS**, which identifies the branch as XML attributes of the parent element. The **AS** node contains the **<XML Attribute>** nodes as child nodes. Each **<XML Attribute>** node has two child nodes: **EQ** to represent the equal sign (=) in the attribute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

### 6.4.5 Element Overview

The following diagram illustrates the entire e\*Insight ETD tree. Note that this is only a diagrammatic representation of the tree, since the actual tree conforms to the node structure described in “e\*Insight ETD for Monk—eX\_Standard\_Event.ssc” on page 89.

Figure 31 The e\*Insight ETD



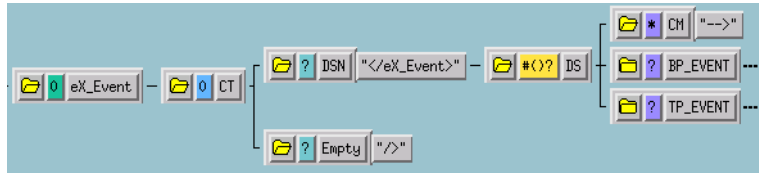
All data pertinent to e\*Insight is contained in the XML element **eX\_Event**. **eX\_Event** contains two distinct “trees”: **BP\_EVENT** and **TP\_EVENT**. **BP\_EVENT** contains all of the information pertaining to e\*Insight. **TP\_EVENT** contains all of the information pertaining to e\*Xchange. 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 32 shows the ETD structure for this element.

**Figure 32 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**.

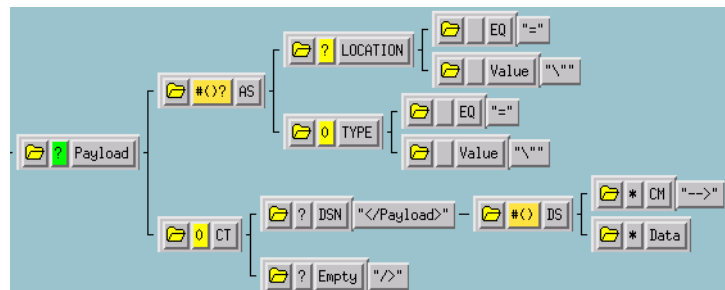
The **DS** node always contains a **CM** child node to hold XML comments. In this example, the **eX\_Event** element does not hold data directly, but contains two sub-elements—**BP\_EVENT** and **TP\_EVENT**—which have similar facilitator node branches associated with them.

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 33 shows the ETD structure for these attributes.

**Figure 33 XML Attribute Type**



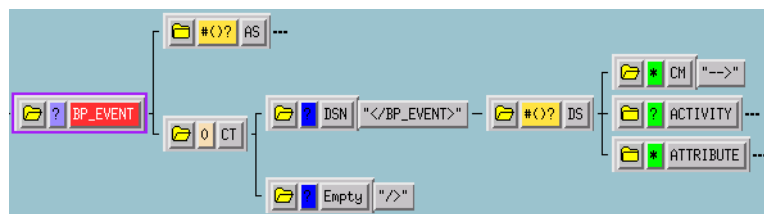
## 6.5 Using eX\_Standard\_Event.ssc

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

### 6.5.1 BP\_EVENT

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

**Figure 34** BP\_EVENT



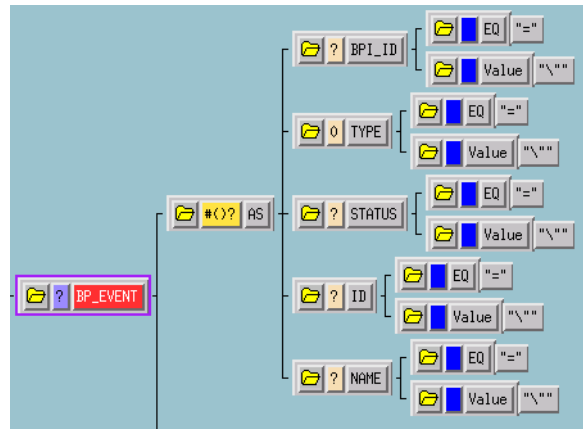
Three of the nodes shown in Figure 34 are collapsed (followed by three dashes) indicating there are additional nodes underneath these nodes. Each of these nodes contains a different type of information pertinent to e\*Insight.

- `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 35.

Figure 35 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.

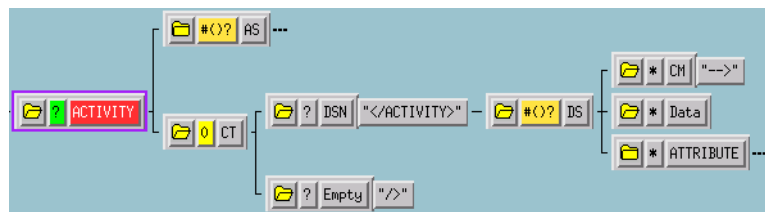
**NAME**

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 36 shows the location of these nodes in the e\*Insight ETD.

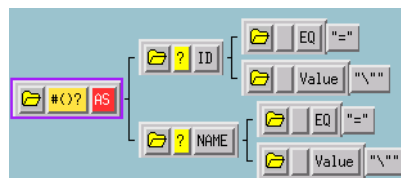
**Figure 36** BP\_EVENT.CT.DSN.DS.ACTIVITY



**ACTIVITY.AS Nodes**

This location in the e\*Insight ETD contains ID information about the current activity in two nodes as shown in Figure 37.

**Figure 37** 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**

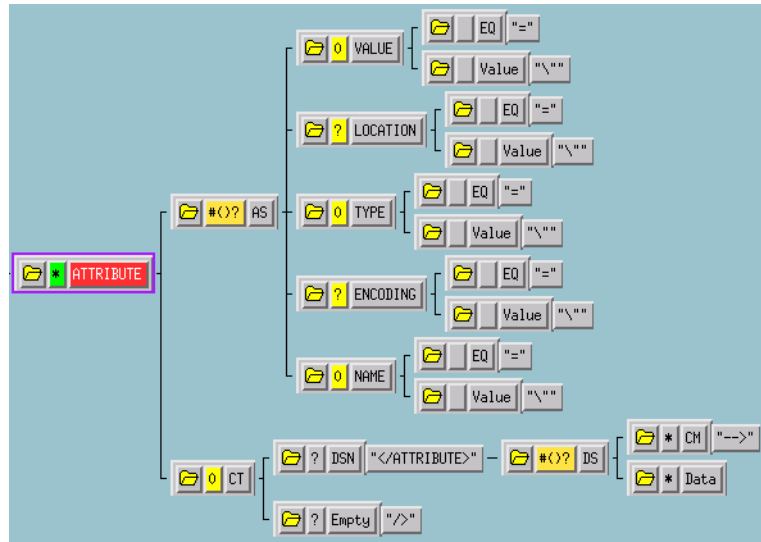
This repeating node structure contains the local attribute information defined for the current activity. The structure itself is exactly the same as the global attribute node

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 38:

**Figure 38** 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 <b>VALUE.Value</b> node.
"DB"	Indicates that the value for the attribute can be found in the e*Insight database at the location specified in the <b>VALUE.Value</b> node.
"URL"	Indicates that the value for the attribute can be found at the URL location specified in the <b>VALUE.Value</b> node.
"EMBEDDED"	Indicates that the value for the attribute is contained in the current e*Insight Event in the <b>VALUE.Value</b> 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 <b>VALUE.Value</b> 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 <b>VALUE.Value</b> node is XML data that has been encoded using the scheme described in the <b>ENCODING</b> node. Currently only base 64 encoding is supported.
"STRING"	Indicates that the data in the <b>VALUE.Value</b> node is string data.
"TRANSIENT"	Indicates that the data in the <b>VALUE.Value</b> 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 <b>VALUE.Value</b> 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 <b>VALUE.Value</b> 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.

# 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.

---

## 7.1 Copy the e\*Insight Schema

When beginning an integration project, make a copy of the e\*Insight schema, that is installed from the CD. 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 Enterprise Manager GUI. Secondly, you install the support files that are used by the Schema into the Default repository, for example the configuration files, Event Type Definitions, and Collaboration Rules Scripts. You also install some files required for e\*Gate to communicate with e\*Insight, for example, workflow.jar.

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

### Using the e\*Insight GUI

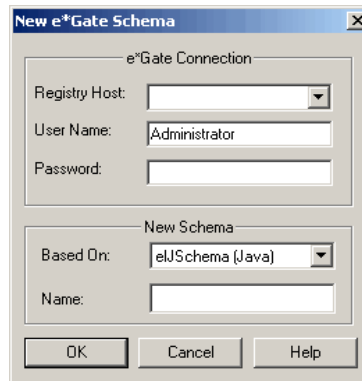
When you install the e\*Insight GUI a text file is created on your local machine containing e\*Insight schema information. This text file is used to create your schema on the Registry host. This process does not effect any of the files associated with the e\*Insight schemas.

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

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

- 1 Open the e\*Insight GUI.
- 2 From the File menu, select Create New e\*Gate Schema.
- 3 The New e\*Gate Schema dialog appears. See Figure 39.

**Figure 39** New e\*Gate Schema Dialog



- 4 Enter or select the name of the Registry Host where you want to create the schema.
- 5 Enter a user name and password that are valid on the Registry Host.
- 6 From the Based On drop-down list, select either **eIJSchema (Java)** or **eISchema (Classic)**.
- 7 Enter a name for the schema in the **Name** box.
- 8 Click **OK**.

### Copying the Schema from the Registry Host

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

#### To create a copy of the e\*Insight Schema

- 1 Open the eISchema in the e\*Gate Enterprise Manager GUI.
  - A Start the e\*Gate Enterprise Manager.
  - B Log in to eIJSchema or eISchema.
- 2 Export the schema to a file `c:\eGate\client\<e*Insight schema backup file name>`.
  - A Select **Export Schema Definitions to File ...** from the **File** pull-down menu.
  - B In the **Select archive File** dialog box enter `<e*Insight schema backup file name>` in the **File name** text box, and then click **Save**.
- 3 Create a new schema using the export file as a template.
  - A Select **New Schema** from the **File** pull-down menu.

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

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

## Installing from the CD

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

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

---

## 7.2 Sending Messages to the e\*Insight Engine (eIJSchema)

### 7.2.1 Starting a Business Process (eIJSchema)

The Collaboration in the e\*Way (or BOB) that feeds data into the business process must publish an **eX\_to\_eBPM** Event Type to the **eIcp\_eInsightJMS** connection point. This “Start” Event must include the following:

- the name of the business process
- a unique ID for the business process instance
- an event type of “START\_BP”
- all the input global attributes required for the event

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

The business process name, unique ID, and “START\_BP” event type are set by setting a value in the relevant node. The three nodes required to start the BPI populated in the **eI\_StandardEvent.xsc** ETD are described in Table 7.

**Table 7** Configuring a start message

eI_StandardEvent	How to populate
BP_EVENT.NAME	Must be filled with the exact name of the BP as it appears in the e*Insight GUI.

**Table 7** 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 101](#).

## 7.2.2 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

```
getEIActivity1Out().getEI_StandardEvent().getBP_EVENT().setAttribute("
Bonus", "NUMBER", "9000");
```

```
getEIActivity1Out().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.

### 7.2.3 Getting Attributes (eIJSchema)

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

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

### 7.2.4 Sending the “Done” Event Back to e\*Insight (eIJSchema)

When an activity completes (successfully or not), a “Done” Event must be sent back to the e\*Insight engine carrying the status of the activity. To do this, the activity Collaboration sending this Event must publish an **eX\_to\_eBPM** Event Type to the **eIcr\_eInsightJMS** connection point.

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

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

**Table 8** Done Event in Active control mode\

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

Java Node Location	How to populate
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 9** Done Event in Passive control mode

Java Node Location	How to populate
BP_EVENT.ID	Copy from source to destination. It is the responsibility of the developer to ensure that this value is stored within the event as it is passed through e*Gate.
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 101.

---

## 7.3 Sending Messages to the e\*Insight Engine (eISchema)

### 7.3.1 Starting a Business Process (eISchema)

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

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

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

The business process name, unique ID, and “START\_BP” event type are set by setting a value in the relevant node. The three nodes required to start the BPI populated in the **eX\_Standard\_Event.ssc** (Monk), or **eI\_StandardEvent.xsc** (Java) ETDs are described in Table 7.

**Table 10** Configuring a start message (eISchema)

Monk Node Location	Java Node Location	How to populate
BP_EVENT.AS.NAME.Value	BP_EVENT.NAME	Must be filled with the exact name of the BP as it appears in the e*Insight GUI.
BP_EVENT.AS.ID.Value	BP_EVENT.ID	Must be filled with a unique ID (for example, a timestamp).
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 \(eISchema\)” on page 101](#).

### 7.3.2 Setting Attributes (eISchema)

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

#### Setting Attributes in a Monk Collaboration

The Monk Collaboration must do one of the following:

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

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

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

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

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



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

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

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

- ♦ replace **<root-path>** with **~input%eX\_Event** or **~output%eX\_Event**
- ♦ replace **<attribute>** with the exact name of the attribute as it appears in the e\*Insight GUI
- ♦ replace **<value>** with the attribute value
- ♦ replace **<type>** with one of the strings “STRING”, “XML”, or “BIN” as appropriate

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

## Setting Attributes in a Java Collaboration

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

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

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

### 7.3.3 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

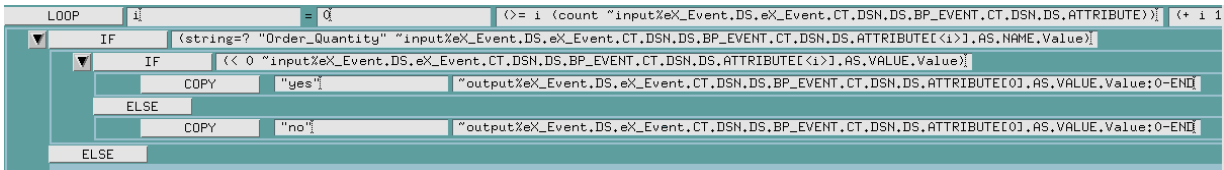
This Collaboration must do one of the following:

- 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 40 shows an example of a LOOP rule that does this.

**Figure 40** 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

### 7.3.4 Sending the “Done” Event Back to e\*Insight (eISchema)

When an activity completes (successfully or not), a “Done” Event must be sent back to the e\*Insight engine carrying the status of the activity. To do this, the activity Collaboration sending this Event must publish an **eX\_to\_eBPM** Event Type to the **eX\_eBPM** IQ.

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

The “Done” Event must have the following nodes in either the **eX\_Standard\_Event.ssc** (Monk), or **eI\_Standard\_Event** (Java) ETD populated. Table 8 describes the required nodes when an activity is set to Active control and Table 9 describes the required nodes when an activity is set to Passive control.

**Table 11** Done Event in Active control mode

Monk Node Location	Java Node Location	How to populate
BP_EVENT.AS.BPI_ID.Value	BP_EVENT.BPI_ID	Copy from source to destination.
BP_EVENT.AS.ID.Value	BP_EVENT.ID	Copy from source to destination.
BP_EVENT.AS.NAME.Value	BP_EVENT.NAME	Copy from source to destination.
BP_EVENT.AS.TYPE.Value	BP_EVENT.TYPE	Copy from source to destination.
BP_EVENT.AS.STATUS.Value	BP_EVENT.STATUS	Copy either the string “SUCCESS” or “FAILURE”, depending on whether or not the activity completed successfully.
BP_EVENT.CT.DSN.DS.ACTIVITY.AS.ID.Value	BP_EVENT.ACTIVITY.ID	Copy from source to destination. (Not absolutely required, but recommended to speed processing.)
BP_EVENT.CT.DSN.DS.ACTIVITY.AS.NAME.Value	BP_EVENT.ACTIVITY.NAME	Copy from source to destination.
BP_EVENT.DS.ATTRIBUTE[i].AS.VALUE.Value	BP_EVENT.ATTRIBUTE[i].VALUE	Use a loop or the e*Insight helper function <b>eX-get-attribute</b> , or <b>getATTRIBUTE_VALUE</b> to determine the value of the machine-assigned attribute, <b>eX_eBPMServer</b> , 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. ( <b>Active mode only</b> )
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. ( <b>Active mode only</b> )
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. ( <b>Active mode only</b> )

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

### Using the helper function eX-copy-no-attribute to send the “Done” Event

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

```
eX-copy-no-attribute ~input%eX_Event ~output%eX_Event
```

```
eX-set-BP_EVENT ~output%eX_Event "STATUS" "SUCCESS"
```

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

**Table 12** Done Event in Passive control mode

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

In addition, the “Done” Event must carry with it the values for any attributes (global or local) specified as “Output” or “Input/Output”. These can be set as shown in [“Setting Attributes \(eISchema\)” on page 101](#).

# e\*Insight Implementation (eIJSchema)

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

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

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

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

## 8.1 Overview

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

**Table 13** Overview of implementation tasks

	Task	Section
1	Create the business process (BP) in the e*Insight GUI	<a href="#">“Create the Payroll BP in e*Insight” on page 113</a>
2	Use the e*Insight GUI to configure the e*Gate schema that supports e*Insight	<a href="#">“Configure the Integration Schema (e*Insight)” on page 116</a>
3	Configure the e*Insight Engine	<a href="#">“Configure the e*Insight Engine” on page 120</a>
4	Add and configure the user-defined e*Gate components	<a href="#">“Configure User-defined e*Gate Components” on page 121</a>
5	Run and test the scenario	<a href="#">“Run and Test the e*Insight Scenario” on page 131</a>

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

### 8.1.1 Case Study: Payroll Processing

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

Figure 41 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 41** e\*Insight Data Flow Diagram

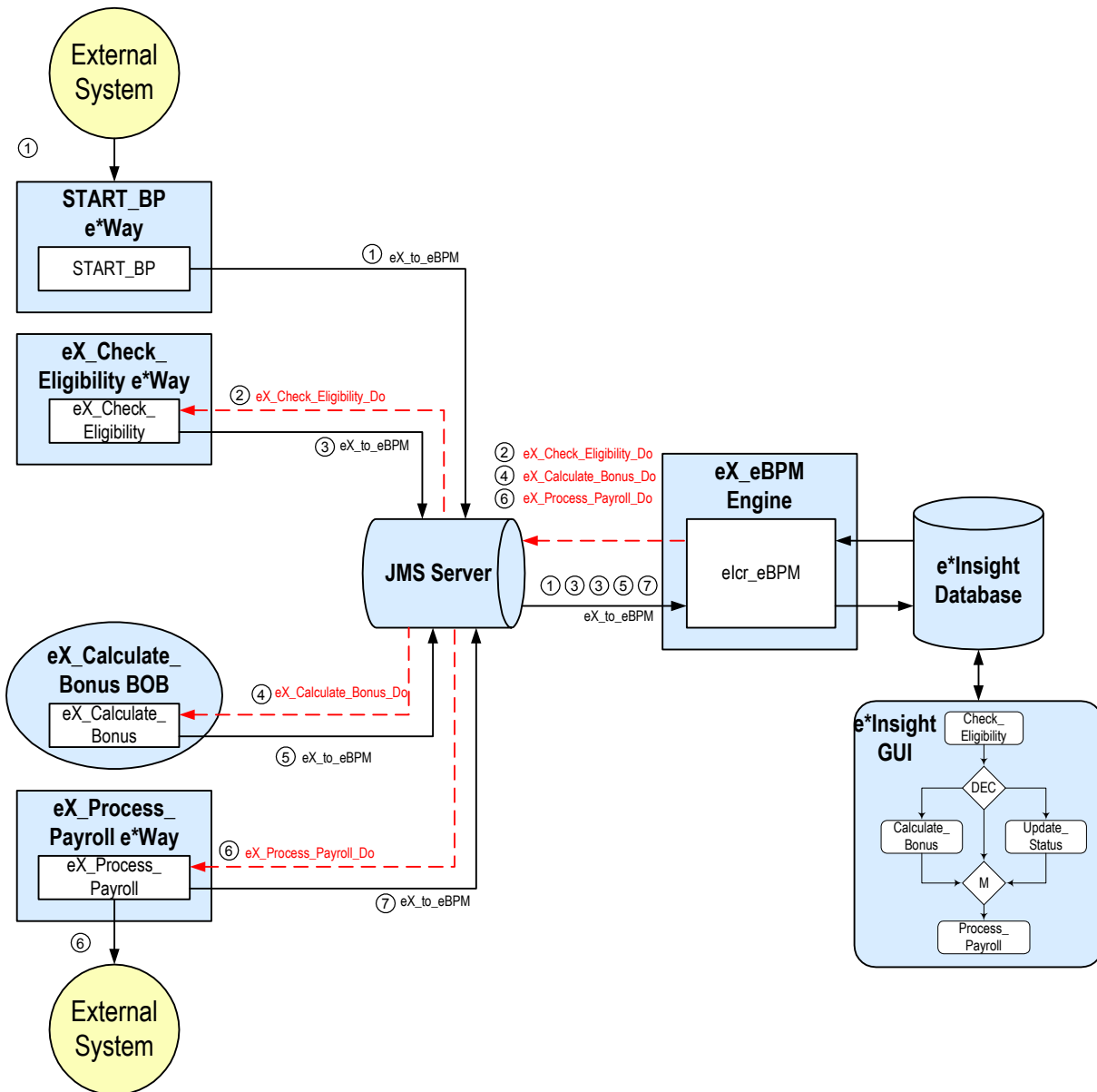
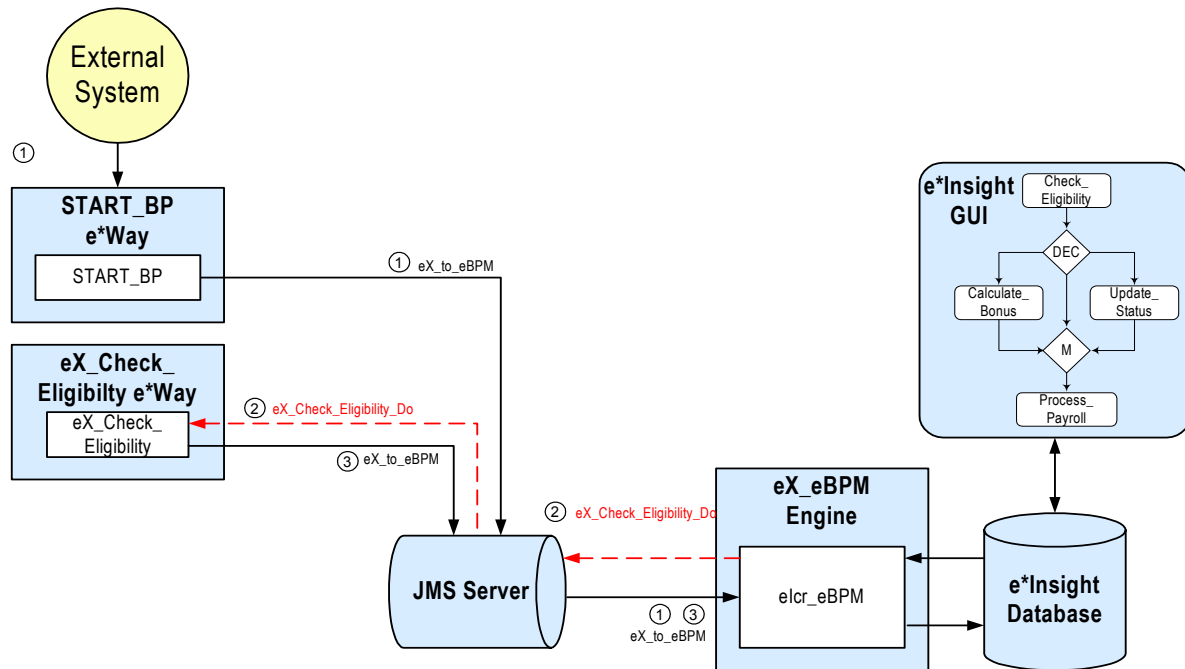


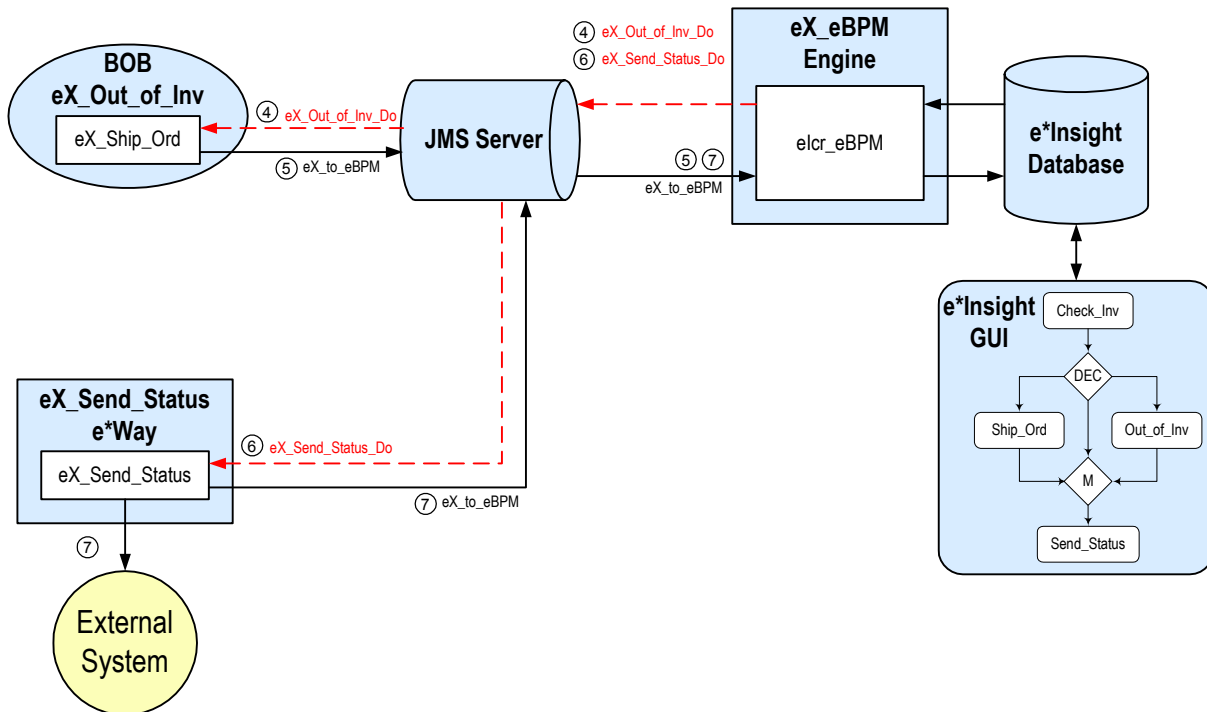
Figure 42 e\*Insight Data Flow Diagram (Part 1)



- ① The user-defined **START\_BP** e\*Way picks up the text file containing the order information from a shared location on the network, uses the order information to create the e\*Insight Event that causes the e\*Insight engine to start a business process instance, and publishes it using the **eX\_to\_eBPM** Event Type to the JMS Server. The e\*Insight engine retrieves the Event and uses the information it contains to start the BPI.
- ② The e\*Insight engine publishes a “Do” Event (**eX\_Check\_Eligibility\_Do**) for the first activity in the business process (**Check\_Eligibility**). **eX\_Check\_Eligibility** e\*Way, the e\*Gate component that corresponds to this activity in the business process, retrieves this Event from the JMS Server and uses the information it contains to check the availability of the items ordered.
- ③ When the **Check\_Eligibility** activity is finished, the **eX\_Check\_Eligibility** BOB publishes a “Done” Event using the **eX\_to\_eBPM** Event Type. The e\*Insight engine retrieves the “Done” Event, updates the BPI to reflect whether the employee is eligible for a bonus, and then moves forward to the next activity in the business process based on the result of a decision gate. If the employee is eligible for a bonus, the next activity is **Calculate\_Bonus**; if the employee is in the sales department then the next activity is **Process\_Payroll**, if the employee has been employed for less than three months then the next activity is **Update\_Status**.

Let’s assume the employee has been employed for less than three months. The e\*Insight engine processes the e\*Insight script corresponding to the **Update\_Status** activity in the business process, and then moves forward to the next activity in the business process—**Process\_Payroll**.

Figure 43 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 **Calculate\_Bonus** activity is finished, the **eX\_Calculate\_Bonus** BOB publishes a "Done" Event indicating that the bonus has been calculated. The e\*Insight engine retrieves this Event, updates the BPI, and then moves forward to the next activity in the business process—**Process\_Payroll**.
- ⑥ The e\*Insight engine publishes a "Do" Event (**eX\_Process\_Payroll\_Do**) corresponding to the **Process\_Payroll** activity in the business process. The **eX\_Process\_Payroll** e\*Way retrieves this Event and uses the information it contains to send a order status report to the payroll system.
- ⑦ The **eX\_Process\_Payroll** e\*Way publishes two Events: one containing the status report to be sent to the payroll system, and also the "Done" Event. The e\*Insight engine retrieves the "Done" Event and uses the information it contains to update the BPI to indicate that the final activity in the business process has completed successfully.



## 8.2 Create the Payroll BP in e\*Insight

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

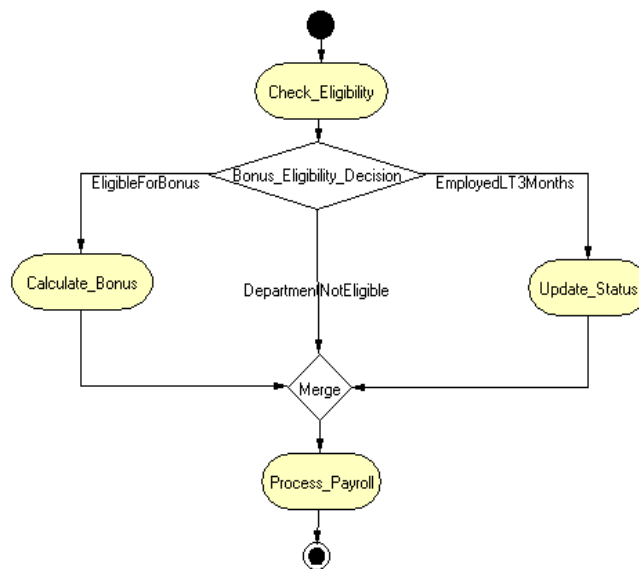
- 1 Create a Business Process named **Payroll**.
- 2 Add the activities.
- 3 Add the decision gates.
- 4 Make the connections between the activities and gates.
- 5 Add all the global attributes (see Table 14).
- 6 Assign global attributes to activities (see Table 15).
- 7 Add the logic to the decision gates (see Table 16).
- 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 44 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 44** Payroll Business Process Model



**Table 14** 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 15** 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 16** Decision Gate

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

**Table 16** Decision Gate

Link	Target Activity	Expression
EligibleForBonus	Calculate_Bonus	(Default)

### 8.2.1 Creating the processes performing the Activities

The activities in our scenario can either be performed by e\*Gate or an e\*Insight Script. Three of the activities (**Check\_Eligibility**, **Calculate\_Bonus**, and **Process\_Payroll**) use e\*Gate. This is described in [“Configure the Integration Schema \(e\\*Insight\)” on page 116](#).

The **Update\_Status** activity is performed by an e\*Insight Script. This is described below.

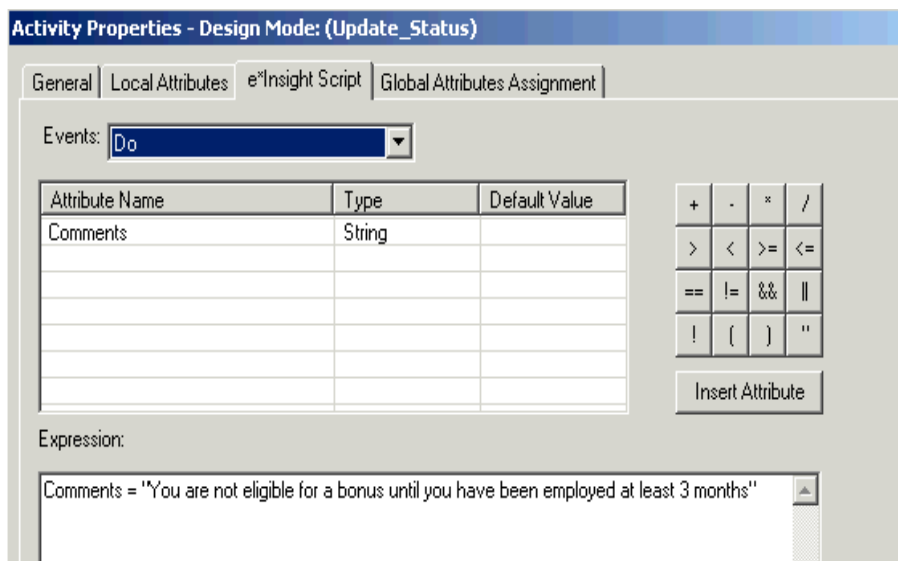
### Configuring the e\*Insight Script for Update\_Status

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 45.

**Figure 45** Update\_Status e\*Insight Script Tab



## 8.3 Configure the Integration Schema (e\*Insight)

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

### To create a copy of eIJSchema

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

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

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

### Integration Schema Activity Components Summary

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

**Table 17** Integration Schema Activity Components

Name	Type	Participating Host	Configuration Instructions
eX_Check_Eligibility	Multi-Mode e*Way	localhost	<a href="#">“Creating the eX_Check_Eligibility Multi-Mode e*Way” on page 117</a>
eX_Calculate_Bonus	BOB	localhost	<a href="#">“Creating the eX_Calculate_Bonus BOB” on page 119</a>
eX_Process_Payroll	Single-Mode e*Way	localhost	<a href="#">“Process_Payroll e*Way Configuration” on page 120</a>

For information on how to use the e\*Insight GUI to configure the e\*Gate Registry see the *e\*Insight Business Process Manager 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.

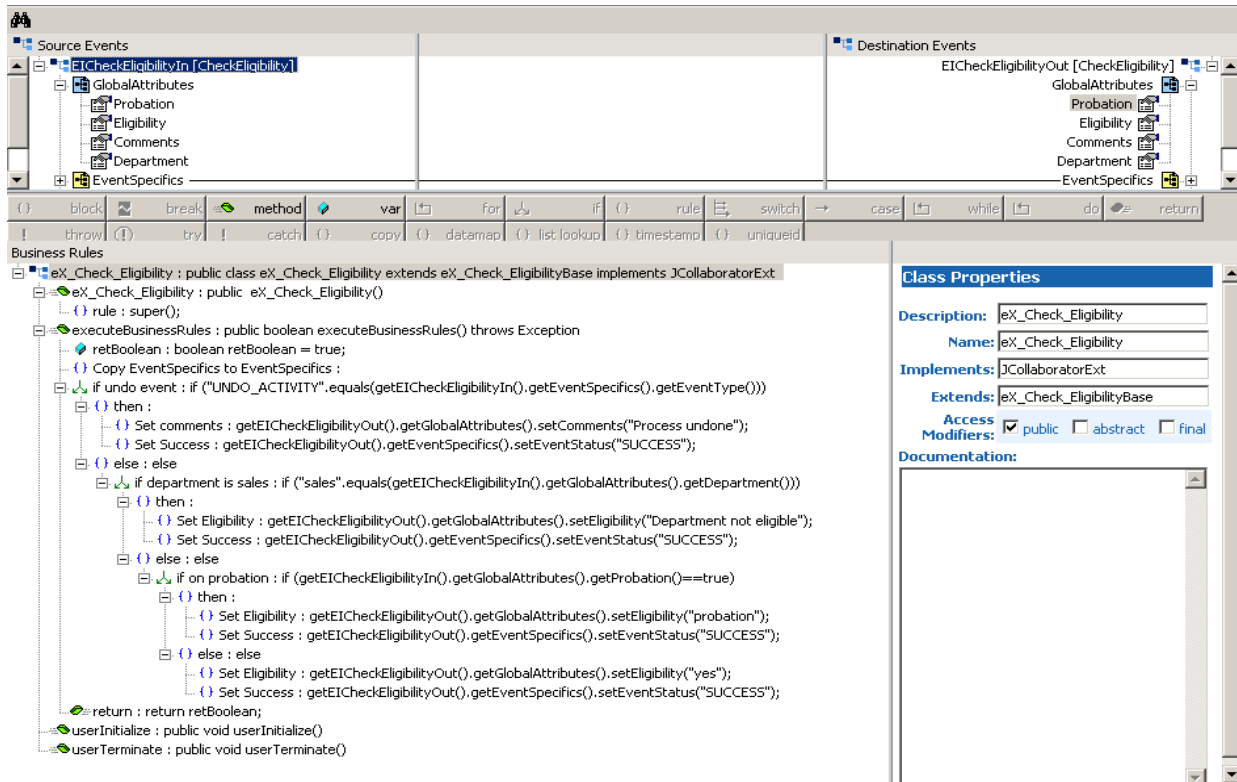
Any other employee is treated as being eligible for a bonus and the CRS set the value of **Eligible** to “yes” and sends “SUCCESS”.

### 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 46, on the following page, shows the **eX\_Check\_Eligibility.xpr** CRS used in this example.

Figure 46 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

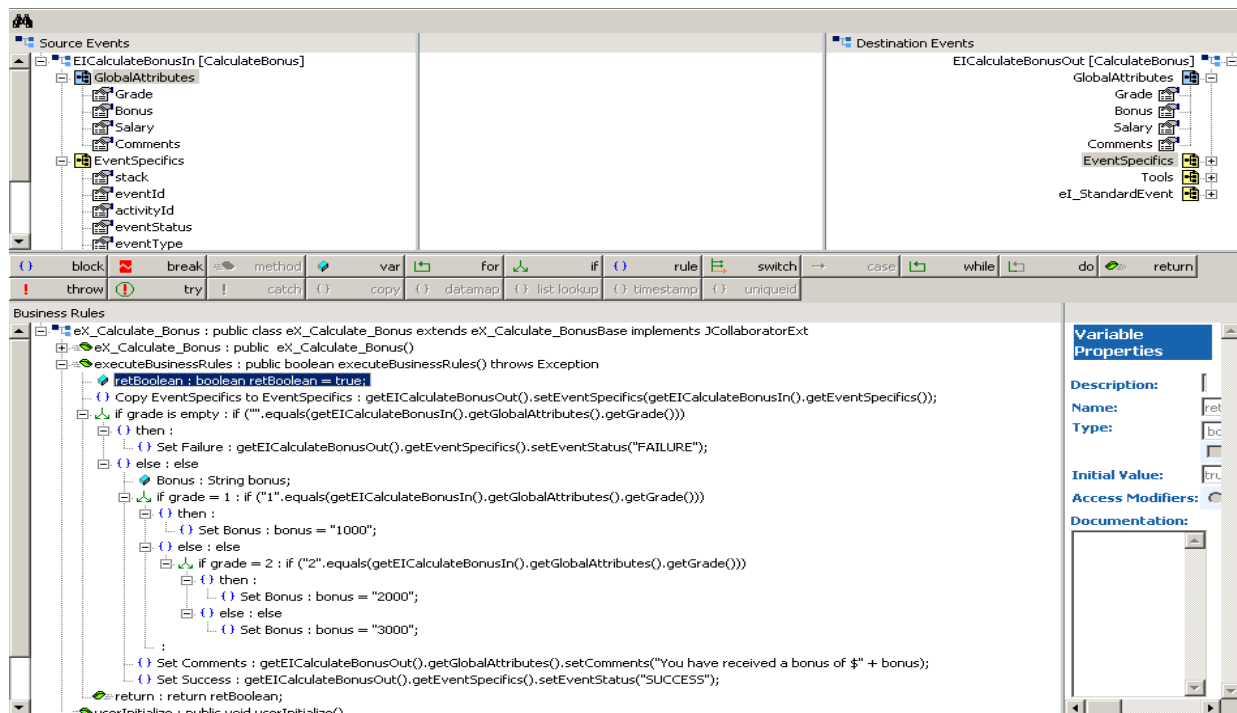
- 1 In the e\*Insight GUI, open the **Calculate\_Bonus** activity properties.
- 2 On the **General** tab, **e\*Gate Module** section, select a **Module Type** of **BOB**.
- 3 Click **New**.

The **Define Collaboration** dialog appears.

- 4 Click **OK**.
- 5 Create eX\_Calculate\_Bonus.xpr.

Figure 47 shows the eX\_Calculate\_Bonus CRS used in this example.

**Figure 47** 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 127](#).

---

## 8.4 Configure the Integration Schema (e\*Gate)

The remaining components used in this implementation are configured from the e\*Gate Enterprise Manager. You must start the e\*Gate Enterprise Manager and open the Payroll schema that you created in [“To create a copy of eIJSchema” on page 116](#).

### 8.4.1 Configure the e\*Insight Engine

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

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

### Edit the elcp\_eInsightEngine Connection Configuration File

Most of the parameter settings in the elcp\_eInsightEngine connection’s configuration file should not be changed. [“Using XA” on page 34](#) discusses the parameters that may need to be changed depending on the implementation. Use the e\*Way Editor and the information in [“Using XA” on page 34](#) to make the required changes for the Payroll example.

### 8.4.2 Configure the JMS Connection

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

#### To configure the JMS Connection

- 1 From the e\*Gate Enterprise Manager GUI components view, select the **e\*Way Connections** folder.
- 2 Select the **eI\_cpeInsightJMS** connection, and click the **Properties** tool.
- 3 In the **e\*Way Connection Configuration File** section, click **Edit**.



- 4 From the **Goto Section** list, select **Message Service**.
- 5 Enter a **Server Name** and **Host Name** where your JMS server resides.

**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.

## 8.5 Configure User-defined e\*Gate Components

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

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

### Configuration Order for the User-defined Components

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

**Table 18** Configuration Order for User-defined Components

	Task	Section
1	Add and configure the START_BP e*Way	<a href="#">“Configure the START_BP e*Way” on page 121</a>
2	Configure the Process_Payroll e*Way	<a href="#">“Configure the Process_Payroll e*Way” on page 127</a>

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

#### 8.5.1 Configure the START\_BP e\*Way

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

The **START\_BP** e\*Way is completely user defined and must be added to the **eIJSchema** in the e\*Gate Enterprise Manager. In an actual implementation, the choice of e\*Way (or BOB) would depend on the requirements of the situation. For example, if the data were coming from an SAP system, you might select an SAP ALE e\*Way; or if the data were

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

Table 19 shows the steps to configure the **START\_BP** e\*Way.

**Table 19** Configuration steps for the **START\_BP** e\*Way

	Step	Section
1	Add the e*Way and create the e*Way configuration file	<a href="#">“Step 1: Create the START_BP e*Way” on page 122</a>
2	Create the Input ETD	<a href="#">“Step 2: Create the Input ETD” on page 122</a>
3	Create the START_BP Collaboration Rules script (CRS)	<a href="#">“Step 3: Create the START_BP Collaboration” on page 123</a>
4	Configure the Collaboration in the GUI	<a href="#">“Step 4: Configure the Collaboration in the GUI” on page 126</a>

### Step 1: Create the **START\_BP** e\*Way

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

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

**Table 20** Start e\*Way Parameters

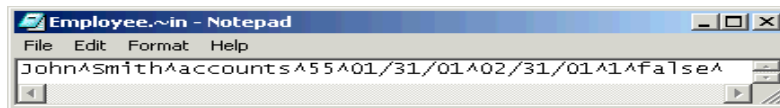
Screen	Parameter	Setting
General Settings	(All)	(Default)
Outbound (send) settings	(All)	(Default)
Poller (inbound) settings	PollDirectory	<eGate>\client\data\Payroll
	(All others)	(Default)
Performance Testing	(All)	(Default)

### Step 2: Create the Input ETD

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

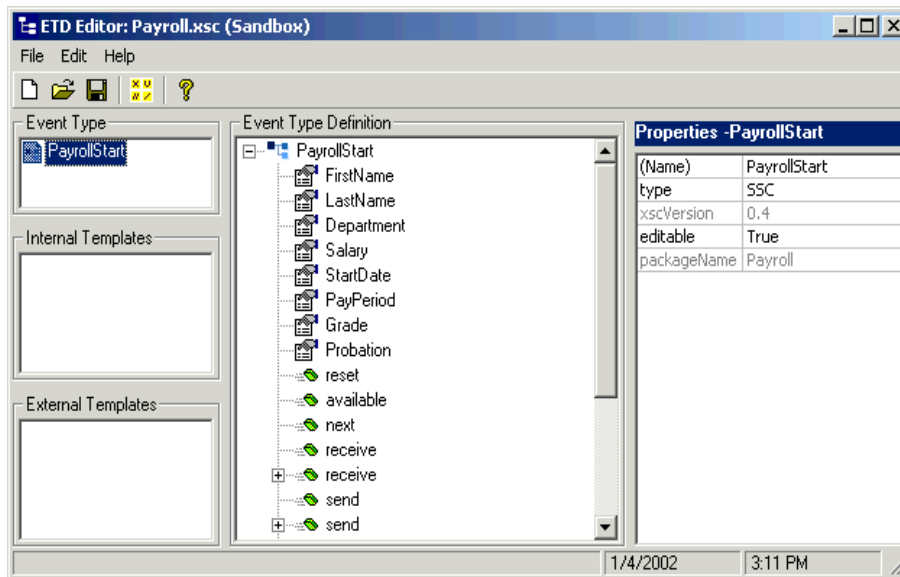
The input data file used in this example is shown in Figure 48. Place this data file at the directory location **c:\eGate\client\data\Payroll**.

**Figure 48** 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 49. Set the global delimiter to a ^ character. For more information on using the ETD Editor see the ETD Editor's online help.

**Figure 49** Input ETD: PayrollStart.xsc (Java)



### Step 3: Create the START\_BP Collaboration

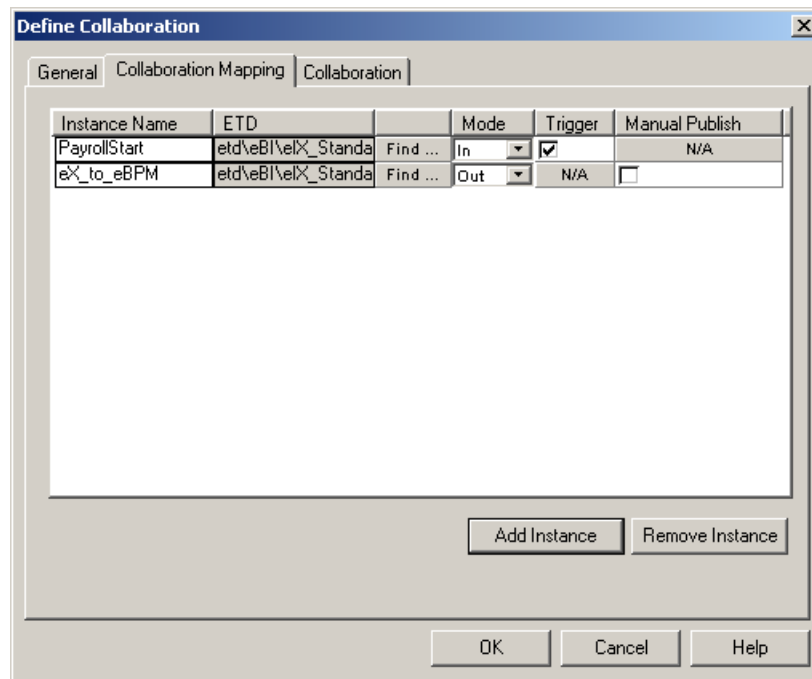
The Collaboration that sends the Event that starts the BPI must do two things:

- 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.

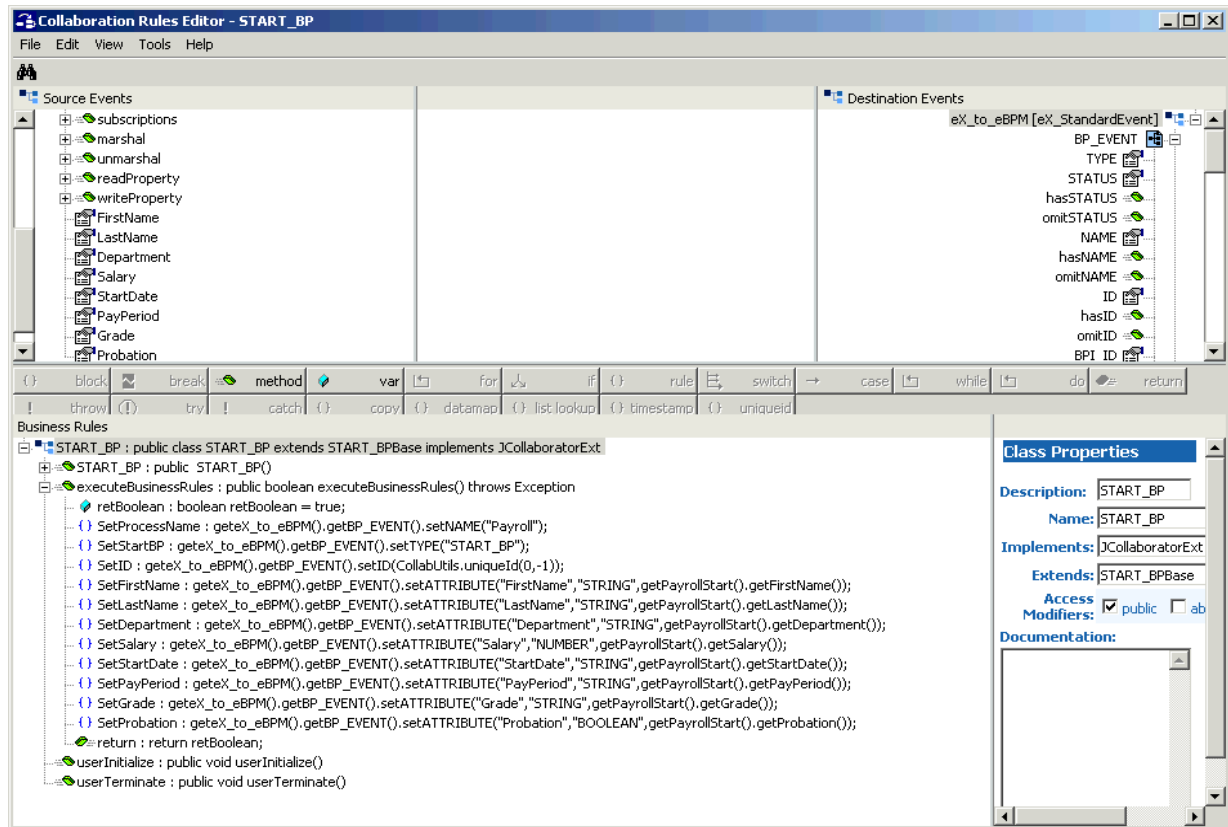
- 1 Create a Collaboration Rule, **START\_BP**, that uses the Java service.
- 2 Configure the Collaboration Mapping tab, as shown in Figure 50.

**Figure 50** 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 51.

Figure 51 START\_BP CRS

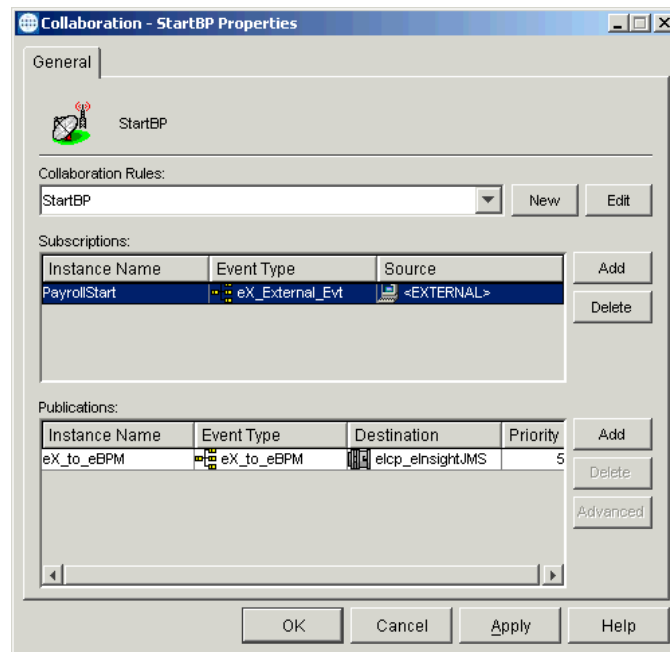


## Step 4: Configure the Collaboration in the GUI

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 Enterprise Manager GUI.

- 1 Create a Collaboration for the **Start\_BP** e\*Way configured as shown in Figure 52.

**Figure 52** Start\_BP Collaboration



## 8.5.2 Configure the Process\_Payroll e\*Way

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

This e\*Way must accomplish two tasks:

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

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

Table 21 shows the steps to configure the Process\_Payroll e\*Way.

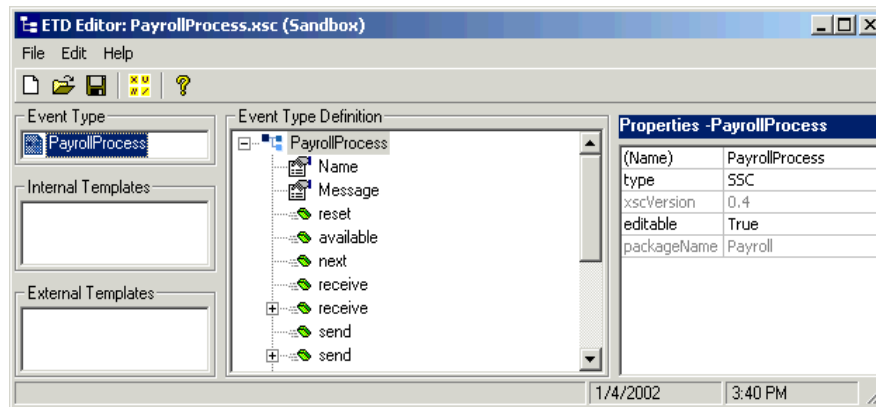
**Table 21** Process\_Payroll e\*Way configuration steps

	Step	Section
1	Find the executable and create the e*Way configuration file	<a href="#">“Step 1: Configure the e*Way” on page 131</a>
2	Create the Output ETD	<a href="#">“Step 1: Create the Output ETD:PayrollProcess.xsc using Java” on page 127</a>
3	Create the eX_Process_Payroll.tsc CRS	<a href="#">“Step 3: Create the Process_Payroll Collaboration Rule” on page 128</a>
4	Configure the Collaboration in the GUI	<a href="#">“Step 4: Configure the Collaboration” on page 129</a>

### 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 53. Set the global delimiter to a , (comma) character. For more information on using the ETD Editor see the ETD Editor’s online help.

Figure 53 PayrollProcess.xsc ETD



### Step 3: Create the Process\_Payroll Collaboration Rule

This CRS must accomplish three things:

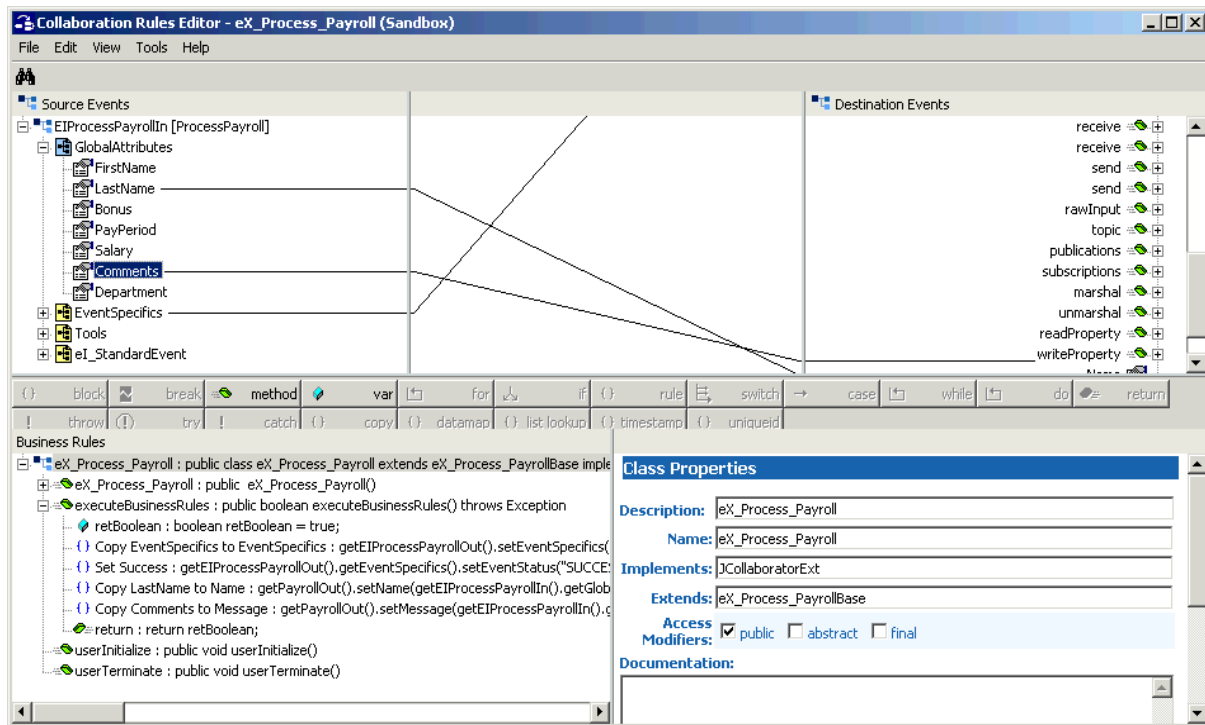
- put the output data into a readable format that can be written to a file
- set the BP status node to "SUCCESS"
- send the "Done" Event back to the e\*Insight engine

#### 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 to open the Collaboration Editor, and add code as show in Figure 54.



Figure 54 eX\_to\_Process\_Payroll.xpr CRS



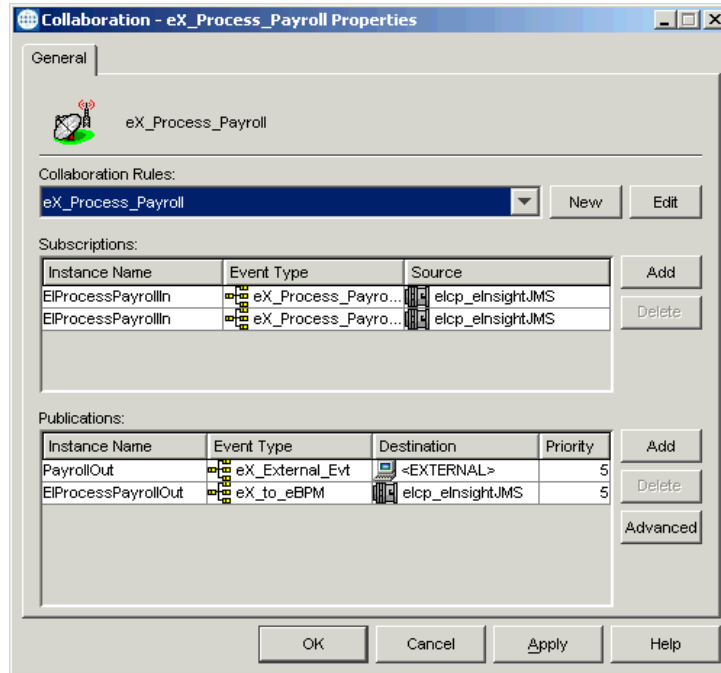
- 7 Compile and save the CRS.
- 8 Close the editor.
- 9 In the **Process\_Payroll** Activity properties, click **Configure e\*Gate Schema**.
- 10 Click **OK** to close the information dialog.
- 11 Close the **Process\_Payroll** Activity properties.

### 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 55.

Figure 55 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 22** Process\_Payroll e\*Way Parameters

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

### 8.5.3 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
```

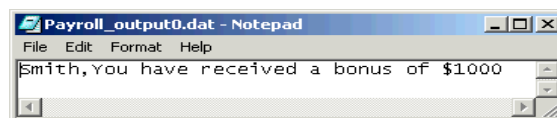
Substitute the appropriate username and password for your installation.

- 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 48 on page 123](#) (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 56](#) appears in the directory indicating successful completion of the BPI.

**Figure 56** In Stock Output File

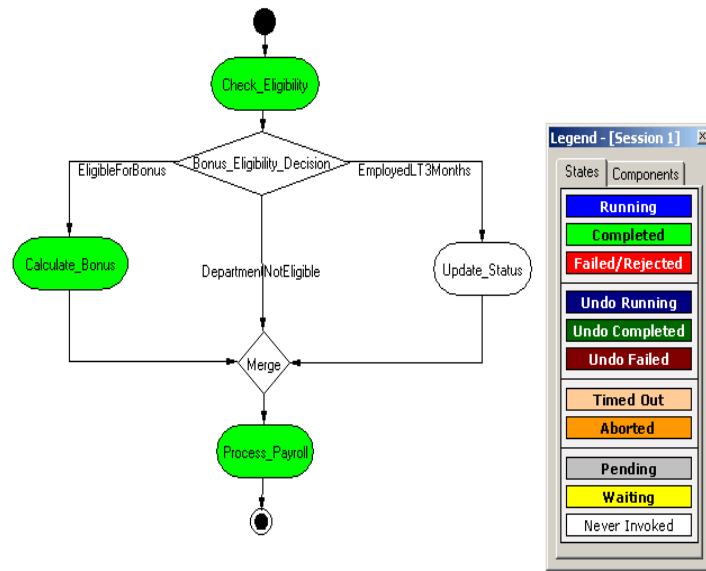


- 6 Switch to the e\*Insight GUI and, while in monitor mode, select the most recent BPI from the **List** tab, and then select the **Diagram** tab to observe the path that the data has taken.

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 57](#) illustrates how the successfully completed BPI appears in the e\*Insight GUI.

**Figure 57** Eligible for Bonus Completed BPI Diagram

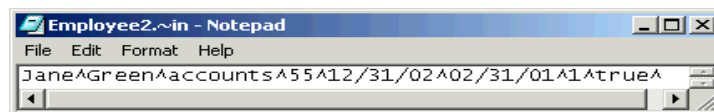


## Not Eligible Processing

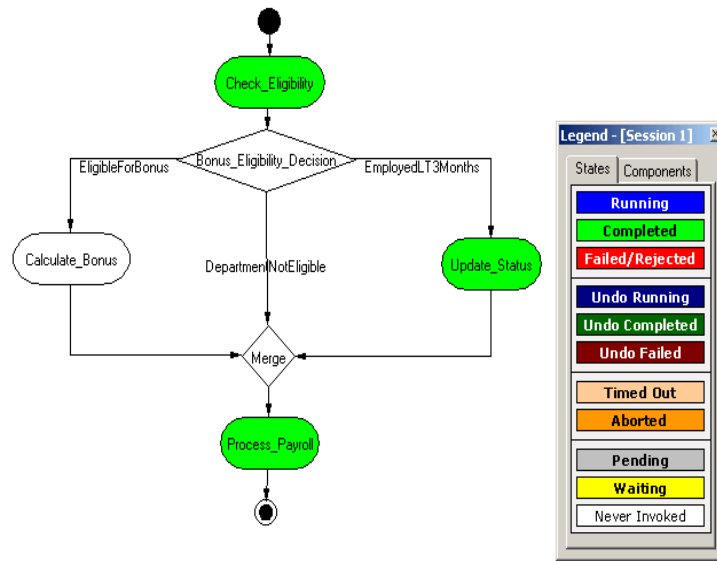
Testing the functionality for employees that are not eligible for a bonus uses exactly the same procedure as that for eligible processing except that different input data is submitted.

- Verify that sending in the data shown in Figure 58 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 59 and the output file shown in Figure 60.

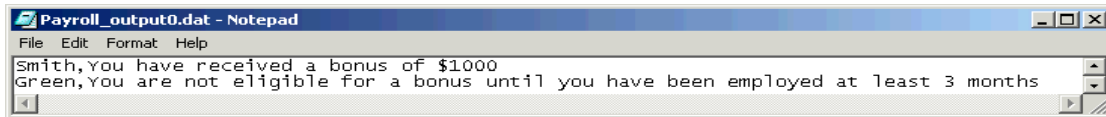
**Figure 58** EmployedLT3Months Input File



**Figure 59** Employed Less Than 3 months Completed BPI Diagram

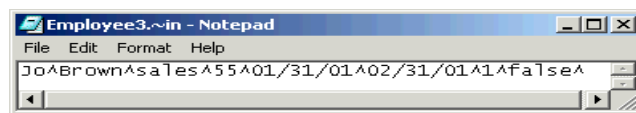


**Figure 60** Out of Stock Output File

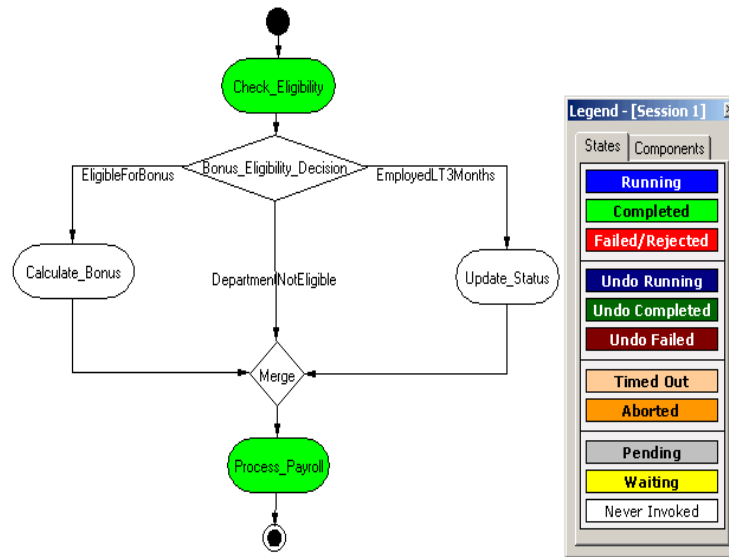


- Verify that sending in the data shown in Figure 61 with a department of sales causes the business process to take the “DepartmentNotEligible” branch of the decision gate and create the diagram shown in Figure 62 and the output file shown in Figure 63.

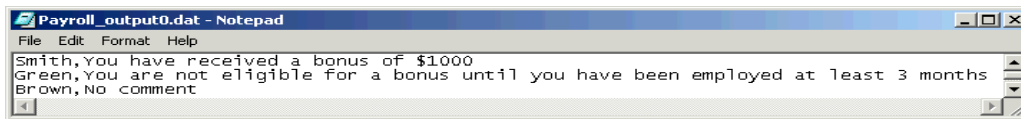
**Figure 61** DepartmentNotEligible Input File



**Figure 62** Department Not Eligible Completed BPI Diagram



**Figure 63** Department Not Eligible Output File



### 8.5.4 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.

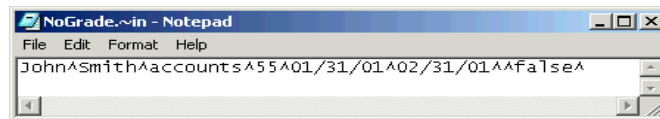
If an activity fails and the **Manual Restart** option is selected on the **General** tab of the **Activity Properties** dialog box for that activity, then e\*Insight marks the activity as “Failed” in the GUI. Then the activity waits for the user to initiate the next course of action; skip, restart, 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 131.
- 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 64 and change the extension to **“.fin”**.

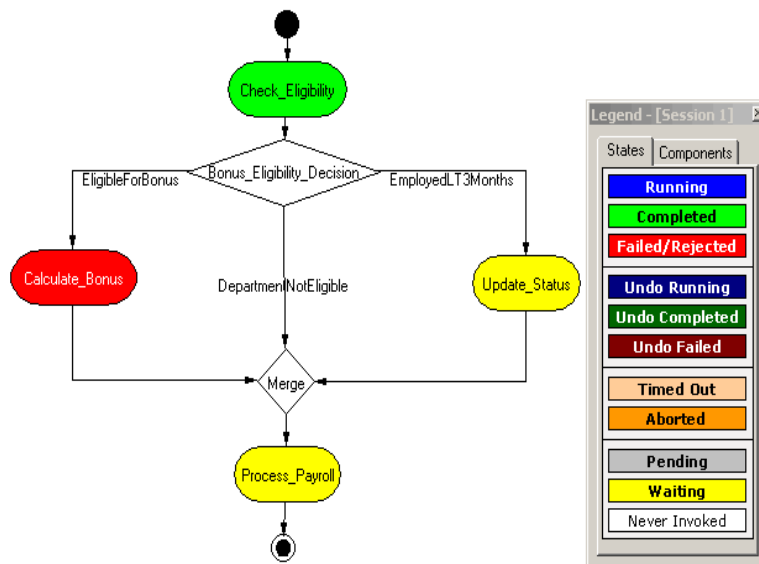
**Figure 64** Manual Undo Input File



**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 65 on the next page.

**Figure 65** 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.

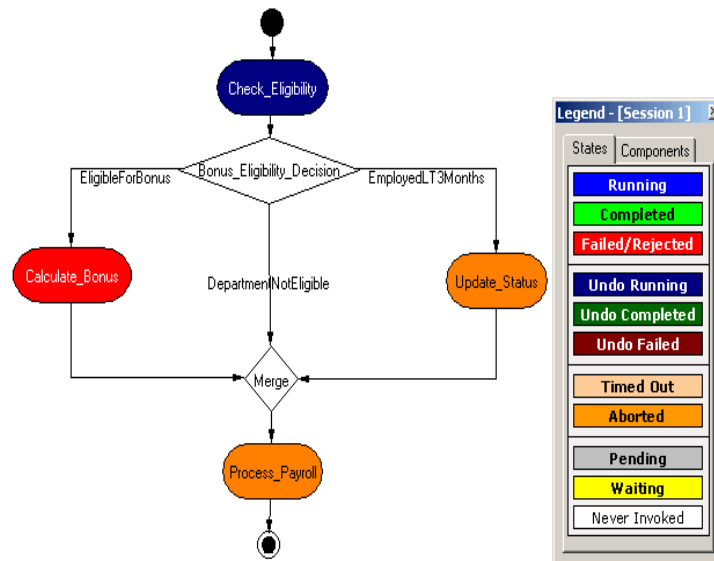
- 5 Right-click the **Calculate\_Bonus** activity from the tree view, then select **Properties** from the popup menu.

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 66** Manual Undo Completed BPI Diagram



### 8.5.5 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.

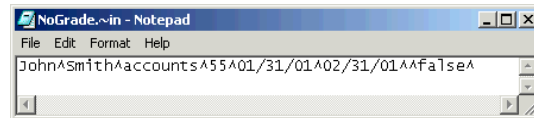
#### Repairing a String Attribute

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 131**.
- 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 67, and change the extension to **“.fin”**.

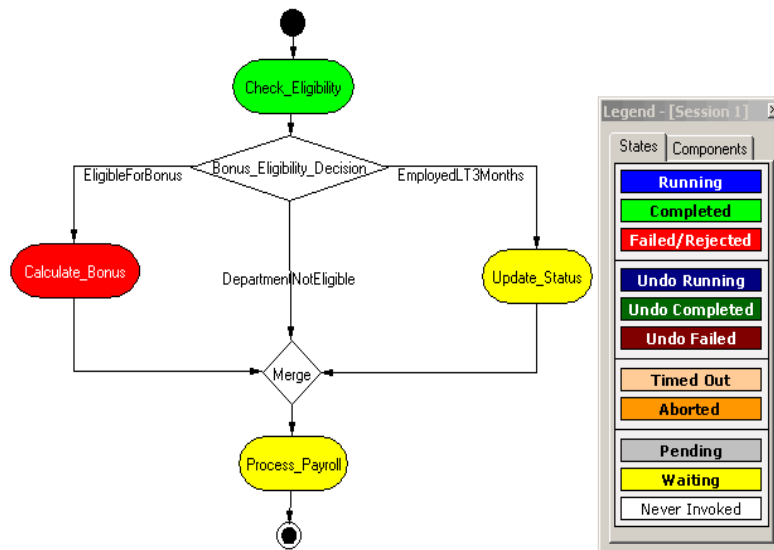
**Figure 67** Attribute Repair Input File



**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.

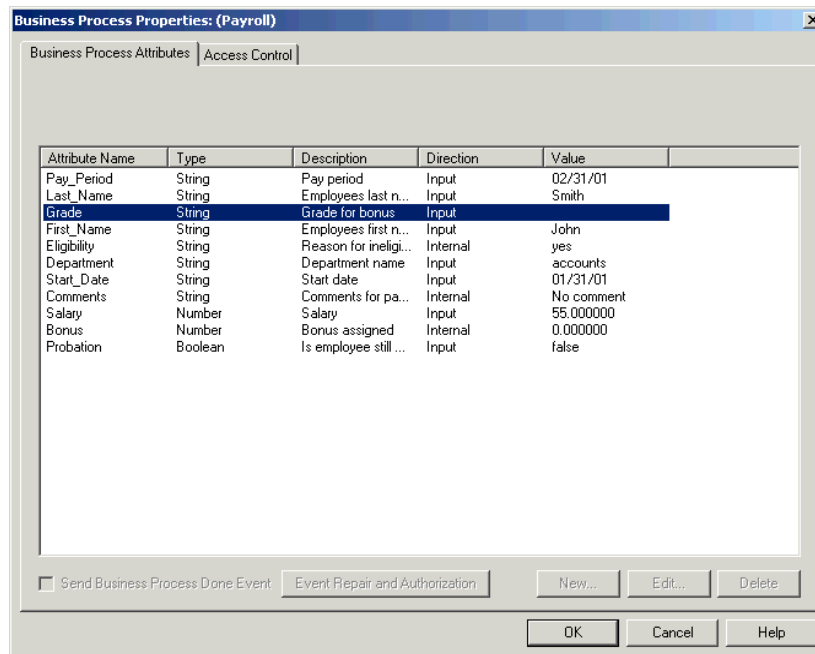
**Figure 68** Attribute Repair—Failed BPI Diagram



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 69** Attribute Repair—Business Process Properties



- 6 On the **Business Process Attributes** tab in the **Business Process Properties** dialog
- 7 In the **Edit Business Process Attribute** dialog box, change the value of the attribute in the **Current Value:** box to 3, and then click **OK**.
- 8 Click **OK** to close the **Business Process Properties** dialog box.
- 9 Right-click the **Check\_Eligibility** activity from the tree view, then select **Properties** from the popup menu.

The **Activity Properties - Monitor Mode: (Check\_Eligibility)** dialog box displays.

- 10 Select the **Result/Recourse** tab.
- 11 Click **Restart Activity**, and then click **OK**.

The BPI now completes successfully.

# 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 109](#) for the initial configuration instructions.

---

## 9.1 Overview

The major steps in the implementation are:

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

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

### 9.1.1 Case Study: Payroll Processing

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

#### Important Considerations

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

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

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

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

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

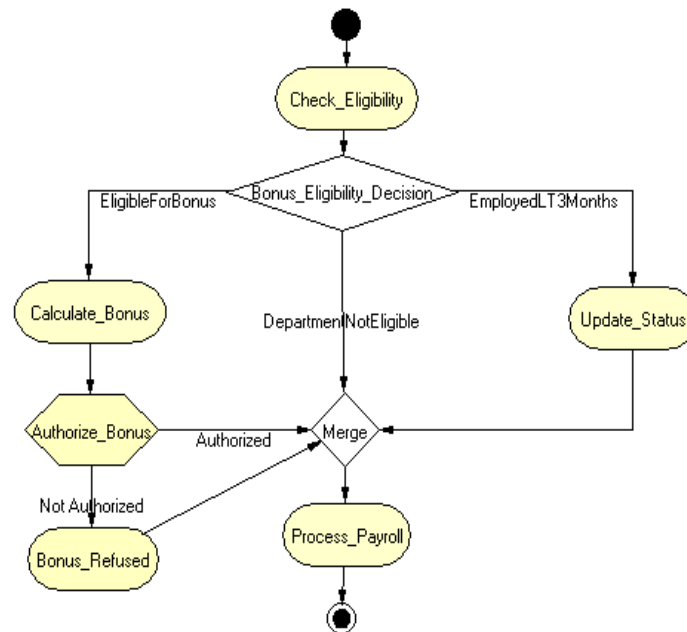
**Note:** *The Authorization Activity has two fixed Local Attributes—assignedTo (the user to whom the Authorization process is assigned) and performedBy (automatically assigned at run time as the user logged in to the e\*Insight GUI). These two values must match in order for the user to Authorize, Reject,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

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

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

Use the diagram shown in Figure 70 and the following tables to create the BP in e\*Insight.

**Figure 70** Payroll Business Process Model



### 9.2.1 Creating the processes performing the Activities

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

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

The **Bonus\_Refused** activity is performed by an e\*Insight Script. This is described below.

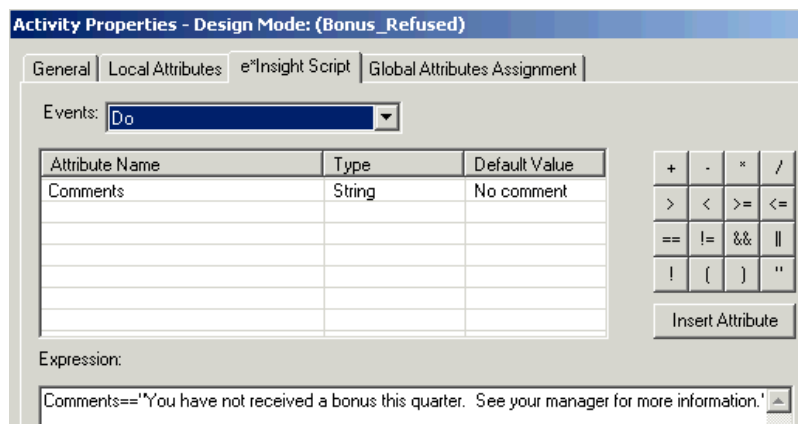
### Configuring the e\*Insight Script for Bonus\_Refused

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

To configure the e\*Insight Script for Update\_Status

- 1 From the **Bonus\_Refused** properties, **Activity Performed by** area, select **e\*Insight Script**.
- 2 Select the **e\*Insight Script** tab.
- 3 Configure the script as shown in Figure 71.

Figure 71 Update\_Status e\*Insight Script Tab



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

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

### 9.3.1 Testing the Standard Business Logic

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

#### Authorized Processing

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

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

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

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

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

Substitute the appropriate username and password for your installation.

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

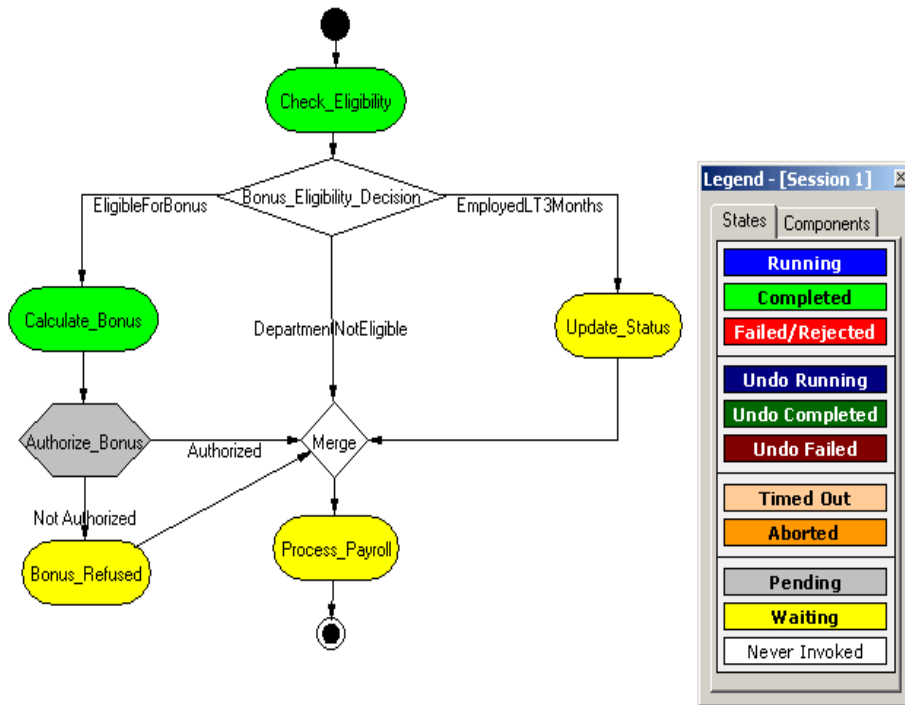
*Note:* The change of the extension to “.~in” indicates that the data file has been picked up by the **START\_BP** e\*Way.

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

The **Authorize\_Bonus** Authorization Activity should appear gray. This shows that the activity is pending.

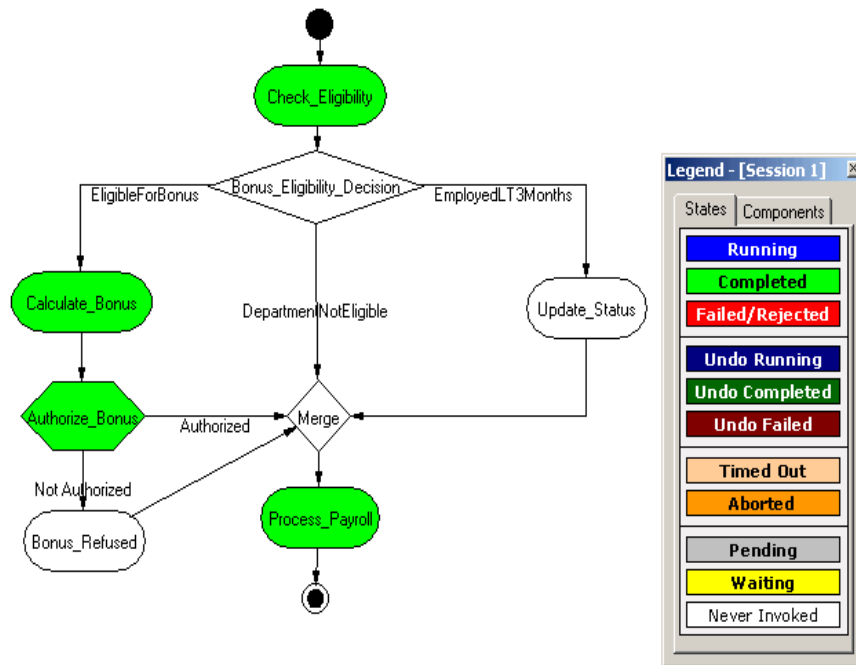


**Figure 72** Authorize\_Bonus Pending BPI Diagram



- 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 73.

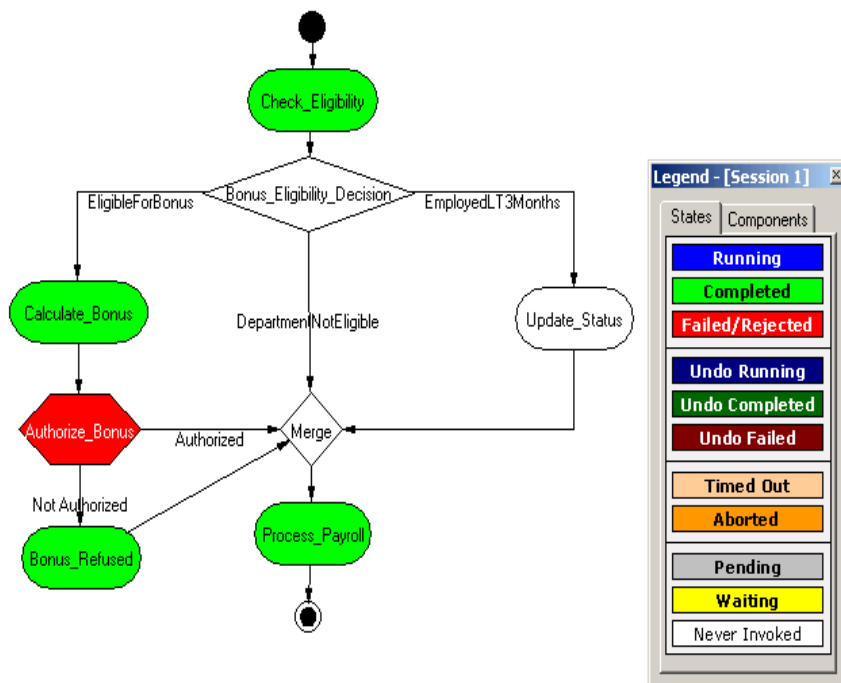
Figure 73 Authorization\_Quantity - Authorized



### Not Authorized Processing

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

Figure 74 Authorize\_Bonus - Not Authorized



# e\*Insight Implementation (eISchema)

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

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

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

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

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

---

## 10.1 Overview

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

**Table 23** Overview of implementation tasks

	Task	Section
1	Create the business process (BP) in the e*Insight GUI	<a href="#">“Create the ProcessOrder BP in e*Insight” on page 151</a>
2	Use the e*Insight GUI to configure the e*Gate schema that supports e*Insight	<a href="#">“Configure the Integration Schema” on page 154</a>
3	Configure the e*Insight Engine	<a href="#">“Configure the e*Insight Engine” on page 162</a>
4	Add and configure the user-defined e*Gate components	<a href="#">“Configure User-defined e*Gate Components” on page 162</a>
5	Run and test the scenario	<a href="#">“Run and Test the e*Insight scenario” on page 176</a>

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

### 10.1.1 Case Study: Order Processing

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

Figure 76 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 75** e\*Insight Data Flow Diagram

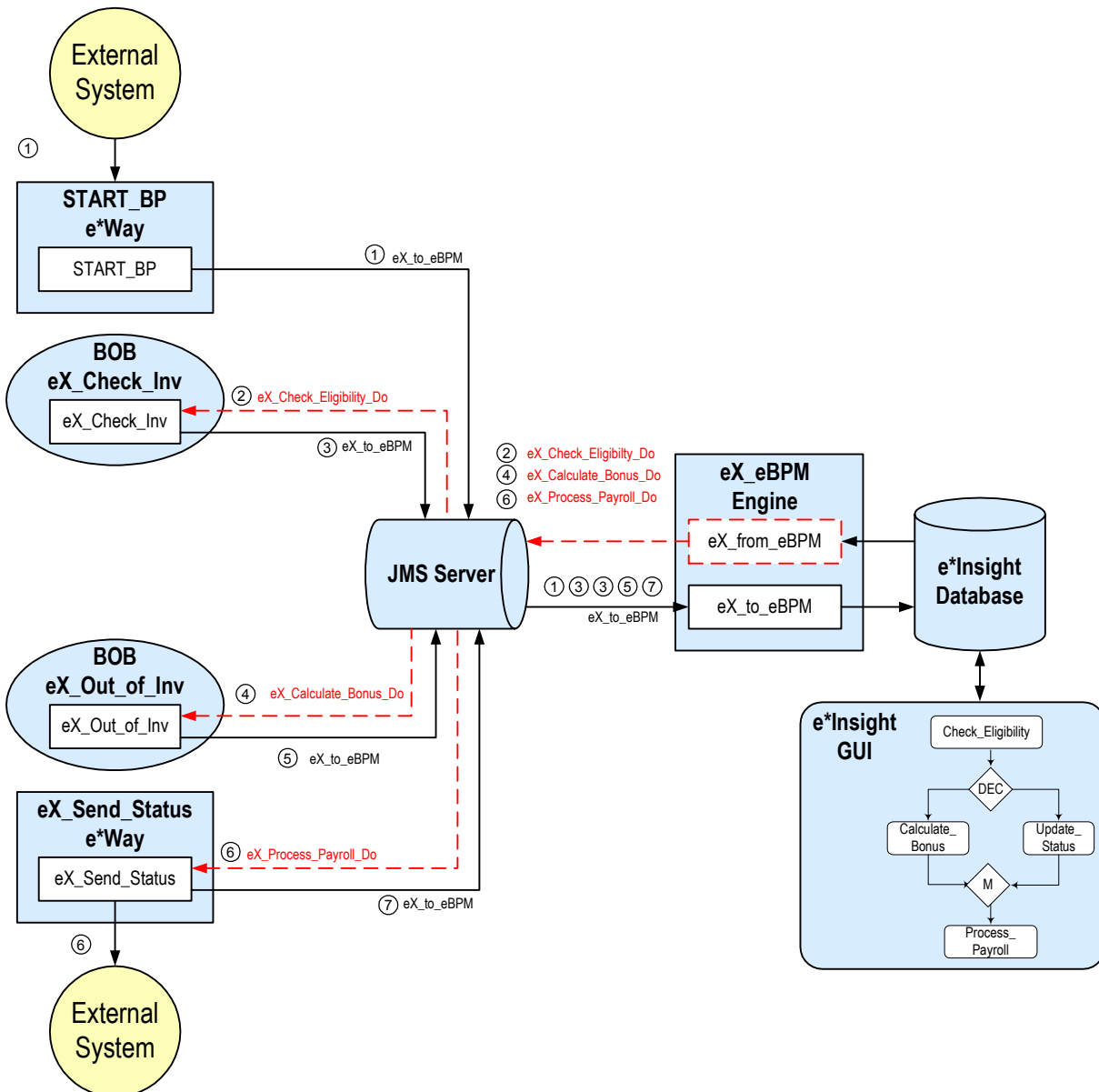
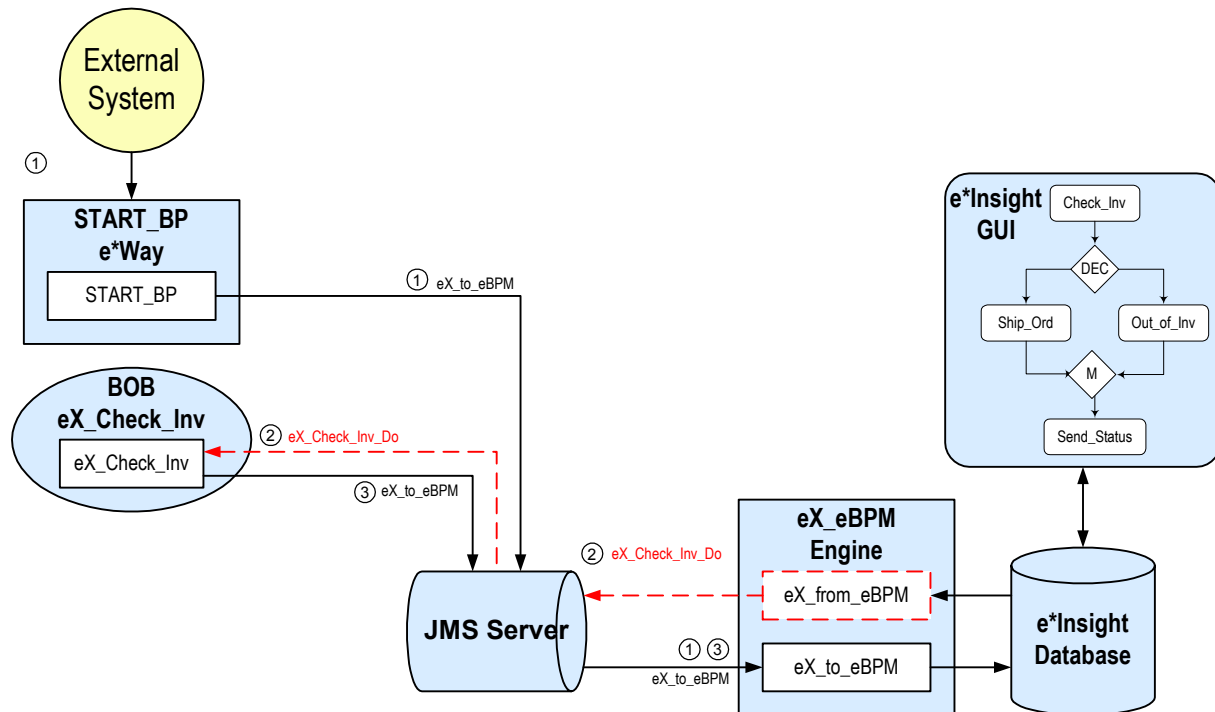


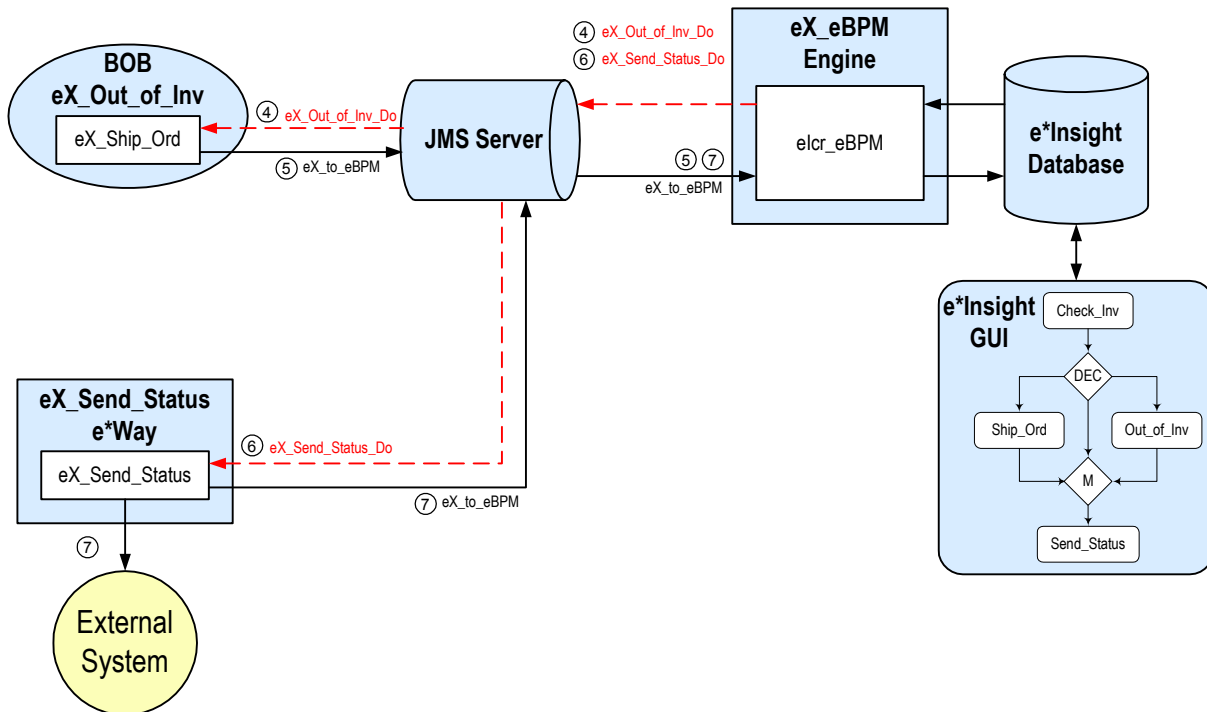
Figure 76 e\*Insight Data Flow Diagram (Part 1)



- 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 77 e\*Insight Data Flow Diagram (Part 2)



- ④ Let's assume the items are not in stock. The e\*Insight engine publishes a "Do" Event (**eX\_Out\_of\_Inv\_Do**) corresponding to the **Out\_of\_Inv** activity in the business process. The **eX\_Out\_of\_Inv** BOB retrieves this Event from the **eX\_eBPM** IQ and uses the information it contains to ship the order to the customer.
- ⑤ When the **Ship\_Ord** activity is finished, the **eX\_Out\_of\_Inv** BOB publishes a "Done" Event to the **eX\_eBPM** IQ indicating that the order has been shipped. The e\*Insight engine retrieves this Event, updates the BPI, and then moves forward to the next activity in the business process – **Send\_Status**.
- ⑥ The e\*Insight engine publishes a "Do" Event (**eX\_Send\_Status\_Do**) corresponding to the **Send\_Status** activity in the business process. The **eX\_Send\_Status** e\*Way retrieves this Event from the **eX\_eBPM** IQ and uses the information it contains to send a order status report to the customer.
- ⑦ The **eX\_Send\_Status** e\*Way publishes two Events: one containing the status report to be sent to the customer to the external system responsible for sending out the report, and also the "Done" Event to the **eX\_eBPM** IQ. The e\*Insight engine retrieves the "Done" Event from the **eX\_eBPM** IQ and uses the information it contains to update the BPI to indicate that the final activity in the business process has completed successfully.

## 10.2 Create the ProcessOrder BP in e\*Insight

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

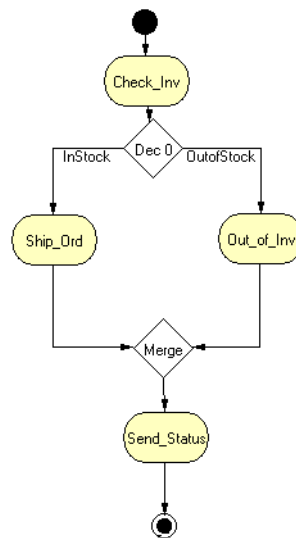
- 1 Create a Business Process named ProcessOrder.
- 1 Add the activities.
- 2 Add the decision gates.
- 3 Make the connections between the activities and gates.
- 4 Add all the global attributes.
- 5 Assign global attributes to activities.
- 6 Add the logic to the decision gates.
- 7 Configure the properties for the activities.

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

Use the diagram shown in Figure 78 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 78** ProcessOrder Business Process Model



**Table 24** BP Global Attributes

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

**Table 25** Activity Attributes

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



**Table 26** Decision Gates

Feeding Activity	Expression
Check_Inv	In_Stock==true

### 10.2.1 Creating the processes performing the Activities

The activities in our scenario can either be performed by e\*Gate or an e\*Insight Script. Three of the activities (Check\_Inv, Out\_of\_Inv, and Send\_Status) use e\*Gate. This is described in [“Configure the Integration Schema” on page 154](#).

The Ship\_Ord activity is performed by an e\*Insight Script. This is described below.

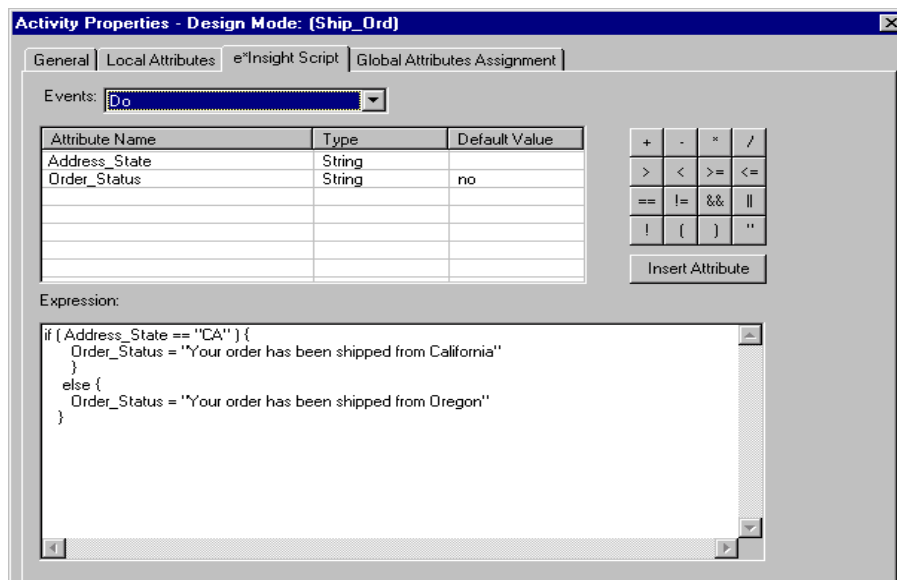
### Configuring the e\*Insight Script for Ship\_Ord

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 79.

**Figure 79** Ship\_Order e\*Insight Script Tab



## 10.3 Configure the Integration Schema

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

### To create a copy of eISchema

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

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

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

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

### Integration Schema Activity Components Summary

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

**Table 27** Integration Schema Activity Components

Name	Type	Participating Host	Configuration Instructions
eX_Check_Inv	BOB	localhost	<a href="#">“Creating the eX_Check_Inv BOB” on page 155</a>
eX_Out_of_Inv	BOB	localhost	<a href="#">“Creating the eX_Out_of_Inv BOB” on page 158</a>
eX_Send_Status	e*Way	localhost	<a href="#">“Send_Status e*Way Configuration” on page 161</a>

For information on how to use the e\*Insight GUI to configure the e\*Gate Registry see the *e\*Insight Business Process Manager 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 and 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.

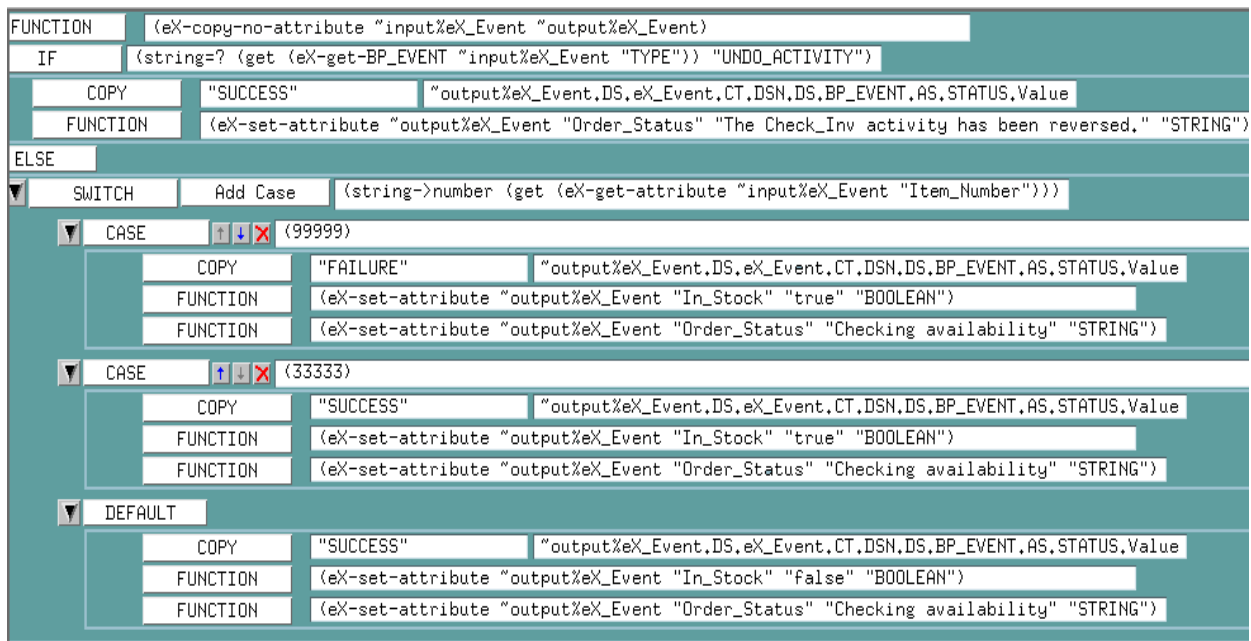
Any other Item\_Number is treated as being out of stock and the CRS set the value of In\_Stock to "no" and sends "SUCCESS".

### 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 80, on the following page, shows the eX\_Check\_Inv.tsc CRS used in this example.

**Figure 80** eX\_Check\_Inv.tsc CRS (Monk)

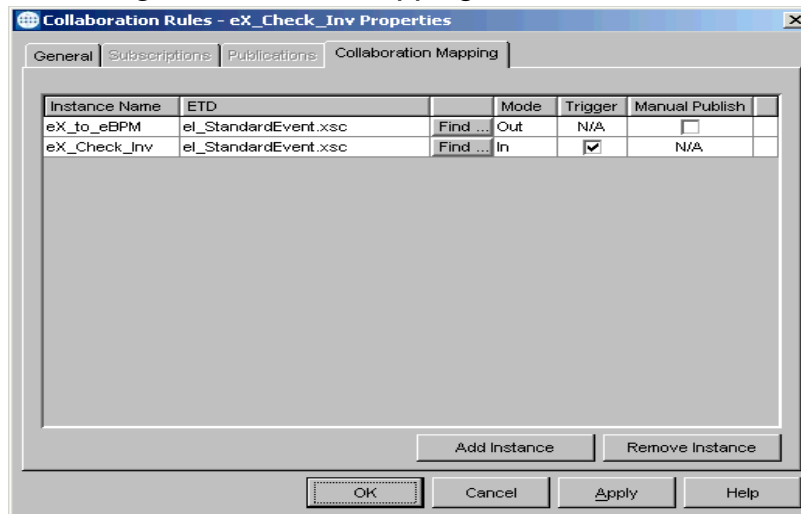


- 7 Validate and save the CRS.
- 8 Close the editor.
- 9 In the **Check\_Inv** activity properties, click **Configure e\*Gate Schema**.  
You may be required to log into e\*Gate.
- 10 Click **OK**, to close the information dialog.
- 11 Close the **Check\_Inv** Activity properties.

### 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 81.

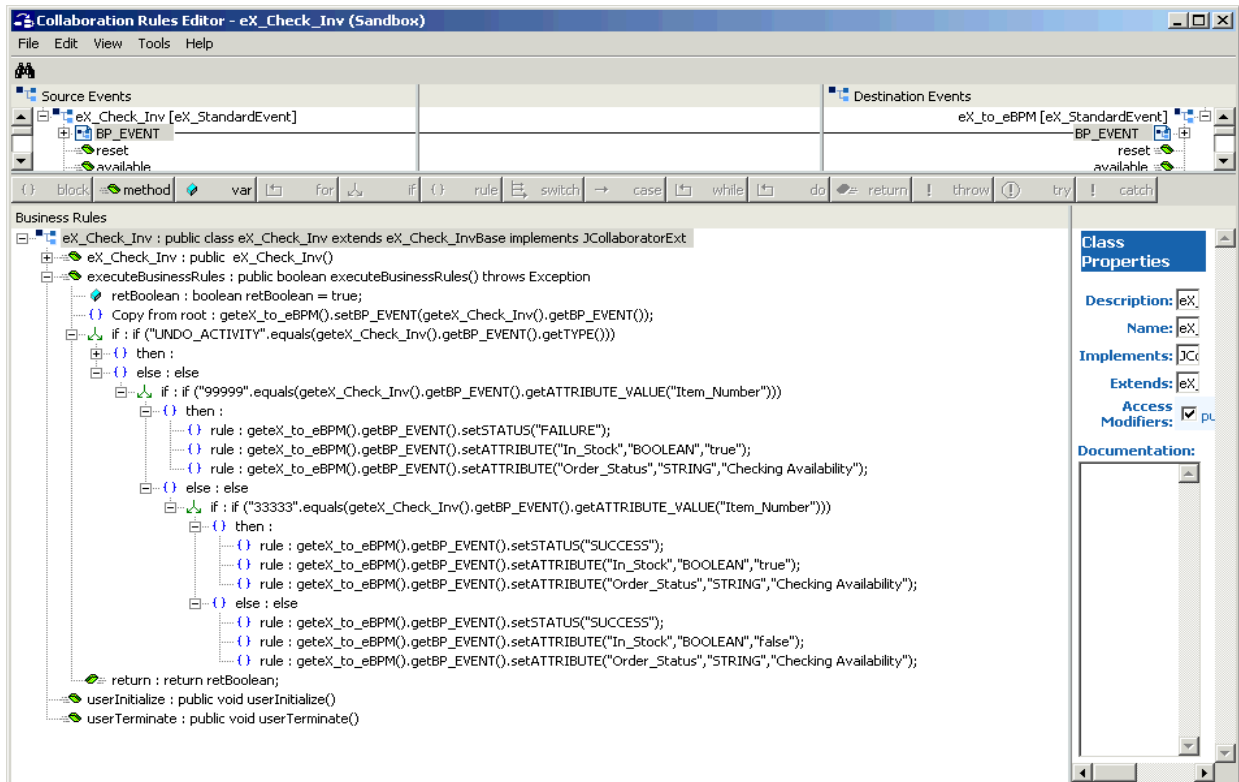
**Figure 81** Define Mapping for eX\_Check\_Inv



- 7 Click **OK**.
- 8 Create eX\_Check\_Inv.xsc.

Figure 82, on the following page, shows the **eX\_Check\_Inv.xsc** CRS used in this example.

Figure 82 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.

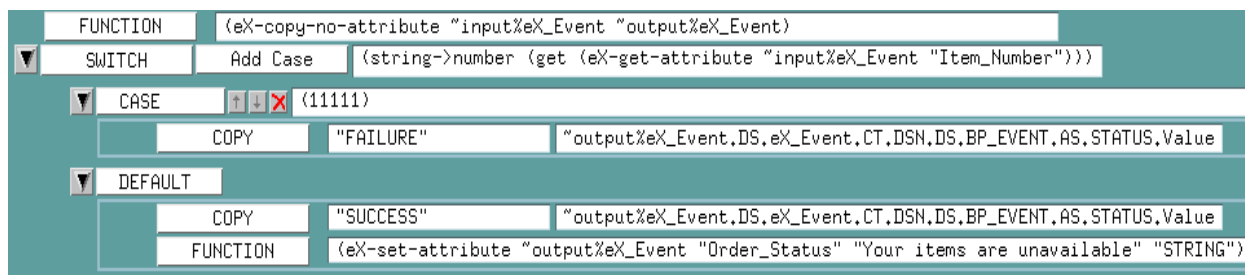
In addition, this translation demonstrates how e\*Insight handles “undoing” a partially completed business process. If Item\_Number 11111 is encountered, then “FAILURE” is returned to the e\*Insight engine which in turn issues “undo” Events for any activities upstream from the failed activity. In this 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 83 shows the eX\_Out\_of\_Inv CRS used in this example.

**Figure 83** eX\_Out\_of\_Inv.tsc CRS (Monk)

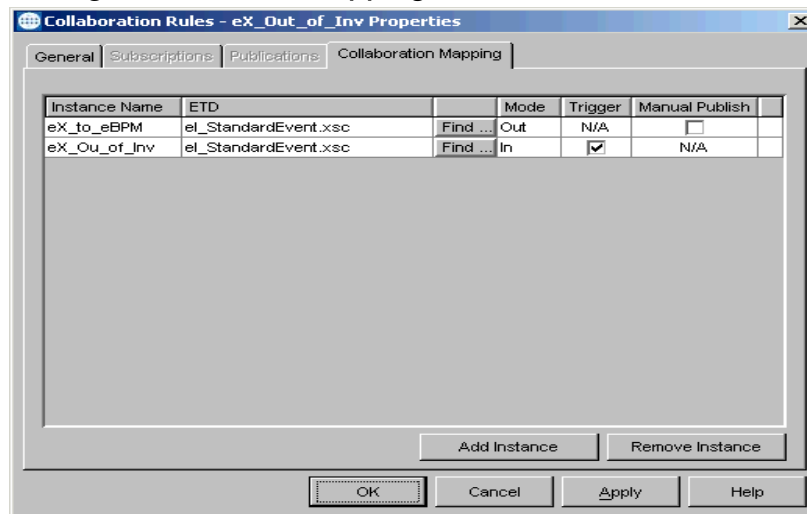


- 7 Validate and save the CRS.
- 8 Close the editor.
- 9 In the **Out\_of\_Inv** Activity properties, click **Configure e\*Gate Schema**.  
You may be required to log into e\*Gate.
- 10 Click **OK**, to close the information dialog.
- 11 Close the **Out\_of\_Inv** Activity properties.

### 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 84.

**Figure 84** Define Mapping for eX\_Out\_of\_Inv (Java)

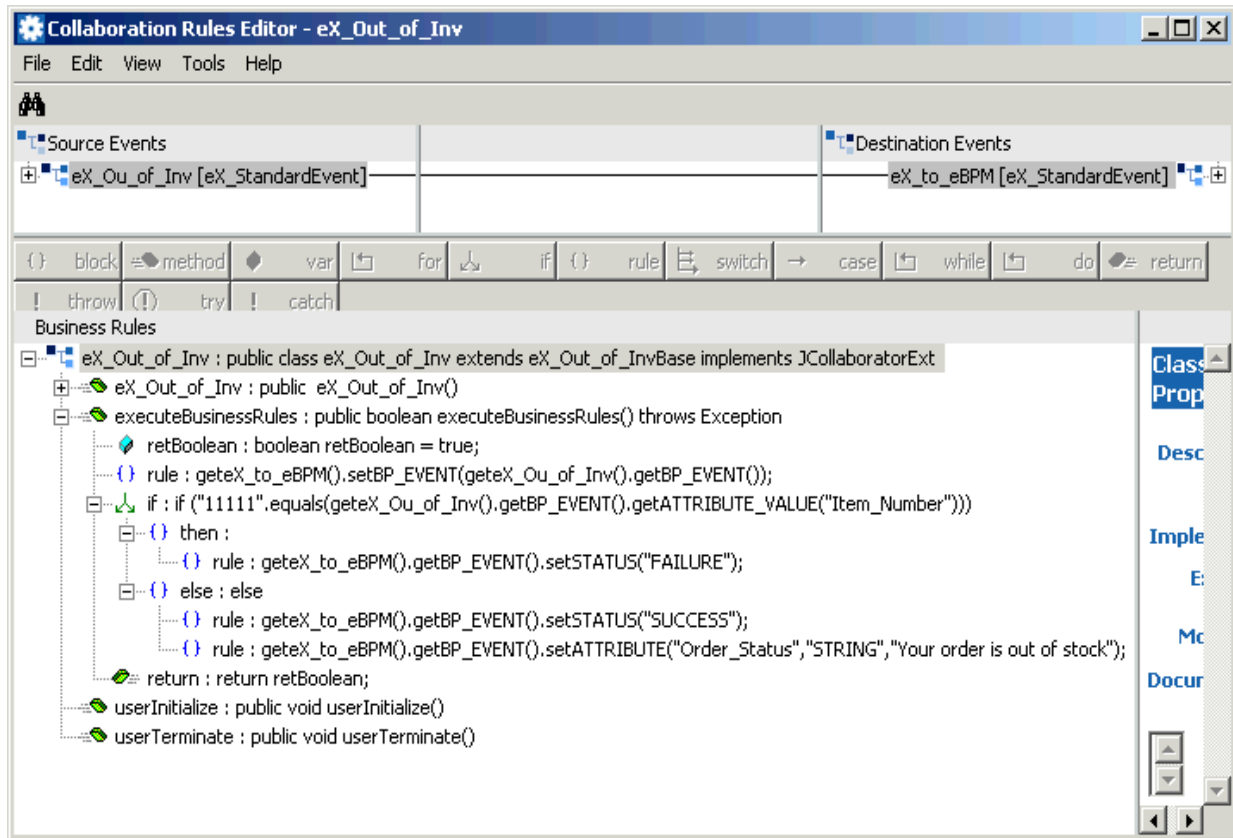


- 7 Click **OK**.
- 8 Create eX\_Out\_of\_Inv.tsc.

Figure 85 shows the eX\_Out\_of\_Inv CRS used in this example.



**Figure 85** 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 170](#). The e\*Way and basic components should be created from within the e\*Insight GUI.

To create the Send\_Status Activity e\*Way

- 1 In the e\*Insight GUI, open the **Send\_Status** activity properties.
- 2 On the **General** tab, select the **e\*Way** e\*Gate module.
- 3 Click **Configure e\*Gate Schema**.  
You may be required to log into e\*Gate.
- 4 Click **OK**, to close the information dialog.

- 5 Close the **Send\_Status** Activity properties.

## 10.4 Configure the e\*Insight Engine

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

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

### Edit the eX\_eBPM Configuration File

Most of the parameter settings in the eX\_eBPM engine’s configuration file should not be changed. Table 28 discusses the parameters that may need to be changed depending on the implementation. Use the e\*Way Editor and the information in **“Configuring the e\*Insight Engine” on page 66** to make the required changes for the ProcessOrder example.

## 10.5 Configure User-defined e\*Gate Components

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

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

### Configuration Order for the User-defined Components

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

**Table 28** Configuration Order for User-defined Components

	Task	Section
1	Add and configure the START_BP e*Way	<b>“Configure the START_BP e*Way” on page 163</b>
2	Configure the Send_Status e*Way	<b>“Configure the Send_Status e*Way” on page 170</b>

**Important:** *All the integration schema associations are displayed in table format at the end of this section. The sections dealing with e\*Way configuration include tables detailing*

*the non-default e\*Way parameter settings. The sections dealing with the Monk and Java Collaboration Rules Scripts show screen shots of these scripts as they appear in the e\*Gate Collaboration Editor.*

### 10.5.1 Configure the START\_BP e\*Way

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

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

Table 29 shows the steps to configure the **START\_BP** e\*Way.

**Table 29** Configuration steps for the START\_BP e\*Way

	Step	Section	
		Monk	Java
1	Add the e*Way and create the e*Way configuration file	<a href="#">“Step 1: Create the START_BP e*Way using Monk” on page 163</a>	<a href="#">“Step 1: Create the START_BP e*Way using Java” on page 166</a>
2	Create the Input ETD	<a href="#">“Step 2: Create the Input ETD using Monk” on page 164</a>	<a href="#">“Step 2: Create the Input ETD using Java” on page 166</a>
3	Create the START_BP Collaboration Rules script (CRS)	<a href="#">“Step 3: Create the START_BP CRS using Monk” on page 164</a>	<a href="#">“Step 3: Create the START_BP Collaboration using Java” on page 167</a>
4	Configure the Collaboration in the GUI	<a href="#">“Step 4: Configure the START_BP Collaboration in the GUI using Monk” on page 165</a>	<a href="#">“Step 4: Configure the Collaboration in the GUI using Java” on page 169</a>

#### Step 1: Create the START\_BP e\*Way using Monk

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

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

**Table 30** Start e\*Way Parameters

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

### Step 2: Create the Input ETD using Monk

The input ETD is based on the format of the input data. The ProcessOrder example uses a delimited text file (**InStock.~in**) that contains the data needed to process the order.

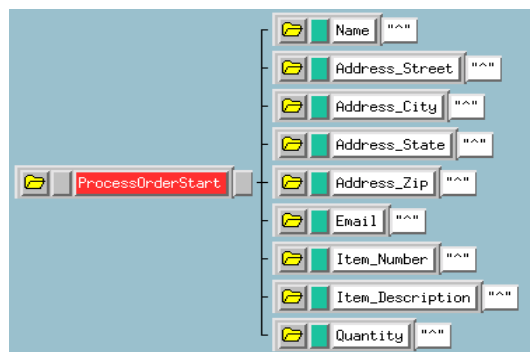
The input data file used in this example is shown in Figure 86. Place this data file at the directory location **<eGate>\client\data\ProcessOrder**.

**Figure 86** 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 87. For more information on using the ETD Editor see the ETD Editor’s online help.

**Figure 87** 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 88, shows the **START\_BP** CRS used in the ProcessOrder example:

**Figure 88** START\_BP CRS (Monk)

COPY	"ProcessOrder"	"output%eX_Event.DS,eX_Event.CT,DSN,DS,BP_EVENT.AS.NAME.Value
COPY	"START_BP"	"output%eX_Event.DS,eX_Event.CT,DSN,DS,BP_EVENT.AS.TYPE.Value
UNIQUE ID	"output%eX_Event.DS,eX_Event.CT,DSN,DS,BP_EVENT.AS.ID.Value	
FUNCTION	(eX-set-attribute "output%eX_Event "Address_Street" "input%ProcessOrderStart.Address_Street "STRING")	
FUNCTION	(eX-set-attribute "output%eX_Event "Address_City" "input%ProcessOrderStart.Address_City "STRING")	
FUNCTION	(eX-set-attribute "output%eX_Event "Address_State" "input%ProcessOrderStart.Address_State "STRING")	
FUNCTION	(eX-set-attribute "output%eX_Event "Address_Zip" "input%ProcessOrderStart.Address_Zip "STRING")	
FUNCTION	(eX-set-attribute "output%eX_Event "Cust_Name" "input%ProcessOrderStart.Name "STRING")	
FUNCTION	(eX-set-attribute "output%eX_Event "Cust_email" "input%ProcessOrderStart.Email "STRING")	
FUNCTION	(eX-set-attribute "output%eX_Event "Item_Description" "input%ProcessOrderStart.Item_Description "STRING")	
FUNCTION	(eX-set-attribute "output%eX_Event "Item_Number" "input%ProcessOrderStart.Item_Number "STRING")	
FUNCTION	(eX-set-attribute "output%eX_Event "Order_Quantity" "input%ProcessOrderStart.Quantity "NUMBER")	

### Step 4: Configure the START\_BP Collaboration in the GUI using Monk

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

#### To configure the Collaboration

- 1 Create a Collaboration Rule, **START\_BP**, that uses the Monk service and the **START\_BP** CRS created in step 2, subscribes to the **eX\_External\_Evt** Event Type, and publishes to the **eX\_to\_eBPM** Event Type.
- 2 Create a Collaboration for the **START\_BP** e\*Way that uses the **START\_BP** Collaboration Rule, subscribes to the **eX\_External\_Evt** Event Type from **<EXTERNAL>**, and publishes the **eX\_to\_eBPM** Event Type to the **eX\_eBPM** IQ.

## Step 1: Create the START\_BP e\*Way using Java

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

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

**Table 31** Start e\*Way Parameters

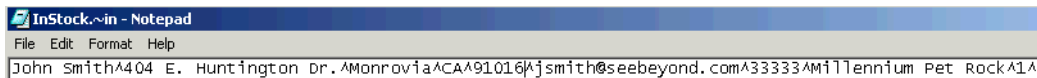
Screen	Parameter	Setting
General Settings	(All)	(Default)
Outbound (send) settings	(All)	(Default)
Poller (inbound) settings	PollDirectory	<eGate>\client\data\ProcessOrder
	(All others)	(Default)
Performance Testing	(All)	(Default)

## Step 2: Create the Input ETD using Java

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

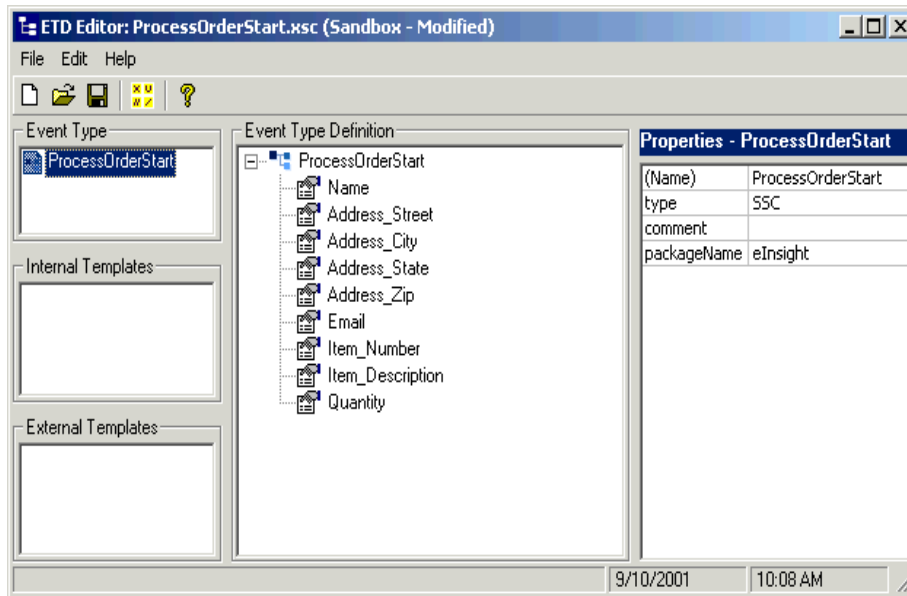
The input data file used in this example is shown in Figure 89. Place this data file at the directory location **c:\eGate\client\data\ProcessOrder**.

**Figure 89** 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 90. Set the global delimiter to a ^ character. For more information on using the ETD Editor see the ETD Editor’s online help.

**Figure 90** Input ETD: ProcessOrderStart.xsc (Java)



### Step 3: Create the START\_BP Collaboration using Java

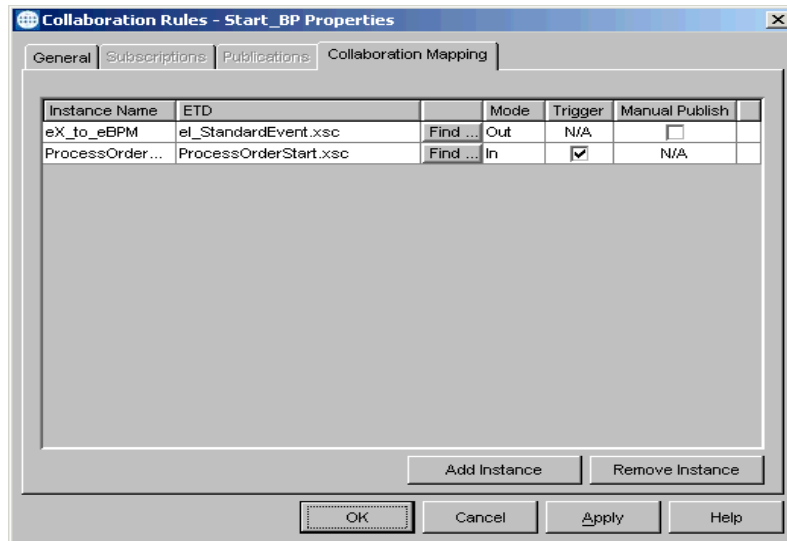
The Collaboration that sends the Event that starts the BPI must do two things:

- 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.

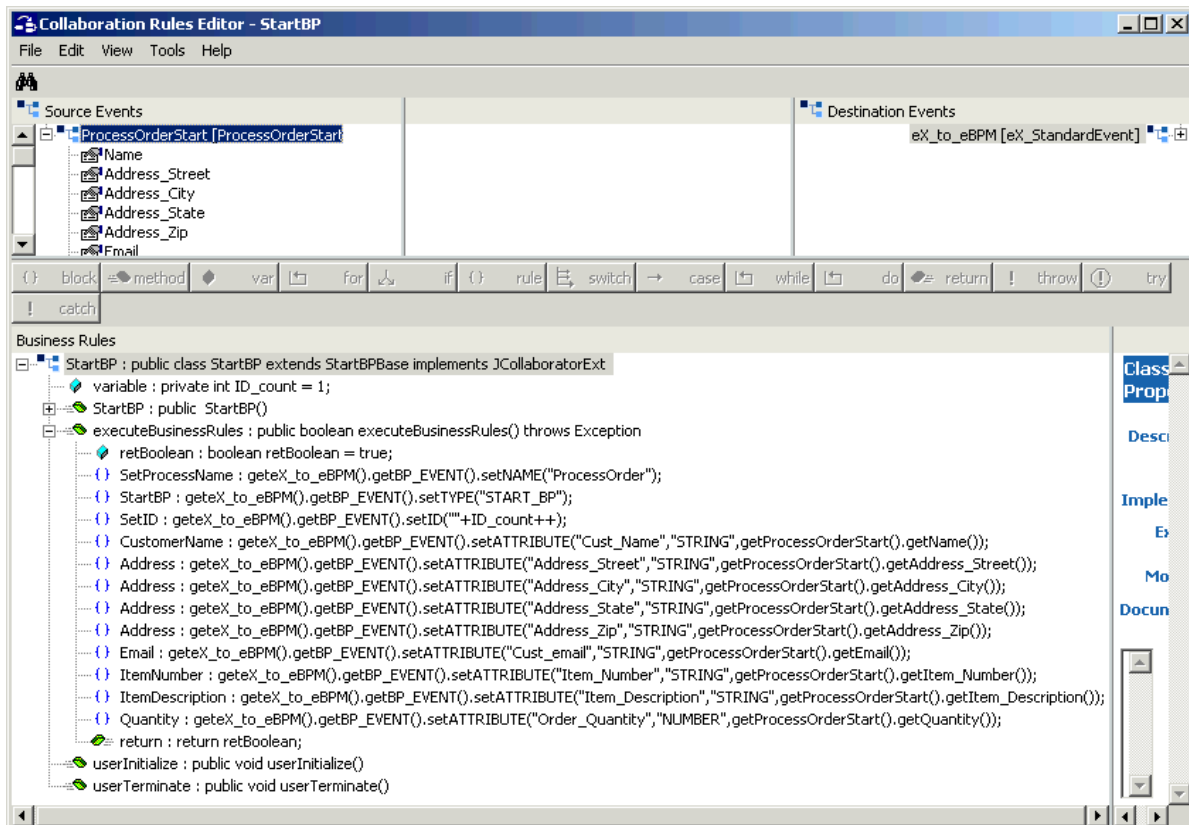
- 1 Create a Collaboration Rule, **START\_BP**, that uses the Java service.
- 2 Configure the Collaboration Mapping tab, as shown in Figure 91.

**Figure 91** 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 92.

**Figure 92** START\_BP CRS (Java)



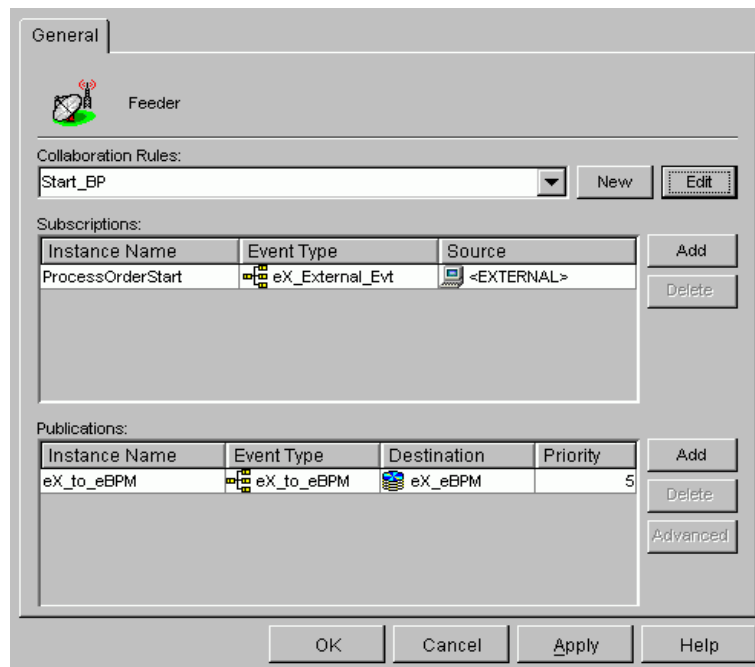


## Step 4: Configure the Collaboration in the GUI using Java

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 Enterprise Manager GUI.

- 1 Create a Collaboration for the **START\_BP** e\*Way configured as shown in Figure 93.

**Figure 93** START\_BP Collaboration



## 10.5.2 Configure the Send\_Status e\*Way

The last component that must be configured in the ProcessOrder example is the **Send\_Status** e\*Way.

This e\*Way must accomplish two tasks:

- Create a file containing the text of e-mail message that can be sent to an order-status message via e-mail (simulated; no actual mail is sent)
- Return "SUCCESS" to the e\*Insight engine

This e\*Way simulates sending an e-mail order status message by writing the customer's e-mail address and a short status message to a text file. When this is successful, an Event is returned to the e\*Insight engine with the status node set to "SUCCESS".

Table 32 shows the steps to configure the Send\_Status e\*Way.

**Table 32** Send\_Status e\*Way configuration steps

	Step	Section	
		Monk	Java
1	Find the executable and create the e*Way configuration file	<a href="#">"Step 1: Configure the eX_Send_Status e*Way using Monk" on page 170</a>	<a href="#">"Step 1: Configure the e*Way using Java" on page 173</a>
2	Create the Output ETD	<a href="#">"Step 2: Create the Output ETD using Monk" on page 171</a>	<a href="#">"Step 2: Create the Output ETD: SendStatus.xsc using Java" on page 173</a>
3	Create the eX_Send_Status.tsc CRS	<a href="#">"Step 3: Create the eX_Send_Status.tsc CRS using Monk" on page 171</a>	<a href="#">"Step 3: Create the Send_Status Collaboration Rule using Java" on page 174</a>
4	Configure the Collaboration in the GUI	<a href="#">"Step 4: Configure the Collaboration using Monk" on page 172</a>	<a href="#">"Step 4: Configure the Collaboration using Java" on page 175</a>

### Step 1: Configure the eX\_Send\_Status e\*Way using Monk

First find the executable, then create the configuration file.

The **eX\_Send\_Status** e\*Way is a simple file e\*Way (**stcewfile.exe**) that writes a text file (**ProcessOrder\_output%d.dat**) to the directory `<egate>\client\data\ProcessOrder`.

The file created contains the e-mail address of the person who placed the order, along with the status of the order. Use the following table to set the e\*Way parameters in the configuration file:

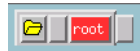
**Table 33** Send\_Status e\*Way Parameters

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

### Step 2: Create the Output ETD using Monk

Use the e\*Gate ETD Editor to create a single node ETD like that shown in Figure 94.

**Figure 94** 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 95 accomplishes these tasks. The source ETD is eX\_Standard\_Event.ssc and the destination ETD is root.ssc.

**Figure 95** eX\_Send\_Status.tsc CRS (Monk)

```

COPY      (string-append (get (eX-get-attribute "input%eX_Event "Cust_email")) "^" (get
IF        (eX-set-BP_EVENT "input%eX_Event "STATUS" "SUCCESS"))
FUNCTION  (iq-put "eX_to_eBPM" (get "input%eX_Event") (list "eX_Send_Status") 0 0 0)
ELSE
DISPLAY   ("Failed to set SUCCESS")
newline

(get (eX-get-attribute "input%eX_Event "Order_Status")) "output%root
0 0 0)

```

## Step 4: Configure the Collaboration using Monk

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

### To configure the collaboration

- 1 Highlight the **eX\_Send\_Status** e\*Way.
- 2 Delete the two Collaborations **eX\_to\_Send\_Status** and **eX\_from\_Send\_Status**.
- 3 Add a Collaboration named **eX\_Send\_Status**.
- 4 Highlight the **Collaboration Rules** folder.
- 5 Delete the two Collaboration Rules **eX\_to\_Send\_Status** and **eX\_from\_Send\_Status**.
- 6 Add a Collaboration Rule named **eX\_Send\_Status**.
- 7 Edit the Collaboration Rule.
- 8 In the **Collaboration Rules Properties** dialog box, select the **Monk** service.
- 9 Find the CRS **eX\_Send\_Status.tsc** and associate it with the Collaboration Rule.
- 10 On the **Subscriptions** tab, move the **eX\_Send\_Status\_Do** and **eX\_Send\_Status\_Undo** Event Types to the **Selected Input Event Types** box.
- 11 On the **Publications** tab, move the **eX\_External\_Evt** and the **eX\_to\_eBPM** Event Types to the **Selected Output Event Types** box.  
Verify that the **eX\_External\_Evt** Event Type is marked as the default.
- 12 Click **OK** to close the **Collaboration Rules Properties** dialog box.
- 13 Highlight the **eX\_Send\_Status** e\*Way and edit the **eX\_Send\_Status** Collaboration you associated with it in step 3.
- 14 In the **Collaboration Properties** dialog box, select the **eX\_Send\_Status** Collaboration Rule.
- 15 Under **Subscriptions** add the **eX\_Send\_Status\_Do** and **eX\_Send\_Status\_Undo** Event Types from the **eX\_from\_eBPM** source.
- 16 Under **Publications** add the Event Type **eX\_External\_Evt** with destination **<EXTERNAL>** and Event Type **eX\_to\_eBPM** with destination **eX\_eBPM IQ**.

## Step 1: Configure the e\*Way using Java

First find the executable, then create the configuration file.

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

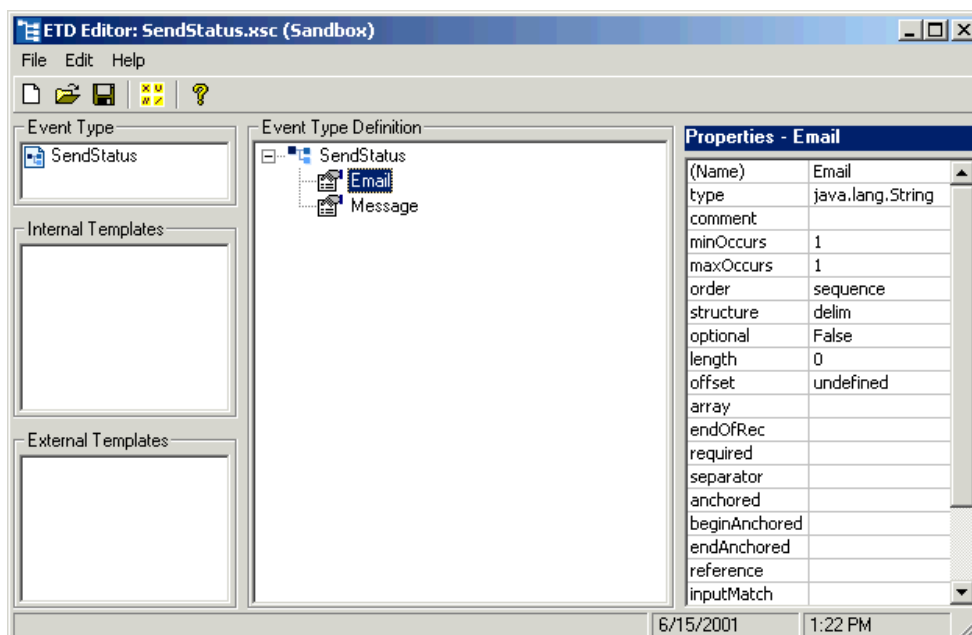
**Table 34** Send\_Status e\*Way Parameters

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

## Step 2: Create the Output ETD: SendStatus.xsc using Java

Use the e\*Gate ETD Editor to create an ETD like that shown in Figure 96.

**Figure 96** SendStatus.xsc ETD (Java)



### Step 3: Create the Send\_Status Collaboration Rule using Java

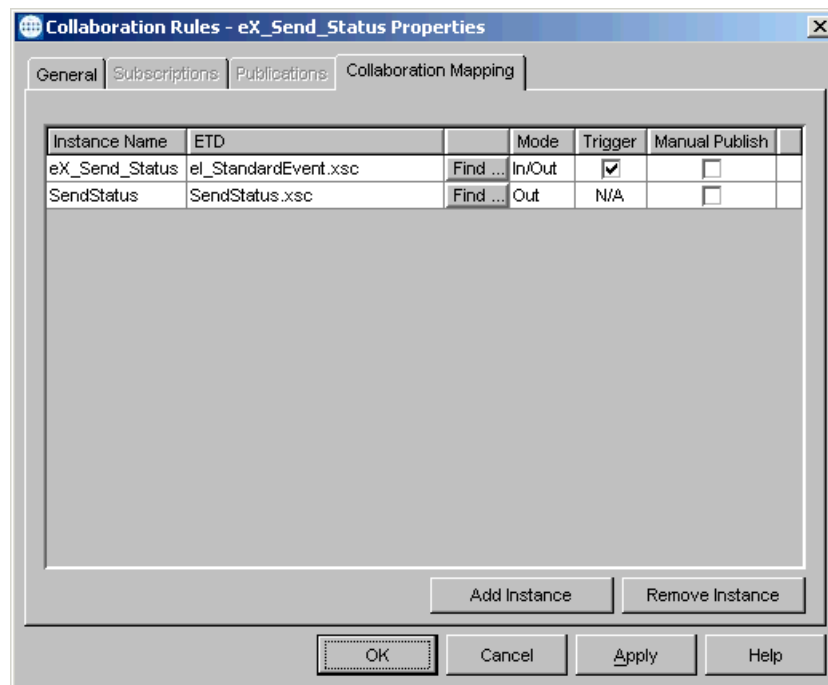
This CRS must accomplish three things:

- 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

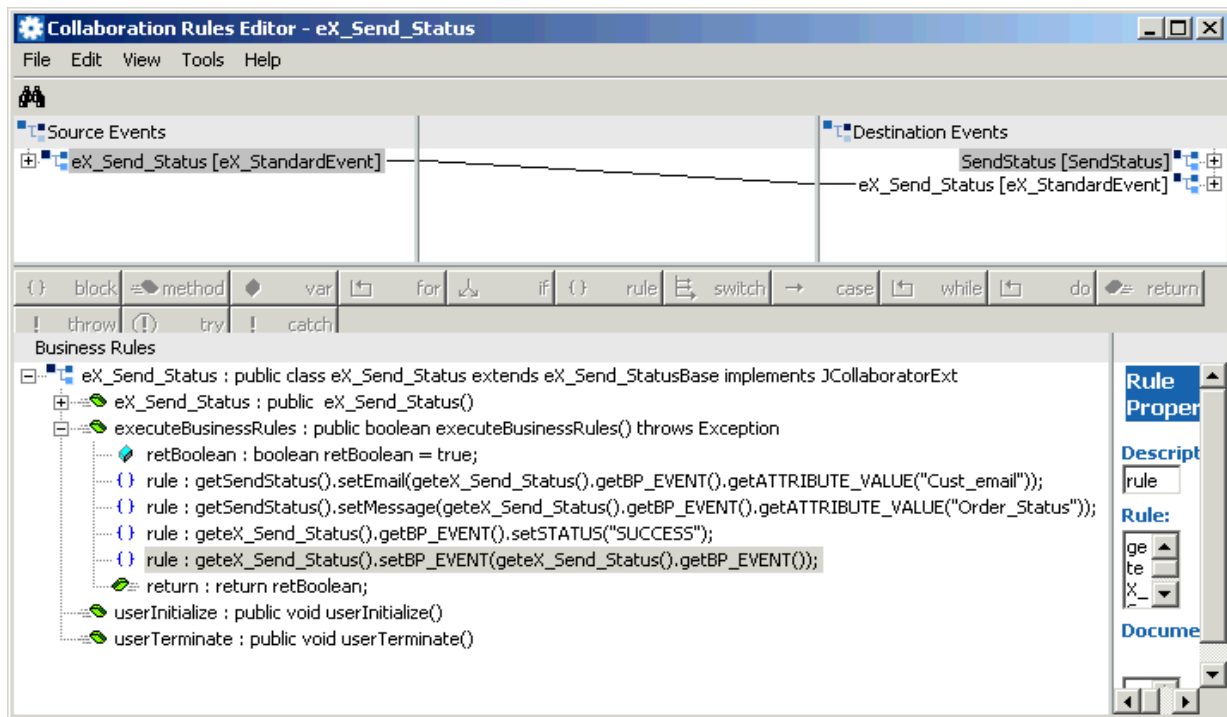
- 1 From the eX\_Send\_Status Collaboration Rule General tab, select the Java Collaboration Service.
- 2 From the eX\_Send\_Status Collaboration Rule Collaboration Mapping tab, create two new instances as shown in Figure 97.

**Figure 97** 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 98.

**Figure 98** eX\_Send\_Status.xsc CRS (Java)



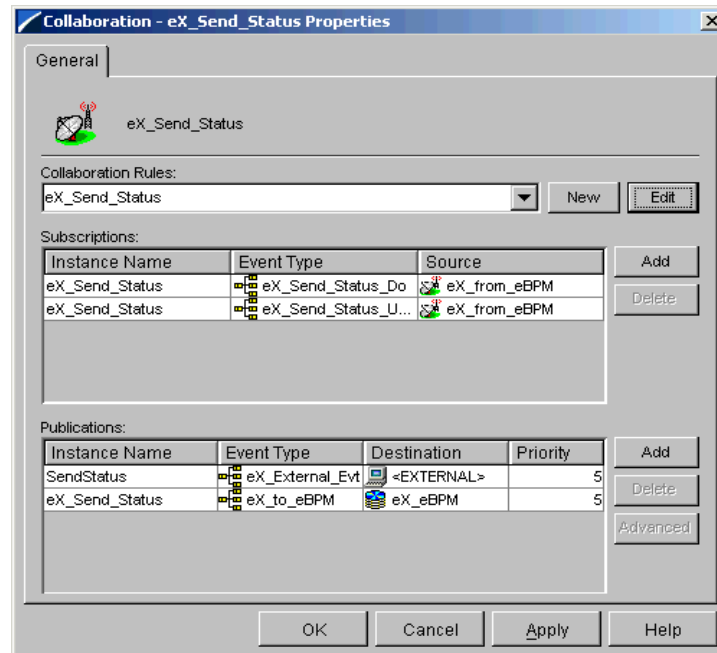
### Step 4: Configure the Collaboration using Java

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 99.

**Figure 99** eX\_Send\_Status Collaboration (Java)



## 10.6 Run and Test the e\*Insight scenario

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

### 10.6.1 Testing the Standard Business Logic

The following procedure tests the standard business logic of the e\*Insight ProcessOrder case study example. That logic is as follows: a check is made to see whether or not the item ordered is available. If it is in stock the Ship\_Order activity is invoked and a message is generated that can be sent to the customer indicating that his order has been shipped to him. If the item is unavailable, then the Out\_of\_Inv activity is invoked which creates a message informing the customer that his item is unavailable.

The test is made by sending in data with different item numbers and verifying the correct processing. Input data with an item number of 33333 is interpreted as being in stock and any other number except for the three special numbers (11111, 22222, and 99999) is interpreted as being out of stock.

### In-Stock Processing

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

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



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

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

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

Substitute the appropriate username and password for your installation.

- 4 Start the e\*Gate Monitor, and check the status of all the components. Any components displayed in red are not running. Any e\*Insight components that are not running should be investigated before feeding data into the system.
- 5 Navigate to the location for the input data file, **InStock.~in**, shown in **Figure 86 on page 164** (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.

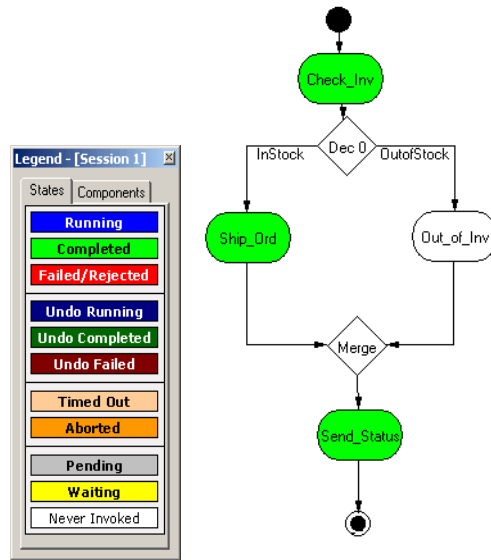
If everything is working correctly, an output file (**ProcessOrder\_output#.dat**) as appears in the directory indicating successful completion of the BPI.

- 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 100 illustrates how the successfully completed BPI appears in the e\*Insight GUI.

Figure 100 In Stock Completed BPI Diagram

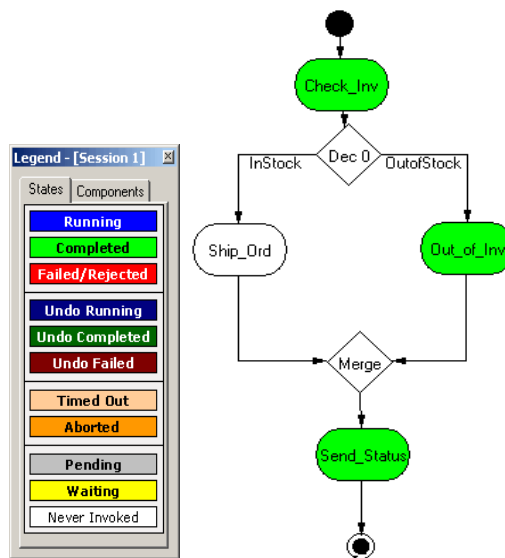


## Out-of-Stock Processing

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 101.

Figure 101 Out of Stock Completed BPI Diagram



## 10.6.2 Demonstrating Business Process Undo Functionality

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

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

If an activity fails and **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

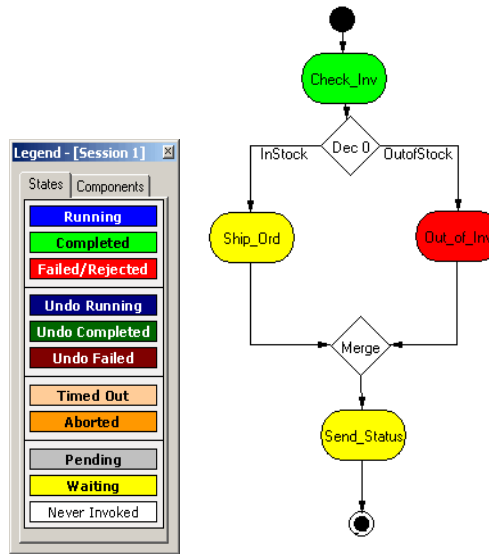
Use the following procedure to test the functionality of manual undo in the e\*Insight scenario.

- 1 Perform steps 1 through 4 outlined in **“In-Stock Processing” on page 176**.
- 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 102 on the next page.

**Figure 102** 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.

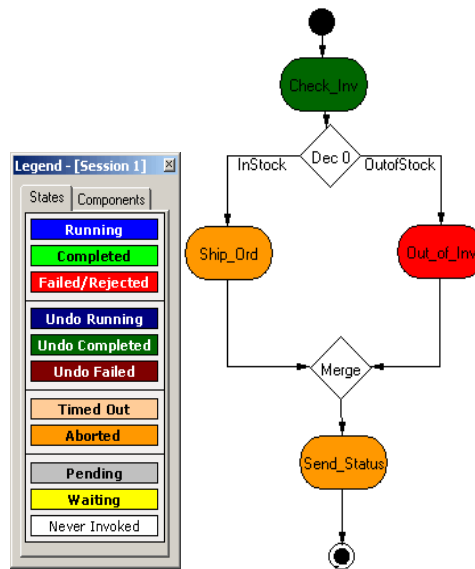
- 5 Right-click the **Out\_of\_Inv** activity from the tree view, then select **Properties** from the popup menu.

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 103** Manual Undo Completed BPI Diagram



### 10.6.3 Demonstrating Business Process Restart Functionality

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

#### Repairing a String Attribute

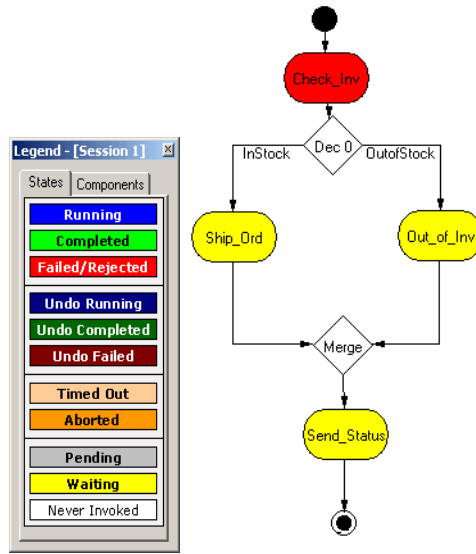
Attributes can be of various types; Boolean, number, string, and XML. The following example shows the procedure to repair an attribute of type string. For information on repairing an attribute with type XML, see the *e\*Insight Business Process Manager User's Guide*.

- 1 Perform steps 1 through 4 outlined in **"In-Stock Processing"** on page 176.
- 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.

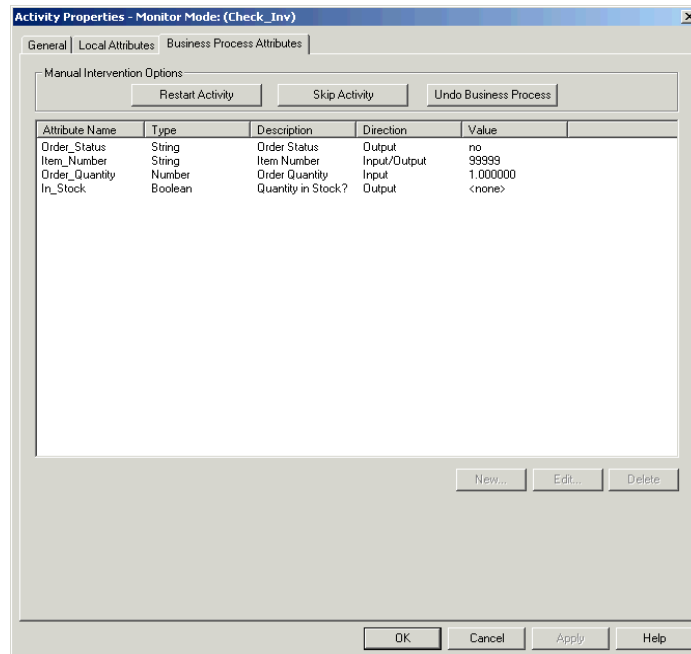
**Figure 104** Attribute Repair—Failed BPI Diagram



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.

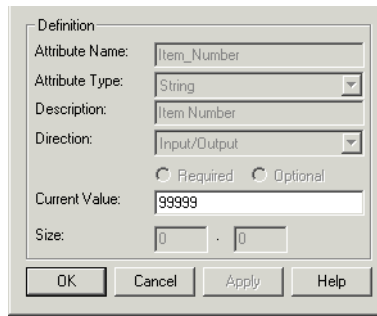
**Figure 105** Attribute Repair—Business Process Attributes tab



- 6 Highlight the **Item\_Number** attribute line, and then click **Edit**.

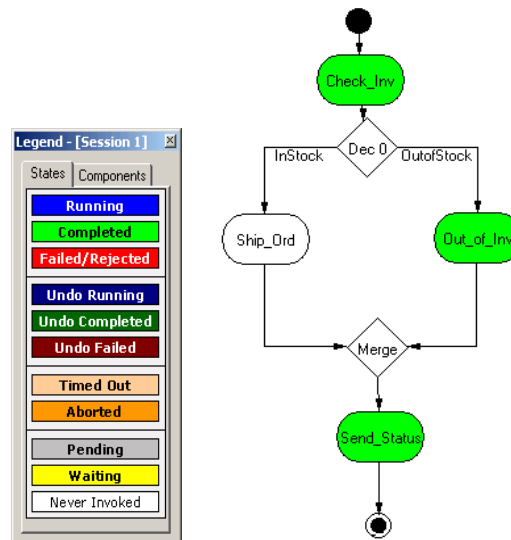
The **Edit Business Process Attribute** dialog box is displayed, as shown in Figure 106.

**Figure 106** 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 107 on the next page.

**Figure 107** 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.

# 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 147](#) for the initial configuration instructions.

---

## 11.1 Overview

The major steps in the implementation are:

- 1 Create and configure the Decision gate and Authorization Activity in the e\*Insight GUI.
- 2 Use the e\*Insight GUI to configure the e\*Gate schema that supports e\*Insight.
- 3 Add and configure the user-defined e\*Gate components.
- 4 Run and test the scenario.

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

### 11.1.1 Case Study: Order Processing

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

Figure 108, 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 148](#).



Figure 108 e\*Insight Data Flow Diagram

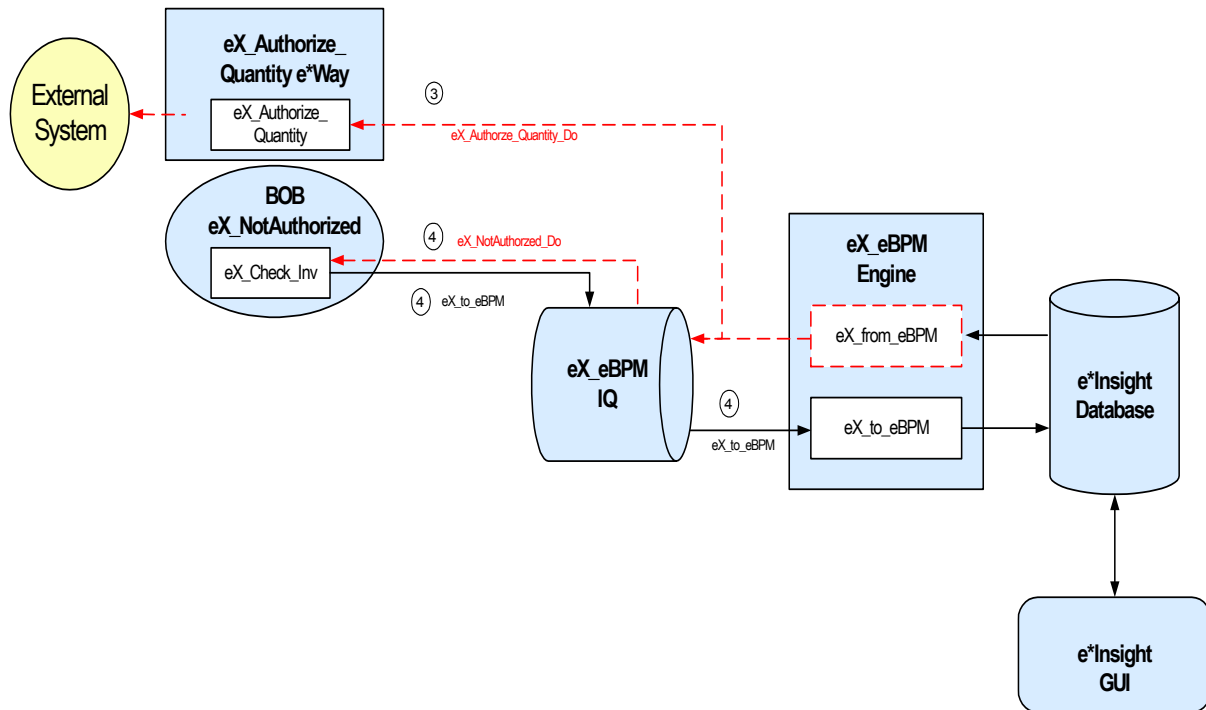


Figure 108 data flow description

- ③ When the **Check\_Inv** activity is finished, the **eX\_Check\_Inv** BOB publishes a “Done” Event using the **eX\_to\_eBPM** Event Type to the **eX\_eBPM IQ**. The e\*Insight engine retrieves the “Done” Event from the IQ, updates the BPI to reflect whether the items ordered are in stock, and then moves forward to the next activity in the business process based on the result of a decision gate. If the items are in stock, the next activity is another decision gate; otherwise the next activity is **Out\_of\_Inv**.

Let’s assume the items are in stock. The BPI moves to the next activity based on the result of the second decision gate. If the quantity is less than 100, the next activity is **Ship\_Ord**; otherwise the next activity is **eX\_Authorize\_Quantity**.

Let’s assume the quantity is greater than or equal to 100. The e\*Insight engine publishes a “Do” Event (**eX\_Authorize\_Quantity\_Do**) corresponding to the **Authorize\_Quantity** authorization activity in the business process. The **eX\_Authorize\_Quantity** e\*Way retrieves this Event from the **eX\_eBPM IQ** and uses the information it contains to retrieve the BPI ID and alert the relevant person that this instance requires authorization.

When the **eX\_Authorize\_Quantity** activity has either been authorized or rejected, the e\*Insight engine moves forward to the next activity in the business process. This is **Not\_Authorized** if the quantity was not authorized; otherwise the next activity is **Ship\_Order**.

- ④ Let's assume the quantity is not authorized. The e\*Insight engine publishes a "Do" Event (**eX\_Not\_Authorized\_Do**) corresponding to the **Not\_Authorized** activity in the business process. The **eX\_Not\_Authorized** BOB retrieves this Event from the **eX\_eBPM** IQ and updates the status. When the **Not\_Authorized** activity is finished, the **eX\_Not\_Authorized** BOB publishes a "Done" Event using the **eX\_to\_eBPM** Event Type to the **eX\_eBPM** IQ. The e\*Insight engine retrieves the "Done" Event from the IQ, updates the BPI to reflect the status, and then moves forward to the **Send\_Status** activity.

Let's assume that either the quantity is authorized, or the item was in stock, but the quantity was less than 100. The e\*Insight engine processes the e\*Insight script corresponding to the **Ship\_Ord** activity in the business process, and then moves forward to the **Send\_Status** activity.

### Important Considerations

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

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

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

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

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

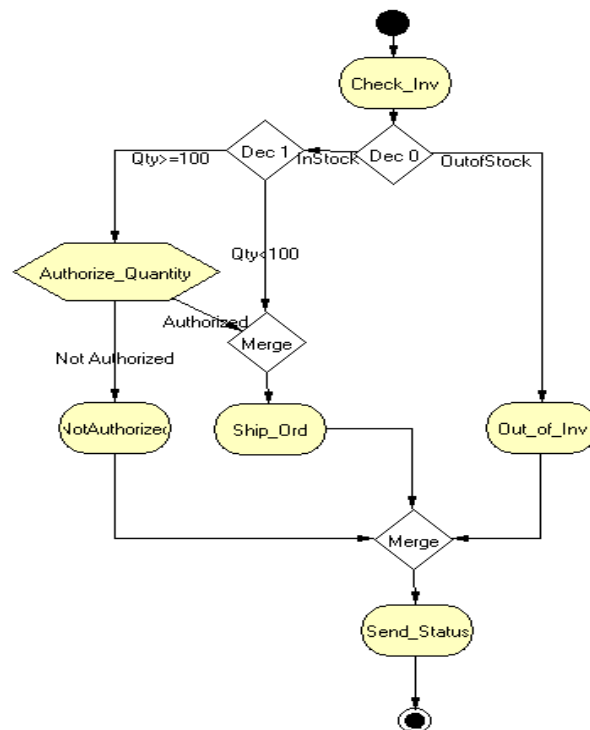
**Note:** The Authorization Activity has two fixed Local Attributes—assignedTo (the user to whom the Authorization process is assigned) and performedBy (automatically assigned at run time as the user logged in to the e\*Insight GUI). These two values must match in order for the user to Authorize, Reject,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

- 3 Add the additional Decision gate.
- 4 Make the connections between the activities and gates.
- 5 Add the logic to the Decision gate.

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

Use the diagram shown in Figure 109 and the following tables to create the BP in e\*Insight.

**Figure 109** ProcessOrder Business Process Model



## 11.3 Step 2: Configure the Integration Schema

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

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

In this example, you use an additional BOB for the **NotAuthorized** Activity, and an additional e\*Way for the **Authorize\_Quantity** Authorization Activity. The **Send\_Status** activity must be associated with an e\*Way because it interfaces with an external component.

### Integration Schema Activity Components Summary

Use the information in Table 35 to configure the e\*Gate schema that supports this example.

**Table 35** Integration Schema Activity Components

Name	Type	Participating Host	Active/Passive	Manual Restart	TimeOut
eX_Authorize_Quantity	e*Way	localhost	Active	Yes	Not used
eX_NotAuthorized	BOB	localhost	Active	Yes	Not used

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

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

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

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

#### Not\_Authorized CRS

The Not\_Authorized translation simulates the activity of sending out an item that is in stock. It sets the value of the Order\_Status attribute to a short message indicating that the order has not been authorized, and returns "SUCCESS" to the e\*Insight engine.

Figure 110, 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 110** Not\_Authorized.tsc CRS

FUNCTION	(eX-copy-no-attribute "input%eX_Event" "output%eX_Event")	
COPY	"SUCCESS"	"output%eX_Event.DS,eX_Event.CT,DSN,DS,BP_EVENT,AS,STATUS,Value"
FUNCTION	(eX-set-attribute "output%eX_Event" "Order_Status" "Your order has not been authorized," "STRING")	

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

Once you have created the CRS for the **eX\_NotAuthorized** BOB, you must associate it with the corresponding Collaboration Rule in the e\*Gate GUI. You must:

- 1 Highlight the BOB's Collaboration.
- 2 Open the **Collaboration Properties** dialog box for the Collaboration.
- 3 Edit the Collaboration Rules.
- 4 Change the Service to **Monk**.
- 5 Find the "Not\_Authorized.tsc" file and associate it with the Collaboration Rule.
- 6 Click **OK** to continue.

### 11.4.1 Configure the Authorize\_Quantity e\*Way

The last component that must be configured in the ProcessOrder example is the **Authorize\_Quantity** e\*Way.

This e\*Way must create a file containing the text of e-mail message that can be sent to advise that an order requires authorization that can be sent via e-mail (simulated; no actual mail is sent)

This e\*Way simulates sending an e-mail order status message simply by writing a short status message to a text file giving the Business Process Instance ID. An Event is not returned to the e\*Insight engine as it is expecting authorization via the e\*Insight GUI in this example.

Follow these steps to configure the Send\_Status e\*Way:

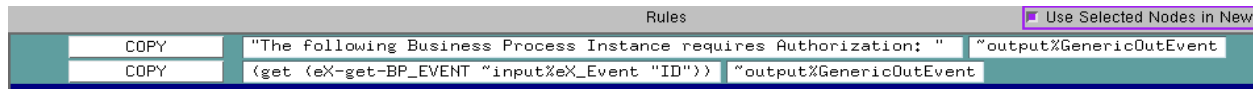
- 1 Create the **Authorize\_Quantity.tsc** CRS.
- 2 Find the executable and create the e\*Way configuration file.
- 3 Configure the Collaboration in the GUI.

## Step 2: Create the Authorize\_Quantity.tsc CRS

The Authorize\_Quantity translation copies a fixed text message and the Business Process Instance ID into the outgoing message. No message is sent back to the e\*Insight engine.

The Authorize\_Quantity.tsc CRS is shown in Figure 111. The source ETD is eX\_Standard\_Event.ssc and the destination ETD is GenericOutEvent.ssc

**Figure 111** 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 36** Send\_Status e\*Way Parameters

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

### Step 4: Configure the Collaboration

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

In the Enterprise Manager:

- 1 Highlight the **eX\_Authorize\_Quantity** e\*Way.
- 2 Delete the two Collaborations **eX\_to\_Authorize\_Quantity** and **eX\_from\_Authorize\_Quantity**.
- 3 Add a Collaboration named **eX\_Authorize\_Quantity**.
- 4 Highlight the **Collaboration Rules** folder.
- 5 Delete the two Collaboration Rules **eX\_to\_Authorize\_Quantity** and **eX\_from\_Authorize\_Quantity**.
- 6 Add a Collaboration Rule named **eX\_Authorize\_Quantity**.
- 7 Edit the Collaboration Rule.
- 8 In the **Collaboration Rules Properties** dialog box, select the **Monk** service.

- 9 Find the CRS **Authorize\_Quantity.tsc** and associate it with the Collaboration Rule.
- 10 On the **Subscriptions** tab, move the **eX\_Authorize\_Quantity\_Do** and **eX\_Authorize\_Quantity\_Undo** Event Types to the **Selected Input Event Types** box.
- 11 On the **Publications** tab, move the **eX\_External\_Evt** to the **Selected Output Event Types** box.
- 12 Click **OK** to close the **Collaboration Rules Properties** dialog box.
- 13 Highlight the **eX\_Authorize\_Quantity** e\*Way and edit the **eX\_Authorize\_Quantity** Collaboration you associated with it in step 3.
- 14 In the **Collaboration Properties** dialog box, select the **eX\_Authorize\_Quantity** Collaboration Rule.
- 15 Under **Subscriptions** add the **eX\_Authorize\_Quantity\_Do** and **eX\_Authorize\_Quantity\_Undo** Event Types from the **eX\_from\_eBPM** source.
- 16 Under **Publications** add the Event Type **eX\_External\_Evt** with destination **<EXTERNAL>**.

---

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

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

### 11.5.1 Testing the Standard Business Logic

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

#### Authorized Processing

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

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

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

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

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

Substitute the appropriate username and password for your installation.

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

**Note:** The change of the extension to “.~in” indicates that the data file has been picked up by the **START\_BP** e\*Way.

If everything is working correctly, an output file (**output#.dat**) as shown in Figure 112 appears in the directory indicating that an order requires authorization.

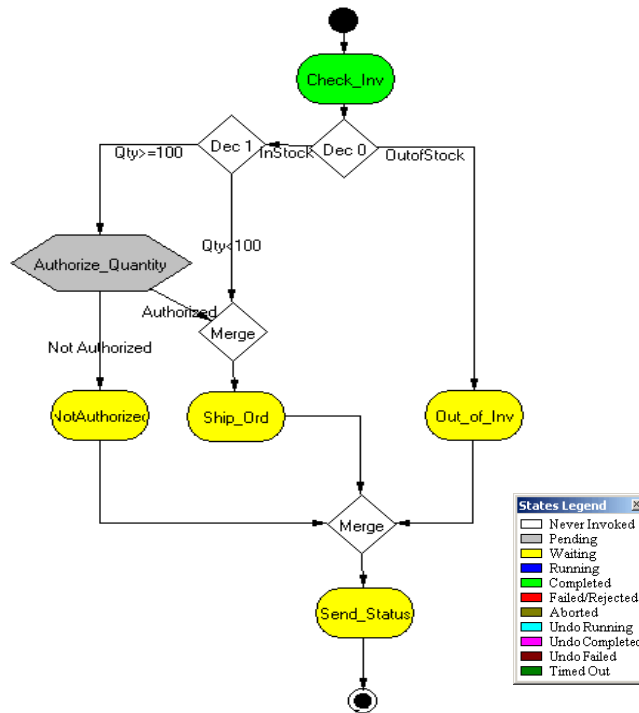
**Figure 112** In Stock Output File



- 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.

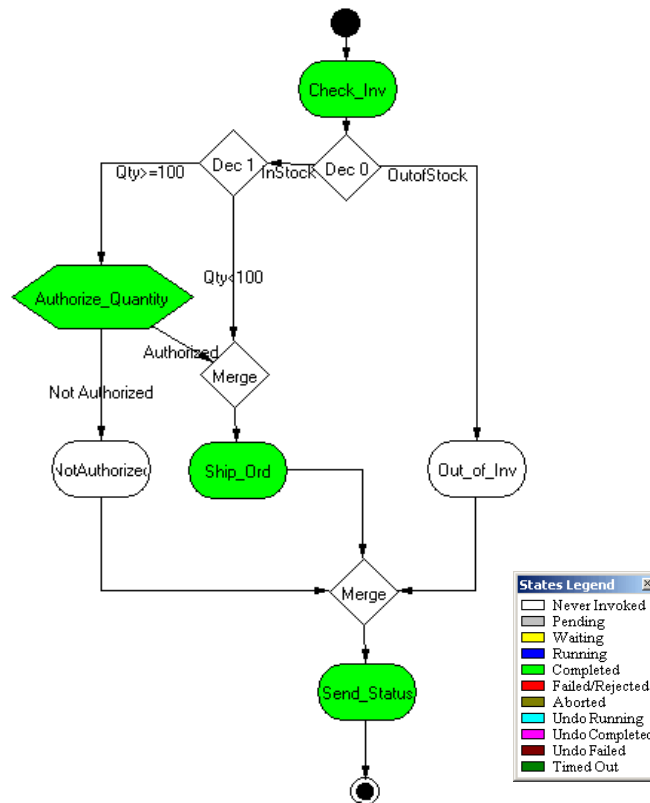
**Figure 113** Authorize\_Quantity Pending BPI Diagram





- 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 114.

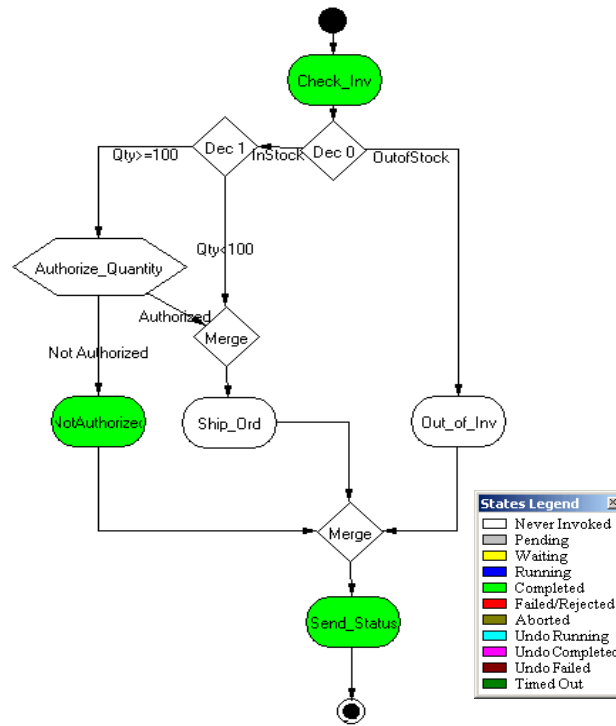
**Figure 114** Authorization\_Quantity - Authorized



## Not Authorized Processing

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

Figure 115 Authorization\_Quantity - Not Authorized



# e\*Insight User Activity Implementation

This chapter discusses the implementation of the User Activity considering aspects of deployment and security.

This chapter also considers how the previous case studies could be enhanced to include the User Activity.

---

## 12.1 Overview of the User Activity

User Activities allow external applications to access attributes in the business process using an API. The API provides a set of functions that allow the external application to access attributes for the User Activity from the e\*Insight database. The e\*Insight engine uses the returned value of the attributes to continue the business process.

For more information on the API, see [“e\\*Insight User Activity API Methods” on page 411](#).

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

### 12.1.1 User Activity Security

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

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

#### To create a user for the initial connection

- 1 Use e\*Insight Administrator to create a user (for example, Connection\_User), and assign a password.

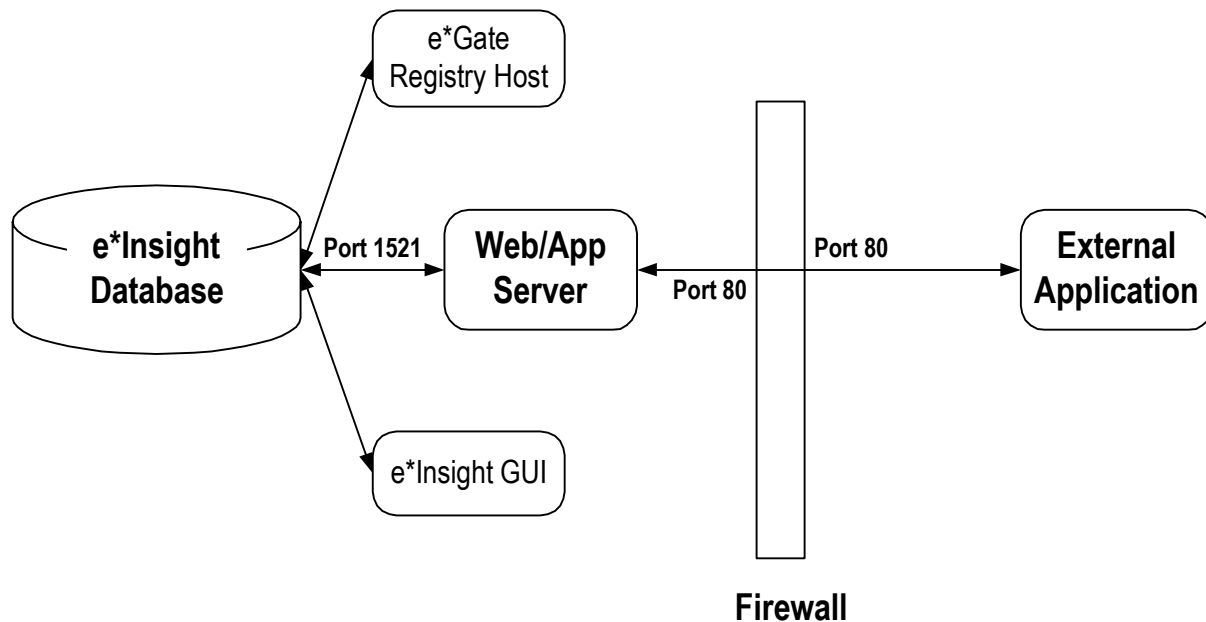
- 2 Do not give this user any authorization rights within e\*Insight.

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

### 12.1.2 Deployment of the User Activity

The application used with the User Activity may connect directly to the database, or it may be a Web-based application that connects to the database via a Web/Application Server. The security described in “User Activity Security” on page 195 is still valid when connecting via a Web/App Server, however there are additional considerations. Figure 116 shows how e\*Insight, the Web/App Server, and the external application may be deployed.

**Figure 116** User Activity Deployment



The firewall is configured to use port 80 to communicate with the Web/App Server. The Web/App Server is configured to communicate with the database using a direct database connection.

An example User Activity scenario using a Web-based application is provided with e\*Insight. For information on installing this, see the *e\*Insight Business Process Manager Installation Guide*.

---

## 12.2 Overview of the Payroll BP

In this example, we assume we have a custom Web application developed for the purpose of allowing various users to set the bonus values. The current example uses an Automated Activity to do a similar function from within the e\*Insight GUI. By

replacing the Automated Activity with a User Activity, the business process can now integrate with an external application such as a Web-based application. This application would allow users to login, view lists of the business process instances, make appropriate changes, and have those changes re-enter the e\*Insight business model for continued processing.

While this example specifically uses a custom Web application, design and development of the external application is left up to the developer. The external application uses the Java API's provided with the e\*Insight engine (a complete description and list of these APIs is available in "[e\\*Insight User Activity API Methods](#)" on page 411). The role of the APIs is to allow the external application to communicate with the e\*Insight engine for the purpose of setting security and verifying users, retrieving business model related information, getting and returning attribute types and values, tracking instances, and many other functions related to the business process instances.

---

## 12.3 Overview

The major steps in the implementation are:

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

### 12.3.1 Case Study: Payroll Processing with User Activity

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

---

## 12.4 Step 1: Update the Payroll BP in e\*Insight

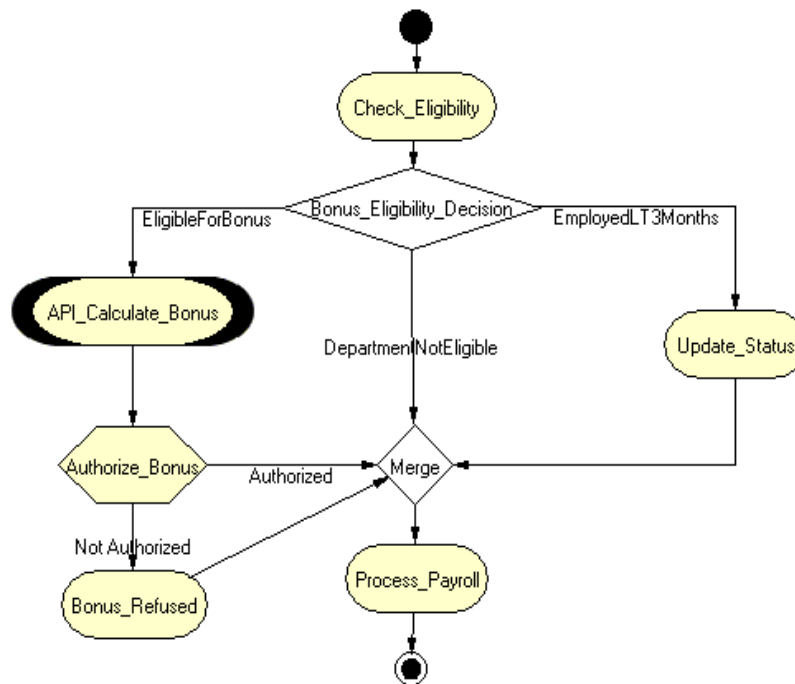
The following is a summary of the procedures for adding a User Activity in the e\*Insight GUI.

- 1 Delete the **Calculate\_Bonus** Activity.
- 2 Add a User Activity called **API\_Calculate\_Bonus**.
- 3 Make the connections between the Decision gates, Merge gate, and User Activity, as shown in Figure 117.

**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 117** Payroll Business Process Model



---

## 12.5 Step 2: Configure the Integration Schema

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

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

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

---

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

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

### 12.6.1 Testing the User Activity

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

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

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

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

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

Substitute the appropriate username and password for your installation.

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

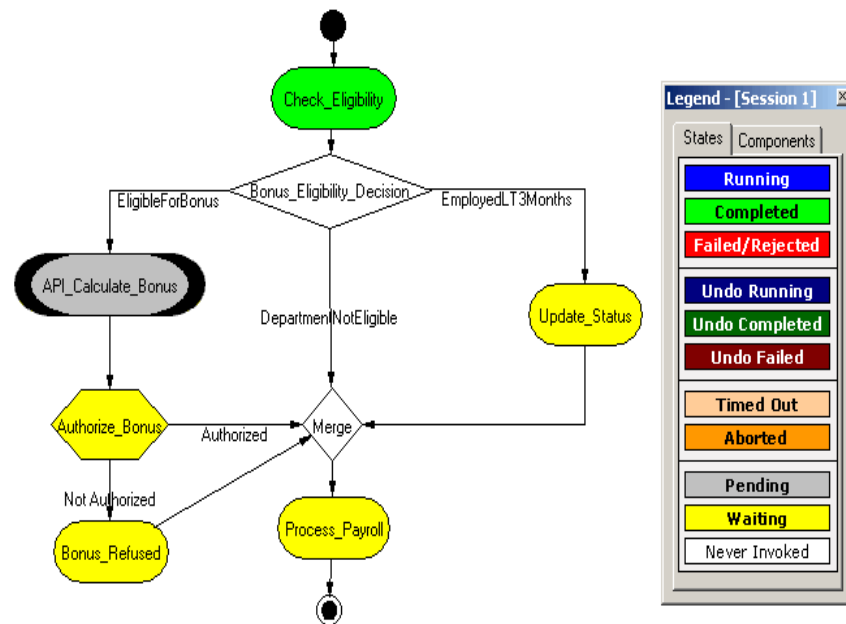
**Note:** *The change of the extension to **“.~in”** indicates that the data file has been picked up by the **START\_BP** e\*Way.*

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

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

Figure 118 shows how the BPI Diagram appears.

**Figure 118** 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 119 on page 201](#).

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).

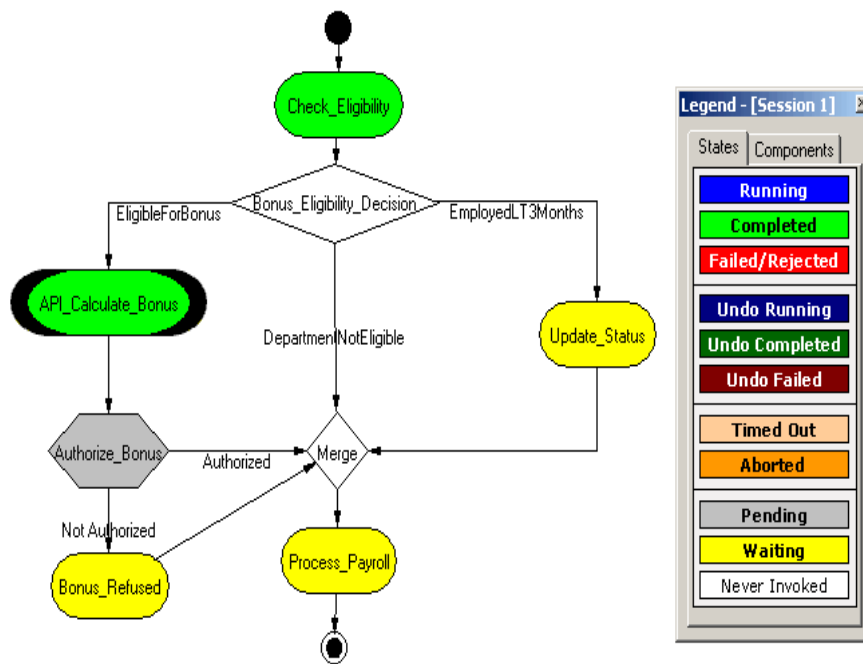
- ◆ **setUser**—This function passes the name of the user to the e\*Insight engine if the external application uses login/password.
- ◆ **checkUserPrivileges**—This function establishes the security rights for the session using security levels as set within e\*Insight.
- ◆ **getGlobalAttributeValue** 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 are shipped with e\*Insight 4.5.2. This example can be installed by you and set up and used to demonstrate a business process using a Web-based application to allow users to log in, and view lists of instances that need to be reviewed and adjusted. The external application then sends the information back to e\*Insight, thus allowing the business process to continue.

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

**Figure 119** User Activity - Bonus set



## 12.7 Overview of the ProcessOrder BP

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

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

**Methods” on page 411).** The role of the APIs is to allow the external application to communicate with the e\*Insight engine for the purpose of setting security and verifying users, retrieving business model related information, getting and returning attribute types and values, tracking instances, and many other functions related to the business process instances.

---

## 12.8 Overview

The major steps in the implementation are:

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

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

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

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

---

## 12.9 Step 1: Update the ProcessOrder BP in e\*Insight

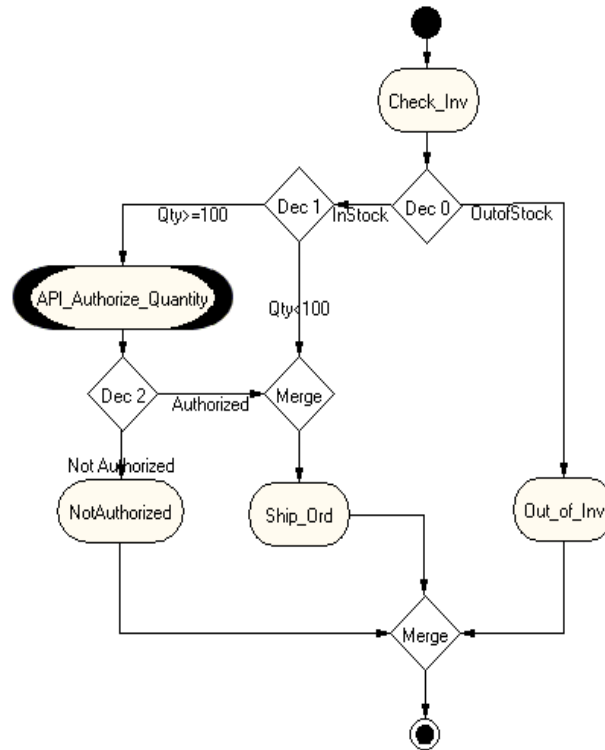
The following is a summary of the procedures for adding a User Activity in the e\*Insight GUI.

- 1 Delete the **Authorize\_Quantity** Activity.
- 2 Add a User Activity called **API\_Authorize\_Order**.
- 3 Add a Decision gate to receive the output from the User Activity.
- 4 Make the connections between the Decision gates, Merge gate, and Not Authorized Activity, as shown in **Figure 117 on page 198**.

*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 120** ProcessOrder Business Process Model



---

## 12.10 Step 2: Configure the Integration Schema

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

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

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

---

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

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

### 12.11.1 Testing the User Activity

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

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

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

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

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

Substitute the appropriate username and password for your installation.

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

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

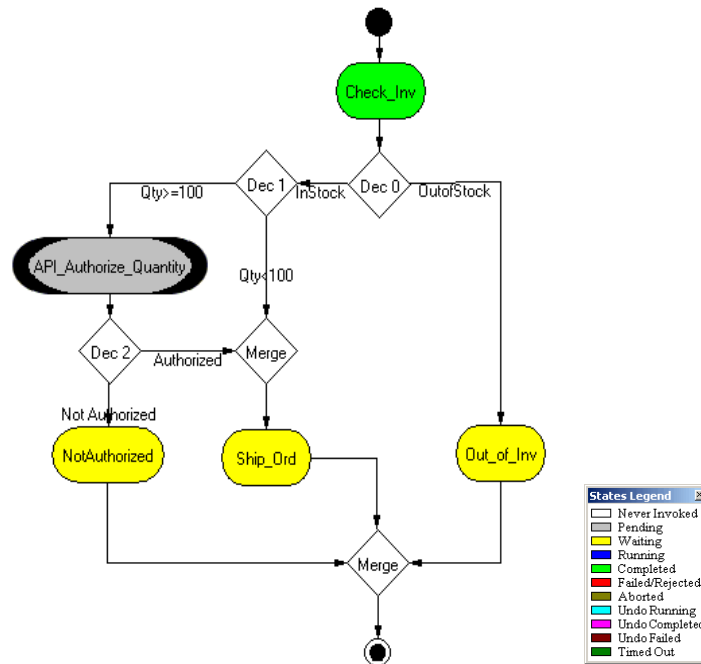
**Note:** *The change of the extension to ".~in" indicates that the data file has been picked up by the START\_BP e\*Way.*

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

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

Figure 121 shows how the BPI Diagram appears.

**Figure 121** 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 122 on page 206](#).

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

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

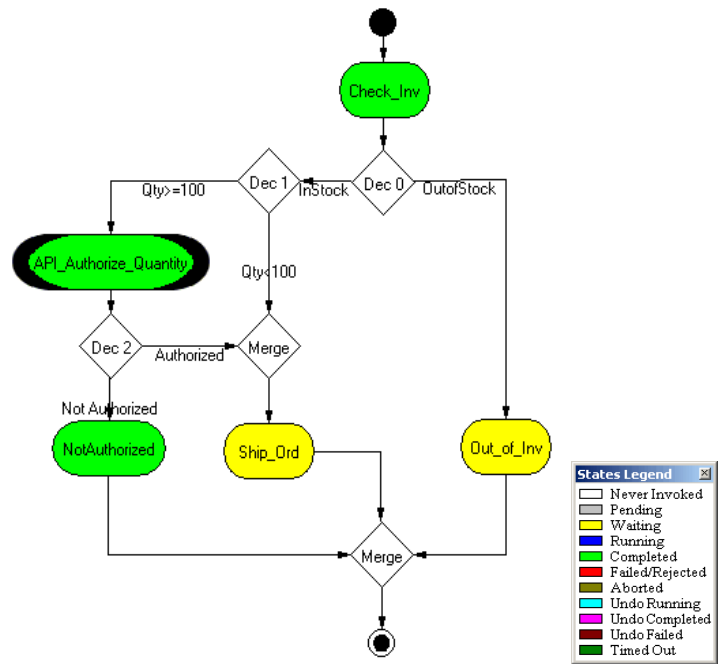
- ◆ **setUser**—This function passes the name of the user to the e\*Insight engine if the external application uses login/password.
- ◆ **checkUserPrivileges**—This function establishes the security rights for the session using security levels as set within e\*Insight.
- ◆ **getGlobalAttributeValue**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

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

A example of a business model including a User Activity and an external application are shipped with e\*Insight 4.5.2. This example can be installed by you and set up and used to demonstrate a business process using a Web-based application to allow users to log in, and view lists of instances that need to be reviewed and adjusted. The external application then sends the information back to e\*Insight, thus allowing the business process to continue.

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

**Figure 122** User Activity - Item in stock



# 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 109](#) for the initial configuration instructions.

---

## 13.1 Overview of the Sub-Process Example

The major steps in the implementation are:

- 1 Create and configure a new business process to perform the activities required for checking inventory.
- 2 Update the Payroll business process to replace the Calculate\_Bonus activity with a Sub-Process.
- 3 Use the e\*Insight GUI to configure the e\*Gate schema that supports e\*Insight.
- 4 Add and configure the user-defined e\*Gate components.
- 5 Run and test the scenario.

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

## 13.2 Create the CalculateBonus BP in e\*Insight

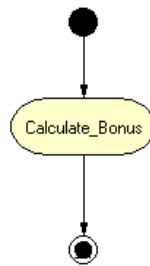
The following is a summary of the procedure for creating a BP in the e\*Insight GUI.

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

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

Use the diagram shown in Figure 123 and the following tables to create the BP in e\*Insight.

**Figure 123** CalculateBonus Business Process Model





**Table 37** BP Global Attributes

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

**Table 38** Activity Attributes

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

---

## 13.3 Configure the Integration Schema for CalculateBonus

The Calculate\_Bonus activity uses the eX\_Calculate\_Bonus BOB that was created in [“Creating the eX\\_Calculate\\_Bonus BOB” on page 119](#).

To configure the Calculate\_Bonus Activity

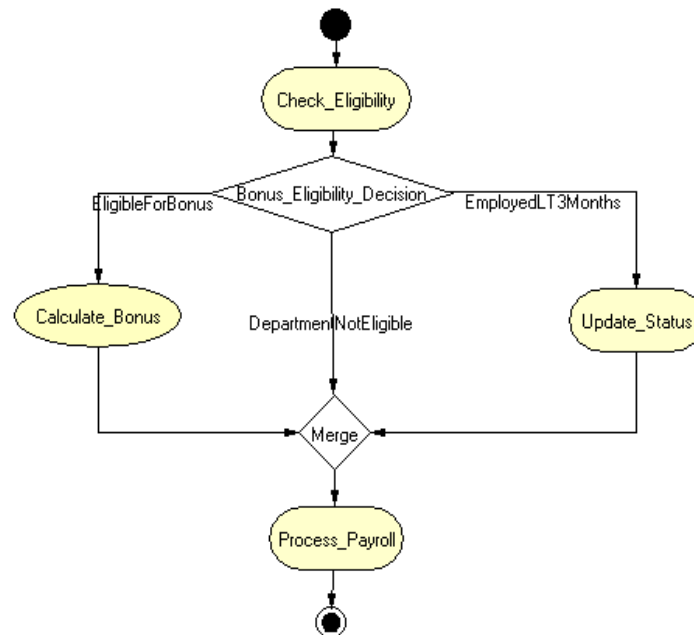
- 1 In the e\*Insight GUI, open the Calculate\_Bonus activity properties.
- 2 On the **General** tab, select the **BOB** e\*Gate module.
- 3 Check the module name is **eX\_Calculate\_Bonus**, and modify if necessary.
- 4 Check that the correct Participating Host is selected.
- 5 Close the **Calculate\_Bonus**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

## 13.4 Modify the Payroll BP in e\*Insight

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

- 1 Delete the Calculate\_Bonus activity.
- 2 Add the Calculate\_Bonus Sub-Process.
- 3 Make the connections between the sub-process, decision gate and activity as shown in Figure 124.

**Figure 124** Payroll BP with Sub-Process



- 4 Configure the Calculate\_Bonus sub-process properties.
  - A Assign the CalculateBonus business process as the sub-process.
  - B Map the sub-process attributes to business process attributes as defined in Table 39.

**Table 39** Sub-Process attribute mapping

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

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

## 13.5 Configure the Integration Schema for Payroll

The schema created in “[e\\*Insight Implementation \(eJSchema\)](#)” on page 109 does not need to be modified for this example.

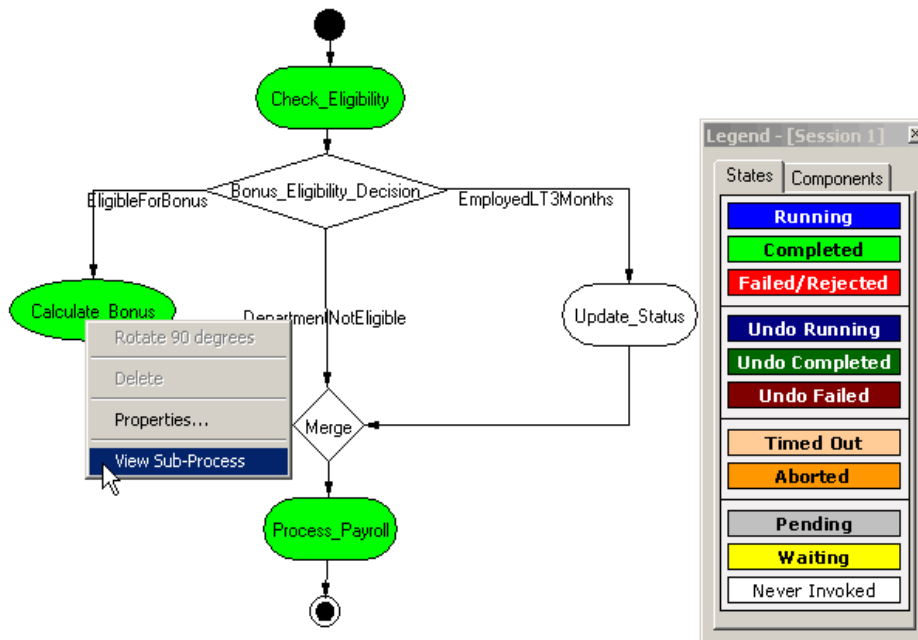
## 13.6 Run and Test the e\*Insight scenario

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

Use the instructions in “[Run and Test the e\\*Insight Scenario](#)” on page 131 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 125.

**Figure 125** Accessing the CalculateBonus business process



---

## 13.7 Overview of the Dynamic Sub-Process Example

The Dynamic Sub-Process Example is a continuation of the previous example.

In this example the Business Process that is called as the Sub-Process depends on where the department that the employee works for. If the department is accounts, then the **Calculate\_Bonus\_accounts** Business Process is called, and if the department is marketing, then the **Calculate\_Bonus\_marketing** Business Process is called.

The major steps in the implementation are:

- 1 Create and configure two new business processes (**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.

---

## 13.8 Create the accounts BP in e\*Insight

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

- 1 Export business process named CalculateBonus.

**Note:** The exported file is used in [“Create the marketing BP in e\\*Insight” on page 215.](#)

- 2 Rename the CalculateBonus business process **accounts**.
- 3 Select the enabled version, and from the **File** menu, select **Save as New Version**.
- 4 Rename the activity **Calculate\_Bonus\_accounts**.
- 5 Enable the business process version.

---

## 13.9 Configure the Integration Schema for accounts

To configure the Calculate\_Bonus\_accounts activity

- 1 In the e\*Insight GUI, open the **Calculate\_Bonus\_accounts** activity properties.
- 2 On the **General** tab, select the **BOB** e\*Gate module.
- 3 Click **New**.

The Define Collaboration dialog appears.

- 4 Click **OK**.
- 5 Create eX\_Calculate\_Bonus\_marketing.xpr.

The actual CRS is created later in [“To create eX\\_Calculate\\_Bonus\\_accounts.xpr” on page 214.](#)

- 6 Save the CRS.
- 7 Close the editor.
- 8 Click **Configure e\*Gate**.  
You may be required to log into e\*Gate.
- 9 Click **OK**, to close the information dialog.
- 10 Close the **Calculate\_Bonus\_accounts** Activity properties.

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

## Creating the CRS in e\*Gate

Since the CRS for eX\_Calculate\_Bonus\_accounts BOB is very similar to the eX\_Calculate\_Bonus CRS previously created you can copy this instead of creating a new script from scratch.

To create eX\_Calculate\_Bonus\_accounts.xpr

- 1 Open the Collaboration Editor.
- 2 Open eX\_Calculate\_Bonus.xpr.
- 3 From the **File** menu, select **Save As**, and enter the name **eX\_Calculate\_Bonus\_accounts.xpr**.

**Important:** You should replace the existing file in *Collaboration\_Rules\Calculate\_Bonus\_accounts*.

- 4 Compile and then close the editor.

---

## 13.10 Create the marketing BP in e\*Insight

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

- 1 Import the business process that was exported in [“Create the accounts BP in e\\*Insight” on page 213](#).
- 2 Rename the imported CalculateBonus business process **marketing**.
- 3 Rename the activity Calculate\_Bonus\_marketing.
- 4 Enable the business process version.

---

## 13.11 Configure the Integration Schema for marketing

To configure the Calculate\_Bonus\_marketing activity

- 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 216](#).
- 6 Save the CRS.
- 7 Close the editor.

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

- 8 In the **Calculate\_Bonus\_marketing** activity properties, check the module name is eX\_Calculate\_Bonus\_marketing, and modify if necessary.
- 9 Click **Configure e\*Gate**.  
You may be required to log into e\*Gate.
- 10 Click **OK**, to close the information dialog.
- 11 Close the **Calculate\_Bonus\_marketing** Activity properties.

### Creating the CRS for eX\_Calculate\_Bonus\_marketing in e\*Gate

Since the CRS for eX\_Calculate\_Bonus\_marketing BOB is very similar to the eX\_Calculate\_Bonus CRS previously created you can copy this instead of creating a new script from scratch.

To create eX\_Calculate\_Bonus\_marketing.xpr

- 1 Open the Collaboration Editor.
- 2 Open eX\_Calculate\_Bonus\_marketing.xpr.
- 3 From the **File** menu, select **Save As**, and enter the name **eX\_Calculate\_Bonus\_marketing.xpr**.

***Important:** You should replace the existing file in **Collaboration\_Rules\Calculate\_Bonus\_marketing**.*

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

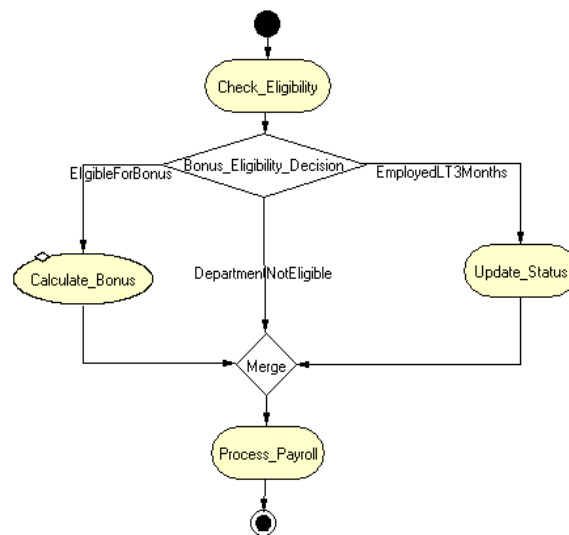


## 13.12 Modify the Payroll BP in e\*Insight

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

- 1 Save the enabled **Payroll** business process as a new version.
- 2 Delete the **Calculate\_Bonus** activity.
- 3 Add the **Calculate\_Bonus** Dynamic Sub-Process.
- 4 Make the connections between the dynamic sub-process, decision gate and activity as shown in Figure 126.

**Figure 126** 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.

## 13.13 Configure the Integration Schema for Payroll

The schema created in “**e\*Insight Implementation (eIJSchema)**” on page 109 does not need to be modified for this example.

---

## 13.14 Run and Test the e\*Insight scenario

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

Use the instructions in [“Run and Test the e\\*Insight Scenario” on page 131](#) to test your sub-process.

# 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 147](#) for the initial configuration instructions.

---

## 14.1 Overview of the Sub-Process Example

The major steps in the implementation are:

- 1 Create and configure a new business process to perform the activities required for checking inventory.
- 2 Update the ProcessOrder business process to replace the Check\_Inv activity with a Sub-Process.
- 3 Use the e\*Insight GUI to configure the e\*Gate schema that supports e\*Insight.
- 4 Add and configure the user-defined e\*Gate components.
- 5 Run and test the scenario.

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

## 14.2 Create the CheckInventory BP in e\*Insight

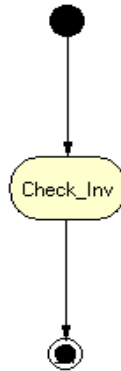
The following is a summary of the procedure for creating a BP in the e\*Insight GUI.

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

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

Use the diagram shown in Figure 127 and the following tables to create the BP in the e\*Insight.

**Figure 127** CheckInventory Business Process Model



**Table 40** BP Global Attributes

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

**Table 41** Activity Attributes

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

## 14.3 Configure the Integration Schema for CheckInventory

The Check\_Inv activity uses the eX\_Check\_Inv BOB that was created in [“Creating the eX\\_Check\\_Inv BOB” on page 155](#).

To configure the Check\_Inv Activity

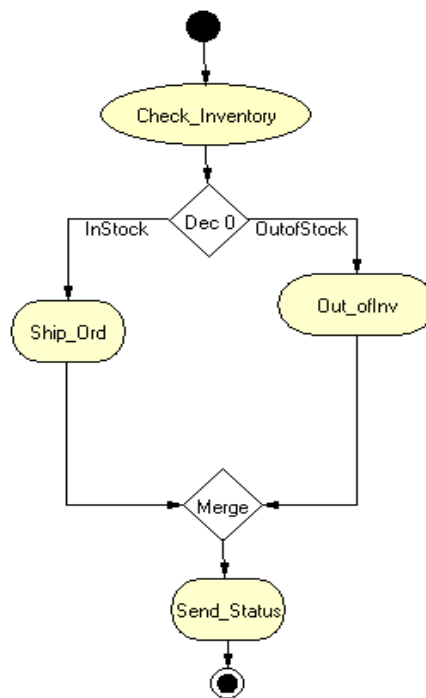
- 1 In the e\*Insight GUI, open the Check\_Inv activity properties.
- 2 On the **General** tab, select the **BOB** e\*Gate module.
- 3 Check the module name is **eX\_Check\_Inv**, and modify if necessary.
- 4 Check that the correct Participating Host is selected.
- 5 Close the **Check\_Inv** Activity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

## 14.4 Modify the ProcessOrder BP in e\*Insight

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

- 1 Delete the Check\_Inv activity.
- 2 Add the Check\_Inventory Sub-Process.
- 3 Make the connections between the sub-process, decision gate and activity as shown in Figure 128.

**Figure 128** 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 42.

**Table 42** Sub-Process attribute mapping

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

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

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

---

## 14.5 Configure the Integration Schema for ProcessOrder

The schema created in [“e\\*Insight Implementation \(eISchema\)” on page 147](#) does not need to be modified for this example.

---

## 14.6 Run and Test the e\*Insight scenario

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

Use the instructions in [“Run and Test the e\\*Insight scenario” on page 176](#) to test your sub-process.

---

## 14.7 Overview of the Dynamic Sub-Process Example

The Dynamic Sub-Process Example is a continuation of the previous example.

In this example the Business Process that is called as the Sub-Process depends on where the order is being delivered. If the order is for California, then the **Check\_Inv\_CA** Business Process is called, and if the order is for Oregon, then the **Check\_Inv\_OR** Business Process is called.

The major steps in the implementation are:

- 1 Create and configure two new business processes (**Check\_Inv\_CA** for California and **Check\_Inv\_OR** for Oregon) to perform the activities required for checking inventory.
- 2 Update the ProcessOrder business process to replace the Check\_Inventory sub-process with a dynamic sub-process.
- 3 Use the e\*Insight GUI to configure the e\*Gate schema that supports e\*Insight.
- 4 Add and configure the user-defined e\*Gate components.
- 5 Run and test the scenario.

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



---

## 14.8 Create the CA BP in e\*Insight

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

- 1 Export business process named CheckInventory.

*Note:* The exported file is used in [“Create the OR BP in e\\*Insight” on page 228](#).

- 2 Rename the CheckInventory business process **CA**.
- 3 Select the enabled version, and from the **File** menu, select **Save as New Version**.
- 4 Rename the activity **Check\_Inv\_CA**.
- 5 Enable the business process version.

---

## 14.9 Configure the Integration Schema for CA

To configure the Check\_Inv\_CA activity using Monk

- 1 In the e\*Insight GUI, open the **Check\_Inv\_CA** activity properties.
- 2 On the **General** tab, select the **BOB** e\*Gate module.
- 3 Click **New**.

The **Define Collaboration** dialog appears.

- 4 Click **OK**.
- 5 Create eX\_Check\_Inv\_CA.tsc. The source and destination Event Type Definitions are eX\_Standard\_Event.

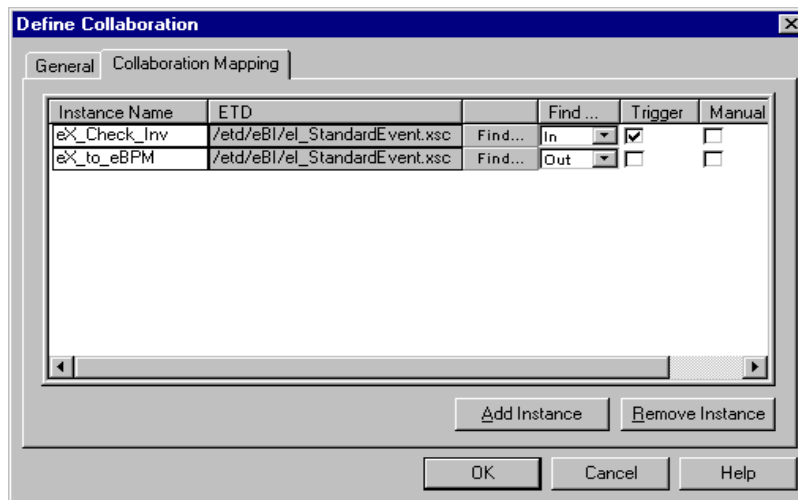
The actual CRS script is created later in [“To create eX\\_Check\\_Inv\\_CA.tsc using Monk” on page 227](#).

- 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 129.

**Figure 129** 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 227](#).
- 8 Save the CRS.
- 9 Close the editor.
- 10 In the **Check\_Inv\_CA** activity properties, check the module name is **eX\_Check\_Inv\_CA**, and modify if necessary.
- 11 Click **Configure e\*Gate**.  
You may be required to log into e\*Gate.
- 12 Click **OK**, to close the information dialog.
- 13 Close the **Check\_Inv\_CA** Activity properties.

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

## Creating the CRS in e\*Gate

Since the CRS for eX\_Check\_Inv\_CA BOB is very similar to the eX\_Check\_Inv CRS previously created you can copy this instead of creating a new script from scratch.

### To create eX\_Check\_Inv\_CA.tsc using Monk

- 1 Open the Collaboration Editor.
- 2 Open eX\_Check\_Inv.tsc.
- 3 From the **File** menu, select **Save As**, and enter the name **eX\_Check\_Inv\_CA.tsc**. You should replace the existing file.
- 4 Close the editor.

### To create eX\_Check\_Inv\_CA.xsc using Java

- 1 Open the Collaboration Editor.
- 2 Open eX\_Check\_Inv.xsc.
- 3 From the **File** menu, select **Save As**, and enter the name **eX\_Check\_Inv\_CA.xsc**. You should replace the existing file.
- 4 Compile and then close the editor.

---

## 14.10 Create the OR BP in e\*Insight

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

- 1 Import the business process that was exported in [“Create the CA BP in e\\*Insight” on page 225](#).
- 2 Rename the imported CheckInventory business process **OR**.
- 3 Rename the activity Check\_Inv\_OR.
- 4 Enable the business process version.

---

## 14.11 Configure the Integration Schema for OR

To configure the Check\_Inv\_OR activity using Monk

- 1 In the e\*Insight GUI, open the **Check\_Inv\_OR** activity properties.
  - 2 On the **General** tab, select the **BOB** e\*Gate module.
  - 3 Click **New**.
- The **Define Collaboration** dialog appears.
- 4 Click **OK**.
  - 5 Create eX\_Check\_Inv\_OR.tsc. The source and destination Event Type Definitions are eX\_Standard\_Event.

The actual CRS script is created later in [“To create eX\\_Check\\_Inv\\_OR.tsc using Monk” on page 230](#).

- 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.

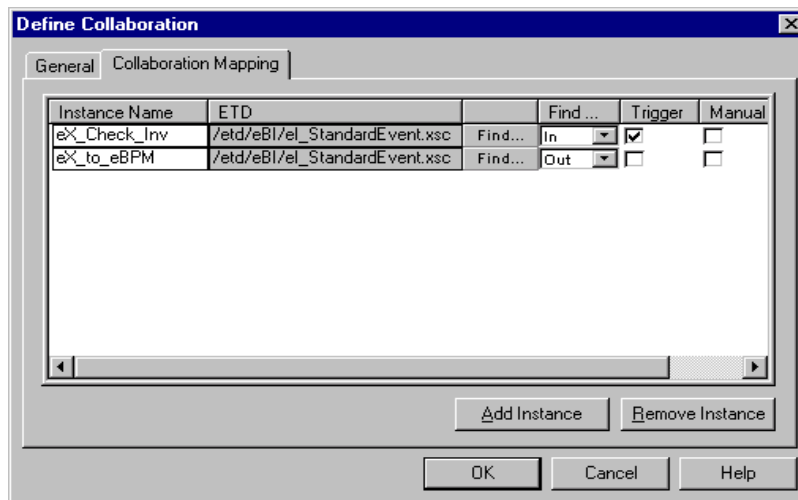
### To configure the Check\_Inv\_OR activity using Java

- 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 130.

**Figure 130** 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 230](#).
- 8 Save the CRS.
- 9 Close the editor.
- 10 In the **Check\_Inv\_OR** activity properties, check the module name is **eX\_Check\_Inv\_OR**, and modify if necessary.
- 11 Click **Configure e\*Gate**.  
You may be required to log into e\*Gate.
- 12 Click **OK**, to close the information dialog.
- 13 Close the **Check\_Inv\_OR** Activity properties.

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

## Creating the CRS in e\*Gate

Since the CRS for eX\_Check\_Inv\_OR BOB is very similar to the eX\_Check\_Inv CRS previously created you can copy this instead of creating a new script from scratch.

### To create eX\_Check\_Inv\_OR.tsc using Monk

- 1 Open the Collaboration Editor.
- 2 Open eX\_Check\_Inv.tsc.
- 3 Update the script to define 11111 as in stock.
- 4 From the **File** menu, select **Save As**, and enter the name **eX\_Check\_Inv\_OR.tsc**. You should replace the existing file.
- 5 Close the editor.

### To create eX\_Check\_Inv\_OR.xsc using Java

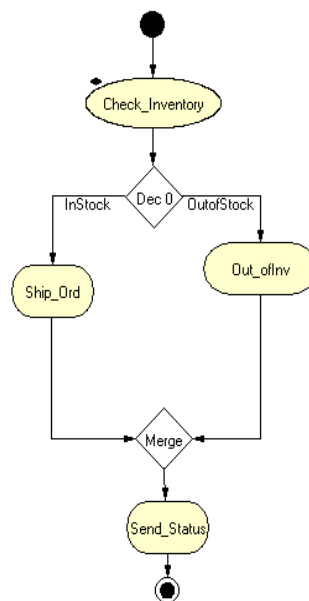
- 1 Open the Collaboration Editor.
- 2 Open eX\_Check\_Inv\_OR.xsc.
- 3 From the **File** menu, select **Save As**, and enter the name **eX\_Check\_Inv\_OR.xsc**. You should replace the existing file.
- 4 Update the script to define 11111 as out of stock.
- 5 Compile and then close the editor.

## 14.12 Modify the ProcessOrder BP in e\*Insight

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

- 1 Save the enable ProcessOrder business process as a new version.
- 2 Delete the Check\_Inv activity.
- 3 Add the Check\_Inventory Dynamic Sub-Process.
- 4 Make the connections between the dynamic sub-process, decision gate and activity as shown in Figure 131.

**Figure 131** 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 42.

**Table 43** Global attribute mapping

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

- C From the Local Attributes tab, edit **subBPName**, and select Address\_State from the Value drop down list.

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

---

## 14.13 Configure the Integration Schema for ProcessOrder

The schema created in [“e\\*Insight Implementation \(eISchema\)” on page 147](#) does not need to be modified for this example.

---

## 14.14 Run and Test the e\*Insight scenario

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

Use the instructions in [“Run and Test the e\\*Insight scenario” on page 176](#) to test your sub-process.



# 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 109](#) or [“e\\*Insight Implementation \(eISchema\)” on page 147](#) for the initial configuration instructions.

---

## 15.1 Overview

The major topics covered in this chapter are:

- Installation and configuration of Tomcat
- Implementation steps

---

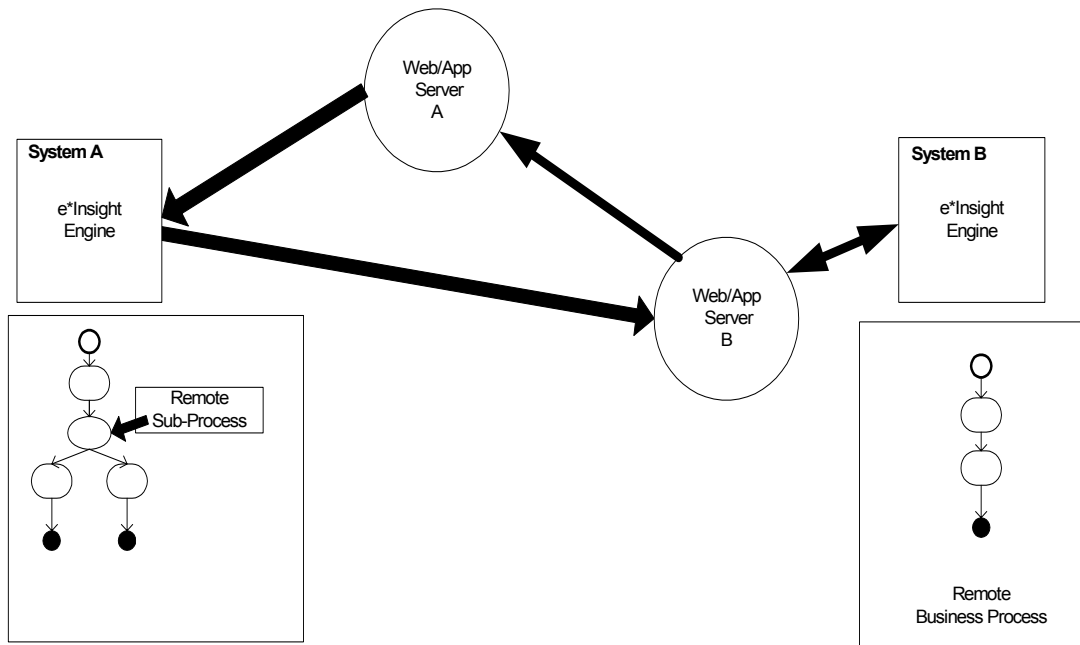
## 15.2 Overview of the Remote Sub-Process

The Remote Sub-Process allows you to access a Business Process defined on a different machine. Business Process messages are sent between the two e\*Insight engines via a Web/Application Server. The Web/Application Server routes the Business Process message to the correct location.

**Note:** *e\*Insight uses Apache/Tomcat Web/Application Server.*

The Remote Sub-Process can be used across machines with the LAN, WAN, or the Internet. The deployment of machines that are separated by a firewall is considered in [“Installation of Tomcat and e\\*Insight on Different Hosts” on page 236](#).

**Figure 132** 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.

## 15.3 Installation and Configuration of Tomcat

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

**Table 44** Steps for Installation and Configuration of Tomcat

	Step	Procedure
1	Install Tomcat	<a href="#">"Installing Tomcat" on page 234</a>
2	Configure Tomcat	<a href="#">"Configuring Tomcat" on page 235</a>
3	Deploy the SOAP service	<a href="#">"Deploying the SOAP Service" on page 235</a>

### 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 **eInsightDeploymentDescriptor.xml** in any text editor. Modify the following parameter:

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

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

- 4 In a command prompt, change directory to **<einsight>\soap**.
- 5 Run **RegisterSoapService.cmd**.

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

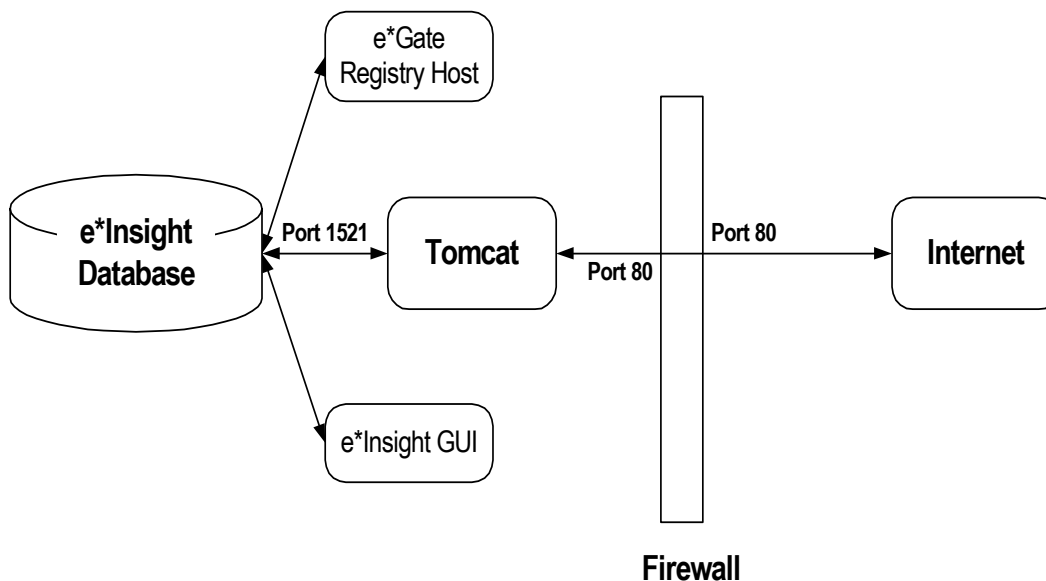
- 6 Start Internet Explorer and go to url: **http://<webhost>:8080/soap/admin/index.html**
- 7 Click List to check that the URN has been successfully deployed.

---

## 15.4 Installation of Tomcat and e\*Insight on Different Hosts

There are a number of considerations when deploying Tomcat and e\*Insight on different hosts. Figure 133 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 133** 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 235](#).

---

## 15.5 Overview of the Remote Sub-Process Example (eIJSchema)

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

The Payroll example must be completed before the remote sub-process example. See [“e\\*Insight Implementation \(eIJSchema\)” on page 109](#) for the initial configuration instructions.

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

The major steps in the implementation are:

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

---

## 15.6 Install and configure Tomcat

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

Follow the instructions in **“Installation and Configuration of Tomcat” on page 234** for both machines. Define the URN’s as follows:

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

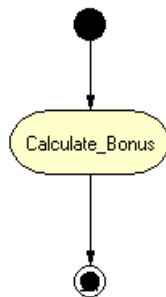
## 15.7 Create the CalculateBonus BP in e\*Insight

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

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

Use the diagram shown in Figure 136 and the following tables to create the BP in e\*Insight.

**Figure 134** CalculateBonus Business Process Model



**Table 45** BP Global Attributes

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

**Table 46** Activity Attributes

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

- 5 In the business process properties, select **Send Business Process Done Event**.
- 6 Enable the business process version.

### To configure the Partner Information

- 1 From the **Options** menu, select **Define Information for Partners**.
- 2 Enter **UUID**.  
This is a unique name that identifies your e\*Insight database to other Trading Partners. For example, **SystemB**.
- 3 Enter the URN.  
This corresponds to the URN setup in the SOAP client. For example, **urn:CalculateBonus**.
- 4 Enter the URL.  
This is the location of the SOAP client. By default this is **http://localhost:8080/soap/servlet/rpcrouter**.
- 5 Click **OK**.

---

## 15.8 Configure the Integration Schema for CalculateBonus

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

### 15.8.1 Create the CalculateBonus Schema

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

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

## 15.8.2 Configure the CalculateBonus Schema

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

### Create the Calculate\_Bonus activity BOB

- 1 In the **SystemA** e\*Gate Enterprise Manager GUI, open the **Payroll** schema.
- 2 Select the eX\_Calculate\_Bonus **BOB**.
- 3 From the File menu select **Export Module Definitions to File**.  
The **Select Archive File** dialog appears.
- 4 Enter eX\_Calculate\_Bonus, and then click **OK**.  
The file **eX\_Calculate\_Bonus.zip** is created.
- 5 Log into the **SystemB** e\*Gate Enterprise Manager GUI and open the **CalculateBonus** schema.
- 6 From the File menu select **Import Module Definitions from File**.
- 7 Locate the file created in step 4.
- 8 Click **OK**.

### Edit the elcp\_eInsightEngine Connection Configuration File

Most of the parameter settings in the elcp\_eInsightEngine connection's configuration file should not be changed. ["Using XA" on page 34](#) discusses the parameters that may need to be changed depending on the implementation. Use the e\*Way Editor and the information in ["Using XA" on page 34](#) to make the required changes for the CalculateBonus example.

### Configure the JMS Connection

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

#### To configure the JMS connection

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

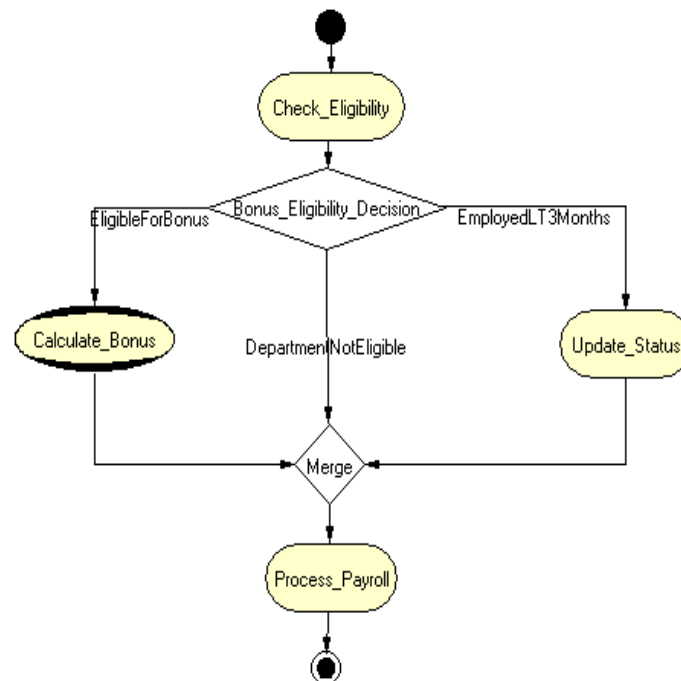


## 15.9 Modify the Payroll BP in e\*Insight

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

- 1 Delete the Calculate\_Bonus activity.
- 2 Add the Calculate\_Bonus Remote Sub-Process.
- 3 Make the connections between the remote sub-process, decision gate and activity as shown in Figure 137.

**Figure 135** 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 50.

**Table 47** Sub-Process attribute mapping

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

7 Enable the business process version.

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

---

## 15.10 Configure the Integration Schema for Payroll

The schema created in “[e\\*Insight Implementation \(eISchema\)](#)” on page 147 does not need to be modified for this example.

---

## 15.11 Run and Test the e\*Insight scenario

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

### To test the Remote Sub-Process scenario

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

### To monitor a remote business process

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

---

## 15.12 Overview of the Remote Sub-Process Example (eISchema)

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

The Process Order example must be completed before the remote sub-process example. See [“e\\*Insight Implementation \(eISchema\)” on page 147](#) for the initial configuration instructions.

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

The major steps in the implementation are:

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

---

## 15.13 Install and configure Tomcat

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

Follow the instructions in [“Installation and Configuration of Tomcat” on page 234](#) for both machines. Define the URN’s as follows:

SystemA	urn:ProcessOrder
SystemB	urn:CheckInventory

---

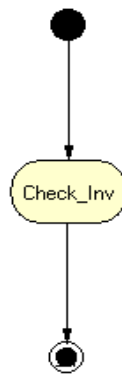
## 15.14 Create the CheckInventory BP in e\*Insight

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

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

Use the diagram shown in Figure 136 and the following tables to create the BP in e\*Insight.

**Figure 136** CheckInventory Business Process Model



**Table 48** BP Global Attributes

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

**Table 49** Activity Attributes

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

- 5 In the business process properties, select **Send Business Process Done Event**.
- 6 Enable the business process version.

**To configure the Partner Information**

- 1 From the **Options** menu, select **Define Information for Partners**.
- 2 Enter **UUID**.  
This is a unique name that identifies your e\*Insight database to other Trading Partners. For example, **SystemB**.
- 3 Enter the URN.  
This corresponds to the URN setup in the SOAP client. For example, **urn:CheckInventory**.
- 4 Enter the URL.  
This is the location of the SOAP client. By default this is **http://localhost:8080/soap/servlet/rpcrouter**.
- 5 Click **OK**.

---

## 15.15 Configure the Integration Schema for CheckInventory

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

### 15.15.1 Create the CheckInventory Schema

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

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

- A Start the e\*Gate Enterprise Manager.
- B Log in to eISchema.
- 2 Export the eISchema to a file <eGate>\client\eISchema.zip.
  - A Select **Export Schema Definitions to File ...** from the **File** pull-down menu.
  - B In the **Select archive File** dialog box enter **eISchema.zip** in the **File name** text box, and then click **Save**.
- 3 Create a new schema using the eISchema export file as a template.
  - A Select **New Schema** from the **File** pull-down menu.
  - B Enter **CheckInventory** in the text box.
  - C Mark the **Create from export** check box.
  - D Click **Find** and browse for the eISchema.zip file created in step 2 above.
  - E Click **Open**.

## 15.15.2 Configure the e\*Insight engine

Most of the parameter settings in the eX\_eBPM engine's configuration file should not be changed. Use the e\*Way Editor and the information in "[Configuring the e\\*Insight Engine](#)" on page 66 to make the required changes for the CheckInventory example.

## 15.15.3 Create the Check\_Inv activity BOB

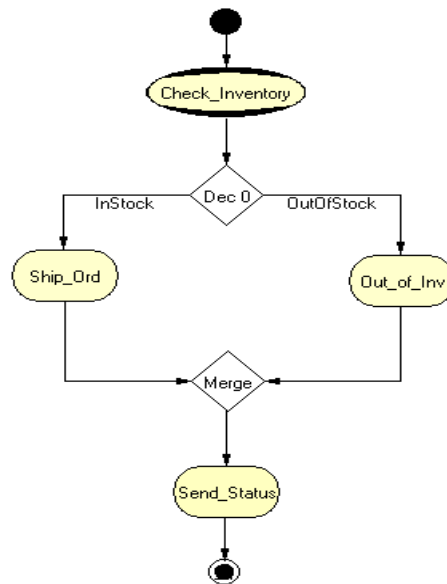
- 1 In the **SystemA** e\*Gate Enterprise Manager GUI, open the **ProcessOrder** schema.
- 2 Select the eX\_Check\_Inv **BOB**.
- 3 From the File menu select **Export Module Definitions to File**.  
The **Select Archive File** dialog appears.
- 4 Enter eX\_Check\_Inv, and then click **OK**.  
The file **eX\_Check\_Inv.zip** is created.
- 5 Log into the **SystemB** e\*Gate Enterprise Manager GUI and open the CheckInventory schema.
- 6 From the File menu select **Import Module Definitions from File**.
- 7 Locate the file created in step 4.
- 8 Click **OK**.

## 15.16 Modify the ProcessOrder BP in e\*Insight

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

- 1 Delete the Check\_Inv activity.
- 2 Add the Check\_Inventory Remote Sub-Process.
- 3 Make the connections between the remote sub-process, decision gate and activity as shown in Figure 137.

**Figure 137** 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 50.

**Table 50** Sub-Process attribute mapping

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

- 6 Enable the business process version.

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

## 15.17 Configure the Integration Schema for ProcessOrder

The schema created in “[e\\*Insight Implementation \(eISchema\)](#)” on page 147 does not need to be modified for this example.

## 15.18 Run and Test the e\*Insight scenario

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

### In-Stock Processing

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



**To test the Remote Sub-Process scenario**

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

**To monitor a remote business process**

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

# 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.

---

## 16.1 Overview

The difference between running an activity in active and passive mode is how the activity is started. In active mode e\*Insight sends a message to e\*Gate to start 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.

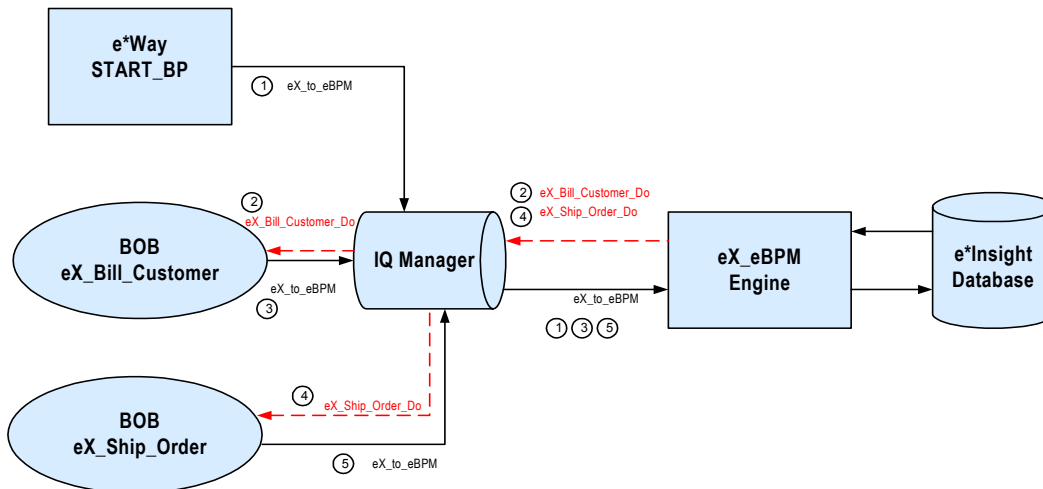
### 16.1.1 Case Study

The case study discussed in this chapter illustrates a simplified implementation of order processing. In this case, e\*Insight receives an incoming order as a delimited text file. Once e\*Insight has received the order the customer is billed and then the order is shipped. See Figure 140 for the Business Process diagram.

## 16.1.2 Case Study - Active Control Mode

Figure 138 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 138** e\*Insight Data Flow Diagram



**Figure 138 data flow description**

- ① The user-defined **START\_BP** e\*Way picks up the text file containing the order information from a shared location on the network, uses the order information to create the event that causes the e\*Insight engine to start a business process instance, and publishes it using the **eX\_to\_eBPM** Event Type. The e\*Insight engine retrieves the Event and uses the information it contains to start the BPI.
- ② The e\*Insight engine publishes a “Do” Event (**eX\_Bill\_Customer\_Do**) for first activity in the business process (**Bill\_Customer**). **eX\_Bill\_Customer** BOB, the e\*Gate component that corresponds to this activity in the business process, retrieves this Event and uses the information it contains to check the availability of the items ordered.
- ③ When the **Bill\_Customer** activity is finished, the **eX\_Bill\_Customer** BOB publishes a “Done” Event using the **eX\_to\_eBPM** Event Type.
- ④ The e\*Insight engine publishes a “Do” Event (**eX\_Ship\_Order\_Do**) corresponding to the **Ship\_Order** activity in the business process. The **eX\_Ship\_Order** BOB retrieves this Event and uses the information it contains to ship the order to the customer.
- ⑤ When the **Ship\_Order** activity is finished, the **eX\_Ship\_Order** BOB publishes a “Done” Event indicating that the order has been shipped.

### 16.1.3 Case Study - Passive Control Mode

We are now going to change the configuration of the Ship\_Order activity to use passive control, that is e\*Gate controls the Activity rather than e\*Insight. The e\*Gate configuration is modified so the Bill\_Customer collaboration publishes eX\_Ship\_Order\_Do in addition to returning a “Done” message to the e\*Insight engine.

Figure 139 e\*Insight Data Flow Diagram

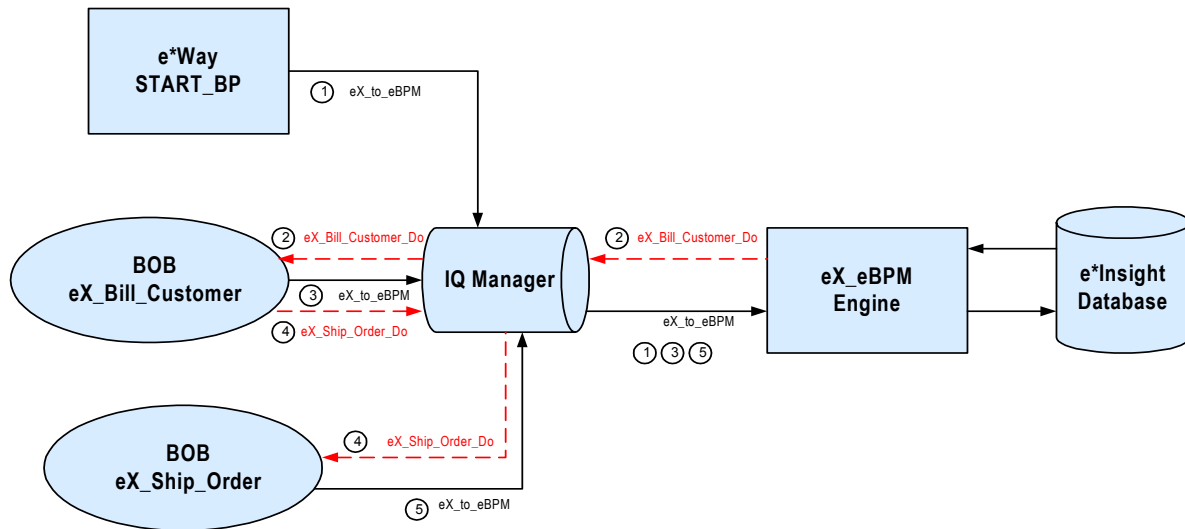


Figure 138 data flow description

- 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 254](#)**
- Step 2, 3, and 4:
  - ♦ **[“Configure the Integration Schema \(eIJSchema\)” on page 255](#)** or
  - ♦ **[“Configure the Integration Schema \(eISchema\)” on page 264](#)**
- Step 5: **[“Run and Test the e\\*Insight scenario” on page 269](#)**

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 270](#)**
- B **[“Modify User-defined e\\*Gate Components \(eIJSchema\)” on page 270](#)**
- C **[“Run and Test the e\\*Insight scenario” on page 273](#)**

Use the following steps if you are using eISchema:

- A **[“Modify the Order BP in e\\*Insight \(eISchema\)” on page 274](#)**
- B **[“Modify User-defined e\\*Gate Components \(eISchema\)” on page 274](#)**
- C **[“Run and Test the e\\*Insight scenario” on page 275](#)**

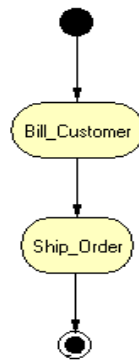
## 16.2 Create the Order BP in e\*Insight

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

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

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

**Figure 140** 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 51** 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 52** Activity Attributes

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

**Table 52** Activity Attributes

Activity	Attribute(s)	Input/Output
Ship_Order	Cust_Name	Input
	Item_Number	Input
	Order_Status	Output

## 16.3 Configure the Integration Schema (eIJSchema)

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

To create a copy of eIJSchema

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

### Integration Schema Activity Components Summary

Use the information in Table 56 to configure the e\*Gate schema that supports the example.

**Table 53** 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

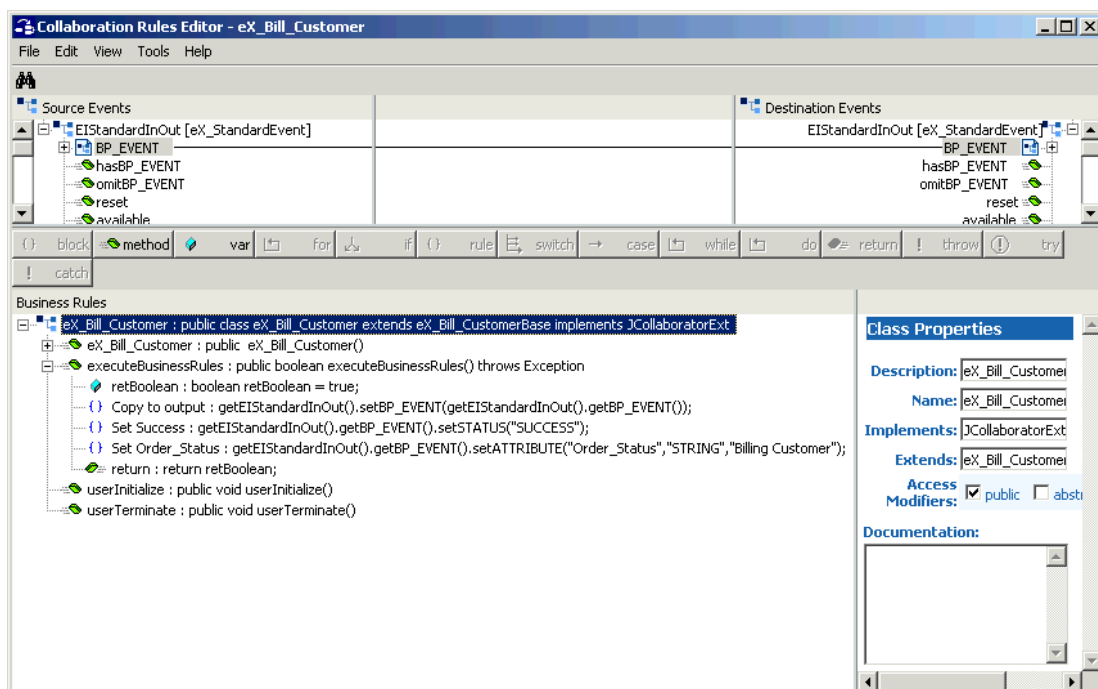
The Bill\_Customer translation implements the logic associated with billing the customer. In this simple example, this component sets the “Order\_Status” attribute value.

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 141 shows the eX\_Bill\_Customer CRS used in the example.

**Figure 141** 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

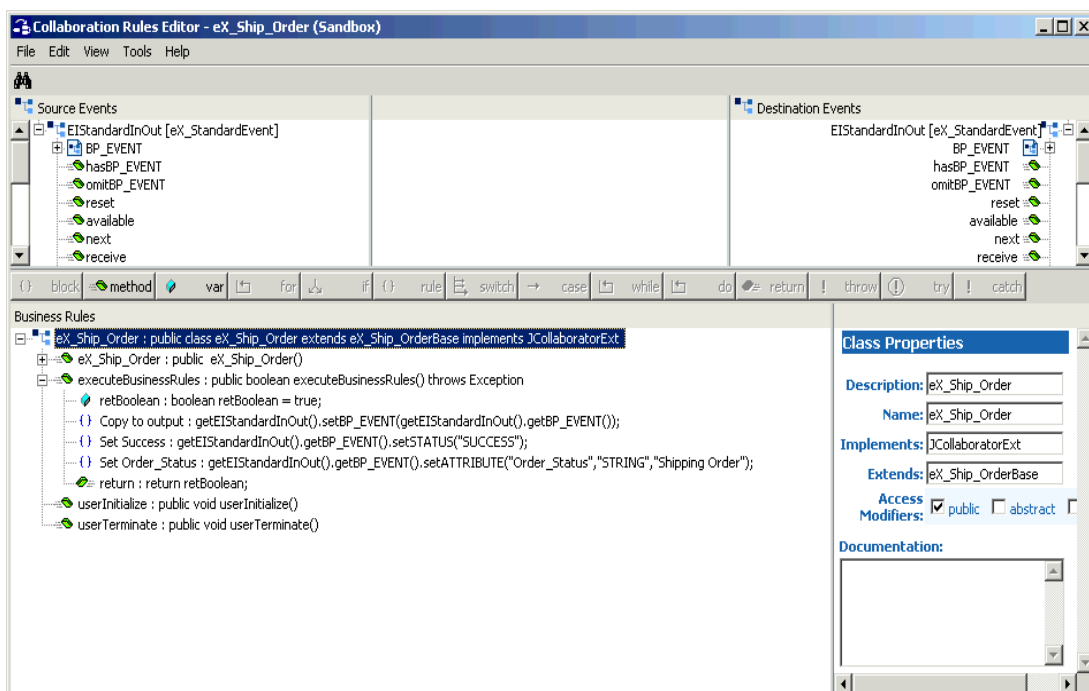
The Ship\_Order translation implements the logic associated with shipping to the customer. In this simple example, this component sets the “Order\_Status” attribute value.

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 141 shows the eX\_Ship\_Order CRS used in the example.

**Figure 142** 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.

---

## 16.4 Configure the e\*Insight Engine (eIJSchema)

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

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

### Edit the elcp\_eInsightEngine Connection Configuration File

Most of the parameter settings in the elcp\_eInsightEngine connection's configuration file should not be changed. ["Using XA" on page 34](#) discusses the parameters that may need to be changed depending on the implementation. Use the e\*Way Editor and the information in ["Using XA" on page 34](#) to make the required changes for the example.

#### 16.4.1 Configure the JMS Connection

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

To configure the JMS connection

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

---

## 16.5 Configure User-defined e\*Gate Components (eIJSchema)

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

The Order example uses a file e\*Way to start the business process and BOBs to run all the other activities.

## Configuration Order for the User-defined Components

- 1 Add and configure the **START\_BP** e\*Way.
- 2 Configure the Collaborations for the activity components running as BOBs.

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

### 16.5.1 Configure the **START\_BP** e\*Way

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

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

Table 54 shows the steps to configure the **START\_BP** e\*Way.

**Table 54** Configuration steps for the **START\_BP** e\*Way

	Step	Section
1	Add the e*Way and create the e*Way configuration file	<a href="#">“Step 1: Create the START_BP e*Way” on page 259</a>
2	Create the Input ETD	<a href="#">“Step 2: Create the Input ETD” on page 260</a>
3	Create the <b>START_BP</b> Collaboration Rules script (CRS)	<a href="#">“Step 3: Create the START_BP Collaboration” on page 261</a>
4	Configure the Collaboration in the GUI	<a href="#">“Step 4: Configure the Collaboration in the GUI” on page 263</a>

#### Step 1: Create the **START\_BP** e\*Way

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

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

**Table 55** Start e\*Way Parameters

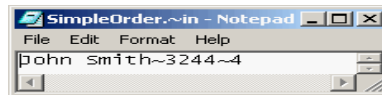
Screen	Parameter	Setting
General Settings	(All)	(Default)
Outbound (send) settings	(All)	(Default)
Poller (inbound) settings	PollDirectory	<eGate>\client\data\Order
	(All others)	(Default)
Performance Testing	(All)	(Default)

## 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.

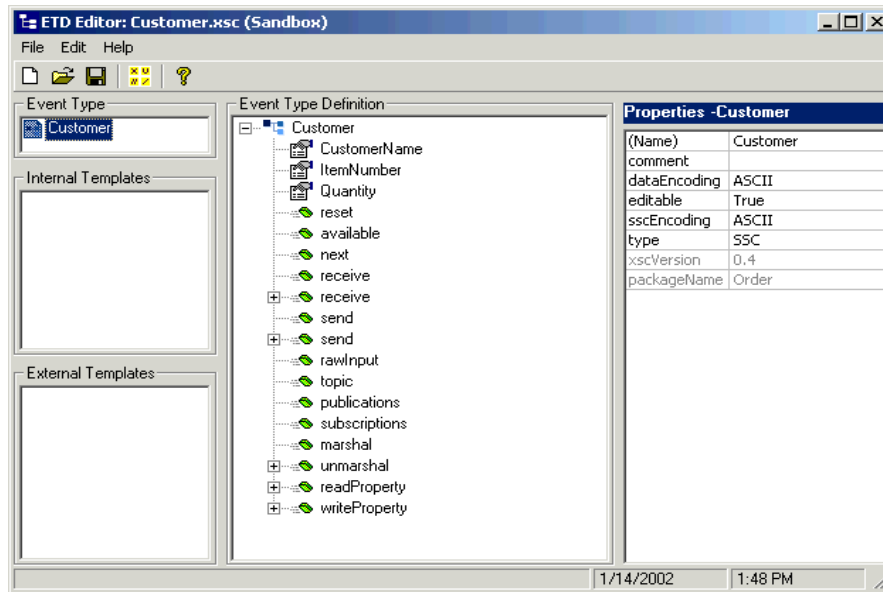
The input data file used in this example is shown in Figure 143. Place this data file at the directory location **c:\eGate\client\data\Order**.

**Figure 143** 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 149. Set the global delimiter to a ~ character. For more information on using the ETD Editor see the ETD Editor’s online help.

Figure 144 Input ETD: Customer.xsc (Java)



### Step 3: Create the START\_BP Collaboration

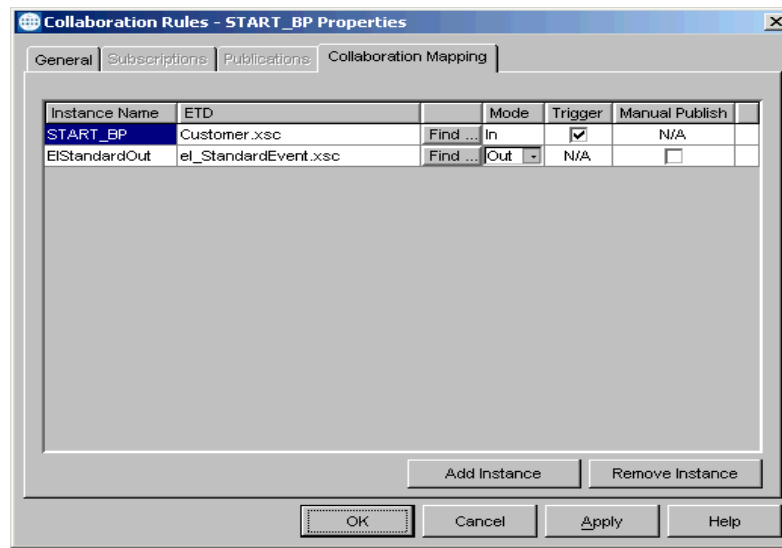
The Collaboration that sends the Event that starts the BPI must do two things:

- 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.

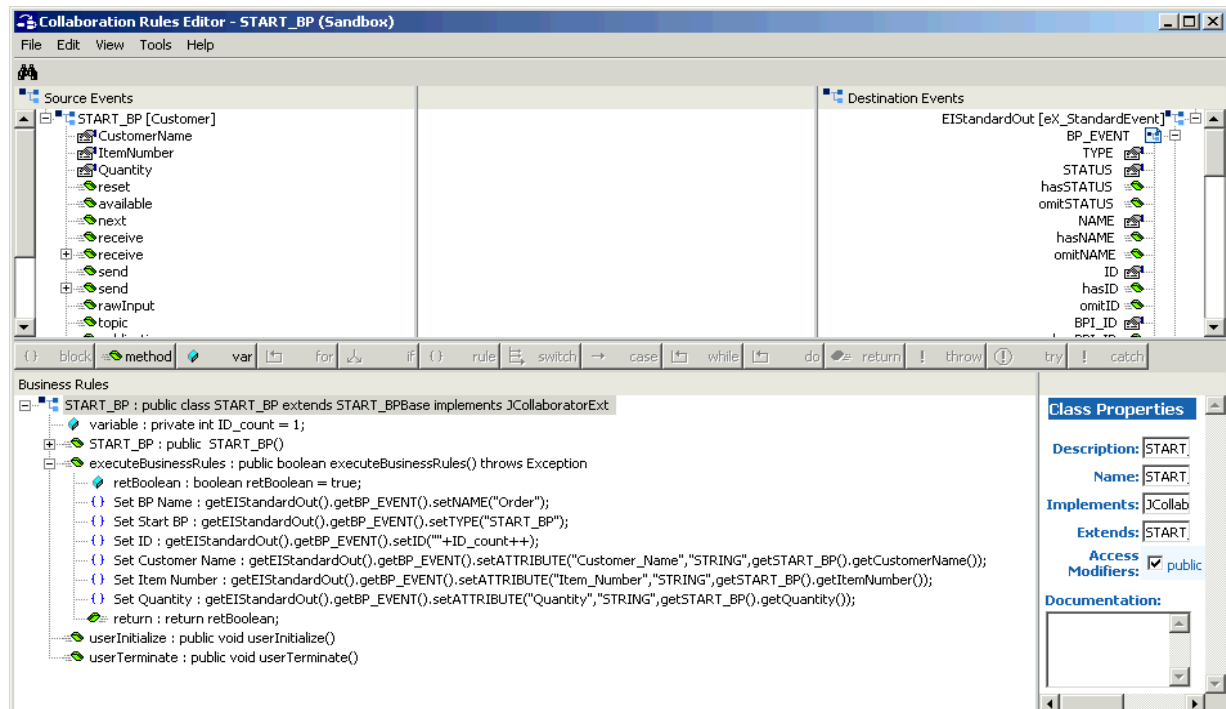
- 1 Create a Collaboration Rule, **START\_BP**, that uses the Java service.
- 2 Configure the Collaboration Mapping tab, as shown in Figure 145.

**Figure 145** 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 146.

**Figure 146** START\_BP CRS

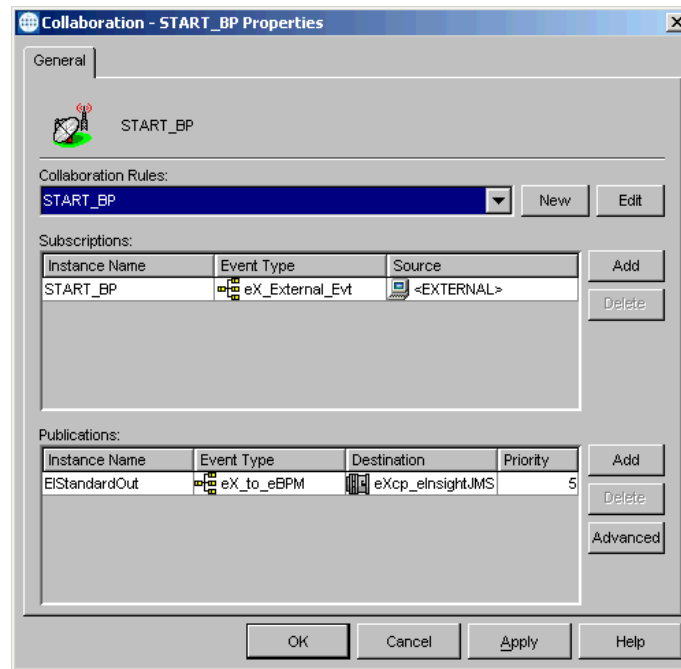


## Step 4: Configure the Collaboration in the GUI

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 Enterprise Manager GUI.

- 1 Create a Collaboration for the **Start\_BP** e\*Way configured as shown in Figure 147.

**Figure 147** Start\_BP Collaboration



## 16.6 Configure the Integration Schema (eISchema)

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

**Important:** Before you begin to make changes to the e\*Gate Registry, make a copy of the e\*Insight schema. See [“Copy the e\\*Insight Schema” on page 98](#) 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 56 to configure the e\*Gate schema that supports the example.

**Table 56** Integration Schema Activity Components

Name	Type	Participating Host	Active/Passive	Manual Restart	TimeOut
eX_Bill_Customer	BOB	localhost	Active	Yes	Not used
eX_Ship_Order	BOB	localhost	Active	Yes	Not used

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

## 16.7 Configure the e\*Insight Engine (eISchema)

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

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

### Edit the eX\_eBPM Engine’s Configuration File

Most of the parameter settings in the eX\_eBPM engine’s configuration file should not be changed. Use the e\*Way Editor and the information in [“Configuring the e\\*Insight Engine” on page 66](#) to make the required changes for the Order example.



---

## 16.8 Configure User-defined e\*Gate Components (eISchema)

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

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

### Configuration Order for the User-defined Components

- 1 Add and configure the **START\_BP** e\*Way.
- 2 Configure the Collaborations for the activity components running as BOBs.

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

#### 16.8.1 Configure the **START\_BP** e\*Way

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

Follow these steps to configure the **START\_BP** e\*Way:

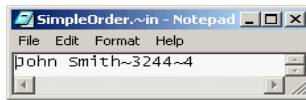
- 1 Create the Input ETD
- 2 Create the **START\_BP** Collaboration Rules script (CRS)
- 3 Add the e\*Way and create the e\*Way configuration file
- 4 Configure the Collaboration in the GUI

### Step 1: Create the Input ETD

The input ETD is based on the format of the input data. The Order example uses a delimited text file (**SimpleOrder.~in**) that contains the data needed to process the order.

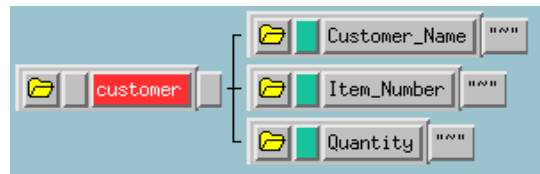
The input data file used in this example is shown in Figure 148. Place this data file at the directory location **c:\eGate\client\data\Order**.

**Figure 148** 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 149. For more information on using the ETD Editor see the ETD Editor's online help.

**Figure 149** 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 150 shows the **START\_BP** CRS used in the Order example:

**Figure 150** START\_BP CRS

Rules		Use Selected Nodes in New R
COPY	"PassiveOrder"	"output%X_Event.DS,eX_Event.CT.DSN,DS.BP_EVENT,AS.NAME,Value
COPY	"START_BP"	"output%X_Event.DS,eX_Event.CT.DSN,DS.BP_EVENT,AS.TYPE,Value
UNIQUE ID		"output%X_Event.DS,eX_Event.CT.DSN,DS.BP_EVENT,AS.ID,Value
FUNCTION	{eX-set-attribute "output%X_Event "Customer_Name" "input%customer.Customer_Name "STRING"}	
FUNCTION	{eX-set-attribute "output%X_Event "Item_Number" "input%customer.Item_Number "STRING"}	
FUNCTION	{eX-set-attribute "output%X_Event "Quantity" "input%customer.Quantity "STRING"}	

## Step 3: Add the e\*Way and Create the e\*Way Configuration File

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

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

**Table 57** Start e\*Way Parameters

Screen	Parameter	Setting
General Settings	(All)	(Default)
Outbound (send) settings	(All)	(Default)
Poller (inbound) settings	PollDirectory	c:\eGate\client\data\Order
	(All others)	(Default)
Performance Testing	(All)	(Default)

## Step 4: Configure the Collaboration in the GUI

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

- 1 Create a Collaboration Rule, **START\_BP**, that uses the Monk service and the **START\_BP** CRS created in step 2, subscribes to the **eX\_External\_Evt** Event Type, and publishes to the **eX\_to\_eBPM** Event Type.
- 2 Create a Collaboration for the **START\_BP** e\*Way that uses the **START\_BP** Collaboration Rule, subscribes to the **eX\_External\_Evt** Event Type from **<EXTERNAL>**, and publishes the **eX\_to\_eBPM** Event Type to the **eX\_eBPM** IQ.

### 16.8.2 Configure the Activity BOBs

You must complete these two tasks in order to configure the Activity BOBs:

- Set up the activity Collaborations in the e\*Gate GUI.
- Create the corresponding CRS used by the activity Collaborations.

**Note:** Any time you create e\*Ways, you must configure those e\*Ways before e\*Insight can communicate with external systems or components.

## Create the Activity BOB CRSs

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.

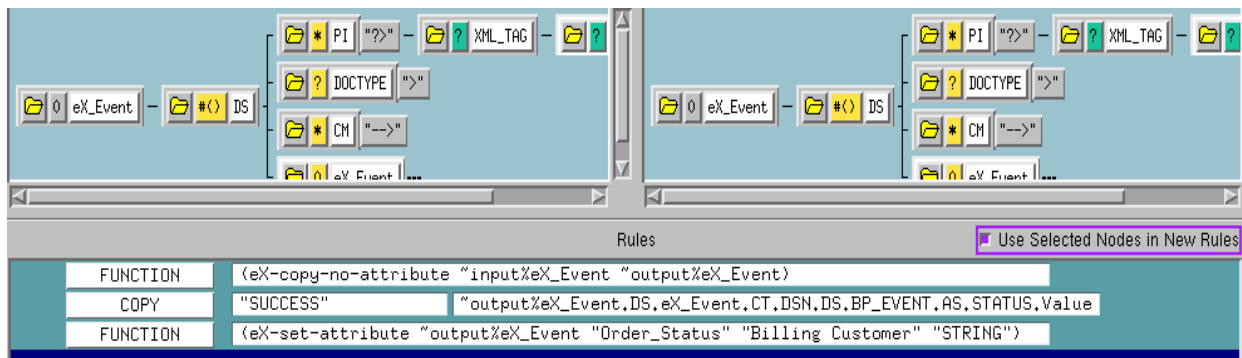
The CRSs associated with the BOBs used in this example are described in the following sections along with a screen capture showing the actual Monk Collaboration Rules Script used.

### 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 151, shows the **Billing\_Customer.tsc** CRS used in the example.

**Figure 151** Billing\_Customer.tsc CRS

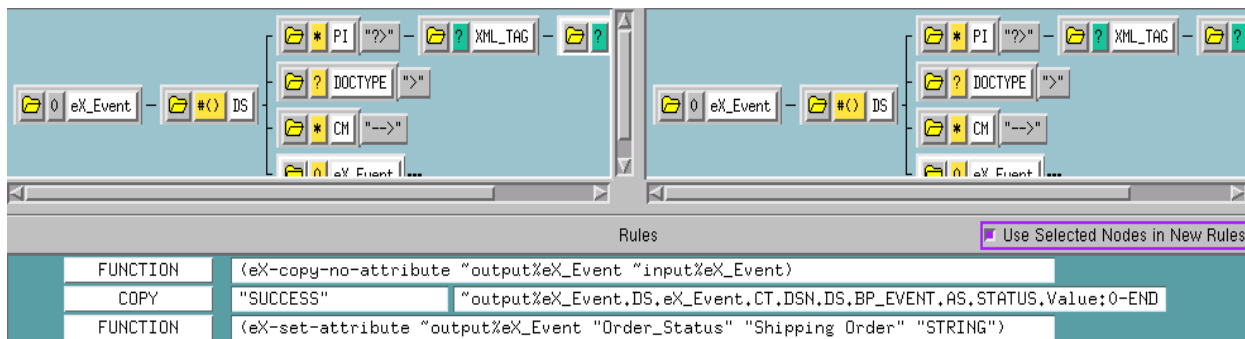


### Ship\_Order CRS

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 152 shows the Ship\_Order CRS used in the example.

**Figure 152** Ship\_Order.tsc CRS



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

Once you have created the CRS for a BOB, you must associate it with the corresponding Collaboration Rule in the e\*Gate GUI. For each BOB you must:

- 1 Highlight the BOB’s Collaboration.
- 2 Open the **Collaboration Properties** dialog box for the Collaboration.
- 3 Edit the Collaboration Rules.

- 4 Change the Service to **Monk**.
- 5 Find the corresponding “.tsc” file and associate it with the Collaboration Rule.
- 6 Click **OK** to continue.

---

## 16.9 Run and Test the e\*Insight scenario

Once the schema has been set up in e\*Gate you can run the scenario. Use the following procedure to test the functionality of the example.

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

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

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

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

Substitute the appropriate username and password for your installation.

- 4 Start the e\*Gate Monitor, and check the status of all the components. Any components displayed in red are not running. Any e\*Insight components that are not running should be investigated before feeding data into the system.
- 5 Navigate to the location for the input data file, **SimpleOrder.~in**, shown in [Figure 148 on page 266](#) (c:\eGate\client\data\Order) and change the extension to “.fin”.

*Note:* The change of the extension to “.~in” indicates that the data file has been picked up by the **START\_BP** e\*Way.

- 6 Switch to the e\*Insight GUI and, while in monitor mode, select the most recent BPI from the **List** pane. The **Diagram** should show the activities as completed (green). If the activities are not green then the e\*Gate component associated with that activity should be investigated for the cause of the problem.

---

## 16.10 Case Study - Passive Control Mode

We are now going to change the configuration of the Ship\_Order activity to use passive control, that is e\*Gate controls the Activity rather than e\*Insight. The e\*Gate configuration is modified so the Bill\_Customer collaboration publishes eX\_Ship\_Order\_Do in addition to returning a “Done” message to the e\*Insight engine. The Ship\_Order CRS is also modified to ensure that the “Done” message contains the correct information.

---

## 16.11 Passive Control Mode (eIJSchema)

### 16.11.1 Modify the Order BP in e\*Insight (eIJSchema)

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

- 1 Save the business process as a new version.
- 2 Change the Ship\_Order activity properties to use passive control.

### 16.11.2 Modify User-defined e\*Gate Components (eIJSchema)

#### Configuration Order for the User-defined Components

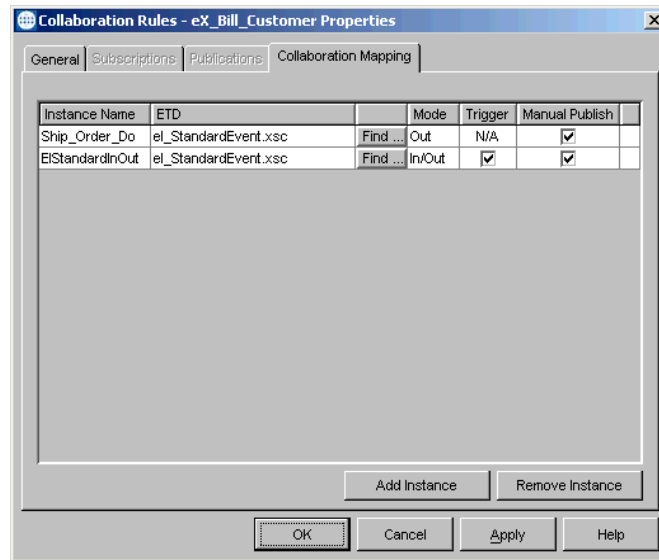
- 1 Modify the Bill\_Customer CRS.
- 2 Modify the Ship\_Order CRS.
- 3 Modify the Bill\_Customer Collaboration Rule and Collaboration.

#### Configure the Bill\_Customer Collaboration Rule and Collaboration

- 1 Update the Bill\_Customer Collaboration Rule with the following Collaboration Mapping.

**Note:** Both instances are now configured to publish the event manually.

Figure 153 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**.

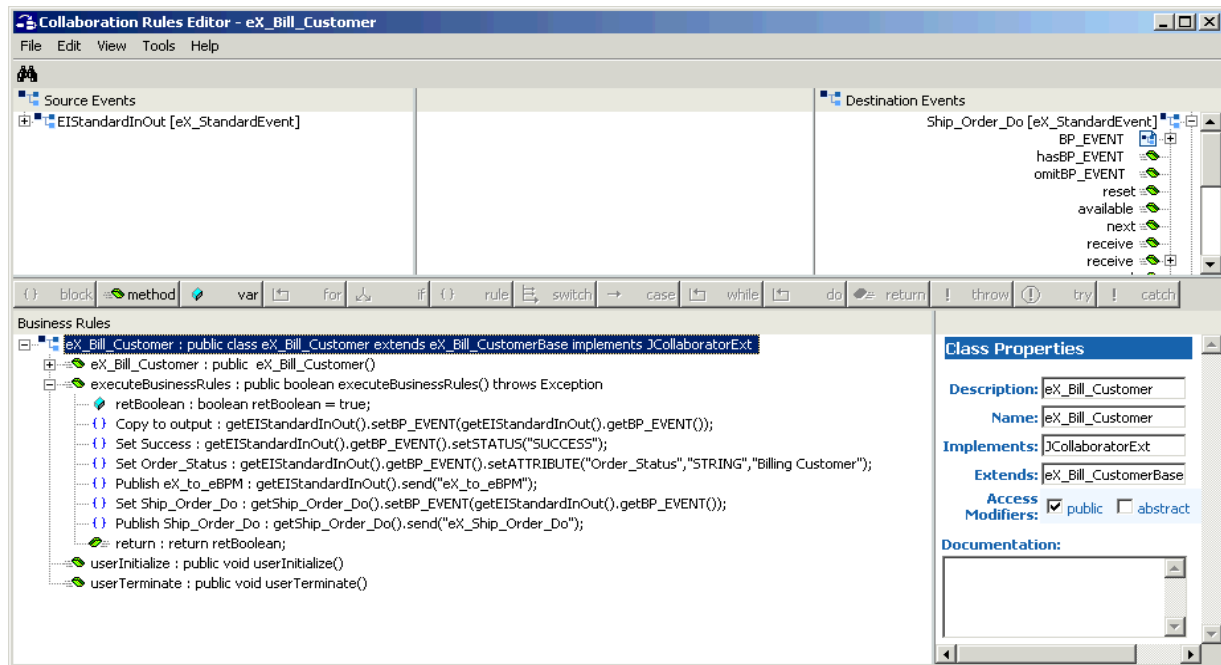
#### Bill Customer CRS (eJSchema)

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 154, shows the **Billing\_Customer.xpr** CRS used in the example.

Figure 154 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.

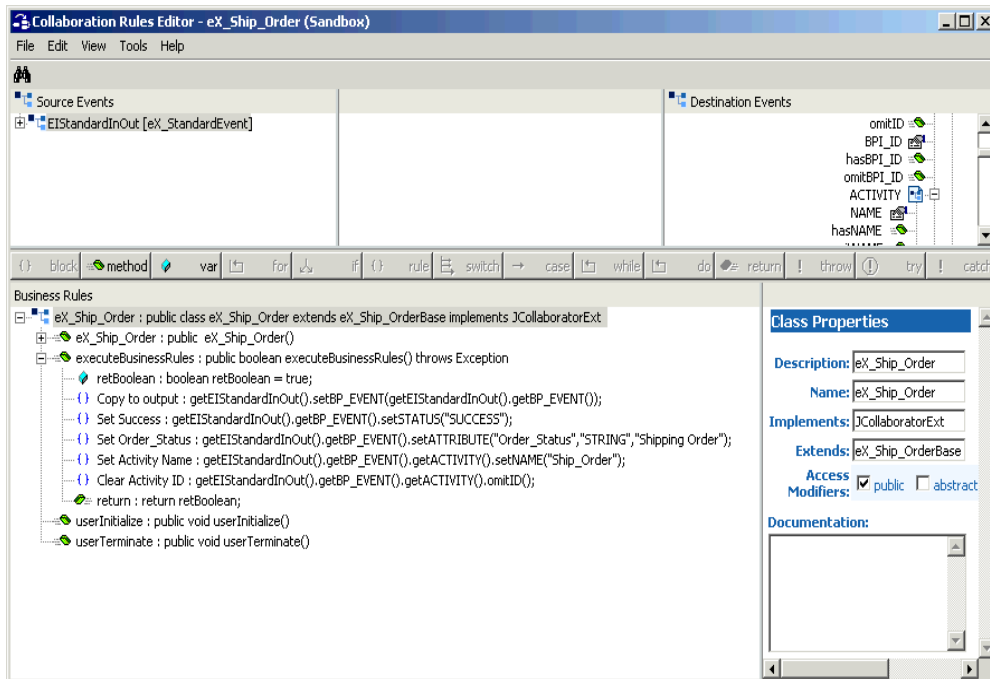
In the active example, the event is sent from the e\*Insight engine with information pertaining to the Ship\_Order activity. In the passive example, the event was originally created by the e\*Insight engine for the Bill\_Customer activity, and the eX\_Bill\_Customer Collaboration is configured to send 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 152 shows the Ship\_Order CRS used in the example.



Figure 155 Ship\_Order.xpr CRS



### 16.11.3 Run and Test the e\*Insight scenario

Run the Schema again as described in [“Run and Test the e\\*Insight scenario” on page 269](#).

## 16.12 Passive Control Mode (eISchema)

This section describes the steps for updating a schema based on eISchema (Classic). For information on eIJSchema, see [“Passive Control Mode \(eIJSchema\)” on page 270](#).

### 16.12.1 Modify the Order BP in e\*Insight (eISchema)

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

- 1 Save the business process as a new version.
- 2 Change the Ship\_Order activity properties to use passive control.

### 16.12.2 Modify User-defined e\*Gate Components (eISchema)

#### Configuration Order for the User-defined Components

- 1 Modify the Bill\_Customer CRS.
- 2 Modify the Ship\_Order CRS.
- 3 Modify the Bill\_Customer Collaboration Rule and Collaboration.

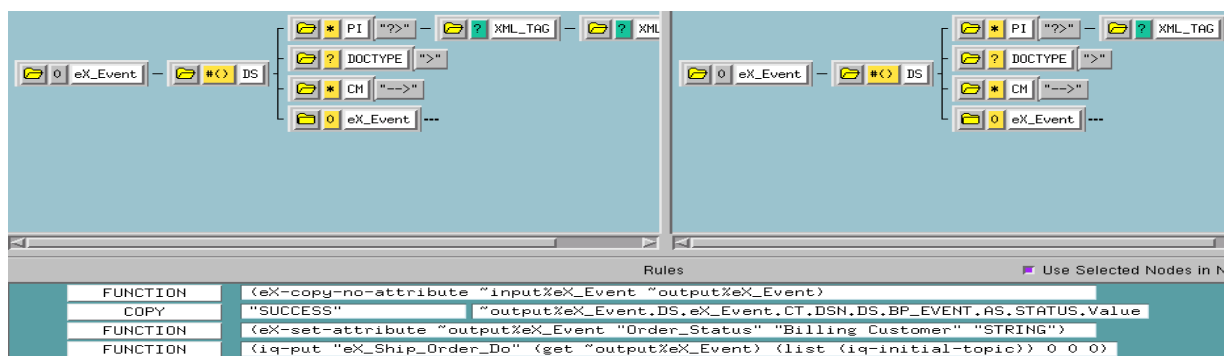
#### Bill Customer CRS

The CRS is already defined to set the value of the Order\_Status attribute to “Billing Customer” and send a “SUCCESS” Event back to the e\*Insight, indicating that the activity has completed successfully.

The script is modified to use an iq-put to publish eX\_Ship\_Order\_Do.

Figure 154, shows the **Billing\_Customer.tsc** CRS used in the example.

**Figure 156** Billing\_Customer.tsc CRS



#### Ship\_Order CRS

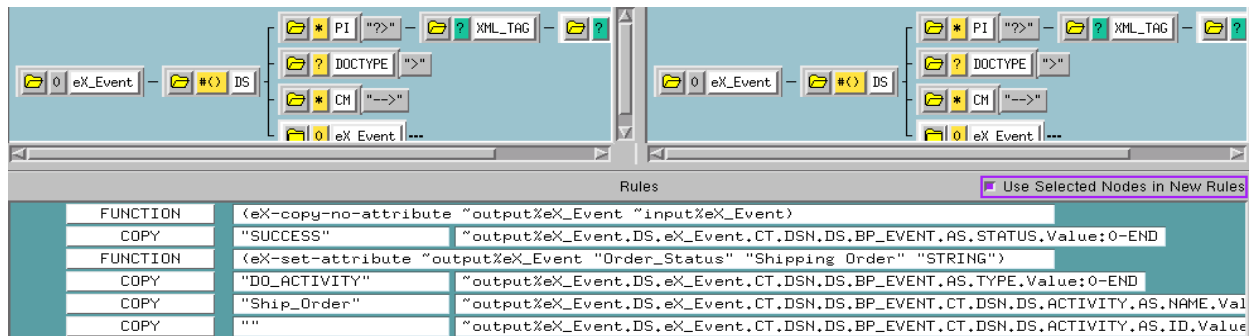
The Ship\_Order translation simulates the activity of sending out an item that is in stock. It sets the value of the Order\_Status attribute to a short message indicating that the order has been sent and returns “SUCCESS” to the e\*Insight engine.

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 152 shows the Ship\_Order CRS used in the example.

**Figure 157** 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**.

### 16.12.3 Run and Test the e\*Insight scenario

Run the Schema again as described in [“Run and Test the e\\*Insight scenario” on page 269](#).

# e\*Insight Performance

The purpose of this chapter is to describe methods that can be used to improve performance. The chapter is divided into three sections; the first describes the performance enhancements that can be made to a schema based on eIJSchema (Java). The second describes the performance enhancements that can be made to a schema based on eISchema (Classic). The last section describes performance enhancements that are not schema specific.

---

## 17.1 Performance Improvements Using eIJSchema

The purpose of this section is to describe methods that can be used to improve performance of the eIJSchema.

- Instance caching
- Using multiple engines
- Using instance caching with multiple engines
- Using binary XML
- Configuring e\*Insight to ignore e\*Xchange ETD
- Changing the Event Type “get” Interval

### 17.1.1 Instance Caching

Instance Caching is the most efficient way to process Business Process Instances. Using instance caching keeps a cache of the instance information throughout the life span of the Business Process Instance. If instance caching is not used the instance information is retrieved from the database instead.

#### To configure the engine to use instance caching

- 1 In the **e\*Way Connections** folder, open the engine’s e\*Way connector properties.
- 2 Click **Edit** to open the engine’s configuration file.
- 3 In the **eBPM Setting** section, set **Instance Caching** to **YES**.

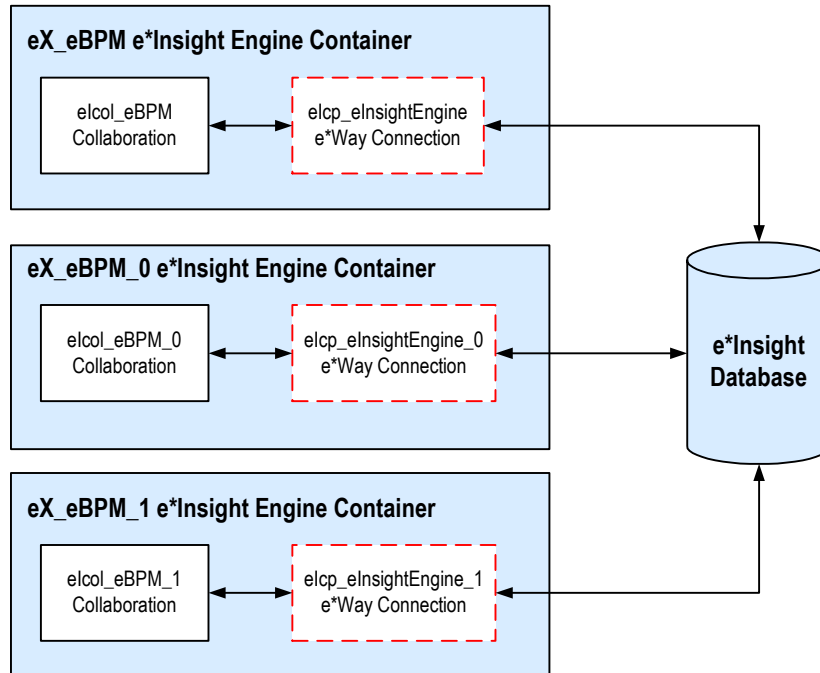
## 17.1.2 Using Multiple e\*Insight Engines

An e\*Insight engine is comprised of a specially configured Collaboration (eIcol\_eBPM) and the e\*Insight e\*Way Connection (eIcp\_eInsightEngine). The e\*Insight engine runs within a Multi-Mode e\*Way (eX\_eBPM), which is referred to as the e\*Insight Engine container. You can use multiple e\*Insight engines to increase performance. This section describes how to add and configure these. You can use instance caching to further improve performance. See “[e\\*Insight Engine Affinity \(eIJSchema\)](#)” on page 280 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 158](#) shows a scenario where three e\*Insight engine containers are used, each with a single engine.

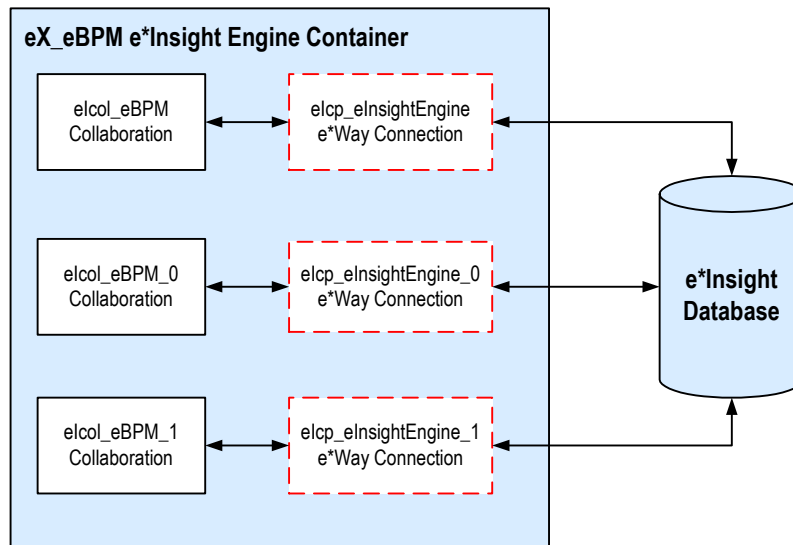
**Important:** A single e\*Insight e\*Way connection cannot be used by multiple Collaborations.

**Figure 158** Multiple Engine Containers with a Single Engine



Alternatively, you can have multiple e\*Insight engines in a single e\*Insight engine container. See [Figure 159](#).

**Figure 159** 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.

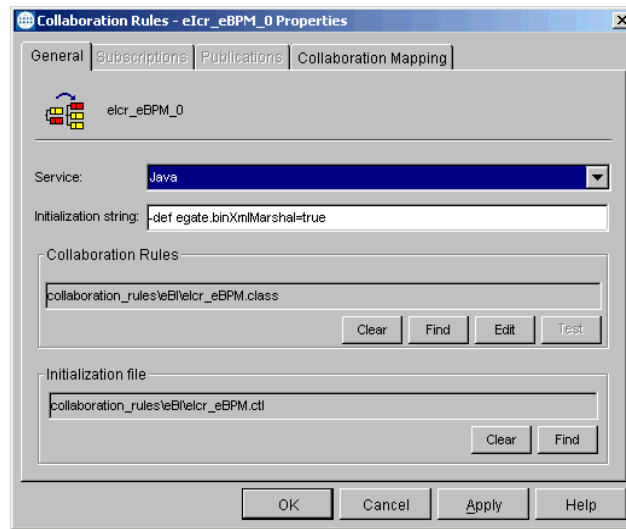
#### To add and configure an e\*Insight e\*Way Connection

- 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 34](#).

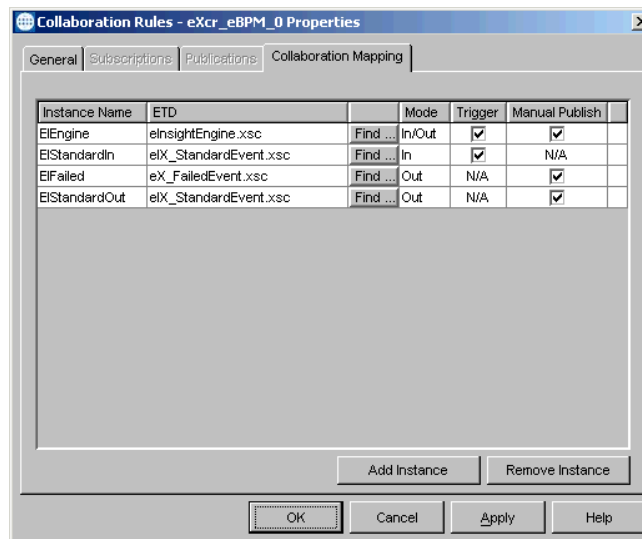
#### To configure an e\*Insight engine container for a new e\*Way Connection

- 1 Create a new e\*Way Connection. See ["To add and configure an e\\*Insight e\\*Way Connection" on page 278](#).
- 2 Add a Collaboration Rule and configure it as shown in Figure 160.

**Figure 160** e\*Insight Engine General Tab



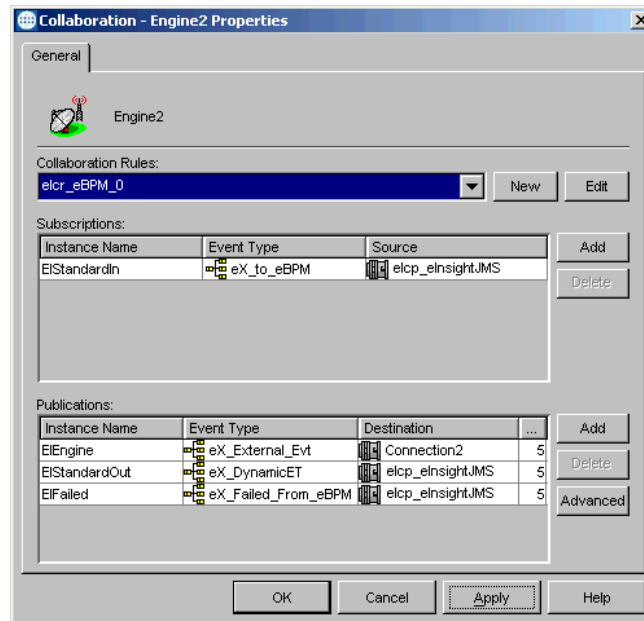
**Figure 161** e\*Insight Engine Collaboration Mapping Tab



**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 162. You should configure the Collaboration to publish to the e\*Way Connection created in step 1.

**Figure 162** e\*Insight Engine Collaboration



### 17.1.3 e\*Insight Engine Affinity (eIJSchema)

e\*Insight Engine Affinity allows e\*Insight engines in a multi-engine environment to cache information about particular Business Process Instances as they flow through the e\*Gate schema using Instance Caching. Since the engines hold the instance information in a cache, it is essential that an individual instance is always processed by the same engine or Collaboration. This is achieved by setting a JMS property in e\*Gate 4.5.2, or using a different Event Type for each engine in e\*Gate 4.5.1.

Using Engine Affinity can possibly improve the overall message throughput but if an engine is shut down for some reason, the instances associated with that engine do not finish being processed until the engine is manually restarted using the e\*Gate Monitor.

### 17.1.4 Using Engine Affinity with e\*Gate 4.5.2 or 4.5.3 — 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 .

**Note:** *The message selector for the inbound eX\_StandardEvent is only set if you are using e\*Gate 4.5.2 and are subscribing to a JMS e\*Way Connection. Otherwise, you must configure Engine Affinity using different Event Types, see [“Using Engine Affinity with e\\*Gate 4.5.1 — Active Mode”](#) on page 281.*



### To configure multiple engines to use e\*Insight Engine Affinity

- 1 Create multiple engines. See [“Using Multiple e\\*Insight Engines” on page 277](#).
- 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:

```
getEISstandardOut.prepareReplyToSender(getEISstandardIn());
```

*Note:* If you are using one instance (such as EISstandardInOut), you do not need to use prepareReplyToSender() since the Engine Affinity JMS Properties already exist.

For backwards compatibility, if the first Business Process Activity is passive, the e\*Insight engine still echoes back the incoming START\_BP message for synchronization. To disable this feature, add another System Property definition, **ei.pasvNoEcho=true**, in the Initialization string of the e\*Insight Engine Collaboration Rules.

### 17.1.5 Using Engine Affinity with e\*Gate 4.5.1 – Active Mode

The procedures described in the next section use unique names for the Event Types based on the Collaboration name. You can alternatively use the e\*Way name if the e\*Way uses a single Collaboration. If multiple Collaborations are used then using the e\*Way name does not provide a unique name for each engine. For information on configuring the e\*Way Connection to use the e\*Way name, see [“Using XA” on page 34](#).

### To configure multiple engines to use e\*Insight Engine Affinity

- 1 Create multiple engines. See [“Using Multiple e\\*Insight Engines” on page 277](#).
- 2 Edit the configuration file for each e\*Insight engine. In the **eBPM Settings**, set the **Instance Caching** parameter to **Yes**.

*Note:* The engines can refer to the same configuration file.

- 3 Ensure that every Collaboration uses a unique Collaboration Rule.

- 4 Create an Event Type named **eX\_to\_<Collaboration Name>** for each e\*Insight engine Collaboration.

**Important:** *An Event Type must be created for the default engine named **eX\_to\_col\_eBPM** in addition to the Event Types required for additional engines.*

- 5 For every e\*Way connection, update the Collaboration that subscribes to eX\_eBPM using the following procedure. Replace <Collaboration name> with the appropriate name.
  - ♦ In the Collaboration properties Subscriptions box, change the **Event Type** to **eX\_to\_<Collaboration name>** with Source **eIcp\_eInsightJMS**.
- 6 For every *Activity* e\*Way or BOB, update the Collaboration that publishes eX\_to\_eBPM with the additional publications for the additional engines.
  - ♦ In the Collaboration properties Subscriptions box, add an entry for every e\*Way Connection Collaboration with the following properties: **EIStandardInOut** instance, **eX\_to\_<Collaboration name>** Event Type, with Destination **eIcp\_eInsightJMS**.

**Note:** *The instance name may be different, depending on your implementation.*

## 17.1.6 Using Engine Affinity with e\*Gate 4.5.2 or 4.5.3 — 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 277](#).
- 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.

## 17.1.7 Using Engine Affinity with e\*Gate 4.5.1 — 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 Collaboration uses unique names for the Event Types based on the Collaboration name for each e\*Insight engine. Each e\*Insight engine is configured to only subscribe to the Event Type based on its name, ensuring 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 277](#).
- 2 Edit the configuration file for each e\*Insight engine. In the **eBPM Settings**, set the **Instance Caching** parameter to **Yes**.

**Note:** *The engines can refer to the same configuration file.*

- 3 Ensure that every Collaboration uses a unique Collaboration Rule.
- 4 Create an Event Type named **eX\_to\_<Collaboration Name>** for each e\*Insight engine Collaboration.

**Important:** *An Event Type must be created for the default engine named **eX\_to\_col\_eBPM** in addition to the Event Types required for additional engines.*

- 5 For every e\*Way connection, update the Collaboration that subscribes to eX\_eBPM using the following procedure. Replace <Collaboration name> with the appropriate name.

- ♦ In the Collaboration properties Subscriptions box, change the **Event Type** to **eX\_to\_<Collaboration name>** with Source **eIcp\_eInsightJMS**.
- 6 For every *Activity* e\*Way or BOB, update the Collaboration that publishes eX\_to\_eBPM with the additional publications for the additional engines.
    - ♦ In the Collaboration properties Subscriptions box, add an entry for every e\*Way Connection Collaboration with the following properties: **EIStandardInOut** instance, **eX\_to\_<Collaboration name>** Event Type, with Destination **eIcp\_eInsightJMS**.
  - 7 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.

### 17.1.8 Using e\*Xchange with e\*Insight (eIJSchema)

The Event Type Definition used in the engine's Collaboration Rules Script, eIX\_StandardEvent.xsc contains two sections, BP\_EVENT, containing information for e\*Insight and TP\_EVENT, containing information for e\*Xchange. If you are not using the e\*Xchange section of the ETD, then you can configure the engine ignore this section, reducing processing.

To configure the engine to ignore e\*Xchange section of the ETD

- 1 Select the e\*Way Connections folder.
- 2 Open the properties of the e\*Insight engine e\*Way Connector.
- 3 Click **Edit**.
- 4 Go to the **eBPM Settings** section, and find **Using e\*Xchange with e\*Insight**.
- 5 Click **No**.
- 6 Close the configuration editor and save the changes.

### 17.1.9 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.*

## 17.1.10 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 286](#)) 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 Enterprise Manager does not allow the use of commas (,) since the e\*Gate Registry uses this delimiter internally. Special characters can be specified by using standard escape codes (such as `\t`, `\n`) or Unicode escape codes (such as `\u003d` for '=').

### Configuring a Separate Collaboration for Do and Undo Events

To remove the requirement for a single Collaboration to subscribe to both “Do” and “Undo” Events, separate the “Do” and “Undo” logic in your Collaboration Rules script. Then create a “Do” and an “Undo” Collaboration, each subscribing to a single Event Type.

## Removing Unnecessary Subscriptions

Subscribing to multiple Event Types is less efficient than subscribing to a single Event Type. If you have not implemented “undo” logic in your business process, then remove the subscriptions to the eX\_<Activity\_Name>\_Undo Event Types.

### 17.1.11 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, then 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).

### 17.1.12 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).

### 17.1.13 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.*

## 17.2 Performance Improvements Using eISchema

The purpose of this section is to describe methods that can be used to improve performance of the eISchema.

- Instance caching
- Using multiple engines
- Using instance caching with multiple engines
- Setting the Exchange Data Interval

### 17.2.1 Instance Caching

Instance Caching is the most efficient way to process Business Process Instances. Using instance caching keeps a cache of the instance information throughout the life span of the Business Process Instance. If instance caching is not used the instance information is retrieved from the database instead. This allows more flexibility and fault tolerance at the cost of performance.

To configure the engine to use instance caching

- 1 Open the engine's properties.
- 2 Click **Edit** to open the engine's configuration file.
- 3 In the **eBPM Setting** section, set **Instance Caching** to **YES**.

### 17.2.2 Using Multiple e\*Insight Engines (eISchema)

You can use multiple e\*Insight engines to increase performance. This section describes how to add and configure additional engines. You can use instance caching to further improve performance. See [“e\\*Insight Engine Affinity \(eISchema\)” on page 288](#) for more information.

You can either create a new engine from scratch, or copy the engine and collaborations. Both procedures are described below.

To create a new e\*Insight engine

- 1 Add a new e\*Insight to the participating host.
- 2 Edit the engine's configuration file.
- 3 Add a Collaboration Rule that subscribes to **eX\_External\_Evt** and publishes **eX\_Failed\_From\_eBPM**.
- 4 Add a Collaboration Rule that subscribes to **eX\_to\_eBPM** and publishes **eX\_External\_Evt**.
- 5 Add a Collaboration to the e\*Insight engine that uses the Collaboration Rule created in step 3. Configure to subscribe to **External** and publish to **eX\_Dead\_Letter\_Queue** IQ.

- 6 Add a Collaboration to the e\*Insight engine that uses the Collaboration Rule created in step 3. Configure to subscribe to **eX\_eBPM** and publish to **External IQ**.

#### To copy an existing engine

- 1 Copy the e\*Insight engine.
- 2 Copy the eX\_to\_eBPM Collaboration.
- 3 Copy the eX\_from\_eBPM Collaboration.

**Important:** *When you copy the above components some elements are then used by both engines. These include the e\*Insight engine configuration file and the Collaboration Rules. If you need to change these for one engine, but not the other, you must create a new version.*

### 17.2.3 e\*Insight Engine Affinity (eISchema)

e\*Insight Engine Affinity allows e\*Insight engines in a multi-engine e\*Gate schema to cache information about particular Business Process Instances as they flow through the e\*Gate schema using Instance Caching. Using Engine Affinity can possibly improve the overall message throughput but if an engine is shutdown for some reason, the instances associated with that engine do not finish being processed until the engine is manually restarted using the e\*Gate Monitor.

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 163.

#### 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 163.

**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 163.

**Figure 163** Monk Collaboration Rule Script for Engine Affinity

FUNCTION	{eX-event-sendback-to-sender ~output%eX_Event}
COPY	"" ~output%eX_Event:0-END

## 17.2.4 Exchange Data Interval (elSchema)

The e\*Insight engine uses the interval to determine how often to poll the control table. The control table contains entries for batch tasks, such as manual restarts, and also contains entries for instances using the User Activity or Authorization Activity. Depending on your implementation of e\*Insight, it may be desirable to reduce the polling frequency to increase performance.

The Exchange Data Interval is set in the e\*Way Configuration file, **Communication Setup** section.

## 17.2.5 Review JVM Settings

You may be able to improve performance by changing the memory allocated for the Java Virtual Machine. This is set in the e\*Insight engine configuration file, **JVM Settings, Maximum Heap Size**. If this is left set to zero (0), the preferred value for the maximum heap size of the Java VM is used.

---

## 17.3 General e\*Insight Performance Tips

This section describes some ways of improving performance that are not specific to the type of e\*Insight schema that you are running.

- Review the *SeeBeyond eBusiness Integration Suite Deployment Guide*.
- Use a third party tool to determine if hardware is the limitation.
- Enhance database access by one of the following:
  - A Use RAID/stripe disks with a multi-controller.
  - B Install the database and e\*Insight engines on the same machine.
  - C Tune the database.
- Use a Model Specific database — The Model Specific database uses a different structure for storing the attribute values, which increases performances. If you are using the Model Specific database, you can modify the database to your specific data requirements by controlling the size allocated to each attribute value. You need to create the necessary database tables before running your e\*Insight schema. Every business process version uses its own set of tables.

See the *e\*Insight Business Process Manager User's Guide* for information on creating the Model Specific database.

- Do not automatically reload models — The Auto Model Reload engine configuration parameter determines if the engine dynamically loads an enabled Business Process Version if the enabled/disabled status of Business Process Version changes. If the value is set to YES then Business Process Versions that are enabled or disabled while the engine is running are immediately recognized. However, setting this value to YES may degrade performance.
- Do not preload unnecessary Business Processes — The Business Processes to Preload engine configuration parameter allows you to load all or a subset of all the business processes stored in the e\*Insight database. The default setting is ALL. Naming specific business processes to preload may improve performance if you have a large number of business processes defined.
- Do not include input only attributes in the “Done” Event — To simplify your Collaboration Rules script, you may decide to copy everything from the source ETD to the destination ETD, especially when using Java. This copies all the input attributes to the destination ETD and, unless they are removed, they are included in the “Done” Event. This requires additional processing by the engine, as it tries to write the attribute to the database but is refused permission.
- 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.

# Troubleshooting

One of the easiest ways to debug your e\*Gate configuration is through the use of log files. All executable components—BOBs, e\*Ways, IQ Managers, and Control Brokers—have the ability to create log files that contain whatever level of debugging information you select.

---

## 18.1 Log Files

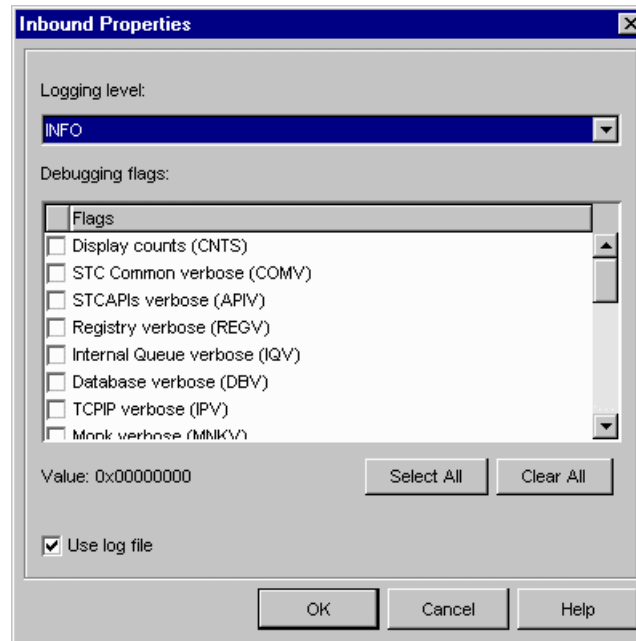
All log files are stored in the `\eGate\client\logs` directory on the Participating Host running the elements that generate the log entries. Logs are named after the component that creates them; for example, the **eX\_eBPM** engine creates a log file called **eX\_eBPM.log**.

### 18.1.1 Generating Log Files

To configure a component to generate a log file:

- 1 In the e\*Gate Enterprise Manager window, select the component that you want to configure and display its properties.
- 2 Select the **Advanced** tab, and then click **Log**.
- 3 Select the desired logging options (see Figure 164).

Figure 164 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 58** 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 eInsight process. The e\*Way engine should use this trace.

#### 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. The e\*Way engine should use this trace. 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.

e\*Insight Trace Example Code:

```
if ( Trace.isEnabled())
    Trace.setTrace(Trace.DEBUG, " Processing message. ");
```

e\*Gate Trace Example Code:

```
EGate.traceln(EGate.TRACE_EWAY, EGate.TRACE_INFORMATION,
"*** Successfully connected to eBPM Backend DataBase! ***");
```

## 18.2 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 59 and Table 60 show a number of common problems and suggested actions you should take. The tables also shows where to look for the error message and the logging required to display the error.

**Table 59** Common errors (elJSchema)

Error Message	Debug Flag	Problem	Resolution
JMSEException: Could not connect to host: [hostname], port: 24053	COL	elcp_eInsightJMS e*Way connection is configured incorrectly	In the elcp_eInsightJMS connection configuration file, <b>Message Service</b> section, check the value for <b>Server Name</b> and <b>Host Name</b> .
Cannot connect to e*Insight Backend DataBase: I/O exception: Invalid number format for port number	EWY	JDBC URL incorrect	In the elcp_eInsightEngine configuration file, <b>eBPM Settings</b> section, check the value defined for <b>JDBC URL String</b> .
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 elcp_eInsightEngine configuration file, <b>eBPM Settings</b> section, check the value defined for <b>JDBC URL String</b> .
Cannot connect to e*Insight Backend DataBase: ORA-01017: invalid username/ password; logon denied	EWY	Incorrect user name or password	In the elcp_eInsightEngine configuration file, <b>eBPM Settings</b> section, check the values defined for <b>Database User Name</b> and <b>Encrypted Password</b> .

**Table 59** Common errors (eJSchema)

Error Message	Debug Flag	Problem	Resolution
startActivity() failed: Definition not found for :Payrolls	EWY	Incorrect business process name	Check that the business process name defined in the CRS matches the business process name defined in e*Insight exactly. Also, check that a business process version has been enabled for this business process. Note: Although the error message appears in the eX_eBPM log, the problem is in the component that sent the message to the e*Insight engine.
startActivity() failed: Invalid BP_EVENT type:STARTBP	EWY	Incorrect BP_EVENT type	Check that the BP_EVENT type defined in the CRS is correct. Note: Although the error appears in the eX_eBPM log, the problem is in the component that sent the message to the e*Insight engine.
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 60** Common errors (eISchema)

Error Message	Debug Flag	Problem	Resolution
eX_eBPM (Fatal): ewjx: the "JNI DLL" (<jvm path>) specified is not a Java 2 version	EBPM	jvm configuration incorrect	In the eX_eBPM configuration file, Java VM Configuration section, check the value defined for JNI DLL.
eX_eBPM (Warning): Cannot connect to eBPM Backend DataBase: 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. bpold not found for :fred	EWY		

**Table 60** Common errors (eISchema)

Error Message	Debug Flag	Problem	Resolution
>>>>MONKEXCEPT:0069: throw: eBPM: Cannot process event	MNK	Incorrect BP_EVENT type	Check that the BP_EVENT type defined in the CRS is correct. Note: Although the error appears in the eX_eBPM log, the problem is in the component that sent the message to the e*Insight engine.
ewjx: Exception getMessage(): processOutgoing(): eBPM cannot process XML data: Invalid BP_EVENT type:START_BD	EWY		
>>>>MONKEXCEPT:0069: throw: eBPM: Cannot process event	MNK	No ID set	Check that the ID is defined in the CRS. Note: Although the error appears in the eX_eBPM log, the problem is in the component that sent the message to the e*Insight engine.
ewjx: Exception getMessage(): processOutgoing(): eBPM cannot process XML data: ORA-01407: cannot update ("EX_ADMIN"."BUSINESS_PR OCESS_INSTANCE"."BPI_NM ") to NULL	EWY		
ERROR - CONTINUING: BP attribute not found :Cust_Addres in createBPI() bpild:1016	EBPM	Attribute does not exist in e*Insight	Check that the attribute name matches the name defined in the CRS. Note: Although the error appears in the eX_eBPM log, the problem is in the component that sent the message to the e*Insight engine.
(get (eX-get-attribute ~input%eX_Event "Item_Number")) >>>>MONKEXCEPT:0009: get: argument 1 must be a valid path.	MNK	Attribute does not exist in the message sent to an e*Way or BOB for an activity	Check that the attribute is defined as an input attribute for the activity. Also, check that the attribute name matches the name defined in the CRS. Note: This error appears in the log for the e*Way or BOB for the activity.

---

## 18.3 General Troubleshooting Tips

### 18.3.1 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.

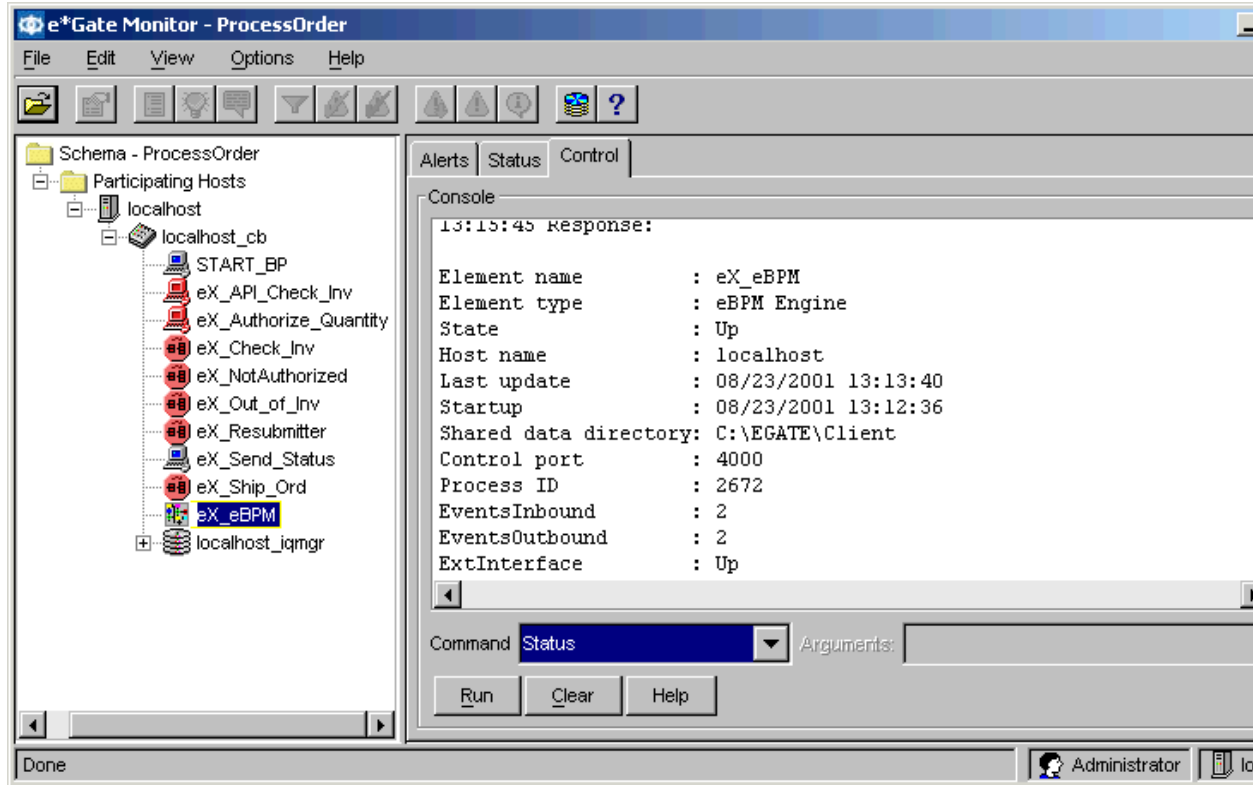
Use the e\*Gate Enterprise Monitor GUI to check whether a message has been processed by a component. The number of inbound and outbound messages for a component is displayed by the status command.

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

Figure 165 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".

### 18.3.2 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 **200108231344480169**, and there are a number of global attributes defined (namely, **Cust\_Address**, **Cust\_Name**, **Cust\_email**, **Item\_Description**, **Item\_Number**, and **Order\_Quantity**).

```
13:44:48.219 IQV D 2488 (iqput.cxx:114): sending to iq manager Data Follows (bytes 581):
```

<pre> 3C 65 58 5F 45 76 65 6E 74 3E 0A 3C 42 50 5F 45 56 45 4E 54 0A 54 59 50 45 3D 22 53 54 41 52 54 5F 42 50 22 0A 49 44 3D 22 32 30 30 31 30 38 32 33 31 33 34 34 34 38 30 31 36 39 22 0A 4E 41 4D 45 3D 22 50 72 6F 63 65 73 73 4F 72 64 65 72 22 3E 0A 3C 41 54 54 52 49 42 55 54 45 0A 56 41 4C 55 45 3D 22 34 30 34 20 45 2E 20 48 75 6E 74 69 6E 67 74 6F 6E 20 44 72 2E 22 0A 54 59 50 45 3D 22 53 54 52 49 4E 47 22 0A 4E 41 4D 45 3D 22 43 75 73 74 5F 41 64 64 72 65 73 73 22 3E 3C 2F 41 54 54 52 49 42 55 54 45 3E 0A 3C 41 54 54 52 49                 </pre>	<pre> &lt;eX_Event&gt;.&lt;BP_E VENT.TYPE="START _BP".ID="2001082 31344480169".NAM E="ProcessOrder" &gt;.&lt;ATTRIBUTE.VAL UE="404 E. Hunti ngton Dr.".TYPE= "STRING".NAME="C ust_Address"&gt;&lt;/A TTRIBUTE&gt;.&lt;ATTRI                 </pre>
--	--

```

42 55 54 45 0A 56 41 4C 55 45 3D 22 4A 6F 68 6E
20 53 6D 69 74 68 22 0A 54 59 50 45 3D 22 53 54
52 49 4E 47 22 0A 4E 41 4D 45 3D 22 43 75 73 74
5F 4E 61 6D 65 22 3E 3C 2F 41 54 54 52 49 42 55
54 45 3E 0A 3C 41 54 54 52 49 42 55 54 45 0A 56
41 4C 55 45 3D 22 6A 73 6D 69 74 68 40 73 65 65
62 65 79 6F 6E 64 2E 63 6F 6D 22 0A 54 59 50 45
3D 22 53 54 52 49 4E 47 22 0A 4E 41 4D 45 3D 22
43 75 73 74 5F 65 6D 61 69 6C 22 3E 3C 2F 41 54
54 52 49 42 55 54 45 3E 0A 3C 41 54 54 52 49 42
55 54 45 0A 56 41 4C 55 45 3D 22 4D 69 6C 6C 65
6E 6E 69 75 6D 20 50 65 74 20 52 6F 63 6B 22 0A
54 59 50 45 3D 22 53 54 52 49 4E 47 22 0A 4E 41
4D 45 3D 22 49 74 65 6D 5F 44 65 73 63 72 69 70
74 69 6F 6E 22 3E 3C 2F 41 54 54 52 49 42 55 54
45 3E 0A 3C 41 54 54 52 49 42 55 54 45 0A 56 41
4C 55 45 3D 22 33 33 33 33 33 22 0A 54 59 50 45
3D 22 53 54 52 49 4E 47 22 0A 4E 41 4D 45 3D 22
49 74 65 6D 5F 4E 75 6D 62 65 72 22 3E 3C 2F 41
54 54 52 49 42 55 54 45 3E 0A 3C 41 54 54 52 49
42 55 54 45 0A 56 41 4C 55 45 3D 22 31 22 0A 54
59 50 45 3D 22 4E 55 4D 42 45 52 22 0A 4E 41 4D
45 3D 22 4F 72 64 65 72 5F 51 75 61 6E 74 69 74
79 22 3E 3C 2F 41 54 54 52 49 42 55 54 45 3E 3C
2F 42 50 5F 45 56 45 4E 54 3E 3C 2F 65 58 5F 45
76 65 6E 74 3E

```

```

BUTE.VALUE="John
Smith".TYPE="ST
RING".NAME="Cust
_Name"></ATTRIBU
TE>.<ATTRIBUTE.V
ALUE="jsmith@see
beyond.com".TYPE
="STRING".NAME="
Cust_email"></AT
TRIBUTE>.<ATTRIB
UTE.VALUE="Mille
nium Pet Rock".
TYPE="STRING".NA
ME="Item_Descrip
tion"></ATTRIBUT
E>.<ATTRIBUTE.VA
LUE="33333".TYPE
="STRING".NAME="
Item_Number"></A
TTRIBUTE>.<ATTRI
BUTE.VALUE="1".T
YPE="NUMBER".NAM
E="Order_Quantit
y"></ATTRIBUTE><
/BP_EVENT></eX_E
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
56 45 4E 54 0A 42 50 49 5F 49 44 3D 22 31 36 31
36 2E 30 2E 32 30 32 3A 22 0A 54 59 50 45 3D 22
44 4F 5F 41 43 54 49 56 49 54 59 22 0A 49 44 3D
22 32 30 30 31 30 38 32 30 31 34 33 38 30 34 30
36 34 36 22 0A 4E 41 4D 45 3D 22 50 72 6F 63 65
73 73 4F 72 64 65 72 22 3E 0A 3C 41 43 54 49 56
49 54 59 20 4E 41 4D 45 3D 22 43 68 65 63 6B 5F
49 6E 76 22 20 49 44 3D 22 31 36 31 36 2E 35 36
39 22 20 2F 3E 0A 3C 41 54 54 52 49 42 55 54 45
20 4E 41 4D 45 3D 22 49 74 65 6D 5F 4E 75 6D 62
65 72 22 20 54 59 50 45 3D 22 53 54 52 49 4E 47
22 20 56 41 4C 55 45 3D 22 39 39 39 39 22 20
2F 3E 0A 3C 41 54 54 52 49 42 55 54 45 20 4E 41
4D 45 3D 22 4F 72 64 65 72 5F 51 75 61 6E 74 69
74 79 22 20 54 59 50 45 3D 22 4E 75 6D 62 65 72
22 20 56 41 4C 55 45 3D 22 31 2E 30 22 20 2F 3E
0A 3C 41 54 54 52 49 42 55 54 45 0A 56 41 4C 55
45 3D 22 65 42 50 4D 22 0A 4C 4F 43 41 54 49 4F
4E 3D 22 45 4D 42 45 44 44 45 44 22 0A 54 59 50
45 3D 22 54 52 41 4E 53 49 45 4E 54 22 0A 4E 41
4D 45 3D 22 65 58 5F 65 42 50 4D 53 65 72 76 65
72 22 3E 3C 2F 41 54 54 52 49 42 55 54 45 3E 3C
2F 42 50 5F 45 56 45 4E 54 3E 3C 2F 65 58 5F 45
76 65 6E 74 3E

```

```

<eX_Event>.<BP_E
VENT.BPI_ID="161
6.0.202:".TYPE="
DO_ACTIVITY".ID=
"200108231344480
169".NAME="Proce
ssOrder">.<ACTIV
ITY_NAME="Check_
Inv" ID="1616.56
9" />.<ATTRIBUTE
NAME="Item_Numb
er" TYPE="STRING
" VALUE="99999"
/>.<ATTRIBUTE NA
ME="Order_Quantit
y" TYPE="Number
" VALUE="1.0" />
.<ATTRIBUTE.VALU
E="eBPM".LOCATIO
N="EMBEDDED".TYP
E="TRANSIENT".NA
ME="eX_eBPMServe
r"></ATTRIBUTE><
/BP_EVENT></eX_E
vent>

```

# 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 303](#).

---

## 19.1 e\*Insight Helper Monk Functions

These functions allow you to set information in the e\*Insight Event (eX\_Standard\_Event.ssc ETD) and to get information from it. These functions are contained in the following file:

- **eX-eBPM-utils.monk**

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

These functions are described in detail on the following pages:

[eX-get-attribute](#) on page 304

[eX-count-attribute](#) on page 305

[eX-set-attribute](#) on page 306

[eX-set-BP\\_EVENT](#) on page 307

[eX-get-BP\\_EVENT](#) on page 308

[eX-get-Activity](#) on page 309

[eX-set-Activity](#) on page 310

[eX-string-set-attribute](#) on page 311

[eX-xml-set-attribute](#) on page 312

[eX-bin-set-attribute](#) on page 313

[eX-count-local-attribute](#) on page 314

[eX-get-local-attribute](#) on page 315

[eX-set-local-attribute](#) on page 316

[eX-copy-no-attribute](#) on page 317

[eX-set-all-BP\\_EVENT](#) on page 318

[eX-get-all-attribute](#) on page 319

[eX-get-all-local-attribute](#) on page 320

## 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: <ul style="list-style-type: none"> <li>▪ "BIN" - Interpreted as binary, however, must be suitably encoded for XML.</li> <li>▪ "XML" - Interpreted as XML, however, must be Base64 encoded for XML.</li> <li>▪ "STRING" - Interpreted as a string (default).</li> <li>▪ "TRANSIENT" - Interpreted as a transient. The e*Insight engine does not process the value but simply return it as-is.</li> <li>▪ "NUMBER" - Interpreted as a decimal number, however, must be given as a string.</li> <li>▪ "BOOLEAN" - Interpreted as a boolean, such as "true" and "false".</li> </ul>

### 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-string-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: <ul style="list-style-type: none"> <li>▪ "BIN" - Interpreted as binary, however, must be suitably encoded for XML.</li> <li>▪ "XML" - Interpreted as XML, however, must be Base64 encoded for XML.</li> <li>▪ "STRING" - Interpreted as a string (default).</li> <li>▪ "TRANSIENT" - Interpreted as a transient. The e*Insight engine does not process the value but simply return it as-is.</li> <li>▪ "NUMBER" - Interpreted as a decimal number, however, must be given as a string.</li> <li>▪ "BOOLEAN" - Interpreted as a boolean, such as "true" and "false".</li> </ul>

### 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 102](#) 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"
```



# 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 322](#)
- [“ATTRIBUTE Class” on page 342](#)
- [“BP\\_EVENT Class” on page 360](#)
- [“eX\\_StandardEvent Class” on page 396](#)

---

## 20.1 ACTIVITY Class

```
public class ACTIVITY
extends com.stc.jcsre.XMLETDImpl
implements com.stc.jcsre.ETD
```

A class to represent the ACTIVITY object of an e\*Insight (Business Process Management) XML ETD. It is defined in the following DTD:

```
<!--eBusiness Process Manager Event section-->
<!ELEMENT BP_EVENT (ACTIVITY?, ATTRIBUTE*)>
<!ATTLIST BP_EVENT
  TYPE (START_BP | DO_ACTIVITY | UNDO_ACTIVITY | UNDO_BPI |
  RESTART_ACTIVITY | SKIP_ACTIVITY | RELOAD_BP | AUTHORIZE |
  DONT_AUTHORIZE) #REQUIRED
  STATUS (SUCCESS | FAILURE) #IMPLIED
  NAME CDATA #IMPLIED
  ID CDATA #IMPLIED
  BPI_ID CDATA #IMPLIED
>
<!ELEMENT ATTRIBUTE EMPTY>
<!--ENCODING=base64 or whatever; eBPM only recognizes base64 for
TYPE=XML-->
<!ATTLIST ATTRIBUTE
  TYPE (BIN | XML | STRING | TRANSIENT | NUMBER | BOOLEAN) #REQUIRED
  NAME CDATA #REQUIRED
  VALUE CDATA #REQUIRED
  ENCODING CDATA #IMPLIED
  LOCATION (FILE | DB | URL | EMBEDDED | AUTO) #IMPLIED
>
<!ELEMENT ACTIVITY (ATTRIBUTE*)>
<!ATTLIST ACTIVITY
  NAME CDATA #IMPLIED
  ID CDATA #IMPLIED
>
```

These methods are described in detail on the following pages:

[addATTRIBUTE](#) on page 323

[clearATTRIBUTE](#) on page 324

[countATTRIBUTE](#) on page 325

[getAttribute\\_VALUE](#) on page 326

[getAttribute](#) on page 327

[getID](#) on page 328

[getName](#) on page 329

[hasID](#) on page 330

[hasNAME](#) on page 331

[marshal](#) on page 332

[omitID](#) on page 333

[omitNAME](#) on page 334

[removeATTRIBUTE](#) on page 335

[setAttribute](#) on page 336

[setID](#) on page 338

[setName](#) on page 339

[toString](#) on page 340

[unmarshal](#) on page 341

## 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: <ul style="list-style-type: none"> <li>▪ "BIN" - Interpreted as binary, however, must be suitably encoded for XML.</li> <li>▪ "XML" - Interpreted as XML, however, must be Base64 encoded for XML.</li> <li>▪ "STRING" - Interpreted as a string (default).</li> <li>▪ "TRANSIENT" - Interpreted as a transient. The e*Insight engine does not process the value but simply return it as-is.</li> <li>▪ "NUMBER" - Interpreted as a decimal number, however, must be given as a string.</li> <li>▪ "BOOLEAN" - Interpreted as a boolean, such as "true" and "false".</li> </ul>



Name	Type	Description
encoding	java.lang.String	The encoding used for the value. Examples: <ul style="list-style-type: none"> <li>▪ "base64" - Standard MIME Base64 encoding.</li> <li>▪ Null if plain text.</li> </ul>
location	java.lang.String	The location of the actual data associated with the Attribute value. Examples: <ul style="list-style-type: none"> <li>▪ "FILE" - Attribute value specifies a file where actual data exists.</li> <li>▪ "DB" - Attribute value is a reference to a row in a database table.</li> <li>▪ "URL" - Attribute value specifies a URL of where actual data exists.</li> <li>▪ "EMBEDDED" - Attribute value is the actual data (default).</li> <li>▪ "AUTO" - Attribute value is actual data but storage in e*Insight is automatically determined.</li> </ul>

### Return Values

None.

### Throws

None.

### Example

```
getInstance().getBP_EVENT().getACTIVITY.setATTRIBUTE("In_Stock", "STRING", "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

---

## 20.2 ATTRIBUTE Class

```
public class ATTRIBUTE
extends com.stc.jcsre.XMLETDImpl
implements com.stc.jcsre.ETD
```

A class to represent the ATTRIBUTE object of an e\*Insight (Business Process Management) XML ETD. It is defined in the following DTD:

```
<!--eBusiness Process Manager Event section-->
<!ELEMENT BP_EVENT (ACTIVITY?, ATTRIBUTE*)>
<!ATTLIST BP_EVENT
  TYPE (START_BP | DO_ACTIVITY | UNDO_ACTIVITY | UNDO_BPI |
  RESTART_ACTIVITY | SKIP_ACTIVITY | RELOAD_BP | AUTHORIZE |
  DONT_AUTHORIZE) #REQUIRED
  STATUS (SUCCESS | FAILURE) #IMPLIED
  NAME CDATA #IMPLIED
  ID CDATA #IMPLIED
  BPI_ID CDATA #IMPLIED
>
<!ELEMENT ATTRIBUTE EMPTY>
<!--ENCODING=base64 or whatever; eBPM only recognizes base64 for
TYPE=XML-->
<!ATTLIST ATTRIBUTE
  TYPE (BIN | XML | STRING | TRANSIENT | NUMBER | BOOLEAN) #REQUIRED
  NAME CDATA #REQUIRED
  VALUE CDATA #REQUIRED
  ENCODING CDATA #IMPLIED
  LOCATION (FILE | DB | URL | EMBEDDED | AUTO) #IMPLIED
>
<!ELEMENT ACTIVITY (ATTRIBUTE*)>
<!ATTLIST ACTIVITY
  NAME CDATA #IMPLIED
  ID CDATA #IMPLIED
>
```

These methods are described on the following pages:

[getENCODING](#) on page 343

[getLocation](#) on page 344

[getName](#) on page 345

[getType](#) on page 346

[getValue](#) on page 347

[hasENCODING](#) on page 348

[hasLOCATION](#) on page 349

[marshal](#) on page 350

[omitENCODING](#) on page 351

[omitLOCATION](#) on page 352

[setENCODING](#) on page 353

[setLOCATION](#) on page 354

[setName](#) on page 355

[setType](#) on page 356

[setValue](#) on page 357

[toString](#) on page 358

[unmarshal](#) on page 359

## 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 <b>ROWID</b> 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 hasENCODING()
```

### 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	org.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: <ul style="list-style-type: none"><li>▪ "FILE" - Attribute data is the name of a file where actual data is stored.</li><li>▪ "DB" - Attribute data is a reference such as "ROWID" to a row in a table.</li><li>▪ "URL" - Attribute data is the URL to where the actual data is stored.</li><li>▪ "EMBEDDED" - Attribute data is the actual data (this is the default).</li><li>▪ "AUTO" - The actual data storage location is automatically determined by the e*Insight engine.</li></ul>

### 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: <ul style="list-style-type: none"><li>▪ "BIN" - Attribute data is binary in nature. However, must be safely encoded for XML.</li><li>▪ "XML" - Attribute data represents a XML message. However, must be base64 encoded.</li><li>▪ "STRING" - Attribute data appears as clear-text string (this is the default).</li><li>▪ "TRANSIENT" - Attribute data is not persisted in the e*Insight database; is passed-thru.</li><li>▪ "NUMBER" - Attribute data represents a decimal numeric string.</li><li>▪ "BOOLEAN" - Attribute data represents a boolean string such as "true" or "false".</li></ul>

### Return Values

None.

### Throws

None.

### Example

```
setType("STRING");
```

## setVALUE

### Syntax

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

### Description

**setVALUE** sets the value of the Business Process or Activity Attribute.

### Parameters

Name	Type	Description
val	java.lang.String	The value of the Attribute.

### Return Values

None.

### Throws

None.

### Example

```
setVALUE("yes");
```

## toString

### Syntax

```
java.lang.String toString()
```

### Description

**toString** Converts this ETD object to a printable String form.

### Parameters

None.

### Return Values

#### **java.lang.String**

Returns the XML message represent by this ETD object.

### Throws

None.

### Example

```
toSTRING();
```

# unmarshal

## Syntax

```
void unmarshal(org.xml.sax.InputSource inputSource,  
com.stc.jcsre.sml.SAXLexer lexer)
```

## Description

**unmarshal** takes a serialized (marshalled) form of the e\*Insight XML Event and distributes (unmarshals) the data into this ETD object.

## Parameters

Name	Type	Description
inputSource	org.xml.sax.InputSource	The input source for the serialized data.
lexer	com.stc.jcsre.xml.SAXLexer	The SAX Lexer (parser) to distribute the data.

## Return Values

None.

## Throws

org.xml.sax.SAXException, when the data cannot be parsed.

com.stc.jcsre.UnmarshalException, when the data cannot be unmarshalled.

---

## 20.3 BP\_EVENT Class

```
public class BP_EVENT
extends com.stc.jcsre.AMLETImpl
implements com.stc.eBIPkg.BPEventETD
```

The BP\_EVENT class represents the e\*Insight Business Process Manager section of the SeeBeyond eBI Standard XML ETD which is used to communicate with the e\*Insight engine. The DTD is:

```
<!--eBusiness Process Manager Event section-->
<!ELEMENT BP_EVENT (ACTIVITY?, ATTRIBUTE*)>
<!ATTLIST BP_EVENT
    TYPE (START_BP | DO_ACTIVITY | UNDO_ACTIVITY | UNDO_BPI
        |RESTART_ACTIVITY | SKIP_ACTIVITY)
    STATUS (SUCCESS | FAILURE) #IMPLIED
    NAME CDATA #IMPLIED
    ID CDATA #IMPLIED
    BPI_ID CDATA #IMPLIED
>
<!ELEMENT ATTRIBUTE EMPTY>
<!--ENCODING=base64 or whatever; eBPM only recognizes base64 for
TYPE=XML-->
<!ATTLIST ATTRIBUTE
    TYPE (BIN | XML | STRING | TRANSIENT | NUMBER | BOOLEAN)
    #REQUIRED
    NAME CDATA #REQUIRED
    VALUE CDATA #REQUIRED
    ENCODING CDATA #IMPLIED
    LOCATION (FILE | DB | URL | EMBEDDED | AUTO) #IMPLIED
>
<!ELEMENT ACTIVITY (ATTRIBUTE*)>
<!ATTLIST ACTIVITY
    NAME CDATA #IMPLIED
    ID CDATA #IMPLIED
>
```

These methods are described in detail on the following pages:

<a href="#">addATTRIBUTE</a> on page 362	<a href="#">omitACTIVITY</a> on page 379
<a href="#">clearATTRIBUTE</a> on page 363	<a href="#">omitBPI_ID</a> on page 380
<a href="#">countATTRIBUTE</a> on page 364	<a href="#">omitID</a> on page 381
<a href="#">getACTIVITY</a> on page 365	<a href="#">omitNAME</a> on page 382
<a href="#">getATTRIBUTE_VALUE</a> on page 366	<a href="#">omitSTATUS</a> on page 383
<a href="#">getATTRIBUTE</a> on page 367	<a href="#">removeATTRIBUTE</a> on page 384
<a href="#">getBPI_ID</a> on page 368	<a href="#">setACTIVITY</a> on page 385
<a href="#">getID</a> on page 369	<a href="#">setATTRIBUTE</a> on page 386
<a href="#">getName</a> on page 370	<a href="#">setBPI_ID</a> on page 388
<a href="#">getStatus</a> on page 371	<a href="#">setEventInfo</a> on page 389
<a href="#">getType</a> on page 372	<a href="#">setID</a> on page 390
<a href="#">hasACTIVITY</a> on page 373	<a href="#">setName</a> on page 391



[hasBPI\\_ID](#) on page 374

[hasID](#) on page 375

[hasNAME](#) on page 376

[hasSTATUS](#) on page 377

[marshal](#) on page 378

[setStatus](#) on page 392

[setType](#) on page 393

[toString](#) on page 394

[unmarshal](#) on page 395

## 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 exists 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 exists; 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	org.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: <ul style="list-style-type: none"> <li>▪ "BIN" - Interpreted as binary, however, must be suitably encoded for XML.</li> <li>▪ "XML" - Interpreted as XML, however, must be Base64 encoded for XML.</li> <li>▪ "STRING" - Interpreted as a string (default).</li> <li>▪ "TRANSIENT" - Interpreted as a transient. The e*Insight engine does not process the value but simply return it as-is.</li> <li>▪ "NUMBER" - Interpreted as a decimal number, however, must be given as a string.</li> <li>▪ "BOOLEAN" - Interpreted as a boolean, such as "true" and "false".</li> </ul>
encoding	java.lang.String	The encoding used for the value. Examples: <ul style="list-style-type: none"> <li>▪ "base64" - Standard MIME Base64 encoding.</li> <li>▪ Null if plain text.</li> </ul>

Name	Type	Description
location	java.lang.String	The location of the actual data associated with the Attribute value. Examples: <ul style="list-style-type: none"><li>▪ "FILE" - Attribute value specifies a file where actual data exists.</li><li>▪ "DB" - Attribute value is a reference to a row in a database table.</li><li>▪ "URL" - Attribute value specifies a URL of where actual data exists.</li><li>▪ "EMBEDDED" - Attribute value is the actual data (default).</li><li>▪ "AUTO" - Attribute value is actual data but storage in e*Insight is automatically determined.</li></ul>

**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: <ul style="list-style-type: none"><li>▪ "SUCCESS" - Indicates that the current Activity has successfully completed.</li><li>▪ "FAILURE" - Indicates that the current Activity has not successfully completed.</li></ul>

### 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	<p>The command represented by the Business Process object. Examples:</p> <ul style="list-style-type: none"> <li>▪ "START_BP" - Instructs the e*Insight engine to start a Business Process Instance.</li> <li>▪ "DO_ACTIVITY" - Indicates a "Do" Event for the current Activity.</li> <li>▪ "UNDO_ACTIVITY" - Indicates an "Undo" Event for the current Activity.</li> <li>▪ "UNDO_BPI" - Indicates an "Undo" for the entire Business Process Instance.</li> <li>▪ "RESTART_ACTIVITY" - Indicates restarting the current Activity after it has paused.</li> <li>▪ "SKIP_ACTIVITY" - Indicates the current Activity should be skipped.</li> <li>▪ "RELOAD_BP" - Indicates the Business Process definition should be reloaded.</li> <li>▪ "AUTHORIZE" - Indicates that the current Activity should be authorized.</li> <li>▪ "DONT_AUTHORIZE" - Indicates that the current Activity should not be authorized.</li> </ul>

### Return Values

None.

### Throws

None.

### Example

```
setTYPE("DO_ACTIVITY");
```

## toString

### Syntax

```
java.lang.String toString()
```

### Description

**toString** converts this ETD object to a printable String form.

### Parameters

None.

### Return Values

#### **java.lang.String**

Returns the XML message to represent by this ETD object.

### Throws

None.

### Example

```
toSTRING();
```

# unmarshal

## Syntax

```
void unmarshal(org.xml.sax.InputSource inputSource,  
com.stc.jcsre.sml.SAXLexer lexer)
```

## Description

**unmarshal** takes a serialized (marshalled) form of the e\*Insight XML Event and distributes (unmarshals) the data into this ETD object.

## Parameters

Name	Type	Description
inputSource	org.xml.sax.InputSource	The input source for the serialized data.
lexer	com.stc.jcsre.xml.SAXLexer	The SAX Lexer (parser) to distribute the data.

## Return Values

None.

## Throws

org.xml.sax.SAXException, when the data cannot be parsed.

com.stc.jcsre.UnmarshalException, when the data cannot be unmarshalled.

## 20.4 eX\_StandardEvent Class

```
public class eX_StandardEvent
extends com.stc.jcsre.SMLETDImp1
implements com.stc.jcsre.ETD
```

eX\_StandardEvent class is an ETD class used to represent the standard XML message that is used to interchange information with the e\*Insight Business Process Manager engine. The DTD is:

```
<!ELEMENT eX_Event (BP_EVENT?, TP_EVENT?)>
<!--eBusiness Process Manager Event section-->
<!ELEMENT BP_EVENT (ACTIVITY?, ATTRIBUTE*)>
<!ATTLIST BP_EVENT
    TYPE (START_BP | DO_ACTIVITY | UNDO_ACTIVITY | UNDO_BPI |
        RESTART_ACTIVITY | SKIP_ACTIVITY | RELOAD_BP | AUTHORIZE
        | DONT_AUTHORIZE) #REQUIRED
    STATUS (SUCCESS | FAILURE) #IMPLIED
    NAME CDATA #IMPLIED
    ID CDATA #IMPLIED
    BPI_ID CDATA #IMPLIED
>
<!ELEMENT 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 #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=0
    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 398

[getBP\\_EVENT](#) on page 399

[getTP\\_EVENT](#) on page 400

[hasBP\\_EVENT](#) on page 401

[hasTP\\_EVENT](#) on page 402

[marshal](#) on page 403

[omitBP\\_EVENT](#) on page 404

[omitTP\\_EVENT](#) on page 405

[setBP\\_EVENT](#) on page 406

[setTP\\_EVENT](#) on page 407

[to\\_eBPMConvert](#) on page 408

[toString](#) on page 409

[unmarshal](#) on page 410

## 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.

# 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.

## 21.0.1 User Activity Security

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

Once that connection has been made, use the **authenticate** method to pass the user name and password for a user that has privileges for the Business Process. This user should have the necessary authority for the Business Processes that they are accessing. For subsequent messages sent during the 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 413](#)
- [“UserActivityMessage Class” on page 436](#)
- [“IClient Interface” on page 437](#)
- [“EbpmMonitor Class” on page 475](#)

## 21.0.2 Defining the Classpath

In order to use the User Activity API methods, you must define the following files in your classpath:

activation.jar	DGutil.jar	stcjs.jar
antlrall.jar	eIX_StandardEvent.jar	workflow.jar
classes12.zip	jconn2.jar	xerces.jar
DGbase.jar	js.jar	xml.jar
DGsqlserver.jar	mail.jar	
DGsybase.jar	soap.jar	

These files are all located in <eInsight>\Integrator.

---

## 21.1 Imessage Interface

These methods are described in detail on the following pages:

### clearMessage

#### Syntax

```
void clearMessage()
```

#### Description

**clearMessage** clears the message.

#### Parameters

None.

#### Return Values

None.

#### Throws

None.

#### Example

```
clearMessage();
```

## getActivityAttributesCount

### Syntax

```
int getActivityAttributesCount()
```

### Description

**getActivityAttributesCount** gets the activity attribute count.

### Parameters

None.

### Return Values

#### integer

Returns an integer in the range 0 to n depending on the number of activity attributes.

### Throws

None.

### Example

```
getActivityAttributesCount();  
=> 3
```

## getActivityAttributeValue

### Syntax

```
java.lang.String getActivityAttributeValue(java.lang.String  
attributeName)
```

### Description

**getActivityAttributeValue** is used to determine the value of an activity attribute.

### Parameters

Name	Type	Description
attributeName	java.lang.String	The attribute name.

### Return Values

#### java.lang.String

Returns a string containing the activity attribute value.

### Throws

None.

### Example

```
getActivityAttributesValue("In_Stock");  
=> "yes"
```

## getActivityName

### Syntax

```
java.lang.String getActivityName()
```

### Description

**getActivityName** gets the activity name.

### Parameters

None.

### Return Values

#### **java.lang.String**

Returns a 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: <ul style="list-style-type: none"><li>▪ "BIN" - Interpreted as binary, however, must be suitably encoded for XML.</li><li>▪ "XML" - Interpreted as XML, however, must be Base64 encoded for XML.</li><li>▪ "STRING" - Interpreted as a string (default).</li><li>▪ "TRANSIENT" - Interpreted as a transient. The e*Insight engine does not process the value but simply return it as-is.</li><li>▪ "NUMBER" - Interpreted as a decimal number, however, must be given as a string.</li><li>▪ "BOOLEAN" - Interpreted as a boolean, such as "true" and "false".</li></ul>
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");
```

## setBPISStack

### Syntax

```
void setBPISStack(java.lang.String bpiStack)
```

### Description

**setBPISStack** 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

```
setBPISStack("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: <ul style="list-style-type: none"><li>▪ "BIN" - Interpreted as binary, however, must be suitably encoded for XML.</li><li>▪ "XML" - Interpreted as XML, however, must be Base64 encoded for XML.</li><li>▪ "STRING" - Interpreted as a string (default).</li><li>▪ "TRANSIENT" - Interpreted as a transient. The e*Insight engine does not process the value but simply return it as-is.</li><li>▪ "NUMBER" - Interpreted as a decimal number, however, must be given as a string.</li><li>▪ "BOOLEAN" - Interpreted as a boolean, such as "true" and "false".</li></ul>

### 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: <ul style="list-style-type: none"><li>▪ START_BP</li><li>▪ DO_ACTIVITY</li><li>▪ UNDO_ACTIVITY</li><li>▪ UNDO_BPI</li><li>▪ RESTART_ACTIVITY</li><li>▪ SKIP_ACTIVITY</li><li>▪ RELOAD_BP</li><li>▪ AUTHORIZE</li><li>▪ DONT_AUTHORIZE</li></ul>

### 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: <ul style="list-style-type: none"><li>▪ SUCCESS</li><li>▪ FAILURE</li></ul>

### Return Values

None.

### Throws

None.

### Example

```
setStatus("SUCCESS");
```

## toXML

### Syntax

```
java.lang.String toXML()
```

### Description

**toXML** converts the message to XML.

### Parameters

None.

### Return Values

#### **java.lang.String**

Returns the message in XML format.

### Throws

None.

### Example

```
toXML();  
=> <BP_EVENT NAME="api" STATUS="SUCCESS" ID="231"  
    BPI_ID="231.0.21:" TYPE="DO_ACTIVITY"><ATTRIBUTE  
    NAME="In_Stock" TYPE="String" VALUE="no" /><ACTIVITY  
    NAME="API_Check_Inv" /></BP_EVENT>
```

---

## 21.2 UserActivityMessage Class

Implements IMessage.

See [“Imessage Interface” on page 413](#).

---

## 21.3 IClient Interface

These methods are described in detail on the following pages:

<a href="#">getActivityGlobalAttributeNames</a> on page 440	<a href="#">getGlobalAttributeType</a> on page 457
<a href="#">getActivityInstanceEndTime</a> on page 441	<a href="#">getGlobalAttributeValue</a> on page 458
<a href="#">getActivityInstanceStartTime</a> on page 442	<a href="#">getLocalAttributeNames</a> on page 459
<a href="#">getActivityInstanceStatus</a> on page 443	<a href="#">getLocalAttributeType</a> on page 460
<a href="#">getActivityNames</a> on page 444	<a href="#">getLocalAttributeValue</a> on page 461
<a href="#">getAssignedBPIIdByState</a> on page 445	<a href="#">getMessageStatus</a> on page 462
<a href="#">getAuthorizationActivityNames</a> on page 446	<a href="#">getUser</a> on page 463
<a href="#">getBPIStack</a> on page 447	<a href="#">getUserActivityNames</a> on page 464
<a href="#">getBusinessModelInstancesIds</a> on page 448	<a href="#">getUUID</a> on page 465
<a href="#">getBusinessModelInstanceName</a> on page 449	<a href="#">initialize</a> on page 466
<a href="#">getBusinessModelInstanceStatus</a> on page 450	<a href="#">refreshCachedMemory</a> on page 467
<a href="#">getBusinessModelName</a> on page 451	<a href="#">releaseActivityInstance</a> on page 468
<a href="#">getEnabledBusinessModelId</a> on page 452	<a href="#">resetUser</a> on page 470
<a href="#">getEnabledBusinessModelsIds</a> on page 453	<a href="#">sendMessage</a> on page 471
<a href="#">getGlobalAttributeDefaultValue</a> on page 454	<a href="#">setGlobalAttributeValue</a> on page 472
<a href="#">getGlobalAttributeDirection</a> on page 455	<a href="#">setLocalAttributeValue</a> on page 473
<a href="#">getGlobalAttributeNames</a> on page 456	<a href="#">setUser</a> on page 474

## 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
<code>businessModelId</code>	<code>java.lang.String</code>	The business model ID.
<code>businessModelInstanceId</code>	<code>java.lang.String</code>	The business model instance ID.
<code>activityName</code>	<code>java.lang.String</code>	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"}
```

## getAssignedBPIdByState

### Syntax

```
java.lang.String[] getActivityNames(java.lang.String businessModelId,  
java.lang.String activityName, java.lang.String status)
```

### Description

**getAssignedBPIdByState** 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

```
getAssignedBPIdByState("Pending");  
=> {"603", "604"}
```

## getAuthorizationActivityNames

### Syntax

```
java.lang.String[] getAuthorizationActivityNames(java.lang.String  
businessModelId)
```

### Description

**getAuthorizationActivityNames** gets a String[] of authorization activity names.

### Parameters

Name	Type	Description
businessModelId	java.lang.String	The business model id.

### Return Values

**java.lang.String[]**

Returns the authorization activity names.

### Throws

java.lang.Exception

### Example

```
getAuthorizationActivityNames("12345");  
=> {"Authorize_Quantity", "Authorize_Total"}
```

## getBPIShstack

### Syntax

```
java.lang.String getBPIShstack(java.lang.String businessInstanceId)
```

### Description

**getBPIShstack** 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

```
getBPIShstack();  
=>("637.0.133")
```

## getBusinessModelInstanceIds

### Syntax

```
java.lang.String[] getBusinessModelInstanceIds(java.lang.String  
businessModelId)
```

### Description

**getBusinessModelInstanceIds** 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

**getBusinessModelInstanceStatus** 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

```
getBusinessModelInstanceStatus("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
<code>businessModelId</code>	<code>java.lang.String</code>	The business model id.
<code>businessModelInstanceId</code>	<code>java.lang.String</code>	The business model instance id.
<code>activityName</code>	<code>java.lang.String</code>	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");
```

## 21.4 EbpmMonitor Class

This is the e\*Insight monitor client to the e\*Insight engine. The user can access the e\*Insight business model with sufficient user privileges. This class also allows the user to send e\*Insight messages to the e\*Insight engine.

The EbpmMonitor Class implements IClient interface. See [IClient Interface](#) on page 437, 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");
```

# XML Structure for the e\*Insight Event

This appendix shows the XML structure for the e\*Insight Event Type Definition. If your data conforms to this structure, you do not need to convert it upon entry to the e\*Insight system.

**Note:** *BP\_EVENT attribute names must match the attributes defined in the business process created in the e\*Insight GUI. These names are case-sensitive. If the attributes defined in the e\*Insight GUI do not match the incoming XML structure you must create a Collaboration to map the incoming data to the correct attribute names; otherwise, the e\*Insight system will not function correctly.*

---

## A.1 XML Structure

```
<!--DTD for eX_Standard_Event.ssc $Id: eX_event.dtd,v 1.1.2.10
2000/09/07 04:43:14 galbers Exp $-->
<!ELEMENT eX_Event (BP_EVENT?, TP_EVENT?)>
<!--eBusiness Process Manager Event section-->
<!ELEMENT BP_EVENT (ACTIVITY?, ATTRIBUTE*)>
<!ATTLIST BP_EVENT
    TYPE (START_BP | DO_ACTIVITY | UNDO_ACTIVITY) #REQUIRED
    STATUS (SUCCESS | FAILURE) #IMPLIED
    NAME CDATA #IMPLIED
    ID CDATA #IMPLIED
    BPI_ID CDATA #IMPLIED
>
<!ELEMENT ATTRIBUTE EMPTY>
<!--ENCODING=base64 or whatever; eBPM only recognizes base64 for
TYPE=XML-->
<!ATTLIST ATTRIBUTE
    TYPE (BIN | XML | STRING | TRANSIENT | NUMBER | BOOLEAN) #REQUIRED
    NAME CDATA #REQUIRED
    VALUE CDATA #REQUIRED
    ENCODING CDATA #IMPLIED
    LOCATION (FILE | DB | URL | EMBEDDED | AUTO) #IMPLIED
>
<!ELEMENT ACTIVITY (#PCDATA | ATTRIBUTE)*>
<!ATTLIST ACTIVITY
    NAME CDATA #IMPLIED
    ID CDATA #IMPLIED
>
<!--ePartner Manager Input/Output Event section-->
<!ELEMENT TP_EVENT (PartnerName?, InternalName?, Direction?,
MessageID?, OrigEventClass?, UsageIndicator?, Payload?, CommProt?,
Url?, SSLClientKeyFileName?, SSLClientKeyFileType?,
SSLClientCertFileName?, SSLClientCertFileType?, MessageIndex?,
TPAttribute?)>
```

```
<!--External Partner Name-->
<!ELEMENT PartnerName (#PCDATA)>
<!--Internal Sending ERP (ex. SAP)-->
<!ELEMENT InternalName (#PCDATA)>
<!--Direction of Transaction to/from Trading Partner (ex. Outbound=O
Inbound=I)-->
<!ELEMENT Direction (#PCDATA)>
<!--Original Request ID from Internal Sending ERP-->
<!ELEMENT MessageID (#PCDATA)>
<!--Original Event Classification (ex. QAP for Query Price and
Availability)-->
<!ELEMENT OrigEventClass (#PCDATA)>
<!--Usage Indicator of EDI message by Trading Partner (Production=P
Test=T)-->
<!ELEMENT UsageIndicator (#PCDATA)>
<!--Payload to carry EDI message-->
<!ELEMENT Payload (#PCDATA)>
<!ATTLIST Payload
    TYPE (RAW | PROCESSED | ENCRYPTED) #REQUIRED
    LOCATION (FILE | DB | URL | EMBEDDED | AUTO) #IMPLIED
>
<!--RAW=Need translation PROCESSED=Already X12 or RN ENCRYPTED=from
Trading Partner-->
<!--Communication Protocol (ex. BATCH, HTTP) for sending to Trading
Partner-->
<!ELEMENT CommProt (#PCDATA)>
<!--URL for EDI message to be exchanged with Trading Partner-->
<!ELEMENT Url (#PCDATA)>
<!--SSL information-->
<!ELEMENT SSLClientKeyFileName (#PCDATA)>
<!ELEMENT SSLClientKeyFileType (#PCDATA)>
<!ELEMENT SSLClientCertFileName (#PCDATA)>
<!ELEMENT SSLClientCertFileType (#PCDATA)>
<!--Message Index for Batched delivery, ex. 1|20 means 1 of 20-->
<!ELEMENT MessageIndex (#PCDATA)>
<!--TP Attribute will contain optional repeating name value pair for
storing of TP data -->
<!ELEMENT TPAttribute (NameValuePair*)>
<!ELEMENT NameValuePair (Name, Value)>
<!ELEMENT Name (#PCDATA)>
<!ELEMENT Value (#PCDATA)>
```

# Glossary

## **attribute**

Attributes pass user-defined control information (programming arguments) to and from the e\*Insight Business Process Manager and its activities.

## **activity**

An activity is an organizational unit for performing a specific function.

## **activity states**

Activity states are the stages that activities within the business process instance go through as the business process version is being run.

## **business process**

A business process is a collection of actions and messages, revolving around a specific business practice, that flow in a specific pattern to produce an end result.

## **business process attributes**

Business process attributes pass user-defined control information (programming arguments) to and from the e\*Insight Business process manager, external sources, and internal components.

## **business process expression**

Business process expressions allow you to dictate business process logic flow based on the ability to perform various types of logic on business process instance attributes.

## **business process instance (BPI)**

A unique instantiation of a business process.

## **business process version**

A form or variant of the original business process model.

## **Collaboration**

A component of an e\*Way or BOB that receives and processes Events and forwards the output to other e\*Gate components. Collaborations perform three functions: they subscribe to Events of a known type, they apply business rules to Event data, and they publish output Events to a specified recipient. Collaborations use Monk translation script files with the extension “.tsc” to do the actual data manipulation.

## **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 eBusiness Integration Suite product suite that facilitates the automation of the business process flow of eBusiness activities.

**e\*Xchange Partner Manager (e\*Xchange)**

An application that allows you to set up and maintain trading partner profiles and view processed messages. e\*Xchange also processes inbound and outbound messages according to certain eBusiness protocols and your validation Collaborations.

**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 eBusiness 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 267  
     creating the Collaboration Rules scripts 267  
 addATTRIBUTE 323, 362, 363  
 APIs  
   eX-bin-set-attribute 313  
   eX-copy-no-attribute 317, 338  
   eX-count-attribute 305, 324  
   eX-count-local-attribute 314, 334  
   eX-get-Activity 309, 328  
   eX-get-all-attribute 319  
   eX-get-all-local-attribute 320  
   eX-get-attribute 304, 323  
   eX-get-BP\_EVENT 308, 327  
   eX-get-local-attribute 315  
   eX-set-Activity 310, 329  
   eX-set-all-BP\_EVENT 318, 339  
   eX-set-attribute 306, 325  
   eX-set-BP\_EVENT 307, 326  
   eX-set-local-attribute 316, 336  
   eX-string-set-attribute 311, 330  
   eX-xml-set-attribute 312, 331  
   getBusinessModelInstanceId 418  
   getBusinessModelName 419  
 attributes  
   getting 102, 105  
   global 80  
   input 81  
   input/output 81  
   local 81  
   output 81  
   setting 101, 104  
   using local attributes to implement undo logic 81  
 authenticate 438

## B

Basic 26  
 BOB  
   Activity BOB 58, 77  
 BP\_EVENT 84, 93  
 business logic, testing 131, 144, 176, 191, 199, 204  
 business processes

attributes, about 80  
 restarting 137, 181  
 starting 100, 103

## C

case study (order processing) 110, 140, 148, 184, 197, 202, 250  
 checkoutActivityInstance 439  
 checkUserPrivileges 475  
 clearATTRIBUTE 324  
 clearMessage 413  
 Collaboration  
   START\_BP Collaboration 53, 73  
 Collaborations  
   configuring 126, 165, 169, 263, 267  
   Do and Undo logic in 56, 76  
   eX\_Activity 54, 58, 78  
   eX\_from\_eBPM 50, 70  
   eX\_Resubmitter 52, 71  
   eX\_to\_Activity 56, 76  
   eX\_to\_eBPM 70  
 configuring  
   e\*Insight engine 34, 66, 258, 264  
   editing the eX\_eBPM engine's configuration file 120, 162, 240, 258, 264  
   eX\_Resubmitter BOB 52, 71  
   eX\_to\_Activity e\*Way 56, 75  
   Send\_Status e\*Way 127, 170, 189  
   START\_BP e\*Way 121, 163, 259, 265  
   starting a business process 100, 103  
   user-defined e\*Gate components 188, 258, 265, 270, 274  
 Configuring the START\_BP Component 53, 73  
 conventions, writing in document 19  
 Copy the eISchema 27  
 countATTRIBUTE 364

## D

demonstrating business process restart 137, 181  
 demonstrating business process undo 135, 179  
 Do and Undo logic in an Activity collaboration 56, 76  
 Driver Type 41

## E

e\*Gate Schema for e\*Insight 30, 61  
 e\*Insight  
   schema components 30, 61, 276  
   sending the "Done" Event back 102, 106  
 e\*Insight Authorization Activity Implementation

140–146, 184–194  
 e\*Insight Business Process Manager 24  
 e\*Insight engine  
   configuring 258, 264  
 e\*Insight ETD, understanding 79–97  
 e\*Insight Helper Monk Functions 302–320  
 e\*Insight Implementation 109–139, 147–183  
 e\*Insight Java Helper Methods 321–410  
 e\*Insight Schema components overview 30, 61  
 e\*Insight Sub-Process Implementation 207–218, 219–232  
 e\*Insight User Activity Implementation 195–206  
 e\*Insight User Activity Methods 411–475  
 e\*Ways  
   Send\_Status, configuring 127, 170, 189  
   START\_BP, configuring 121, 163, 259, 265  
 error handling 52, 72  
   connection failure handling 52  
   data failure handling 52  
   normal event failure handling 72  
   special event failure handling 72  
 error type 52, 71  
   connection 52, 71  
   data 52, 71  
 ETD Monk  
   structure 89  
 eX 396  
 eX-bin-set-attribute 313  
 eX-copy-no-attribute 317, 338  
 eX-count-attribute 305, 324  
 eX-count-local-attribute 314, 334  
 eX-get-Activity 309, 328  
 eX-get-all-attribute 319  
 eX-get-all-local-attribute 320  
 eX-get-attribute 304, 323  
 eX-get-BP\_EVENT 308, 327  
 eX-get-local-attribute 315  
 eXSchema 61  
 eXSchema, copying 27  
 eX-set-Activity 310, 329  
 eX-set-all-BP\_EVENT 318, 339  
 eX-set-attribute 306, 325  
 eX-set-BP\_EVENT 307, 326  
 eX-set-local-attribute 316, 336  
 eX-string-set-attribute 311, 330  
 eX-xml-set-attribute 312, 331

## F

from 398  
 from\_eBPMConvert 398

## G

getACTIVITY 365  
 getActivityAttributesCount 414  
 getActivityAttributeValue 415  
 getActivityGlobalAttributeNames 440  
 getActivityInstanceEndTime 441  
 getActivityInstanceStartTime 442  
 getActivityInstanceStatus 443  
 getActivityName 416, 444  
 getAssignedBPIIdByState 445  
 getATTRIBUTE 327, 367  
 getATTRIBUTE\_VALUE 326, 366  
 getAuthorizationActivityNames 446  
 getBP\_EVENT 399  
 getBPI\_ID 368  
 getBPISStack 447  
 getBusinessModelId 417  
 getBusinessModelInstanceId 418  
 getBusinessModelInstanceName 449  
 getBusinessModelInstancesIds 448  
 getBusinessModelInstanceStatus 450  
 getBusinessModelName 419, 431, 451  
 getEnabledBusinessModelId 452  
 getEnabledBusinessModelsIds 453  
 getENCODING 343  
 getGlobalAttributeCount 420  
 getGlobalAttributeDefaultValue 454  
 getGlobalAttributeDirection 455  
 getGlobalAttributeNames 456  
 getGlobalAttributeType 421, 457  
 getGlobalAttributeValue 422, 458  
 getID 328, 369  
 getLocalAttributeNames 459  
 getLocalAttributeType 460  
 getLocalAttributeValue 461  
 getLocation 344  
 getMessageStatus 462  
 getMsgType 423  
 getName 329, 345, 370  
 getStatus 371  
 getTP\_EVENT 400  
 getType 346, 372  
 getUser 463  
 getUserActivityNames 464  
 getUUID 465  
 getValue 347  
 global attributes 80

## H

hasACTIVITY 373  
 hasBP\_EVENT 401  
 hasBPI\_ID 374

hasENCODING 348  
 hasID 330, 375  
 hasLOCATION 349  
 hasNAME 331, 376  
 hasSTATUS 377  
 hasTP\_EVENT 402

## I

## implementation

- basic information 25, 26
- configuring the e\*Gate components 28
- configuring the e\*Insight schema based on the business process 27
- copying the eXSchema 27
- creating a business process 27
- e\*Insight 109–139, 147–183
- e\*Insight Authorization Activity 140–146, 184–194
- e\*Insight User Activity 195–206
- Remote Sub-Process 233–249
- road map 26
- Sub-Process 207–218, 219–232
- testing and tuning the system 28

initialize 466

input attributes 81

input/output attributes 81

Introducing e\*Insight Business Process Manager (e\*Insight) 24

## J

## Java APIs

- addATTRIBUTE 323, 362, 363
- authenticate 438
- checkoutActivityInstance 439
- checkUserPrivileges 475
- clearATTRIBUTE 324
- clearMessage 413
- countATTRIBUTE 364
- from\_eBPMConvert 398
- getACTIVITY 365
- getActivityAttributesCount 414
- getActivityAttributeValue 415
- getActivityGlobalAttributeNames 440
- getActivityInstanceEndTime 441
- getActivityInstanceStartTime 442
- getActivityInstanceStatus 443
- getActivityName 416, 444
- getAssignedBPIdByState 445
- getATTRIBUTE 327, 367
- getATTRIBUTE\_VALUE 326, 366
- getAuthorizationActivityNames 446
- getBP\_EVENT 399

- getBPI\_ID 368
- getBPISStack 447
- getBusinessModelId 417
- getBusinessModelInstanceName 449
- getBusinessModelInstancesIds 448
- getBusinessModelInstanceStatus 450
- getBusinessModelName 431, 451
- getEnabledBusinessModelId 452
- getEnabledBusinessModelsIds 453
- getENCODING 343
- getGlobalAttributeCount 420
- getGlobalAttributeDefaultValue 454
- getGlobalAttributeDirection 455
- getGlobalAttributeNames 456
- getGlobalAttributeType 421, 457
- getGlobalAttributeValue 422, 458
- getID 328, 369
- getLocalAttributeNames 459
- getLocalAttributeType 460
- getLocalAttributeValue 461
- getLOCATION 344
- getMessageStatus 462
- getMsgType 423
- getNAME 329, 345, 370
- getSTATUS 371
- getTP\_EVENT 400
- getTYPE 346, 372
- getUser 463
- getUserActivityNames 464
- getUUID 465
- getVALUE 347
- hasACTIVITY 373
- hasBP\_EVENT 401
- hasBPI\_ID 374
- hasENCODING 348
- hasID 330, 375
- hasLOCATION 349
- hasNAME 331, 376
- hasSTATUS 377
- hasTP\_EVENT 402
- initialize 466
- marshal 332, 350, 378, 403
- omitACTIVITY 379
- omitBP\_EVENT 404
- omitBPI\_ID 380
- omitENCODING 351
- omitID 333, 381
- omitLOCATION 352
- omitNAME 334, 382
- omitSTATUS 383
- omitTP\_EVENT 405
- refreshCachedMemory 467
- releaseActivityInstance 468
- releaseResources 469

removeActivity 424  
 removeATTRIBUTE 335, 384  
 removeGlobalAttribute 425  
 resetUser 470  
 sendMessage 471  
 setACTIVITY 385  
 setActivityAttributeValue 426  
 setActivityName 427  
 setATTRIBUTE 336, 386  
 setBP\_EVENT 406  
 setBPL\_ID 388  
 setBPIShared 428  
 setBusinessModelId 430  
 setBusinessModelInstanceId 429  
 setENCODING 353  
 setEventInfo 389  
 setGlobalAttributeValue 472  
 setID 338, 390  
 setLocalAttributeValue 473  
 setLOCATION 354  
 setMsgType 433  
 setName 339, 355, 391  
 setStatus 392  
 setStatus 434  
 setTP\_EVENT 407  
 setType 356, 393  
 setUser 474  
 setValue 357  
 to\_eBPMConvert 408  
 toString 340, 358, 394, 409  
 toXML 435  
 unmarshal 341, 359, 395, 410  
 Java APIs setGlobalAttributeValue 432

## L

local attributes 81  
     using to implement undo logic 81

## M

marshal 332, 350, 378, 403  
 Monk functions see functions

## O

omitACTIVITY 379  
 omitBP\_EVENT 404  
 omitBPL\_ID 380  
 omitENCODING 351  
 omitID 333, 381  
 omitLOCATION 352  
 omitNAME 334, 382

omitSTATUS 383  
 omitTP\_EVENT 405  
 order processing case study 110, 140, 148, 184, 197,  
 202, 250  
 output attributes 81  
 Overview 201

## R

refreshCachedMemory 467  
 releaseActivityInstance 468  
 releaseResources 469  
 Remote Sub-Process Implementation 233–249  
 removeActivity 424  
 removeATTRIBUTE 335, 384  
 removeGlobalAttribute 425  
 resetUser 470  
 restart, demonstrating 137, 181

## S

schema, copying 27  
 SeeBeyond eBusiness Integration Suite 21–25  
 sending the "Done" Event back to e\*Insight 102, 106  
 sendMessage 471  
 setACTIVITY 385  
 setActivityAttributeValue 426  
 setActivityName 427  
 setATTRIBUTE 336, 386  
 setBP\_EVENT 406  
 setBPL\_ID 388  
 setBPIShared 428  
 setBusinessModelId 430  
 setBusinessModelInstanceId 429  
 setENCODING 353  
 setEventInfo 389  
 setGlobalAttributeValue 432, 472  
 setI 338  
 setID 390  
 setLocalAttributeValue 473  
 setLOCATION 354  
 setMsgType 433  
 setName 339, 355, 391  
 setStatus 392  
 setStatus 434  
 setting attributes 101, 104  
 setTP\_EVENT 407  
 setType 356, 393  
 setUser 474  
 setValue 357  
 START\_BP component, configuring 53, 73  
 START\_BP e\*Way, configuring 121, 163, 259, 265  
 starting a business process 100, 103  
 supporting documents 20

## T

testing the standard business logic **131, 144, 176, 191, 199, 204**  
to\_eBPMConvert **408**  
toString **340, 358, 394, 409**  
toXML **435**

## U

understanding the e\*Insight ETD **79–97**  
Undo, demonstrating **135, 179**  
unmarshal **341, 359, 395, 410**  
user-defined e\*Gate components, configuring **188, 258, 265, 270, 274**

## X

XML  
  element with sub-elements **89**  
  element without sub-elements **90**  
  ETD structure for an XML attribute **90**  
  structure for the e\*Xchange Event **476–477**