

SeeBeyond™ eBusiness Integration Suite

e*Gate Integrator User's Guide

Release 4.5.2



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.

© 2001–2002 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 This work is protected as an unpublished work under the copyright laws.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 20020304154258.

Contents

List of Figures	16
List of Tables	23
<hr/>	
Chapter 1	
Introduction	26
User's Guide Purpose and Scope	26
Intended Audience	26
Organization of Information	27
Writing Conventions	28
Supporting Documents	30
Online Documents	31
Searching the Online Documents	31
SeeBeyond Web Site	31
<hr/>	
Chapter 2	
System Description	32
Overview of e*Gate System	32
Component Organization and Schemas	32
Layered System Architecture	33
View Layer	34
Enterprise Manager	34
e*Gate Editors	34
Monitoring Features	35
Control Layer	35
Registry	36
Control Brokers	36
Business Rules and Data Processing Layer	36
Collaboration Rules	36
Collaborations	37
Intelligent Queuing Layer	37

Contents

Application Connectivity Layer	37
Types of e*Ways	38
Multi-Mode	38
Database Access	39
Applications	39
Generic e*Way Kit	39
Additional Applications	39
Business Object Brokers	39
System Setup	40
<hr/>	
Chapter 3	
 Getting Started	42
Overview of Starting with e*Gate	42
System Requirements, Installations, and Upgrades	42
Add-ons and Client Software	42
Business Analysis and Planning	43
e*Gate System Setup Prerequisites	43
Creating a System Design	44
Information Gathering	44
System Structure	45
Naming System Elements	45
Make a Checklist	45
Naming Conventions	46
System Preparations	48
Enterprise Manager Basic Operation	49
Enterprise Manager Window	50
Components Tree	51
Toolbar Buttons	52
Menu Bar	53
File > New	53
File > Edit File	57
View > Summary	58
Codeveloping in e*Gate: Using the Team Registry	60
Important: User Name Requirements	61
The Sandbox	61
Sandbox Properties	61
Sandbox Operation	62
Team Registry File Operations	63
Testing Schemas: Run-time and Sandbox Considerations	65
Changing Default Check-in/Check-out Actions	65
Sandbox/Run-time Registry Directory Structure	66
Team Registry and Component “Run As” Settings	66
Team Registry and Version-control Systems	66
Adding New Participating Hosts	67
Users, Roles, and Privileges	69
Using the Network View	70

Contents

Introduction: Network View	70
Using Network View	70
Online Help Systems	73
Using Online Help	73
Hypertext Links	74
Accessing Online Help	74
Help Window	75
GUI Features	76
On Entry and Exit	76
Online Help Features	77
Tab Operation	77
Toolbar Buttons	78
Printing Help	78
Closing the Help Window	80

Chapter 4

Setting Up e*Gate	81
Overview of e*Gate Setup	81
e*Gate GUIs	81
Setup Steps	82
System Design Components	83
Component Data Flow Relationships	83
Component Logical Relationships	85
Data Management Relationships	86
Creating a Schema	86
Control Broker Setup	87
Configuring Control Broker Properties	87
Host Activation	95
Creating Event Types and ETDs	95
Selecting the Event Type Definition Editor	96
Creating Event Types	96
Creating Java-enabled Event Type Definitions	97
Creating Monk Event Type Definitions	103
Assigning Definitions to Monk Event Types	105
Creating Collaboration Rules and Scripts	106
Using Collaboration Scripts	107
C-language Scripts	107
Monk Language Scripts	107
Java Language File Types	108
Collaboration Rules Properties	109
Collaboration Services and Types	109
Java Collaborations	110
Creating Java Collaborations	112
Adding Custom Business Rules to the Collaboration	115
Monk Collaborations	116
Creating Monk Collaboration Rules	116
Configuring Monk Collaboration Rules	117

Contents

Adding e*Ways and BOBs	123
e*Way Operation	123
Before Creating an e*Way	125
Explanation of Tree Levels	126
Control Brokers	127
System Files and e*Ways	127
Creating e*Ways	127
Configuring e*Ways	128
Adding Business Object Brokers	130
Adding Multi-Mode e*Ways	131
Before Creating a Multi-Mode e*Way	131
Creating and Configuring a Multi-Mode e*Way	131
Adding e*Way Connections	134
Adding Intelligent Queues	136
IQ Managers	137
Working With IQs	137
Attaching IQs	142
Adding Collaborations	142
Collaboration Setup	143
Creating Collaborations	143
Configuring Collaborations	144
Troubleshooting Collaborations	147
Reviewing and Testing the System	147
Post System Setup Troubleshooting	148
Java Interactive Debugger	148
Monk Test Console	149
Basic Controls	152
Setup Features	154
Input Features	155
Output Features	155

Chapter 5

Event Type Definitions (ETDs)	157
About This Chapter	157
Learning About ETDs	158
What Is an ETD?	158
How Does e*Gate Use ETDs?	159
Java-Enabled ETDs	159
Monk ETDs	159
ETD Editor Overview	159
Feature Overview	159
GUI Overview	160
ETD Editor GUI Areas	161
Main Menu	162
Toolbar	163
Before Using the ETD Editor	163

Contents

Building Java-Enabled ETDs	164
About ETD Types	164
Package Names, Node Names, and .jar File Names	164
Starting the ETD Editor	165
Creating a New Standard ETD	165
Converting a Monk ETD to a Java-enabled Standard ETD	167
Building an Imported ETD	170
Using the SEF Wizard	170
Working With Java-Enabled ETDs	174
About Package Names	174
About Node Names	174
Basic ETD Procedures	175
Opening, Saving, and Renaming ETDs	175
Viewing and Editing Java Properties	176
Working with Elements and Fields	177
Using Templates	180
Compiling an ETD	182
Validating an ETD	183
Promoting to Run Time	185
Global and Local Delimiters	186
About Global Delimiters	186
About Global Delimiter Levels	186
About Local Delimiters and Delimiter Groups	186
About Local Delimiter Groups	186
Using Global Delimiters	187
Using Local Delimiters	188
Standard ETD Properties	191
Event Type Properties	191
Properties of Root and Element Nodes (Parent Nodes)	192
Properties of Field and Reference Nodes (Leaf Nodes)	197
Properties of Delimiters	201

Chapter 6

Monk Event Type Definition Editor	204
Monk ETD Editor Overview	204
Getting Started	204
ETD Creation and Nodes	205
Working with Nodes	205
Naming Nodes	206
Before Using the ETD Editor	207
ETD Editor Window	208
Toolbar	209
Menu Bar	210
Creating and Building ETDs	215
Creating ETD Files	215
Building Delimited ETDs	217
Creating Delimited ETDs	217
Defining Default Delimiters	217

Contents

Creating Root Nodes for Delimited ETDs	222
Adding Delimited-ETD Nodes	228
Specifying HL7 Repeating Fields	231
Building Fixed ETDs	232
Creating Fixed ETDs	232
Creating Root Nodes for Fixed ETDs	232
Adding Fixed-ETD Nodes	235
Specifying Byte Offsets in Fixed ETDs	238
Adding Node Sets	239
Adding Node Subsets	241
Basic ETD Operations	242
Opening ETDs	242
Using the Build Tool	242
Saving ETDs Under New Names	246
Extracting Input Delimiters	247
Testing ETD Files	247
Creating ETD Comments	250
Finding ETD Nodes	250
Editing ETD Files	250
Moving Nodes	251
Using Cut, Copy, and Paste	251
Pruning ETDs	252
Changing Node Details	253
Modifying Internal Templates	253
Deleting ETDs	253
Working With ETD Templates	254
Using External Templates	254
Including External Templates in ETDs	255
Changing ETD Repetition Properties	256
Breaking Template Links	257
Using Internal Templates	257
Creating Internal Templates	257
Converting Existing ETDs	258
Referencing Internal Templates in ETDs	258
Changing ETD Repetition Properties	258

Chapter 7

Java Collaboration Rules	260
About This Chapter	260
Learning About Java Collaboration Rules	261
Files Used by Java Collaboration Rules	261
Where Do Methods Come From?	262
Example: What's for Dinner?	262
Java Collaboration Rules Editor Overview	264
Feature Overview	264
GUI Overview	265
The Editor GUI Panes	266
Menu Commands	267

Contents

Main Toolbar	269
Business Rules Toolbar	269
Working With Java Collaboration Rules	270
Creating a New Java Collaboration Rule	271
Settings for the Collaboration Rules Properties dialog box	274
General Tab	274
Collaboration Mapping Tab	274
The Mapping Pane	275
Dragging and Dropping Fields	277
Using Find and Map	278
The View Commands	278
Saving, Compiling, and Promoting Collaboration Rules	280
Enabling and Disabling ELS	281
Setting Classpath and Package Options	282
Searching and Replacing Within a Collaboration	287
Creating Custom Java Methods	288
Using the Business Rules	290
block	291
case	292
catch	293
copy	294
datamap	296
default	299
do, while	300
else	301
finally	302
for	303
if, then, else	306
list lookup	307
method	310
Methods Presupplied When You Start the Editor	312
return	313
rule	314
switch, case, default	315
then	316
throw	317
timestamp	318
try, catch, finally	321
uniqueid	322
variable	324
while	325
Common Dialog Boxes for Business Rules	326
Dealing with Repeating Nodes	326
Formatting Output	327
Methods for Elements and Fields of ETDs	328
count_MyNode_()	329
get_MyNode_()	330
has_MyNode_()	331
set_MyNode_()	332
Methods for Standard Java-enabled ETDs	333
available()	334
marshal()	335
next()	336
publications()	337
rawInput()	338
readProperty()	339
receive()	341
reset()	342

Contents

send()	343
subscriptions()	344
topic()	345
unmarshal()	346
writeProperty()	347
Subcollaboration Rules	348
Terminology	348
Purpose, Concepts, and Caveats	348
Working with Subcollaboration Rules	349
<hr/>	
Chapter 8	
Monk Collaboration Rules Editor	352
Overview: Monk Collaboration Rules	352
Collaboration Rules Scripts and Types	352
Before You Begin	352
Task List	353
Collaboration Rules Editor Window	353
Toolbar Buttons	355
Other Window Controls	356
ETD Panning Windows	356
Rules Pane Controls	357
Menu Bar	357
File Menu	358
Edit Menu	359
Rules Menu	359
View Menu	360
Options Menu	361
Help Menu	361
Creating Monk Collaboration Rules Scripts	361
Getting Started	362
Creating New Monk Collaboration Rules	362
How e*Gate Processes Event Data	364
Appending Data	364
Trailing Spaces	366
Fixed Data Lengths	366
Adding and Arranging Rules	366
Adding Rules and Elements	366
Arranging Rules	367
Selecting Event Elements	368
Defining ETD Paths	369
Defining Instances of Repeating Event Elements	371
Filling in Rule Details	375
Using the Function Library to Define Rules	377
Basic Collaboration Rules Operations	379
Opening a Collaboration Rules Component	380
Saving a Collaboration Rules Component to a New Name	380
Entering Comments About Collaboration Rules	381
Changing Collaboration Rules Scripts	381
Deleting Rules	381

Contents

Changing Rule Parameters	382
Changing Source/Destination ETDs	382
Validating Collaboration Rules	382
Finding Nodes	383
Converting to and from Double-Byte Character Encodings	384
Using Collaboration Rules	386
Collaboration Rules Reference Table	386
Using the If Rule	387
Comparing an Event Element to a Regular Expression	390
Comparing an Event Element to a Number Using <	390
Comparing an Event Element to a Number Using <=	391
Comparing an Event Element to a Number Using >	392
Comparing an Event Element to a Number Using >=	392
Comparing an Event Element to a Number Using =	393
Comparing an Event Element to a Number Using not =	393
Testing for a False Condition	394
Performing Multiple Tests with an If Rule	394
Performing Alternate Tests with an If Rule	395
Using the Loop Rule	396
Loop Rule Overview	396
Creating a Loop Rule	397
Executing a Loop Rule	399
Defining ETD Paths in a Loop Rule	399
Looping on a Computed Range of Event Elements	401
Looping on a Fixed Range of Event Elements	404
Using the Case Rule	406
Creating Case Rules	408
Using the Comment Rule	410
Using the Copy Rule	411
Using the Display Rule	416
Using the Duplicate Rule	417
Using the Data Map Rule	420
Using the List Lookup Rule	424
Using the Change Pattern Rule	429
Using the Timestamp Rule	432
Using the Unique ID Rule	435
Using the Let Rule	437
Naming Variables in the Let Rule	438
Naming a Condition Using the Let Rule	439
Using the Let Rule to Specify a Variable in an Insert Rule	441
Using the Set! Rule	442
Using the Function Rule	443
Selecting a Prewritten Function	444
Defining Your Own Function	445
Using the User Function Rule	446
Using the Set Regex Rule	446
<hr/>	
Chapter 9	
Working with e*Ways	448
Overview of e*Way Operation	448

Contents

Component Parts	449
e*Ways and the Enterprise Manager	449
Configuring e*Ways with the Enterprise Manager	450
Defining e*Way Components	450
Modifying e*Way Properties	451
Selecting an Executable File	452
Creating or Selecting a Configuration File	453
Changing Command-line Parameters	453
Changing the “Run As” User Name	454
Setting Startup Options or Schedules	454
Activating or Modifying Logging Options	456
Activating or Modifying Monitoring Thresholds	456
e*Ways and Collaborations	457
Configuring e*Ways	457
Concepts	457
Controls	458
Section and Parameter Controls	459
Parameter Configuration Controls	459
Using the e*Way Configuration Editor	460
Navigating Through the Editor	460
Saving Configuration Settings	460
Modifying Configuration Settings	461
Restoring Default Settings	461
Restoring Saved Settings	461
Entering User Notes	461
Creating Business Object Brokers	462
Using the Online Help System	462
Troubleshooting e*Ways	462
In the Enterprise Manager	463
In the e*Way Configuration Editor	463
On the e*Way’s Participating Host	463
In the e*Way’s External Application	463
Multi-Mode e*Way	464
Multi-Mode e*Way Characteristics	464
JVM Settings	466
JNI DLL Absolute Pathname	467
CLASSPATH Prepend	467
CLASSPATH Override	468
Initial Heap Size	469
Maximum Heap Size	470
Maximum Stack Size for Native Threads	470
Maximum Stack Size for JVM Threads	470
Class Garbage Collection	471
Garbage Collection Activity Reporting	471
Asynchronous Garbage Collection	471
Report JVM Info and all Class Loads	471
Disable JIT	471
Remote debugging port number	472
Suspend option for debugging	472
Changing Command-line Parameters	472
Changing the “Run As” User Name	473

Contents

Setting Startup Options or Schedules	474
Advanced Settings for Multi-Mode e*Ways	476
Activating or Modifying Logging Options	476
Activating or Modifying Monitoring Thresholds	476
Configuring Multi-Mode e*Ways with e*Way Connections	477
Controls	477
Section and Parameter Controls	479
Parameter Configuration Controls	479
Using the e*Way Connection Editor	480
Navigating Through the Editor	480
Saving Configuration Settings	480
Modifying Configuration Settings	481
Restoring Default Settings	481
Restoring Saved Settings	481
Entering User Notes	481
JVM Settings	482
e*Insight Business Process Manager Engine	487

Chapter 10

Introduction to e*Gate Monitor 489

e*Gate Monitoring Overview	489
Role of the Control Broker	489
e*Gate Interactive Monitoring	490
e*Gate Monitor Basic Operation	490
e*Gate Monitor Window	491
Toolbar Buttons	492
Menu Bar	493
Controlling e*Gate	495
Starting and Shutting Down Components	496
Displaying Status and Version Information	497
Suspending and Activating Components	498
Non-interactive Monitoring	499
Notification Channels	499
e*Gate Alert Agent	499
e*Gate SNMP Agent	500
Custom User Agents	500
Monitoring Resources and Performance	500
Setting Disk-usage Thresholds	500
Disk-space Quota Limitations	502
Setting Event-processing Thresholds	502

Chapter 11

e*Gate Java Debugger 505

Overview of e*Gate Java Debugger Operation	505
Main Menu	507

Contents

Activating the e*Gate Java Debugger	509
Using the e*Gate Java Debugger	510
Controlling execution of the Collaboration	510
Options Dialog	512

Chapter 12

Event Linking and Sequencing (ELS)	516
Learning About ELS	516
How Does ELS Operate Within e*Gate?	517
ELS Operation	519
About the SeeBeyond-supplied ELS Methods	520
Count-Based Triggers	520
Timer-Based Triggers	522
The ELS Wizard	523
About the ELS Wizard	523
ELS Wizard Operation	523
Sample Implementation	528
Overview	528
Steps	529
Creating the schema and defining the Event Types	529
Building the ETDs	530
Creating the Collaboration Rules	532
Creating the ELS Business Rules for cr_ELS_CombineGrades	533
Creating the Data Transformation Logic Under executeBusinessRules()	535

Chapter 13

XA Transaction Processing	538
Introduction	538
References	538
Architectural Review	539
Operational Summary	540
Working with XA-enabled Collaborations	540
Mixing XA-Compliant and XA-Noncompliant e*Way Connections	541

Appendix A

Java Classes and Methods	542
Index to Methods for Standard Java-enabled ETDs	543
Base64Utils Class (com.stc.eways.util)	549
CollabUtils Class (com.stc.eways.util)	552

Contents

DateUtils Class (com.stc.eways.util)	559
EGate Class (com.stc.common.collabService)	562
FileUtils Class (com.stc.eways.util)	570
JCollabController Class (com.stc.common.collabService)	573
General System Control Methods	574
Character Encoding and Internationalization Methods	580
About the Encoding Methods	580
Character Encodings in the Java Collaboration Rules Editor	581
ELSController Interface Methods	589
JCollaboration Class (com.stc.jcsre)	599
eventSend()	599
JSubCollabMapInfo Class (com.stc.common.collabService)	603
Mainframe Class (com.stc.eways.util)	616
MapUtils Class (com.stc.eways.util)	621
Usage Example	621
QSort Class (com.stc.common.utils)	627
ScEncrypt Class (com.stc.common.utils)	628
STCTypeConverter Class (com.stc.eways.util)	630
StringUtils Class (com.stc.eways.util)	656
Formatting of Output Text	662
Glossary	666
Index	674

List of Figures

Figure 1	e*Gate Layered Architecture	33
Figure 2	Collaboration/e*Way Relationship	38
Figure 3	Sample e*Gate System Diagram	40
Figure 4	Sample e*Gate System Diagram	45
Figure 5	e*Gate Enterprise Manager Login Dialog Box	49
Figure 6	Enterprise Manager Window (Network View)	50
Figure 7	Enterprise Manager Window (Components View)	51
Figure 8	View > Summary	58
Figure 9	Event Type Summary Window	59
Figure 10	Team Registry: Overview	60
Figure 11	The Team Registry in Operation	61
Figure 12	Enterprise Manager Window With Participating Hosts	67
Figure 13	Participating Host Properties Dialog Box	68
Figure 14	Network View with Graphic Representation in Editor Pane	71
Figure 15	Enterprise Manager Help Window	75
Figure 16	Contents Tree	76
Figure 17	Print Topics Dialog Box	79
Figure 18	e*Gate Setup Road Map	82
Figure 19	Basic e*Gate Data Flow Relationships	84
Figure 20	Basic e*Gate Logical Relationships	85
Figure 21	Control Broker Properties Dialog Box: General Tab	88
Figure 22	Control Broker Properties Dialog Box: Notification Setup Tab	89
Figure 23	Control Broker Properties Dialog Box: Timers Tab	90
Figure 24	Timer Event Properties Dialog Box	91
Figure 25	Control Broker Properties Dialog Box: Advanced Tab	92
Figure 26	Control Broker Properties Dialog Box: Security Tab	93
Figure 27	Assign Privileges Dialog Box and Add Role Selection Box	94
Figure 28	Enterprise Manager With Event Types	97
Figure 29	Event Type Definition Editor	99
Figure 30	New Event Type Definition Window	100
Figure 31	Example Wizard: Standard ETD	100
Figure 32	Java Event Type Definition Editor with Sample ETD	101

Figure 33	ETD Editor Window (New)	103
Figure 34	New ETD Dialog Box	104
Figure 35	Event Type Properties Dialog Box	105
Figure 36	Enterprise Manager Window with Services	110
Figure 37	Java Collaboration Rules Editor	111
Figure 38	e*Gate Enterprise Manager: Create New Collaboration Rules	112
Figure 39	Collaboration Rules Properties: Collaboration Mapping Tab	113
Figure 40	Java Collaboration Editor with Sample Collaboration	115
Figure 41	Expanded Java Code	116
Figure 42	Enterprise Manager with Collaboration Rules Icons	117
Figure 43	Collaboration Rules Properties Dialog Box, General Tab	118
Figure 44	SeeBeyond Collaboration Rules Editor Window (Monk)	120
Figure 45	Collaboration Rules Properties Dialog Box, Subscriptions Tab	121
Figure 46	Collaboration Rules Properties Dialog Box, Publications Tab	122
Figure 47	e*Way Operation	124
Figure 48	Contents of Participating Hosts Folder	126
Figure 49	e*Way Properties Dialog Box, General Tab	128
Figure 50	e*Way Editor Window (New)	129
Figure 51	Multi-Mode e*Way Properties Dialog Box	132
Figure 52	Edit Settings Dialog Box – Multi-Mode e*Way	133
Figure 53	*Gate Enterprise Manager: Create New e*Way Connection	134
Figure 54	e*Way Connection Properties Dialog Box	135
Figure 55	STC_Standard IQ Properties Dialog Box, General Tab	138
Figure 56	IQ Properties Dialog Box, Advanced Tab	139
Figure 57	IQ Properties Dialog Box, Database Tab	140
Figure 58	IQ Properties Dialog Box, External Tab	141
Figure 59	Enterprise Manager with Collaboration	144
Figure 60	Collaboration Properties Dialog Box with Selections	146
Figure 61	Monk Test Console, Setup Tab	150
Figure 62	Monk Test Console, Input Tab	151
Figure 63	Monk Test Console, Output Tab	152
Figure 64	ETD Editor GUI Map	160
Figure 65	New Event Type Definition Dialog Box	165
Figure 66	Standard ETD Wizard - Step 1	166
Figure 67	Newly Created Standard ETD	166
Figure 68	ETD Builder Wizards	167
Figure 69	SSC Wizard - Step 1	168
Figure 70	Result of Converting a Monk ETD Using the SSC Wizard	169

Figure 71	ETD Builder Wizards	171
Figure 72	SEF Wizard - Step 1	171
Figure 73	Result of Converting an SEF File Using the SEF Wizard	173
Figure 74	The Java Properties Dialog Box	176
Figure 75	The Comment Property Dialog Box	177
Figure 76	ETD Editor – Test Dialog Showing Parsed Data	183
Figure 77	Sample ETD and Sample Test Data	184
Figure 78	Successful Promotion of an ETD	185
Figure 79	Global Delimiters Dialog Box - Delimiter Level	187
Figure 80	Setting Global Begin and End Delimiters	188
Figure 81	Local Delimiters Dialog Box - Local Delimiters Level	189
Figure 82	Example of a Local Delimiter Group	190
Figure 83	ETD Editor Tree	205
Figure 84	ETD Editor and Tree Structure	208
Figure 85	ETD Dialog Box	216
Figure 86	Set Delimiters Dialog Box	219
Figure 87	Set Delimiters Dialog Box, Other Selected	220
Figure 88	Delimited Root Node Properties Dialog Box, General Tab	223
Figure 89	Delimited Root Node Properties Dialog Box, Content Tab	224
Figure 90	Delimited Root Node Properties Dialog Box, Repetition Tab	225
Figure 91	Delimited Root Node Properties Dialog Box, Delimiters Tab	227
Figure 92	Root Node and Node Icons	228
Figure 93	Delimited Node Properties Dialog Box, General Tab	229
Figure 94	Field Repetition Delimiter Usage	231
Figure 95	Fixed Root Node Properties Dialog Box, General Tab	233
Figure 96	Fixed Root Node Properties Dialog Box, Delimiters Tab	234
Figure 97	Fixed-ETD Node Icons	235
Figure 98	Fixed Node Properties Dialog Box, General Tab	236
Figure 99	Node Set Placeholder	239
Figure 100	Set Properties Dialog Box, Node Set	240
Figure 101	Node Subset Diagram	241
Figure 102	Build an Event Type Definition Dialog Box (Default)	243
Figure 103	Build an Event Type Definition Dialog Box (Library)	244
Figure 104	Build an Event Type Definition Dialog Box (Delimited Data)	245
Figure 105	Select a Test Data File Dialog Box	248
Figure 106	Test Structure Dialog Box	249
Figure 107	Include External Event Type Definition Dialog Box	255
Figure 108	ETD External Template Icons	256

Figure 109 Internal Template Icon	257
Figure 110 Relationship of Collaboration Rules to e*Gate Components	261
Figure 111 Dinner Seen as an Event Type Definition	263
Figure 112 Java Collaboration Rules Editor GUI Map	265
Figure 113 The Collaboration Rule Properties	271
Figure 114 The Collaboration Mapping Tab	272
Figure 115 The New Collaboration Rule	273
Figure 116 Mapping Pane When One Field Is Selected	275
Figure 117 Mapping Pane for Expanded and Collapsed Parent Nodes	276
Figure 118 Mapping Pane When a Business Rule Is Selected	276
Figure 119 Creating Rules by Dragging Fields	277
Figure 120 Find and Map	278
Figure 121 Business Rules Toolbar With Labels Turned Off	279
Figure 122 Business Rules Pane with Code Display Turned On	279
Figure 123 Code Error Exposed by Double-Clicking a Compile Pane Message	279
Figure 124 The View Java Code Window	280
Figure 125 Java Classpaths Dialog Box	282
Figure 126 Java Imports Dialog Box	283
Figure 127 Shortcut Menu in Rule Properties Pane	284
Figure 128 Choose a method Dialog Box	285
Figure 129 Choose a method Dialog Box	286
Figure 130 Results of Inserting a Java Method	286
Figure 131 The Search And Replace Dialog Box	287
Figure 132 The UserFunctions.xml Template for Custom Java Methods	288
Figure 133 Relating UserFunctions.xml, SBYNFunctions.xml, and the GUI	289
Figure 134 The block Rule and Its Properties	291
Figure 135 The Copy Dialog Box	294
Figure 136 Sample Text Files for the dataMap Rule	296
Figure 137 The Data Map Dialog Box	296
Figure 138 The do, while Rule and Its Properties	300
Figure 139 Code Generated When “for” Is Used on a Repeating Node	304
Figure 140 Result of Mapping a Repeating Node in a “for” Loop	305
Figure 141 The if, then, else Rule and Its Properties	306
Figure 142 The List Lookup Dialog Box	307
Figure 143 A method Rule and Its Properties	311
Figure 144 The rule Rule and Its Properties	314
Figure 145 The Insert Timestamp Dialog Box	318
Figure 146 The Insert Unique ID Dialog Box	322

Figure 147	The while Rule and Its Properties	325
Figure 148	The Select Repetition Instance Dialog Box With a Counter	326
Figure 149	The Select Repetition Instance Dialog Box Without a Counter	326
Figure 150	The Output Format Dialog Box	327
Figure 151	Root Collaboration Rule Calling a Subcollaboration Rule	351
Figure 152	Monk Collaboration Rules Editor GUI Map	354
Figure 153	ETD Panning Window	356
Figure 154	New Dialog Box	362
Figure 155	Open Event Type Definition Dialog Box	363
Figure 156	Appending Data with the Copy Rule	364
Figure 157	Appending Data with the Duplicate Rule	365
Figure 158	Offset Data	365
Figure 159	Rules Pane with Added Rules	367
Figure 160	Collapsing and Expanding Rules	367
Figure 161	Specifying Event Elements	368
Figure 162	Select Repetition Instance	372
Figure 163	Rule's Name Button	375
Figure 164	Specifying a Byte Location	377
Figure 165	Library Dialog Box	378
Figure 166	Current Value Text Box	378
Figure 167	Main Comment Dialog Box	381
Figure 168	Error Message with Symbol	383
Figure 169	Find Node in Destination Event Type Definition Dialog Box	384
Figure 170	Find Node in Source Event Type Definition Dialog Box	384
Figure 171	Accessing the Code Conversion Functions	385
Figure 172	If-rule Setup in the Rules Pane	388
Figure 173	Loop Rule Structure	396
Figure 174	Case Rule Setup	407
Figure 175	Case Rule Strings	408
Figure 176	Case Rule Integers	409
Figure 177	Case Rule Example	410
Figure 178	Comment Rule Example	410
Figure 179	How the Copy Rule Handles Input-to-output Event Delimiters	411
Figure 180	Copy Rule Example	412
Figure 181	Copy Dialog Box	413
Figure 182	Set Output Format Dialog Box	415
Figure 183	Display Rule Setup	416
Figure 184	How the Duplicate Rule Handles Input-to-Output Event Delimiters	417

Figure 185	Duplicate Rule Example	418
Figure 186	Duplicate Dialog Box	419
Figure 187	Sample Text File for Data Map Rule	420
Figure 188	Data Map Rule Syntax	422
Figure 189	List Lookup Dialog Box	426
Figure 190	Timestamp Dialog Box	433
Figure 191	Let Rule Bar Structure	437
Figure 192	Functions Dialog Box	444
Figure 193	User Function Rule	446
Figure 194	Set Regex Dialog Box	447
Figure 195	Enterprise Manager View of e*Way Characteristics	449
Figure 196	Enterprise Manager View of e*Way Contents	450
Figure 197	e*Way Properties Dialog Box, General Tab	452
Figure 198	e*Way Properties Dialog Box, Start Up Tab	455
Figure 199	e*Way Configuration Editor GUI Map	458
Figure 200	Sample Selection List Controls	459
Figure 201	e*Way Properties Dialog Box – General Tab	465
Figure 202	Top Portion of the Multi-Mode e*Way Configuration Editor Window	466
Figure 203	Second Portion of the Multi-Mode e*Way Configuration Editor Window	468
Figure 204	Third Portion of the Multi-Mode e*Way Configuration Editor Window	469
Figure 205	Fourth Portion of the Multi-Mode e*Way Configuration Editor Window	470
Figure 206	Multi-Mode e*Way Properties Dialog Box, Start Up Tab	475
Figure 207	e*Way Connection Editor Window Controls	478
Figure 208	Sample Selection List Controls	479
Figure 209	Top Portion of the e*Way Configuration Editor Window	482
Figure 210	Second Portion of the e*Way Configuration Editor Window	484
Figure 211	Third Portion of the e*Way Configuration Editor Window	485
Figure 212	Fourth Portion of the e*Way Configuration Editor Window	486
Figure 213	e*Insight Engine Properties Dialog Box	488
Figure 214	e*Gate Monitor Window, Alerts Tab	491
Figure 215	Threshold Setup	501
Figure 216	Change Disk Threshold	501
Figure 217	Threshold Properties Dialog Box	503
Figure 218	e*Gate Java Debugger Window	506
Figure 219	Stop in Method Dialog Box	509
Figure 220	Context Pane with this Tab Selected	511
Figure 221	Lower Right Pane with Evaluate Tab Selected	512
Figure 222	Options Dialog Box	513

Figure 223 Stop in Class Dialog Box	513
Figure 224 Stop in Method Dialog Box	514
Figure 225 Break on Exception Dialog Box	515
Figure 226 ELS Processing Model	517
Figure 227 ELS Unmarshalling of Events	518
Figure 228 ELS in Context With IQs and non-ELS Collaborations	519
Figure 229 ELS Schematic	521
Figure 230 ELS Buckets With Timers	522
Figure 231 ELS Wizard Step 1 - Specify Field for Link Identifier	524
Figure 232 ELS Wizard Step 2 - Specify Message Count	525
Figure 233 ELS Wizard Step 3 - Specify Expiration	526
Figure 234 Code Generated by the ELS Wizard	527
Figure 235 ELS Collaboration	528
Figure 236 School Example: etd\School\xsc	530
Figure 237 School Example: etd\School\Semester.xsc	531
Figure 238 School Example: Schema After Creating Event Types and ETDs	531
Figure 239 School Example: Properties of cr_ELS_CombineGrades	532
Figure 240 School Example: cr_ELS_CombineGrades Before Modification	533
Figure 241 School Example: Setting the ELS Link Identifier	534
Figure 242 School Example: Setting the ELS Message Count	534
Figure 243 School Example: Generated Code Under retrieveLinkIdentifier()	535
Figure 244 School Example: Completed ELS-enabled Collaboration Rule	537
Figure 245 e*Gate in a Distributed Transaction Processing (DTP) Context	539
Figure 246 Monk Collaboration Rules Editor in a Japanese Environment	581
Figure 247 Java Collaboration Rules Editor in a Japanese Environment	582
Figure 248 Setting the Character Encoding in userInitialize()	583

List of Tables

Table 1	Sample Component Naming Convention	47
Table 2	Additional e*Gate System Names	47
Table 3	Enterprise Manager Toolbar	52
Table 4	Enterprise Manager Menus and Commands	54
Table 5	Tools to Promote Files to Run-time	64
Table 6	Tools to Remove Files from the Sandbox	64
Table 7	Network View Palette Controls	71
Table 8	Tabs Pane Operation	77
Table 9	Help Window Buttons	78
Table 10	Monk Collaboration Script Types	107
Table 11	Java Collaboration File Types	108
Table 12	Monk Test Console Control Buttons	152
Table 13	Monk Test Console Toolbar	153
Table 14	Monk Test Console Input/User Data Buttons	153
Table 15	ETD Editor Menus and Commands (Java)	162
Table 16	Toolbar Buttons	163
Table 17	Node Icons for Java ETDs	179
Table 18	Common Errors When Compiling an ETD	182
Table 19	Properties of the Standard Event Type	191
Table 20	Properties of Standard ETD Parent Node Elements (type=CLASS)	192
Table 21	Properties of Standard ETD Field Nodes (type=FIELD)	197
Table 22	Properties of Global and Local Delimiters	201
Table 23	Acceptable Node-name Characters	206
Table 24	Toolbar Buttons and Functions	209
Table 25	ETD Editor File Menu	211
Table 26	ETD Editor Edit Menu	212
Table 27	ETD Editor Templates Menu	213
Table 28	ETD Editor Templates Menu	213
Table 29	ETD Editor Options Menu	214
Table 30	ETD Editor Help Menu	214
Table 31	Options for Delimited and Fixed ETDs	217
Table 32	Delimiter Variable Repetition Options	230

Table 33	Build an Event Type Definition Dialog Box Options	245
Table 34	Java Collaboration File Types	262
Table 35	Java Collaboration Rules Editor Menu Commands	267
Table 36	Main Toolbar Buttons	269
Table 37	Business Rules Toolbar Buttons	269
Table 38	Parameters for Business Rule copy	295
Table 39	Parameters for Business Rule dataMap	297
Table 40	Parameters for Business Rule lookup	308
Table 41	Parameters for Business Rule timeStamp	319
Table 42	Date and Time Format Codes for the timeStamp Rule	319
Table 43	Parameters for Business Rule uniqueId	322
Table 44	Toolbar Buttons	355
Table 45	Window Controls	356
Table 46	Rules Pane Controls	357
Table 47	File Menu Commands	358
Table 48	Edit Menu Commands	359
Table 49	Rules Menu Commands	359
Table 50	View Menu Commands	360
Table 51	Options Menu Commands	361
Table 52	Help Menu Commands	361
Table 53	Sample Events with Path Locations	370
Table 54	Repeating Event Elements	373
Table 55	Select Repetition Instance Dialog Box Entries	374
Table 56	Showing Byte Locations in Rule Dialog Boxes	376
Table 57	Changing Rule Parameters	382
Table 58	Monk Functions for Code Conversion	385
Table 59	Monk Collaboration Rules	386
Table 60	If-rule Test Setups	389
Table 61	Loop Rule Elements	397
Table 62	Node Repetition Symbols	399
Table 63	Repetition Information for Loop Rule	400
Table 64	Case Rule Control Features	407
Table 65	Copy Rule Use Methods	412
Table 66	Using the Copy/Duplicate Dialog Box	414
Table 67	Using the Set Output Format Dialog Box	416
Table 68	Data Map Dialog Box Entries	423
Table 69	Parameter and Section Controls	459
Table 70	Selection List Controls	460

List of Tables

Table 71	Java 2 JNI DLL Names	467
Table 72	Parameter and Section Controls	479
Table 73	Selection List Controls	479
Table 74	Java 2 JNI DLL Names	483
Table 75	e*Gate Monitor Toolbar	492
Table 76	e*Gate Monitor Menu Commands	493
Table 77	Monitor Commands	495
Table 78	Available Control Tab Commands	495
Table 79	Notification Channels and Delivery Systems	499
Table 80	Event-threshold Monitoring-Event Codes	504
Table 81	e*Gate Java Debugger Main Menu	507
Table 82	Java Classes and Methods	542
Table 83	Trace Events (Logging Levels)	566
Table 84	Trace ID Parameters (for <i>tid</i> Parameter)	567
Table 85	Trace Event Parameters (for <i>event</i> Parameter)	569
Table 86	Sample Inputs and Format Codes and Their Results	664

Introduction

This chapter introduces you to this user's guide, its general purpose and scope, and its organization. It also provides sources of related documentation and information.

1.1 User's Guide Purpose and Scope

This user's guide explains generally how to set up and configure the SeeBeyond Technology Corporation™ (SeeBeyond™) e*Gate Integrator system. e*Gate configuration includes preparation, editing, and initial monitoring procedures.

This guide explains how to use the following graphical user interfaces (GUIs):

- Enterprise Manager and its editor GUIs
 - ◆ Java Event Type Definition Editor
 - ◆ Monk Event Type Definition Editor
 - ◆ Java Collaboration Rules Editor
 - ◆ Monk Collaboration Rules Editor
 - ◆ e*Way (Intelligent Adapter) Editor
- e*Gate Monitor

Important: *Any operation explanations given here are generic, for reference purposes only, and do not necessarily address the specifics of setting up individual e*Gate systems.*

This document does not contain information on software installation and system administration procedures (see "[Supporting Documents](#)" on page 30).

1.2 Intended Audience

The reader of this guide is presumed to be an experienced PC user with the responsibility for helping to set up and/or to maintain a fully functioning e*Gate system. This person must also have expert-level knowledge of Windows NT/Windows 2000 or UNIX operations and be thoroughly familiar with Windows-style GUI operations.

1.3 Organization of Information

This document is organized topically as follows:

- **Chapter 1 “Introduction”** gives a general preview of this document, its purpose, scope, and organization; also provides sources of additional information.
- **Chapter 2 “System Description”** provides a detailed overview of the e*Gate system, including its structure, operation, and GUIs.
- **Chapter 3 “Getting Started”** describes how to run the e*Gate Enterprise Manager, what it is, and its characteristics; also introduces the online Help system.
- **Chapter 4 “Setting Up e*Gate”** explains in detail the essential setup GUIs and features of e*Gate as well as how to use these features for configuring basic system components.
- **Chapter 5 “Event Type Definitions (ETDs)”** explains how to use the Java ETD Editor to define, configure, and modify Java Event Type Definitions.
- **Chapter 6 “Monk Event Type Definition Editor”** explains how to use the Monk ETD Editor to define, configure, and modify Monk Event Type Definitions.
- **Chapter 7 “Java Collaboration Rules”** explains how to use the Java Collaboration Rules Editor to define, configure, and modify Java Collaboration scripts for your Collaborations.
- **Chapter 8 “Monk Collaboration Rules Editor”** explains how to use the Monk Collaboration Rules Editor to define, configure, and modify Collaboration Rules Monk scripts for your Collaborations.
- **Chapter 9 “Working with e*Ways”** explains how to use the Enterprise Manager to create, configure, and modify e*Ways, e*Way Connections, and BOBs.
- **Chapter 10 “Introduction to e*Gate Monitor”** explains the basic features of the e*Gate Monitor GUI and how to use them.
- **Chapter 11 “e*Gate Java Debugger”** explains how to use the Java Collaboration Debugger for Multi-Mode e*Ways.
- **Chapter 12 “Event Linking and Sequencing (ELS)”** explains the purpose and function of Event Linking and Sequencing (ELS) methods available through the Java Collaboration Rules Editor.
- **Chapter 13 “XA Transaction Processing”** explains e*Gate’s role in a Distributed Transaction Processing context and discusses its compliance with the XA standard in terms of architecture and operations.
- **Appendix A “Java Classes and Methods”** lists and describes the Java classes and methods accessible via the Java Collaboration Rule Editor for Standard ETDs, and provides instructions for creating and including your own custom Java methods.

In addition, there is a **Glossary** on page 666 to help you with the e*Gate system’s related terminology.

1.4 Writing Conventions

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

Hypertext Links

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

For information on these and related topics, see “[Parameter, Function, and Command Names](#)” on page 29.

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 -rs schema-name -un user-name  
-up password -ef output-directory
```

Code and Samples

Computer code and samples (including printouts) on a separate line or lines are set in Courier 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.

You can use the **stccb** utility to start the Control Broker.

Additional Conventions

Windows Systems — The e*Gate system is fully compliant with both Windows 2000 and Windows NT platforms. When this document refers to Windows, such statements apply to both Windows platforms.

UNIX and Linux Systems — This guide uses the backslash (“\”) as the separator within path names. If you are working on a UNIX system, including Linux, please make the appropriate substitutions.

Note: *The e*Gate system is fully compatible with Compaq Tru64 UNIX version 4.0F and 5.0A.*

1.5 Supporting Documents

The following SeeBeyond documents provide additional information about the e*Gate Integrator system as explained in this guide:

- *C Generic e*Way Extension Kit Developer's Guide*
- *Creating an End-to-end Scenario with e*Gate Integrator*
- *e*Gate API Kit Developer's 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 Installation Guide*
- *e*Gate Integrator Intelligent Queue Services Reference Guide*
- *e*Gate Integrator Release Notes*
- *e*Gate Integrator SNMP Agent User's Guide*
- *e*Gate Integrator System Administration and Operations Guide*
- *e*Gate Integrator Upgrade Guide*
- *e*Insight Business Process Manager Implementation Guide*
- *e*Insight Business Process Manager User's Guide*
- *e*Xchange Partner Manager Implementation Guide*
- *e*Xchange Partner Manager User's Guide*
- *Java Generic e*Way Extension Kit Developer's Guide*
- *Monk Developer's Reference*
- *Monk Generic e*Way Extension Kit Developer's Guide*
- *SeeBeyond eBusiness Integration Suite Primer*
- *SeeBeyond eBusiness Integration Suite Deployment Guide*
- *SeeBeyond JMS Intelligent Queue User's Guide*
- *Standard e*Way Intelligent Adapter User's Guide*
- *Working with Collaboration IDs*
- *XML Toolkit*

For a complete list of e*Gate-related documentation, consult the *SeeBeyond eBusiness Integration Suite Primer*. You can also refer to the appropriate Microsoft Windows or UNIX documents, if necessary.

For information on how to use a specific add-on product (for example, an e*Way Intelligent Adapter), see the user's guide for that product.

1.6 Online Documents

The documentation for the SeeBeyond eBusiness Integration Suite is distributed as a collection of online documents. These documents are viewable with the Acrobat Reader application from Adobe Systems. Acrobat Reader can be downloaded from:

<http://www.adobe.com>

Note: When downloading Acrobat Reader, make sure to download the version that includes the option for searching .pdf files. This option is required in order to view the searchable master index.

Searching the Online Documents

The collection of online documents includes a searchable master index. This index is a convenient way to find a topic when you are not sure which document to consult. The index requires activation of the SeeBeyond master index.

To activate the SeeBeyond master index

- 1 If you have not already done so, download and install Acrobat Reader; take care to install the version that includes the option for searching .pdf files.
- 2 Start Acrobat Reader.
- 3 On the **Edit** menu, point to **Search**, and then click **Select Indexes**.
- 4 In the **Index Selection** dialog box, click **Add**.
- 5 Locate and open the <eGate>\client\docs\ folder, where <eGate> is the location where e*Gate is installed.
- 6 Double-click **SeeBeyond_Index.pdx**.
- 7 Click **OK** to close the **Index Selection** dialog box.

To search the master index

- 1 On the Acrobat Reader **Edit** menu, point to **Search**, and then click **Query**.
- 2 Type the term or phrase you want to find, and then click **Search**.
A list of documents matching the search criteria appears.
- 3 Select a title from the list, and then click **View**.
- 4 Press CTRL+] and CTRL+[to view the next and previous highlighted results.

1.7 SeeBeyond Web Site

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

<http://www.SeeBeyond.com>

System Description

This chapter gives an overview of the general operation, structure, and architecture of the e*Gate system. In addition, it describes the system's basic components.

2.1 Overview of e*Gate System

The basic purpose of e*Gate is to move data from one point to another. The e*Gate system has the following basic features:

- **Distributed Architecture:** e*Gate uses a distributed architecture that is open and flexible, allowing components to reside on different workstations within a global network.
- **Effective Communication:** e*Gate uses protocols and adapters you choose, which allows e*Gate to communicate with and link multiple applications/databases across different operating systems.
- **Versatile Performance:** e*Gate can function with a large number of hardware, message standards, operating systems, databases, and communication protocols.
- **Multi-mode Operation:** e*Gate operates in both real-time and batch/scheduled integration modes.
- **Convenient Bridges:** e*Gate can bridge between older and newer systems, resulting in a centrally managed and unified enterprise.

The different components of an e*Gate system do not have to reside on the same machine. Instead, you can distribute them across various machines in a total network.

Note: For definitions of e*Gate components and features, see the [Glossary](#) on page 666.

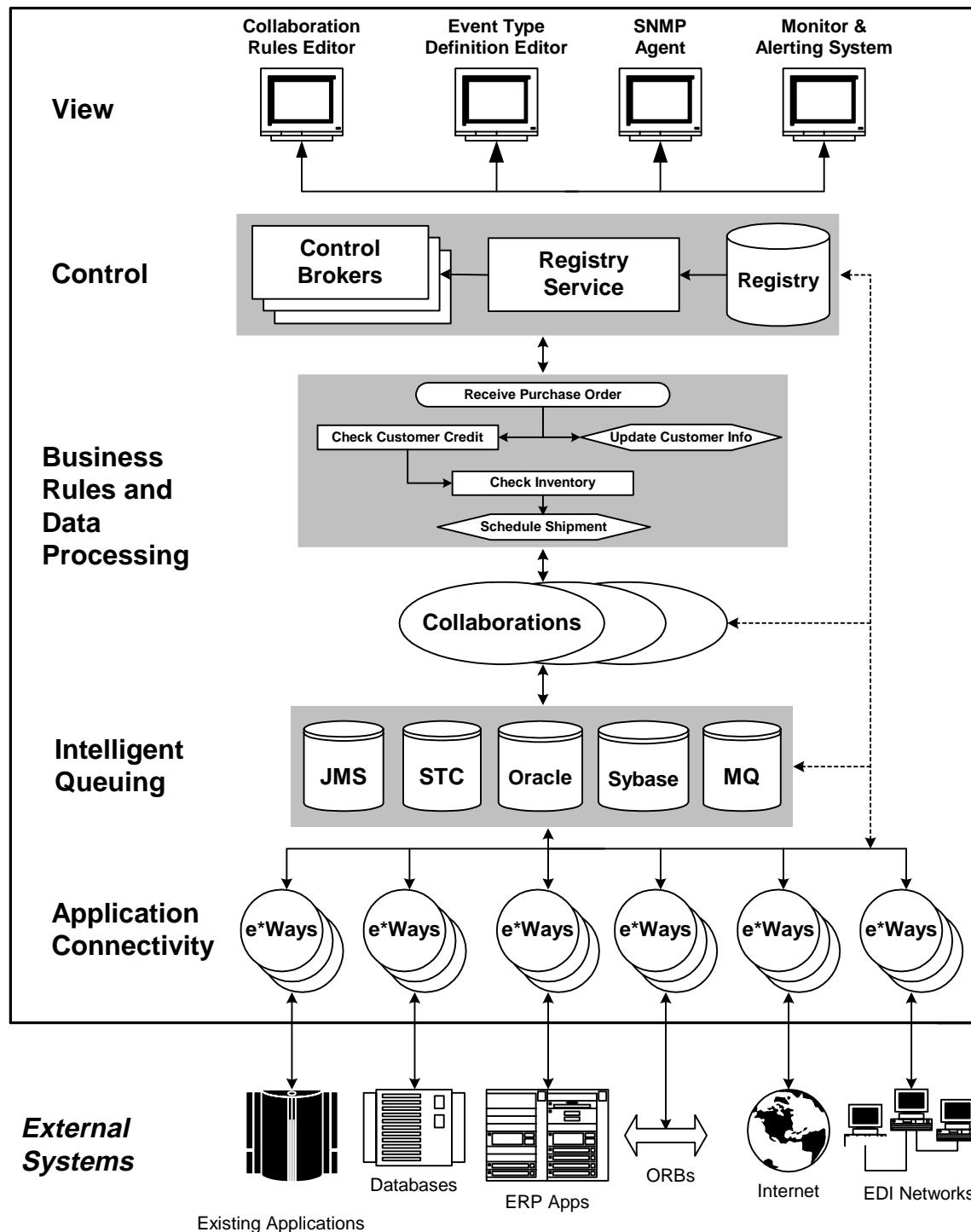
2.1.1 Component Organization and Schemas

You must organize your e*Gate system components into configurations called schemas. e*Gate lets you set up as many customized schemas as you want, to organize and to maintain the system efficiently. See “[Creating a Schema](#)” on page 86 for details on this feature.

2.1.2 Layered System Architecture

The e*Gate system has a layered architecture. The following figure shows a diagram of these layers.

Figure 1 e*Gate Layered Architecture



As shown in Figure 1, e*Gate architecture consists of the following functional layers:

- View
- Control
- Business rules and data processing
- Intelligent queuing
- Application connectivity

The rest of this chapter describes each of these layers, their components, and their general functions in the e*Gate system.

2.2 View Layer

This layer contains those components you interact with, that is, the e*Gate user interfaces. Most View-layer components implement graphical user interfaces (GUIs) to simplify their use. The system contains the following types of GUIs and viewing features:

- e*Gate Enterprise Manager
- e*Gate Editors
- Monitoring features

2.2.1 Enterprise Manager

The Enterprise Manager allows you to create and configure the components of the e*Gate system. Using the Enterprise Manager window, you can define and maintain your configuration schemas. See “[Enterprise Manager Basic Operation](#)” on page 49 for details on this feature.

2.2.2 e*Gate Editors

All e*Gate systems implement specialized editor GUIs, allowing you to set up, view, and revise each element in the system. The e*Gate editors include:

- SeeBeyond Java Event Type Definition (ETD) Editor (see [Chapter 5](#))
- SeeBeyond Monk Event Type Definition (ETD) Editor (see [Chapter 6](#))
- SeeBeyond Java Collaboration Rules Editor (see [Chapter 7](#))
- SeeBeyond Monk Collaboration Rules Editor (see [Chapter 8](#))
- SeeBeyond Collaboration-ID Rules Editor (see the **Note** below)
- e*Way Editor (see [Chapter 9](#))

Note: *The Collaboration-ID Rules Editor is a feature that enables compatibility with e*Gate Version 3.6 (DataGate) only. Normally, you do not need to use this GUI or its features. If necessary, see the **Collaboration-ID Rules Editor User's Guide** for more information on this feature.*

The purpose of the e*Gate Editors is to define the system's data processing structure and the relationships among system components.

Note: *The e*Gate term "Event," as used in this guide, means a package of data (see the **Glossary** on page 666).*

2.2.3 Monitoring Features

e*Gate contains the following GUI and other viewing features to help you with system monitoring, control, and maintenance:

- e*Gate Monitor
- e*Gate Alert Agent configuration tool
- e*Gate SNMP Agent

Note: *The Alert Agent and SNMP Agent are add-ons. See the **e*Gate Integrator Alert Agent User's Guide** and the **e*Gate Integrator SNMP Agent User's Guide** for additional information.*

- **stccmd**, a command-line application program interface (API) monitoring tool

For more information on these features and how they interact with the Enterprise Manager, see **Chapter 10**. For the details of operation, see the *e*Gate Integrator System Administration and Operations Guide*.

2.3 Control Layer

The control layer does the following e*Gate operations:

- It stores and distributes all system configuration information.
- It has the responsibility for starting up and shutting down e*Gate processes.
- It enforces access control mechanisms.
- It forwards Alert, status, and configuration messages to the correct agents.

Control layer components include

- e*Gate Registry, including the Registry Service
- Control Brokers

For more information on these features and how they interact with the Enterprise Manager, see the *e*Gate Integrator System Administration and Operations Guide*.

2.3.1 Registry

The e*Gate Registry does the following tasks:

- It stores all configuration details, either through references to supplemental configuration files or direct containment.
- It handles, through the Registry Service, all requests for configuration updates and changes the content of the Registry when new information is provided.
- It forwards any updates to appropriate clients as necessary.
- It employs a Team Registry concept that divides the Registry into:
 - ◆ Run-time environment.
 - ◆ Sandbox area for user-specific file development.

2.3.2 Control Brokers

The Control Broker component in a schema is responsible for:

- Starting and stopping processes.
- Selectively forwarding alert, status, and configuration messages to appropriate GUIs.
- Routing operational Events to scripts that help perform basic maintenance or administrative actions.

2.4 Business Rules and Data Processing Layer

This layer uses Collaboration Rules, and Collaborations to implement user-defined business logic in response to input Events.

2.4.1 Collaboration Rules

Collaboration Rules components have the following characteristics:

- They specify the details of how applications work together, for example, using data identification and transformation rules.
- You can define these rules, using the Collaboration Rules Editor. See [Chapter 8](#) for details on this feature.
- They can be written in Java or SeeBeyond's Monk language, or you can use another convenient programming language, for example, C.
- The e*Gate Collaboration Services interface with other languages, allowing you to define business rules using non-e*Gate tools.

See "[Creating Collaboration Rules and Scripts](#)" on page 106 for more information. For more information on Collaboration Services, see the *e*Gate Integrator Collaboration Services Reference Guide*.

2.4.2 Collaborations

Collaboration components allow you to:

- Define data mapping from n input Events to m output Events using Collaboration Rules, and
- Define how systems query databases in response to request Events, for example, how APIs having one or more applications can request coordinated action.

See “[Adding Collaborations](#)” on page 142 for more information.

2.5 Intelligent Queuing Layer

This layer provides for data queuing functions. e*Gate Intelligent Queue (IQ) components aid in data communication as follows:

- An IQ’s intelligence comes from its continual recording of Event-state information. This process ensures that the right data goes to the right places, in the correct sequence, and without duplication, even after hardware failures.
- IQs support the publish-and-subscribe (pub/sub) processing in e*Gate as follows:
 - **Publisher Components** send Events to IQs.
 - **Subscriber Components** receive queued Events.

See “[System Design Components](#)” on page 83 for more information on these features.

Specialized IQ features help out in e*Gate queuing as follows:

- **IQ Managers** reorganize IQs, archive IQ information, and lock IQs for maintenance.
- **IQ Services** provide the transport of components within IQs, handling low-level data exchange operations.

In addition to the SeeBeyond Standard IQ, e*Gate supports a number of different IQ types: for example, Java Messaging Service (JMS), memory loopback, Sybase, and Oracle. See “[Adding Intelligent Queues](#)” on page 136 for more information on IQs.

For more detailed information, including specific IQ types (for example, MQSeries), see the *e*Gate Integrator Intelligent Queue Services Reference Guide* and the *SeeBeyond JMS Intelligent Queue User’s Guide*.

2.6 Application Connectivity Layer

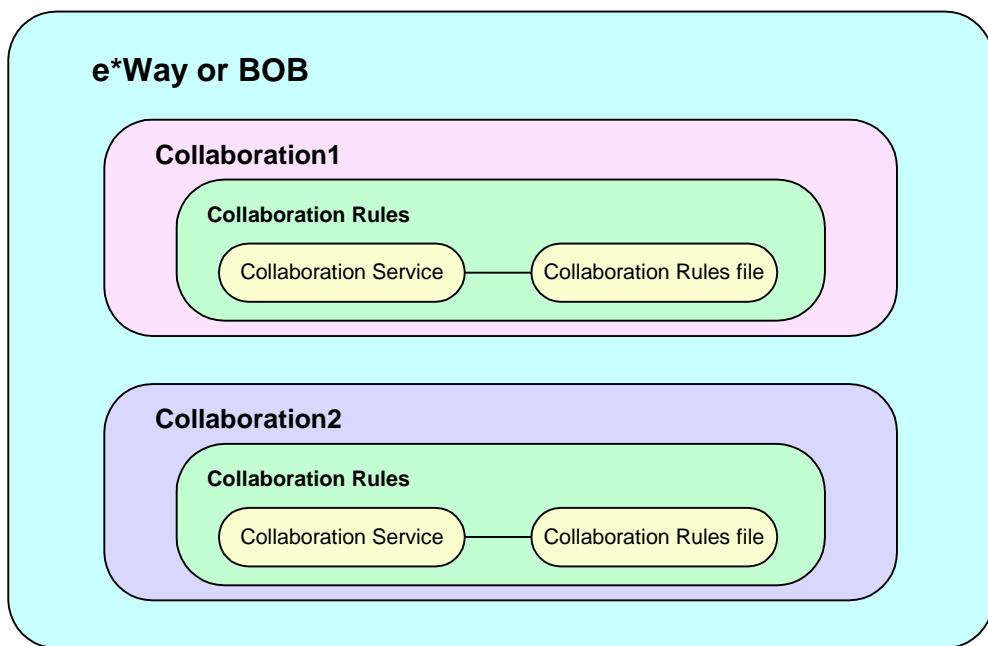
This layer consists of one or more e*Way and optional Business Object Broker (BOB) components. e*Ways are the points of contact between e*Gate and business applications.

e*Ways establish connectivity with external business applications, utilizing the appropriate communication protocols. You can use e*Ways for the following purposes:

- Connecting external business applications with the e*Gate system, while communicating with both external applications and IQs
- Receiving unprocessed data from external components, transforming it into Events and forwarding it to other components within e*Gate via IQs
- Sending processed data to external systems
- Integrating different applications, using the appropriate e*Way as an adapter on each end of the route to enable seamless Event flows

See “[Adding e*Ways and BOBs](#)” on page 123 for more details on these features. The following figure shows the relationship between e*Ways and Collaborations.

Figure 2 Collaboration/e*Way Relationship



Note: BOBs are optional and operate in the same way as e*Ways, except internally.

2.6.1 Types of e*Ways

The e*Gate system supports a number of application-specific e*Ways: for example, Multi-Mode, database access, SAP software, and generic. This section describes some examples of different types of e*Ways.

Multi-Mode

These e*Ways are maximally flexible and extensible; for example, any place you have a BOB, you can replace it with a Multi-Mode e*Way, and any time you have a Java-enabled external system, you can connect to it using a Multi-Mode e*Way and an e*Way Connection specific to that system.

Database Access

These e*Ways enable administrators to incorporate relational database access into enterprise-wide application integration strategies. These e*Ways can query any database in the system and allow instant, system-wide access to any data values. The system can automatically update any data changes across the system.

e*Ways for Database Access use the same GUIs as the rest of the e*Gate system. The GUIs can describe Event flows through the entire enterprise. e*Gate supports a specialized e*Way to access each of the following databases:

- Oracle
- Sybase
- ODBC
- DB2

Applications

The application-specific e*Ways are specifically designed to connect e*Gate to a particular applications or application suite; for example, the e*Way Intelligent Adapters for SAP have been specifically designed to connect e*Gate to SAP R/3 enterprise management software within a network of diverse hardware and software systems. Using one or more SAP e*Ways for this software, e*Gate acts as a hub between SAP R/3 and other software systems, or between differently configured SAP R/3 systems. Other application-specific e*Ways include PeopleSoft, Clarify, and Siebel.

Generic e*Way Kit

The Generic e*Way Intelligent Adapter Extension Kit provides a template, allowing you to design and build custom e*Ways for specific business requirements. The resulting e*Ways still incorporate core e*Gate technology but have more flexibility than specialized e*Ways.

Additional Applications

SeeBeyond continually develops new e*Ways with intelligent adapters for different uses. For a complete list, see the SeeBeyond Web site at the following URL:

<http://www.SeeBeyond.com/>

Also, you can contact SeeBeyond directly for the most current information on the availability of new e*Ways. For more information on how to configure e*Ways, see **Chapter 9**. For detailed information on a specific e*Way, see the appropriate SeeBeyond product's user's guide.

2.6.2 Business Object Brokers

A BOB component is similar to an e*Way in that it establishes connectivity and is capable of data transformation. Starting with e*Gate 4.5, BOBs became unnecessary;

everything that can be done by a BOB can now be done by a Multi-Mode e*Way. BOBs have the following properties:

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

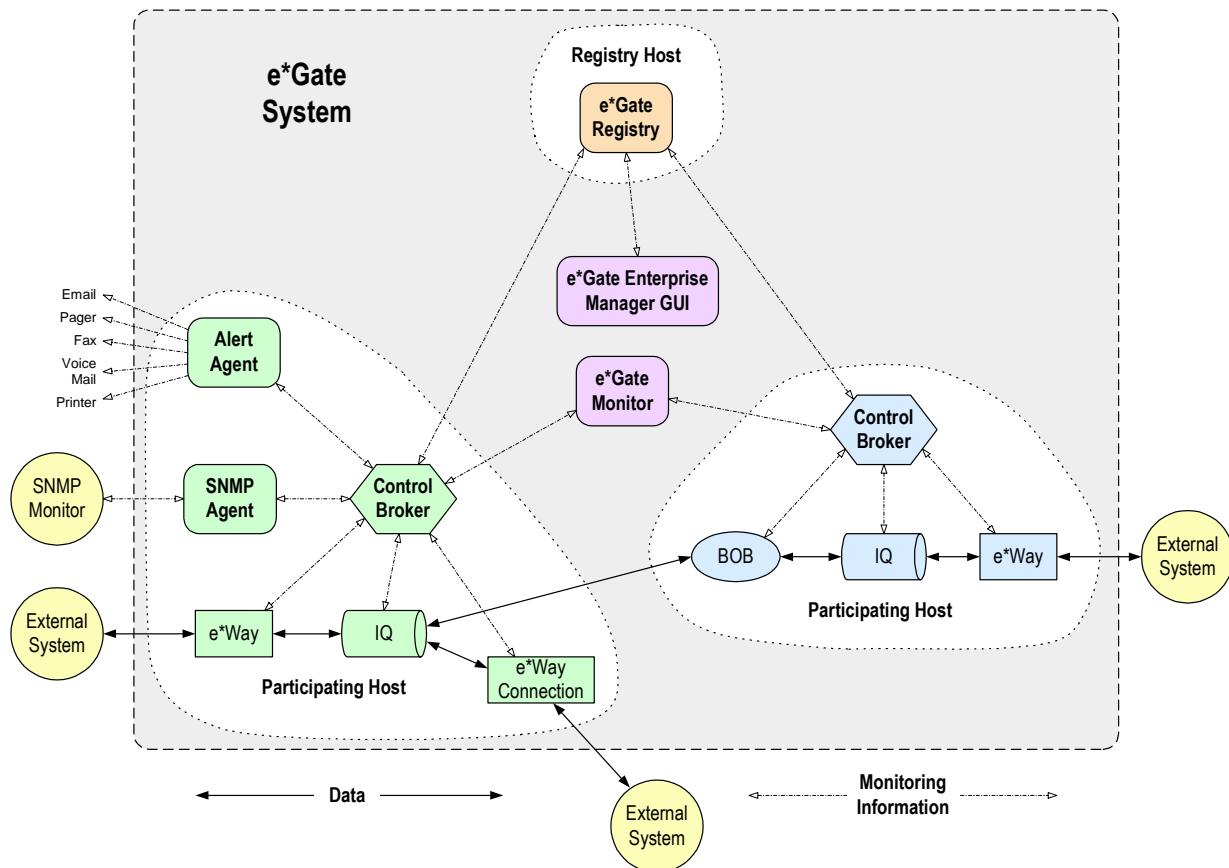
Note: For more information on BOBs, see “[Adding Business Object Brokers](#)” on [page 130](#).

2.7 System Setup

You set up your e*Gate system to meet your own business and information systems (IS) needs. In constructing this system, you must create and configure the components to operate smoothly together to accomplish the desired tasks. Once set up and started, the system runs according to the predefined plans you have implemented.

Figure 3 shows a diagram of a fully functioning sample e*Gate system.

Figure 3 Sample e*Gate System Diagram



As you can see in the previous figure, the different e*Gate layers explained in this chapter operate together in processing, transforming, and transporting data. The diagram shows how all the components of the various layers work with one another. A schema you design can encompass only one host or be spread across two or all the hosts.

The e*Gate GUIs provide you with the tools to set up all the necessary e*Gate schemas and components. See [Chapter 4](#) for a complete explanation of how to set up an e*Gate system.

Getting Started

This chapter gives you an introduction to the basics of preparation and how to use the e*Gate system and Enterprise Manager, including the online Help systems.

3.1 Overview of Starting with e*Gate

Getting started with e*Gate requires the following basic steps:

- Business analysis and preparation
- Enterprise Manager operation

Thorough analysis and knowledge of your business information system (IS) requirements are the necessary preconditions for setting up an e*Gate system. This step requires a complete examination of your current business and network operations, including anticipation of future requirements and expansion. In addition, make sure that your e*Gate system has all necessary e*Way Intelligent Adapters loaded and that all external systems/applications are ready for connection to e*Gate.

The Enterprise Manager feature is a graphical user interface (GUI) or window that allows you complete access to e*Gate. This tool lets you operate all software features necessary for setting up and editing your e*Gate system.

The rest of this chapter explains these steps. In addition, it also gives you a general introduction to the e*Gate online Help systems and how to use them.

3.1.1 System Requirements, Installations, and Upgrades

Because e*Gate system requirements and installation instructions can vary from network to network, see the *e*Gate Integrator Installation Guide* for details on these procedures. This guide also explains necessary considerations for specific types of networks and gives instructions for upgrading e*Gate.

3.1.2 Add-ons and Client Software

Some e*Way setups require add-on software not included in the standard e*Gate system installation. This condition applies to systems that must use any of the specialized e*Ways such as Siebel, SAP ALE, and COM/DCOM. Also, make sure any client software you need in order to connect to an external system is in place, for example, SQL*Net for Oracle databases or SAP GUI to communicate with SAP.

External systems can also require their own setups, installations, and configurations. Such configuration and operation of external systems is beyond the scope of this user's guide. See the appropriate SeeBeyond or other user's guide for details.

3.2 Business Analysis and Planning

Your total IS and business architecture determines the setup of your e*Gate system. e*Gate has a vendor-neutral, technology-neutral architecture that allows for rapid integration into any business/IS setup. As a result, e*Gate can, for example:

- Automate the connections between ERP systems and the rest of the supply chain;
- Integrate traditional EDI with the new XML-based business-to-business exchanges;
- Manage information exchange between already existing systems and Web servers;
- Use message-oriented middleware (MOM) queuing systems from IBM, Oracle, and Microsoft simultaneously, as well as a JMS (Java Message Service) implementation;
- Integrate systems based on COM, CORBA, and Java; and
- Serve as a universal gateway between Oracle, SQL Server, Sybase, Informix, DB2, and older-technology databases as well as graphically moving data between them.

For more definitions or explanations of e*Gate components and functions, see [Chapter 2](#) or the [Glossary](#) on page 666.

3.2.1 e*Gate System Setup Prerequisites

The following prerequisites are required before setting up your e*Gate system:

Deployment Plan

You must first have a complete business/IS analysis of your requirements; then use that information to create a plan, schedule, and budget for your system deployment.

System Design

Because e*Gate has multifaceted capabilities, you must carefully analyze your existing IS networks and applications, including their overall architecture. Also, you must do in-depth research into your business' current needs as well as anticipated future needs. Use this information to create a comprehensive design plan for your e*Gate setup.

See "[Creating a System Design](#)" on page 44.

Naming Conventions

Be sure to develop and use consistent nomenclature for your e*Gate system components. Failure to do so could result in confusion later on. See "[Naming System Elements](#)" on page 45.

System Preparations

Make sure you have the correct hardware and software prepared for your e*Gate installation. Gather all necessary information about the external systems you are using: for example, logical names, port numbers, and software version numbers. This

information is important when configuring and setting up the e*Ways in your e*Gate system. See “[System Preparations](#)” on page 48.

This section presents a brief overview of the necessary e*Gate business analysis and preparation tasks necessary before an e*Gate installation and setup, including the topics in the previous list. For more information, see the *SeeBeyond eBusiness Integration Suite Deployment Guide*.

3.2.2 Creating a System Design

When analyzing your business/IS operations preparing for an e*Gate installation, make sure to ask yourself the following basic questions:

- What are our existing systems, networks, and/or applications?
- What kinds of data and data files do we need to process?
- How do our different types of data exchanges flow?
- What kinds of data interactions are necessary?

Answering these questions correctly and completely diagrams your basic business/IS architecture. This information in turn maps out the basis of your e*Gate system. For more information about system design, see the *SeeBeyond eBusiness Integration Suite Deployment Guide*.

Information Gathering

You must gather the following basic types of information:

- **External Systems:** Determine the number and types of e*Ways you require. You could need different types of e*Ways or e*Way Connections to communicate with each of your existing IS systems, networks, and/or applications.
- **Kinds of Data:** Determine the type or types of data you will process. This information corresponds to e*Gate Event Types and their corresponding Event Type Definitions (ETDs). ETDs are defined data structures for a particular type of Event (packet of data).
- **Data Flows:** Determine where the data comes from and where it has to go. The e*Gate Collaboration components supply the system logic that routes Events through the system.
- **Data Interactions:** Determine the data transformation and verification processes for your system. Collaboration Rules and scripts in e*Gate define what Collaborations do with Events.

Make sure you systematically gather and organize all the information listed above. As you compile and analyze this information, it will suggest the quantity and structure of the e*Gate schemas your system requires.

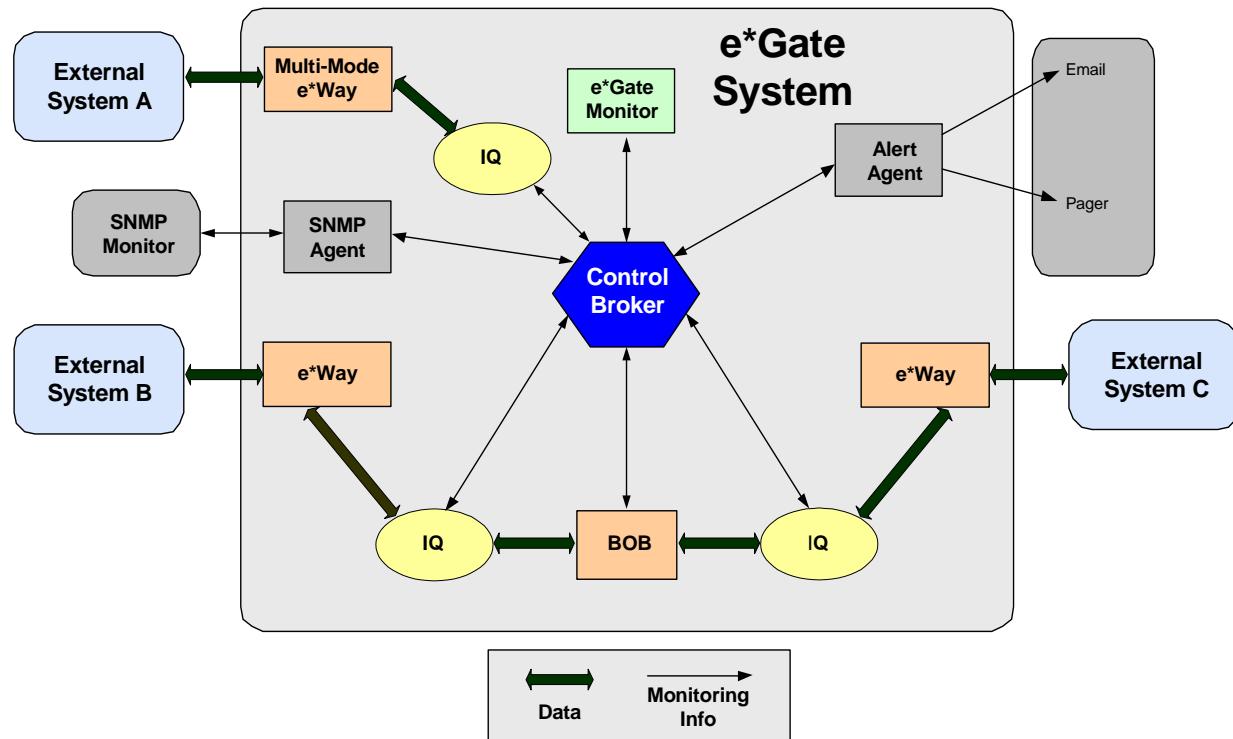
For more information about e*Gate system terms and components discussed in this section, see the [Glossary](#) on page 666 or the appropriate chapter of this user’s guide that explains the component.

System Structure

Figure 4 on page 45 shows the structure of a sample e*Gate system, showing the e*Way, internal-system, and external-system setup. For more information on system design and setting up an e*Gate system, see “[System Design Components](#)” on page 83.

Figure 4 below shows a simple e*Gate system. The system you create to serve your business/IS needs can be as simple or complex as you need it to be.

Figure 4 Sample e*Gate System Diagram



3.2.3 Naming System Elements

As you use the Enterprise Manager to set up components and other parts of your e*Gate system, the GUI prompts you to name them. The components include: e*Ways, Event Types, Collaborations, and Intelligent Queues (IQs). You name these items as you create them, and even though you are free to select any names you desire, make sure you develop consistent, comprehensible naming conventions before starting. Careful planning at the onset will avoid confusion later on.

In addition to components, you will enter names and/or text strings for various other system elements. This section explains the rules and conventions for these elements.

Make a Checklist

Write up a list of all the components and subcomponents linked together in the Enterprise Manager. Enter these names in the appropriate GUI tool, for example, the

e*Way Configuration Editor or **New IQ Component** dialog box. List your component names roughly in the order you will create them. [Chapter 4](#) explains these setup steps in detail.

The following list shows some sample components of a simple e*Gate system:

- Three e*Ways that do the following operations:
 - ◆ Receive inbound data,
 - ◆ Output valid data, and
 - ◆ Output invalid data.
- Three Collaborations that do the following operations:
 - ◆ Validate incoming data,
 - ◆ Output valid data, and
 - ◆ Output invalid data.
- Three types of Events with the following characteristics:
 - ◆ Inbound information,
 - ◆ Valid information, and
 - ◆ Invalid information.
- Two IQs that contain the following types of information:
 - ◆ Valid, and
 - ◆ Invalid.

For more information on creating e*Gate setup checklists, see the *SeeBeyond eBusiness Integration Suite Primer*.

Naming Conventions

This section explains naming conventions and rules in e*Gate.

System Components

Develop a consistent naming convention for all the components you listed in your checklist. Use easily recognizable names that reflect the function, operation, or content of the subject component. Names like “eWay_1” or “Collaboration_A” can be easy to come up with, but do not provide information about the component for later reference.

Make sure your naming convention can help give you a clear idea of what the named item does, what it contains, or how it functions in your e*Gate system. Table 1 shows

examples of a nomenclature for components in the previous naming list. Note how the naming convention reflects the concrete functions or content of named components.

Table 1 Sample Component Naming Convention

e*Way Names	Collaboration Names	Event Type Names	IQ Names
ew_Inbound	Output_Valid	et_ValidEvent	iq_Valid
ew_Inbound_Valid	Output_Invalid	et_InvalidEvent	iq_Invalid
ew_Outbound_Valid	Validate_Incoming	et_InboundEvent	

Naming Rules

The names of all e*Gate system components must obey the following rules:

- Names can be no more than 56 characters long.
- Names are case-sensitive; that is, the system would see CICS_eWay and CICS_EWAY as two different components.
- You can only use alphanumeric characters (letters and numbers), dashes, and underscores. If dashes are used, doublequotes must be used around the component name when it is used in command-line arguments.
- You cannot use spaces, commas, symbols, periods, or other punctuation.

Note: The system treats a User name as a component.

Data Files

In naming data files in your e*Gate system, use the file-naming conventions and maximum number of characters appropriate to the current host's operating system, for example Windows or UNIX. For details, see the user's guides for those systems.

Caution: Path names cannot have more than 256 characters, including the slashes. File and directory names cannot have more than 63 characters, including the extensions.

Other System Entries

Table 2 shows a list of naming conventions for other types of names and/or character strings you must enter in Enterprise Manager/e*Gate Monitor windows, dialog boxes, and properties dialog boxes.

Table 2 Additional e*Gate System Names

System Entry	Maximum Number of Characters	Character Usage
User name	56	Same as a component; case-sensitive.
Login password and confirm	64	Only alphanumeric characters, dashes, and underscores; case-sensitive.
Command arguments	255	Any characters allowed; case-sensitive.

Table 2 Additional e*Gate System Names (Continued)

System Entry	Maximum Number of Characters	Character Usage
Initialization strings	63	Any characters allowed; case-sensitive.
Network-related names	63	Any characters allowed; case-sensitive.
Service functions	255	Any characters allowed; case-sensitive.
ETD Editor (node names)	40 recommended	See " Naming Nodes " on page 206.

Note: For exact naming conventions in the Collaboration Rules and Collaboration-ID Rules editors, see the appropriate user's guide. Elements in these GUIs that are universal across e*Gate (for example, components, file names, and initialization strings) have the same naming conventions as those discussed in this section.

Caution: If you try to open a file with more than 6500 lines in the Collaboration Rules Editor, you get an error message, and the file does not open. If you need to open a larger file, use a text editor or word processor capable of handling large files.

3.2.4 System Preparations

In addition to doing a complete business/IS operations analysis, you must prepare your current systems for integration with e*Gate. This series of tasks entails ensuring that:

- The e*Gate host systems are in place and ready for e*Gate installation,
- Any/all external systems are ready for connection to e*Gate,
- The complete setup of any needed external interface hardware/software is present,
- e*Gate is successfully installed and all necessary e*Ways and BOBs are loaded in your e*Gate system.

Comprehensive instructions on these requirements differ depending on the details of your installation and are beyond the scope of this user's guide. See the *e*Gate Integrator Installation Guide* for more information. Also, see the individual installation instructions for any additional components and add-ons (for example, the installation and implementation chapters in your e*Ways' user's guides).

For more information on necessary preparations and planning, see the *SeeBeyond Business Suite Integration Deployment Guide*. Once you have completed the prerequisite tasks, you are ready to begin e*Gate development/setup operations. See [Chapter 4](#) for an explanation of this process.

3.3 Enterprise Manager Basic Operation

Use the Enterprise Manager GUI to set up and edit all elements of your e*Gate system. This section explains how to access and exit this window as well as its basic features.

To access the Enterprise Manager

- 1 Start the e*Gate Enterprise Manager using the icon on your Windows Desktop; or click the **Start** button and point to **Programs, SeeBeyond eBusiness Integration Suite**, and click **eGate Enterprise Manager**.

The **e*Gate Enterprise Manager** login dialog box appears.

- 2 Enter your Login ID and password. If necessary, select the appropriate Registry Host name from the **Server** list. See Figure 5.

Figure 5 e*Gate Enterprise Manager Login Dialog Box



- 3 Click **Log In** or press ENTER.

The **Open Schema on Registry Host** dialog box appears.

- 4 Select the appropriate schema name, then click **Open** or press ENTER.

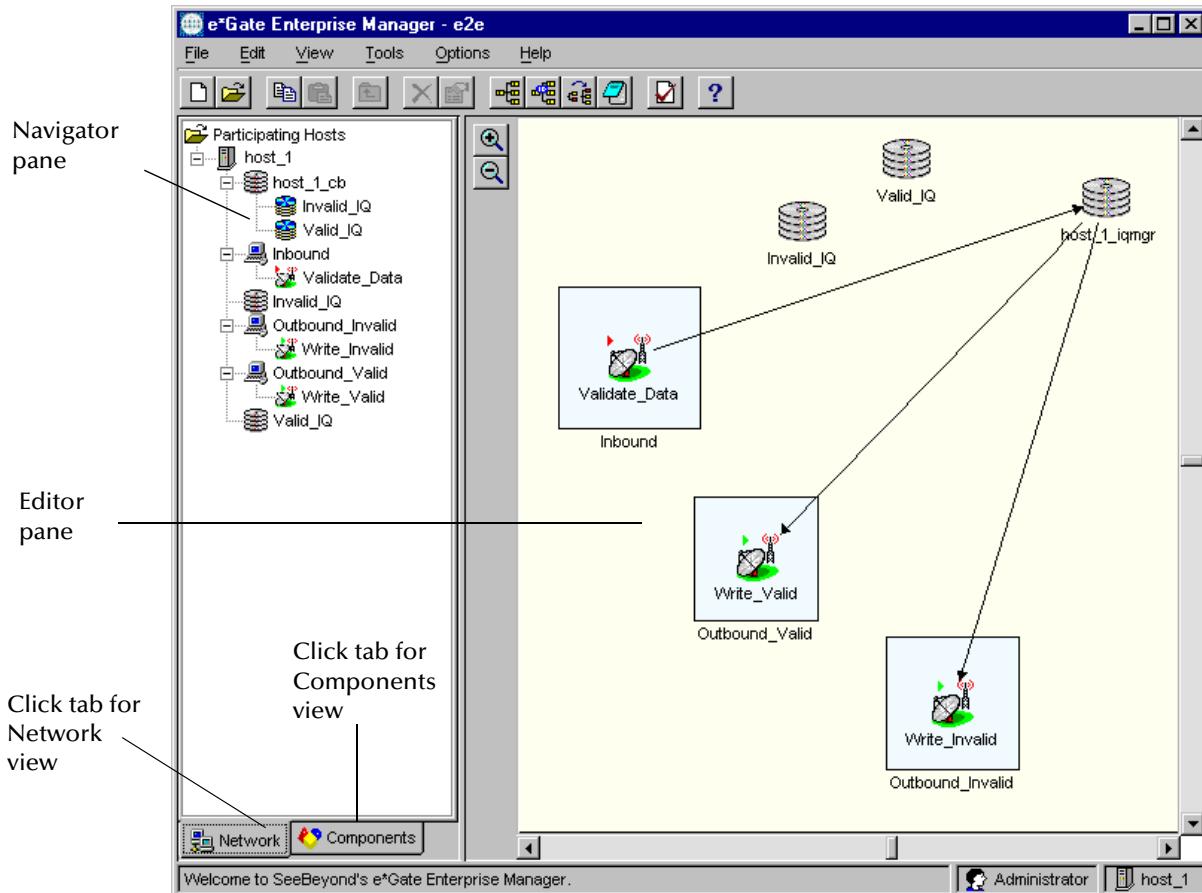
The Enterprise Manager window appears. See Figure 6.

- 5 Size and place the Enterprise Manager window as desired.

3.3.1 Enterprise Manager Window

Figure 6 shows an example of this window (with the Network view open).

Figure 6 Enterprise Manager Window (Network View)



The Enterprise Manager window contains the following major panes:

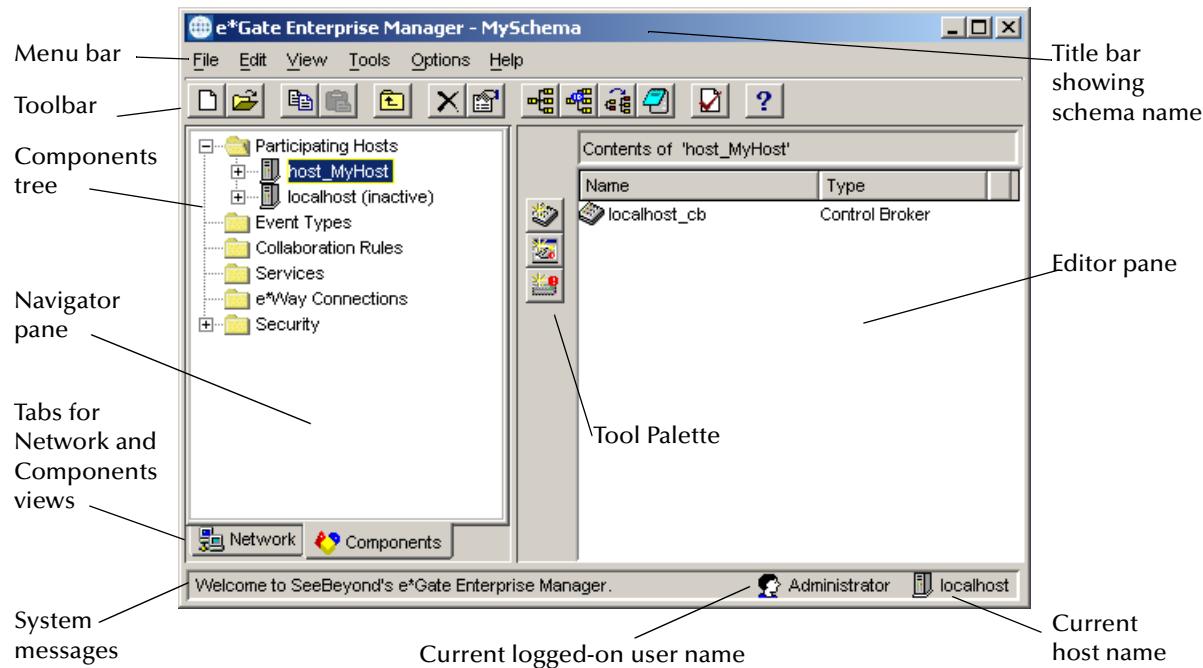
- The **navigator** pane allows you to choose views of the e*Gate system as follows:
 - ◆ The **Components** view lists your basic e*Gate setup components.
 - ◆ The **Network** view shows the basic e*Gate setup components in a graphical arrangement.
- The **editor** pane allows you to create or to edit system components' properties.

The **Network** view only shows the physical e*Gate components and is used for maintenance and troubleshooting purposes. See “[Using the Network View](#)” on [page 70](#) for details on this feature. The **Components** view shows elements you define and/or modify in the e*Gate system.

[Figure 7 on page 51](#) shows an example of the **Components** view in the window.

Note: The contents of the editor pane change according to the view.

Figure 7 Enterprise Manager Window (Components View)



Use the navigator and editor panes to operate the Enterprise Manager. Remember: You can only view components of one schema at a time, and all operations apply only to the current schema. The rest of this section explains basic GUI features of the Enterprise Manager window.

3.3.2 Components Tree

The **Components** view of the navigator pane contains a tree-like structure called the *components tree*. This tree displays the current schema's basic components, under the following folders:

- **Participating Hosts.** Lists all Participating Hosts that have been defined in the schema; you can expand the node to see the Control Broker, e*Ways, Business Object Brokers (BOBs), Collaborations, IQ Managers, and IQs associated with each Participating Host.
- **Event Types.** Lists all Event Types that have been defined in the schema.
- **Collaboration Rules.** Lists all Collaboration Rules defined in the schema.
- **Services.** Lists all the available Collaboration and IQ Services.
- **e*Way Connections.** Lists all e*Way Connections defined in the schema.
- **Security.** When visible, contains subfolders that list: the names of all users with access to the schema; all roles for the schema; and all privileges for the schema.

Note: The **Security** folder does not display for users with less than Administrator-level privileges in the system. See the **e*Gate Integrator System Administration and Operations Guide** for details on e*Gate security and access features.

Set up or edit any components in the components tree by first selecting that item and then using the appropriate Enterprise Manager features such as an editor or a Properties dialog box. [Chapter 4](#) explains how to use the components tree under the section explaining each component.

3.3.3 Toolbar Buttons

Table 3 below lists and describes the Enterprise Manager toolbar buttons. See “[Menu Bar](#)” on page 53 for more information on the buttons’ operation.

Table 3 Enterprise Manager Toolbar

Button	Command	Function
	New Schema	Allows you to create a new schema.
	Open Schema	Allows you to open an existing schema; you can only have one schema open at a time.
	Copy	Allows you to copy a selected item to the Windows clipboard.
	Paste	Allows you to paste a copied item from the Windows clipboard into a desired/appropriate location.
	Go Up to Parent Folder	Allows you to move up a level in the components tree (not associated with a menu command).
	Delete	Allows you to delete a selected item in either pane of the window. There is no undo.
	Properties	Displays a properties dialog box, allowing you to set up or edit properties of a selected item in either window pane.
	ETD Editor	Starts the ETD Editor, allowing you to create, define, and edit ETDs.
	ID Editor	Starts the Collaboration-ID Rules Editor (for backward-compatibility with e*Gate Version 3.6 only). For information on the ID Editor, see the e*Gate online help.
	Collaboration Editor	Starts the Collaboration Rules Editor, allowing you to create and edit Collaboration Rules.
	External Editor	Displays an Open file dialog box, so you can open and edit text files in the e*Gate system; same as the File > Edit File menu command.
	Monk Test Console	Displays the Monk Test Console window, allowing you to test Monk scripts for errors.
	Help	Allows you to access the Enterprise Manager’s online Help system.

Note: Specialized buttons appear in the Palette area of the window, depending on which levels of the components tree are open. **Chapter 4** explains these buttons under the sections for their corresponding components in the components tree.

3.3.4 Menu Bar

Table 4 lists the Enterprise Manager window menu commands and their functions.

Note: Menu command that can also be enabled by buttons display smaller versions of the appropriate buttons to the left of each option name.

File > New

Depending on which folder or component you have selected in the navigator pane, clicking on the **File** and pointing to **New** displays allows you to create the following new elements within the current schema:

- **Participating Host**
- **User**
- **Control Broker**
- **Monitor**
 - ♦ **Alert Agent**
 - ♦ **SNMP Agent**
- **Module**
 - ♦ **BOB**
 - ♦ **e*Way**
 - ♦ **IQ Manager**
 - ♦ **eInsight Engine**
- **Collaboration**
- **IQ**
- **IQ Service**
- **Event Type**
- **Collaboration Rules**
- **Collaboration Service**

For more information on these elements, see the **Glossary** on page 666 or the appropriate sections in **Chapter 4** (Setting Up e*Gate) as well as chapters that explain the appropriate e*Gate feature.

Note: A module is an e*Gate component that requires an executable (.exe) file.

Table 4 Enterprise Manager Menus and Commands

Menu	Command	Function
File	New	Displays a submenu that allows you to create new system components; see “ File > New” on page 53 for details. If an option name is dimmed, you cannot create it within the selected component.
	Login	Displays the e*Gate Open Schema Login dialog box, allowing a different user to log in to the system or the same user to log in to a different system.
	New Schema	Allows you to create a new schema.
	Open Schema	Allows you to open an existing schema; you can only have one schema open at a time.
	Export Schema Definitions to File	Displays the Select archive file dialog box, allowing you to select the archive file to hold export schema definitions and files. <i>Caution:</i> <i>Do not export default repository (registry) <eGate>\client\ folder.</i>
	Export Module Definitions to File	Displays the Select archive file dialog box, allowing you to select the archive file to hold exported module definitions and files. <i>Caution:</i> <i>Do not export default repository (registry) files to the <eGate>\client\ folder.</i>
	Import Definitions from File	Displays the Import from File dialog box, allowing you to select the archive file from which you want to import schema definitions and files.
	Edit File	Displays an Open file dialog box, letting you open and edit text files in the e*Gate system; see “ File > Edit File” on page 57 for details.
	Commit to Sandbox	Displays the Select Local File to Commit dialog box (similar to Open), allowing you to place files in the e*Gate Sandbox, where other users can access them. Here they are not available to the e*Gate system (see Chapter 4 for details).
	Promote to Run Time	Displays the Select File To Promote to Run Time dialog box (similar to Open), allowing you to remove files from the e*Gate Sandbox and place them in run time, that is, making them available to the e*Gate system (see Chapter 4 for details).
	Remove from Sandbox	Displays the Select File to Remove from Sandbox dialog box (similar to Open), allowing you remove files from the e*Gate system Sandbox.
	Retrieve File from Registry	Displays the Select file to retrieve dialog box, allowing you to move through a directory structure until you select the file to retrieve.
	Exit	Exits the Enterprise Manager and closes the window (see “ To exit the Enterprise Manager” on page 59).

Table 4 Enterprise Manager Menus and Commands (Continued)

Menu	Command	Function
Edit	Move	Displays the Move dialog box, allowing you to move IQs, IQ Managers, Collaborations, e*Ways, and BOBs from one Control Broker/Participating Host to another (see Chapter 4 for details).
	Copy	Allows you to temporarily place a copy of a selected item into the Windows clipboard (if the option name is dimmed, you cannot copy the current item); this function only copies the selected item and not its associated components, depending on whether the Recursive Copy feature is activated (see also Recursive Copy under the Options menu).
	Paste	Pastes components you last placed in the Windows clipboard into a selected location in your navigator tree, including into another Participating Host if you want (if the option name is dimmed, there is nothing in the clipboard to paste). Components have limitations on where they can be pasted; see the section discussing a given component, in Chapter 4 .
	Copy Multiple	Operates like Copy except that it copies the selected component only within the current Participating Host, at the current location in the Navigator Tree. It also opens a dialog box asking you how many times you want to copy the component (see also Recursive Copy under the Options menu). <p><i>Note: If you are duplicating a large component or a large number of components, allow plenty of time for this operation to finish.</i></p>
	Rename	Allows you to rename a selected item in either pane of the window (if the option name is dimmed, you cannot rename the current item).
	Delete	Allows you to delete a selected component in either pane of the window (if the option name is dimmed, you cannot delete the current item). This is not a recursive delete, that is, it does not delete all components within a selected item. You can reassign associated components of a deleted item.
	Properties	Displays a properties dialog box, allowing you to set up or edit properties of a selected item in either pane of the window (if the option name is dimmed, there is no properties dialog box for the selected item).

Table 4 Enterprise Manager Menus and Commands (Continued)

Menu	Command	Function
View	Summary	Allows you to view lists of system components and elements (see “ View > Summary ” on page 58 for details on this option).
	Refresh	Reloads the current schema from the e*Gate Registry to display its current components and setup, if, for example, you have made changes. It also renews the monitor display.
	Reset Layout	If you have changed the Network view’s component layout for the current schema, choose this option to return to the system default layout (only available for the Network view).
	Save Layout	If you have changed the Network view’s component layout for the current schema and want to keep it, choose this option to save your layout with the schema (only applies to the Network view).
Tools	ETD Editor	Displays the ETD Editor window, allowing you to create, define, and edit ETDs, that is, the definitions of your Event Types (see Chapter 6 for details).
	ID Editor	Displays the Collaboration-ID Rules Editor window, allowing you to create and edit rules for data verification within a Collaboration, for backward-compatibility with e*Gate Version 3.6 only (see the <i>Collaboration-ID Rules Editor User’s Guide</i> for more information, if necessary).
	Collab Editor	Displays the Collaboration Rules Editor window, allowing you to create and edit Collaboration Rules scripts.
	Monk Test Console	Displays the Monk Test Console window, allowing you to test Monk scripts (see “ Monk Test Console ” on page 149 for details).
Options	Toolbar Text	Acts as a toggle to show or hide text labels for all toolbar and tool palette buttons.
	Roll Over Toolbar	Acts as a toggle to show or hide dynamic shadow boxes around toolbar and tool palette buttons.
	Recursive Copy	Acts as a toggle to affect the result of the Copy or Copy Multiple menu command. When activated, allows you to copy/duplicate and paste a highlighted item and all its associated components; acts as a toggle. This feature is only available in the Components view but, after you use it, the results show up in both views. The operation appends sequential numbers, starting with _0 , to the name of each new component created.
	Update Monk Libraries on Load	Acts as a toggle to affect the behavior when a schema is loaded. When activated, causes all Monk libraries to be refreshed each time a new schema is loaded.
	Change Password	Displays the Change Password dialog box, allowing you to change your current password.
	Default Editor	Allows you to select Java or Monk as the default editor to use for editing Collaborations and ETDs. The system defaults to the editor open when you ended your previous e*Gate session.

Table 4 Enterprise Manager Menus and Commands (Continued)

Menu	Command	Function
Help	e*Gate Help Topics	Allows you to access the Enterprise Manager's online Help system and to open its Help window (see " Online Help Systems " on page 73 for details).
	ESRs Applied	Provides information on code patches that have been applied to your e*Gate system.
	About e*Gate	Provides e*Gate version information and related copyright information.

For complete information on the e*Insight Business Process Manager engine, see the *e*Insight Business Process Manager Implementation Guide*.

For complete information on the SNMP and Alert agents, see the *e*Gate Integrator SNMP Agent User's Guide* and the *e*Gate Integrator Alert Agent User's Guide*.

File > Edit File

This menu command allows you to edit text files within the e*Gate system without having to exit or otherwise leave the Enterprise Manager. You can also use this option to create a new text file for system use, if desired.

For example, you can edit a text file associated with a particular e*Way. Use this feature to open the file, for example, in the Windows Notepad. Then edit the file, save it, and commit it to the e*Gate Sandbox, if desired.

To edit a file, using File > Edit File

- 1 Choose this option to display a Windows **Open** file dialog box.
- 2 Select the desired file, then click **Open**.
- 3 Edit the file, using Windows **Notepad** or another text editor. You can also use **Notepad** to create one or more new files or edit additional files.
- 4 When you save your first file, you receive a prompt with a warning message asking whether you want to commit the file to the Sandbox. Click **Commit** or **Cancel**.

Note: *The system only prompts you to commit when you save the first file opened. If you want to commit more files to the Sandbox, on the File menu, click Commit to Sandbox.*

Sandbox and Run-Time Environments

Whenever you create and edit files within e*Gate, the files are stored in your Sandbox, an area in the e*Gate Registry specific to your e*Gate user name. The Sandbox is a user's local development area. Each user has his or her own Sandbox. Files in a user's Sandbox are available for testing the functions in the file themselves, but they are not available to the run-time schema. The run-time environment is the production environment for a schema.

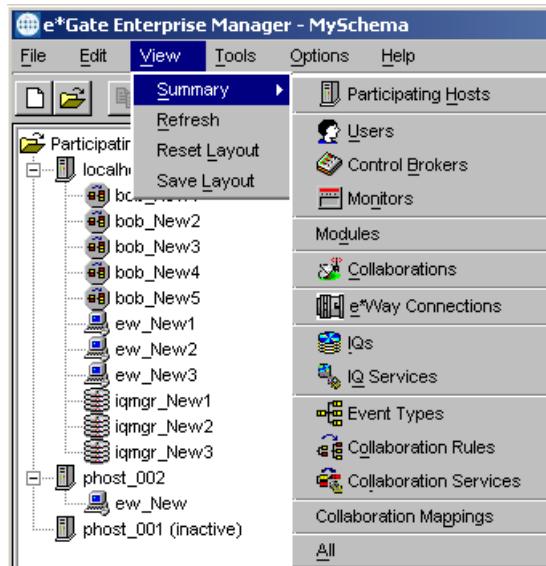
Important: If you perform a full Schema Import (for example, using `stcregutil -fi`), you import all Sandbox files and settings from the export file, overwriting your own.

The e*Gate system's Team Registry allows this division into Sandbox and run-time environments. For more information on working with e*Gate's Team Registry see “[Codeveloping in e*Gate: Using the Team Registry](#)” on page 60.

View > Summary

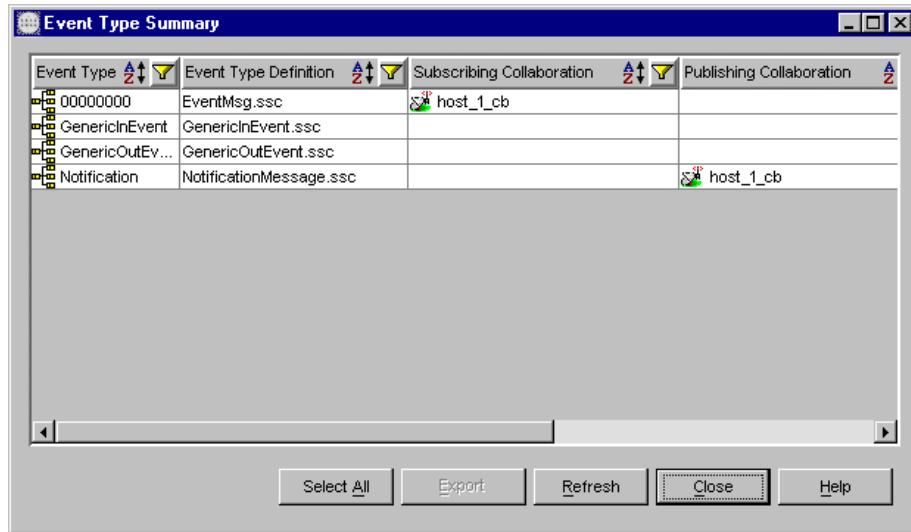
This command lists basic e*Gate system elements; see Figure 8. Choosing an option from this list opens a secondary window, summarizing the components of that element within the current schema.

Figure 8 View > Summary



The command opens a different window appropriate to each of its suboptions. For example, choosing **Event Types** displays the Event Type Summary window; see Figure 9. This window provides a list of the current schema's Event Types and ETDs.

Figure 9 Event Type Summary Window



Many of these summaries give additional information about the components of the selected e*Gate element, for example, subscribing and publishing Collaborations for ETDs (see “[Creating Collaboration Rules and Scripts](#)” on page 106 for details). Other summaries list only the basic components or elements named in the menu command.

For more information on these elements, see the [Glossary](#) on page 666 or the appropriate sections in [Chapter 4](#) (Setting Up e*Gate) and [Chapter 10](#) (Introduction to e*Gate Monitor) as well as chapters on the appropriate e*Gate feature.

Note: Choosing **All** opens the **All Components Summary** window, displaying all system components and elements in the previous list.

Shortcut Menus

The Enterprise Manager window contains a number of shortcut menus that do the same operations as their menu bar or toolbar counterparts, such as accessing properties dialog boxes: Simply right-click the pane or feature and click a command from the shortcut menu that pops up.

To exit the Enterprise Manager

- 1 On the **File** menu, click **Exit**.

A message appears asking if you want to exit.

- 2 Click **OK** to close the Enterprise Manager window and exit the program.

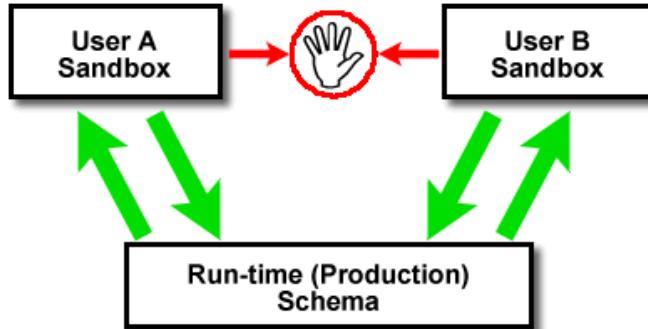
3.4 Codeveloping in e*Gate: Using the Team Registry

e*Gate supports a powerful “Team Registry” system that enables multiple users (developers or system administrators) to develop components of a single schema simultaneously.

Like a standard version-control system, the “Team Registry” system enables individual users to check out files to work on them, and then to promote those files from the test or “sandbox” (see [“The Sandbox” on page 61](#)) environment to the production “run-time” environment. To protect each individual’s work, the system cautions users whenever they attempt to edit a checked-out file. This system enables multiple users to develop and test components of an e*Gate schema while preserving both the individual’s work and the integrity of the running schema.

When a user edits an e*Gate file, such as a Collaboration Rules file, an Event Type Definition (ETD) file, or an e*Way properties definition file, that file is copied to that user’s Sandbox. Each e*Gate user has a unique Sandbox, and the Team Registry will not allow one user to edit another’s file without warning him first.

Figure 10 Team Registry: Overview

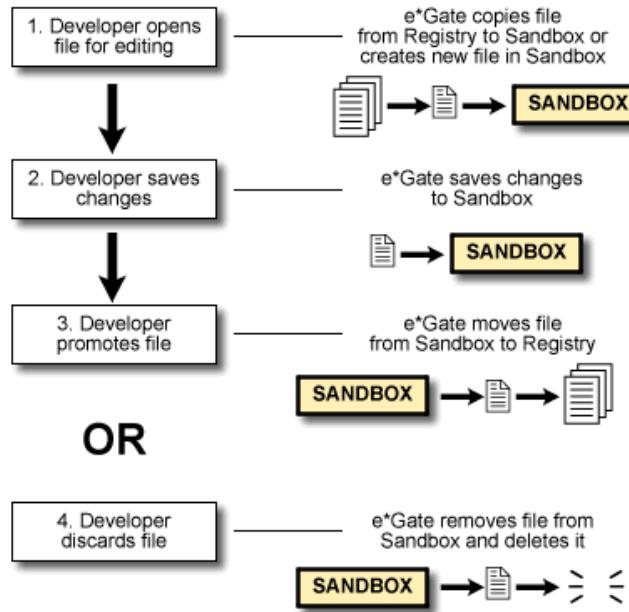


The following is an example of how a typical e*Gate working session might proceed:

- 1 Using the appropriate e*Gate editor, a developer creates a new file (or opens an existing file).
- 2 The developer makes changes, saves them, and tests them.
- 3 When finished working on the file, the developer either
 - moves the file into the running schema
 - or
 - abandons the changes and deletes the file.

Figure 11 is an example of how the process works from the perspective of the Team Registry features:

Figure 11 The Team Registry in Operation



All Team Registry functions are available through the e*Gate editors as well as through the **stcregutil.exe** command-line utility (see the *e*Gate Integrator System Administration and Operations Guide* for more information on this utility).

3.4.1 Important: User Name Requirements

You *must* create a unique e*Gate user name for each developer who will be using Team Registry features. If all users do e*Gate development under a generic user name like “Administrator,” you will lose all benefit from the Team Registry features.

See the “Security” chapter of the *e*Gate Integrator System Administration and Operations Guide* for more information about user names and e*Gate security.

3.4.2 The Sandbox

When you create and edit files within e*Gate, the files are stored in your Sandbox, an area in the e*Gate Registry specific to your e*Gate user name.

Sandbox Properties

The e*Gate Sandbox has the following properties:

- Files in your Sandbox are not part of the run-time schema, so any changes you make will not affect how data is processed in the schema. Files within your Sandbox are not available to the e*Gate components (such as e*Ways or BOBs) that use them. Once you have fully tested a file and are ready to put it into use, you must *promote* (move) it to the run-time schema (see “[Promoting Files](#)” on page 63).
- The first time you open an existing file in an e*Gate editor, e*Gate *commits* (copies) the file to your Sandbox. Once you open a file in the Sandbox, it will remain there—

even if you save it—until you either promote it to the run-time schema or manually remove it from your Sandbox (see “[Removing Files” on page 64](#)).

Important: If you perform a full Schema Import (for example, using `stcregutil -fi`), you import all Sandbox files and settings from the export file, overwriting your own.

Sandbox Operation

When you open an existing file in an e*Gate editor GUI, the file you request appears on screen (assuming no locks—see “[Advisory Locks” on page 62](#)—are placed on that file by another user). Behind the scenes, e*Gate does the following:

- If the file currently exists in your Sandbox, e*Gate opens the file from there, regardless of whether or not the file also exists in the run-time schema.
- If the file does not exist in your Sandbox, e*Gate looks for it in the run-time schema. If it exists there, e*Gate copies the file to your Sandbox and opens the copy in the editor.
- If e*Gate does not find the file in either your Sandbox or the run-time schema, it looks for the file in the Registry Host’s default schema. If it finds the file there, e*Gate copies the file to your Sandbox and opens the copy in the editor.

Committing Files

When you take a file from the run-time schema and move it to the Sandbox, you commit the file to your Sandbox. The system then places an advisory lock on the file.

Advisory Locks

When e*Gate places a file in your Sandbox, it also “checks out” the file by placing an advisory lock on that file within the schema. An advisory lock indicates that the file is checked out by a specific user and a copy exists in that user’s Sandbox. This lock will remain in place until the file is removed from the Sandbox in one of the following ways:

- The user who checked out the file promotes the file.
- The user who checked out the file removes the file from the Sandbox without promoting it.
- The user exits the e*Gate editor without ever saving the file.

The locks are “advisory,” and do not prevent you from making changes to files checked out by other users. When e*Gate encounters a lock, it issues a warning message advising you that the lock exists. You have the choice to either ignore the lock and edit the file, or to respect the lock and abort the attempted edit.

Important: We strongly recommend that you respect advisory locks whenever possible.

In addition to risking loss of work or duplication of effort that a version-control system prevents, ignoring the locks can have the following consequences:

- e*Gate does not “re-lock” a file under a new user name, so subsequent users who come upon the lock will find it assigned to the original user, not the last user who opened it. As a result, you are unable to track the true “ownership” of the file.

- If multiple users have ignored locks and checked out the same file, the last person to promote a file “wins.” All previously promoted versions are overwritten. As a result, multiple developers could promote multiple versions of the same file and cause the running schema to function in an undesirable manner.

Important: *If you want to edit a file that is already checked out by another user, we strongly recommend that you coordinate your efforts with that user, rather than proceeding and ignoring the advisory lock.*

Team Registry File Operations

Creating, Editing, and Unediting Files

Files are always created and edited in your Sandbox. After the file opens in the appropriate editor GUI, edit it as desired.

Caution: *If the editor warns you that an advisory lock exists on the file, we recommend you do not proceed (see “[Advisory Locks](#)” on page 62 for more information).*

Note: *The “Unedit” option performs in a similar fashion as “Undo.” It allows you to undo all of your edits to a file and return it to its state prior to editing.*

Committing Files

When finished editing (or creating) the file, save it. When you save the file, e*Gate commits (saves) it to your Sandbox.

Note: *You cannot save a file directly to the run-time schema using the Save menu command.*

Promoting Files

Once you have fully tested a file in the Sandbox and are ready to put it into use, you must promote it to the run-time schema.

Note: *Before placing the file in the run-time schema, you must first save it to the Sandbox.*

After saving a file to the Sandbox, you are ready to promote it into the run-time schema. When you promote a file, you update the run-time schema to use the new file. If the file already exists in the run-time schema, that file is replaced with the file from the Sandbox. Promoting a file automatically removes it from the user’s Sandbox and, if the file had an advisory lock, releases that lock. For more information on the importance of promoting files—particularly if the file is a new file that does not exist in the run-time schema—see “[Testing Schemas: Run-time and Sandbox Considerations](#)” on page 65.

You can use any of the tools in Table 5 below to promote files from the Sandbox to the run-time environment.

Table 5 Tools to Promote Files to Run-time

Tool to promote files from the Sandbox to run-time	Where to find more information
e*Gate editor	Each editor's Help system or the appropriate chapters in this manual
e*Gate Enterprise Manager	Enterprise Manager Help system or the appropriate sections in this manual
stcregutil.exe (command-line utility)	<i>e*Gate Integrator System Administration and Operations Guide</i>

Caution: Do not promote files that are locked by other users. If you promote a file whose advisory lock is not assigned to you, the advisory lock will remain, but assigned to the original user.

Removing Files

At times you may want to modify different files and test them, but you do not want the changes to go into the run-time schema. In this case, you must remove the unwanted file from your Sandbox to release the advisory lock that was placed on the file when you checked it out.

You can use any of the tools in Table 6 below to remove files from the Sandbox.

Table 6 Tools to Remove Files from the Sandbox

Tool to remove files from the Sandbox	Where to find more information
e*Gate editor	Each editor's Help system or the appropriate chapters in this manual
e*Gate Enterprise Manager	Enterprise Manager Help system or the appropriate sections in this manual
stcregutil.exe (command-line utility)	<i>e*Gate Integrator System Administration and Operations Guide</i>

Important: You must either promote or remove a file from your Sandbox—using the appropriate command from the *File* menu (**promote** or **remove**)—to release the advisory lock. Simply deleting the file from the directory will not release the lock.

Do not remove files that are locked by other users. If you remove a file whose advisory lock is not assigned to you, the advisory lock will remain, assigned to the original user.

3.4.3 Testing Schemas: Run-time and Sandbox Considerations

There is an important difference between testing a file and testing a schema.

- A **file** can be tested using the test facility within the e*Gate Enterprise Manager or the command-line utility **stctrans.exe**; it does not need anything from the run-time environment. You should test files within the Sandbox before promoting them to the run-time environment to make sure that all of the program logic within those files works as expected.
- A **schema** can only be tested when it is running and all components, including the files you want to test, are loaded within that schema. All files required by the schema must be accessible to you, either in the run-time environment or in your own Sandbox. No e*Gate executable component (such as an e*Way or BOBs) can access a file in the Sandbox of another user. In other words, if you want to see how other user files interact with an e*Gate component, you must have the other users promote their files to the run-time environment.

Remember this distinction when you create a new file by starting the e*Gate editor directly from a component's Properties dialog box. For example, you might take the following steps to create and test an e*Way:

- 1 Create the e*Way in the e*Gate Enterprise Manager.
- 2 Assign an executable file to the e*Way and create a configuration file by starting the Configuration Editor from the e*Way Properties dialog box.
- 3 In the Configuration Editor, save the file.

At this point, the file exists in your Sandbox, and the name of the file appears as expected in the e*Way's properties dialog box. However, until you promote the file, it does not actually exist within the run-time schema. If another user tries to run the e*Way before you promote the file, their e*Way will not find the file and an "Unable to load module configuration" error occurs.

Furthermore, if another user now creates and promotes a different version of the same file you have in your Sandbox, you will be unaware of it— your Sandbox version takes priority over the run-time version. When you run the schema, you may see behavior different from the behavior seen by all other users.

To avoid this problem, take the next step *before* exiting the e*Way Editor:

- 4 Promote the file.

Important: *We recommend that you always promote new files before you exit the e*Gate configuration editor in which you create them. This will avoid problems created by e*Ways or BOBs attempting to load files that exist in a Sandbox rather than within the run-time schema.*

Changing Default Check-in/Check-out Actions

A command file governs the actual mechanics of file check out, check in, and promotion to the run-time schema. Most e*Gate installations never require any modifications to this file. If you need more information, see the *e*Gate Integrator System Administration and Operations Guide*.

Sandbox/Run-time Registry Directory Structure

From a user's point of view, the Sandbox and the run-time Registry use the same directory structure, and all files are accessible using the e*Gate Enterprise Manager's file-selection dialog boxes.

System administrators who need additional details regarding the location of files within the Registry's file repository should consult the *e*Gate Integrator System Administration and Operations Guide*.

3.4.4 Team Registry and Component "Run As" Settings

A special feature of the Team Registry enables you to run schemas with Sandbox files, without requiring you to first "check in" the files to the run-time schema.

By default, all e*Gate components are run under the Administrator user name (see the *e*Gate Integrator System Administration and Operations Guide* for more information about e*Gate's "run as" feature). However, if you change a component's "run as" user name to the name of a user who has created Sandbox files, e*Gate allows those components to load files from that user's Sandbox.

For example, if the user "peter" places a Collaboration Rules script in his Sandbox, he can change the "run as" property for an e*Way to "peter." That e*Way will then be able to load the Collaboration Rules script just as though it had been checked into the run-time Registry.

This feature enables you to test Sandbox files within a running schema without affecting any files currently in use. For example, you could switch the "run as" property to test a given user's Collaboration Rules script within the running schema, then switch back to the original version by running the component under the original user.

Important: Whether or not you choose to use this feature, we recommend that you create an e*Gate user name for each developer who creates files for the e*Gate system, and require developers to use their own user name, rather than the Administrator user, when checking files in or out of the e*Gate Registry.

3.4.5 Team Registry and Version-control Systems

You can integrate the Team Registry with your local version-control systems by modifying the command that starts the e*Gate Registry service (**stcregd.exe**). For example, you can specify an external version-control system as an argument for the **-extvcdll** flag.

For instructions and more information, see the section on **stcregd.exe** in the *e*Gate Integrator System Administration and Operations Guide*.

3.5 Adding New Participating Hosts

If necessary, add additional Participating Hosts that your e*Gate system requires before starting. Installation procedures present the option of adding hosts, but you can also add new ones at any time using the Enterprise Manager window.

When your Participating Host is not a GUI Host

If you have a Participating Host running on a remote machine, you must first activate it using the **stcinstd** command:

```
stcinstd.exe -rh <RegHost> -rs <SchemaName> -un <user> -up <password>
```

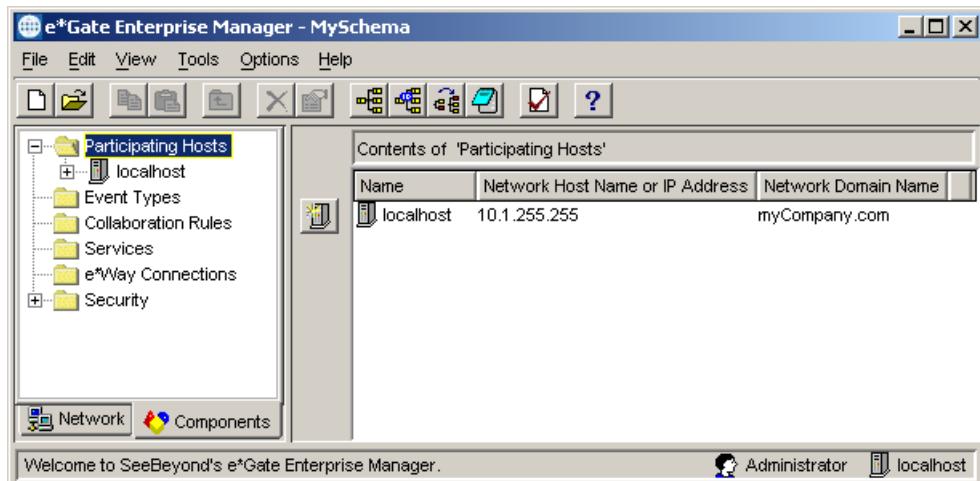
For more information on how to add, configure, and activate Participating Hosts, see the *e*Gate Integrator System Administration and Operations Guide*.

To add a new Participating Host

- 1 In the components tree, click the **Participating Hosts** folder.

A list of one or more Participating Hosts appears in the Editor pane. See Figure 12.

Figure 12 Enterprise Manager Window With Participating Hosts



- 2 In the Palette, click .

The **New Participating Host Component** dialog box appears.

- 3 Enter the name of the new Participating Host and then take one of the following actions:

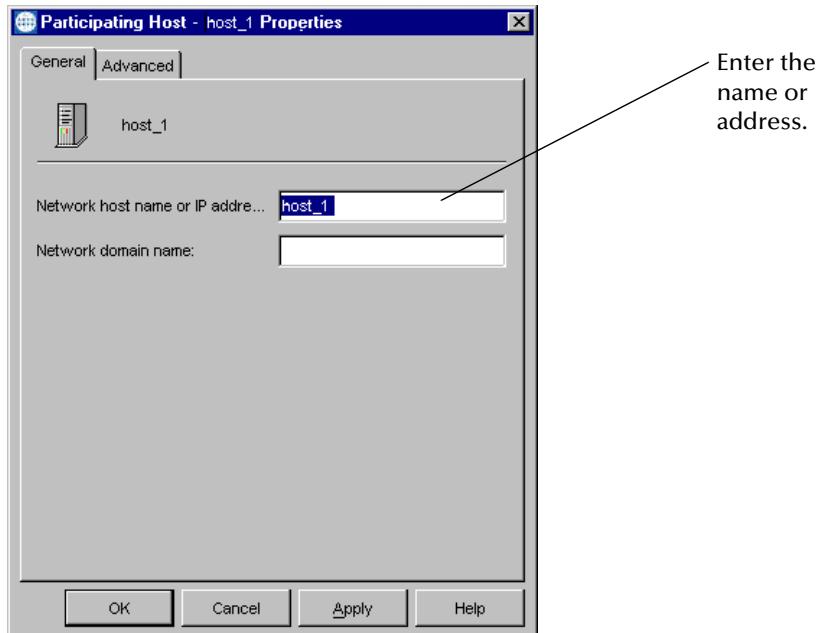
- ♦ Click **Apply** to enter the new Participating Host in the system and leave the dialog box open to create another.
- ♦ Click **OK** to enter the new host in the system and close the dialog box.

The new Participating Host name appears under the **Participating Hosts** folder and in the list in the Editor pane.

- 4 When you are finished naming all hosts, select a newly created host and, on the toolbar or **Edit** menu, click  **Properties**.

The **Participating Host Properties** dialog box appears. See Figure 13.

Figure 13 Participating Host Properties Dialog Box



- 5 In the properties dialog box, enter the following information:
 - ♦ Network host name or IP address
 - ♦ Network domain nameFor a list of the number and types of characters you can use in network-related names, see [Table 2 on page 47](#).

Note: *The Network host name for an activated Participating Host must match the name of the machine where its Control Broker will be run.*

- 6 Click the **Advanced** tab and then click **Threshold Setup** to set the disk threshold parameters, if desired. The **Disk Threshold Settings** dialog box opens.
- 7 To change the current settings, click **Change**. The **Threshold Properties** dialog box opens. See the *e*Gate Integrator System Administration and Operations Guide* for details on how to set or reset these parameters.
- 8 When finished click **OK** twice to return to the **Participating Host Properties** dialog box.
- 9 To save the Participating Host properties, click **Apply** to enter them into the system.
- 10 When finished, click **OK** to close the properties dialog box.

Repeat this procedure as needed to create new Participating Hosts.

Note: *Creating a Participating Host notifies the Registry that a Control Broker will be running on that host, but does **not** create a Control Broker on the host; this can be done either manually or as part of the Participating Host installation.*

3.6 Users, Roles, and Privileges

Before starting, you or another e*Gate user must add all the new users who will use your e*Gate system during its setup operations and later (if desired). A user with Administrator-level privileges can add new users at any time, using the **Security** folder in the Enterprise Manager window.

Note: *The Security folder only appears in the Navigator pane when users have Administrator-level privileges in the system.*

The default user name is:

Administrator

The default password is:

STC

The Administrator user has all possible user privileges in e*Gate. For a list of the number and types of characters you can use in user names and passwords, see “[Naming Conventions](#)” on page 46. The system considers the user name as a component.

Important: *For security purposes the system administrator can assign different levels of roles and privileges in e*Gate. See the **e*Gate Integrator System Administration and Operations Guide** for details.*

Security and access in e*Gate is role-based. That is, you assign the desired privileges to roles (for example Administrator, Operator, or Monitor) then assign the desired roles to users. You can also assign roles and privileges to modules, for example, if you only want one or more specific users to do limited operations with different modules.

For detailed information on e*Gate’s security features, see the following documents:

- *See*[Beyond eBusiness Integration Suite Deployment Guide](#) — For details on reasons for assigning various types of users, roles, and privileges, along with a sample scenario.
- *e*Gate Integrator System Administration and Operations Guide* — For an explanation of how to use the Enterprise Manager to set up and configure users, roles, and privileges in the e*Gate system.

3.7 Using the Network View

This chapter explains how to use the e*Gate Enterprise Manager's Network View feature to view a graphic representation of your schema.

3.7.1 Introduction: Network View

The Network View provides a graphical representation of the entire schema. This feature allows you to check the e*Gate configuration using a graphic system representation, which can be beneficial when troubleshooting problems that may occur in your e*Gate system.

3.7.2 Using Network View

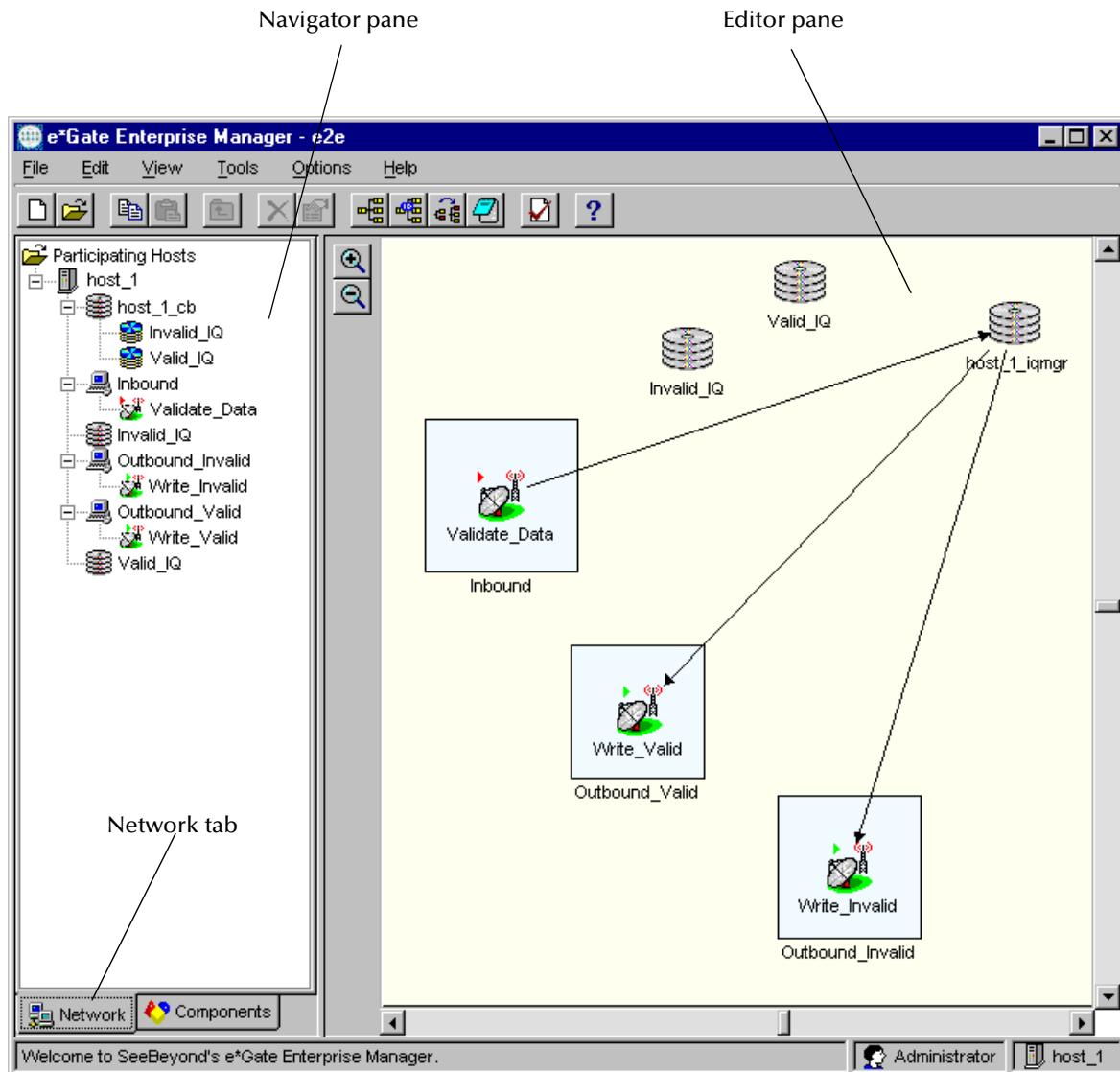
The Navigator pane of the Network View contains a **Participating Hosts** folder and icons for all of the components in your schema (see [Figure 14 on page 71](#)). The Editor pane graphically displays all of the elements and components in the schema in a top-down hierarchical relationship. Event Types are represented as arrows between the component that publishes the Event Type and the component that subscribes to it.

Use the Network View to track how your system components publish and subscribe to other components within the system. When troubleshooting, this graphical view of your system's routing can aid you in locating directional problems within the schema. If a discrepancy is discovered, you can open the offending component's properties dialog box directly from the component (see ["To modify component properties in the Network View" on page 72](#)).

To display the Network View

- 1 Select the **Network** tab at the bottom region of the Navigator pane.

Figure 14 Network View with Graphic Representation in Editor Pane



Network View Palette Controls

Table 7 Network View Palette Controls

Button	Name	Function
	Zoom in	Zooms into your e*Gate system in the Editor pane
	Zoom out	Zooms out of your e*Gate system in the Editor pane

Note: For details about the toolbar buttons and menu commands see [Table 3 on page 52](#) and [Table 4 on page 54](#). Menu commands are grayed out when they are not available to use.

To expand or collapse the view

- 1 Select the Navigator's **Network** tab.
- 2 In the Editor pane, select the component you want to expand or collapse.
 - ♦ To expand a component (display child components), double-click the component.
 - ♦ To collapse a component (display the parent component), double-click the colored area surrounding the component.

To move components to a different location on the screen

- 1 Select the **Network** tab in the Navigator pane.
- 2 In the Editor pane, select the component(s) that you want to move.
- 3 Drag the component(s) to their new location.

Note: *To select more than one component, hold down the **Ctrl** key while you click on each component.*

*To reset the layout to the default view, pull down the **View** menu and select **Reset Layout**.*

To view Event Type names

- 1 Select the **Network** tab in the Navigator pane.
- 2 In the Editor pane, expand the view to see Event Type arrows.
- 3 Move the mouse pointer over an Event Type arrow. The name will appear in a Tooltip. If there are multiple Event Types in opposite directions, the names of the subscribing and publishing components will also appear in the Tooltip.

Note: *If necessary, use the scroll bar to view all of the Event Types in a Tooltip.*

To modify component properties in the Network View

- 1 Double-click on the component in the Editor pane.
- 2 When the component's properties dialog box opens, edit the properties.
- 3 When finished editing the properties, click **OK** to save them and exit the dialog.

Important: *You can modify component properties in the Network View, but you cannot add or delete components.*

3.8 Online Help Systems

The e*Gate Help feature has the following characteristics:

- The online Help contains information about all aspects of using the e*Gate system. It explains topics such as procedures, terminology, and basic concepts of how to use the e*Gate software, including the Enterprise Manager.
- Once you display the Help window, you can move around easily in the current online Help system to get more information at the click of a mouse button.
- The online Help also explains how to work with basic e*Gate system components and other elements. Each Help system includes its own window, table of contents, index, and full-text searching capability. These features aid users in locating information quickly and easily.
- The e*Gate software contains several GUIs in addition to the Enterprise Manager, for example, the Collaboration Rules Editor and the e*Gate Monitor. Each of these GUIs has its own online Help system. Each operates in the same way.

Note: *The e*Gate online Help systems use the Microsoft Internet Explorer for operation. You must have this application installed and available to e*Gate to run the Help feature correctly.*

This section explains:

- [“Using Online Help” on page 73](#)
- [“Help Window” on page 75](#)
- [“Online Help Features” on page 77](#)

3.8.1 Using Online Help

The e*Gate online Help systems offer the following basic features:

- **Instructions for Procedures** which helps you find detailed instructions for e*Gate system procedures in each online Help topic.
- **e*Gate Glossary of Terms** that you can access at any time when using this Help to look up definitions of technical or e*Gate-related terms.
- **Context-sensitive Help** which allows you to press F1 from the active interface to get instant help if you need help with any window, dialog box, or other GUI feature. The Help window appears automatically, displaying the appropriate Help topic.
- **What's This? Help** shows pop-up labels. When you move the mouse pointer over a GUI feature (for example, a button), a yellow text label appears, giving a brief description. When you move the mouse pointer away, the label disappears.

Hypertext Links

The online Help systems use HTML formats. You often see a word or several words in color, for example,

Enterprise Manager

These words represent a hypertext link. Move the mouse pointer over them, and it turns into a pointing finger. Clicking the words instantly jumps the text to the topic associated with that link. Use hypertext links to look up additional information, for example, Glossary terms or cross-references.

Accessing Online Help

To open the Help window

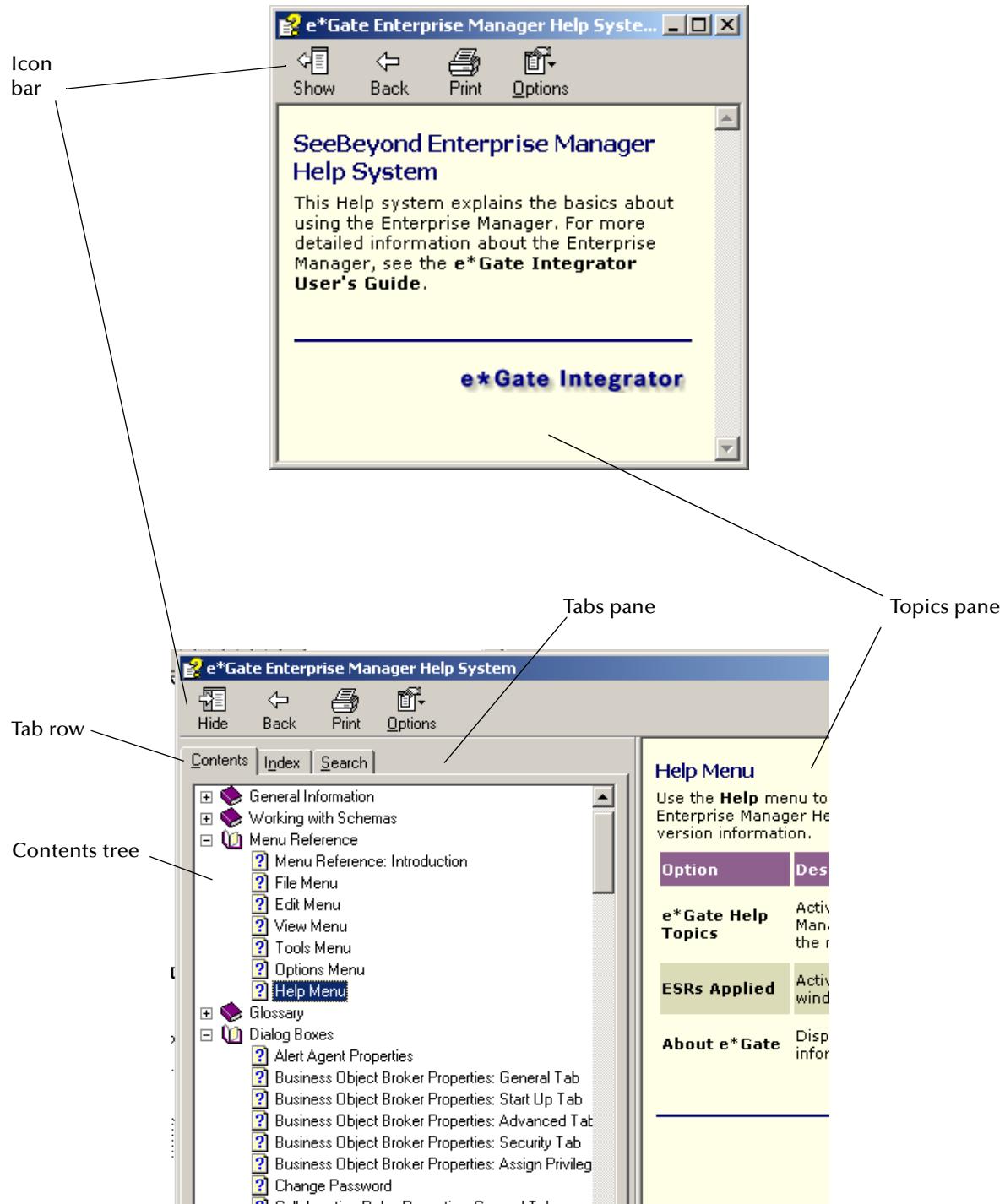
From the current GUI, for example the Enterprise Manager window, use any of the following actions to display its Help window:

- Open the **Help** menu and choose the **e*Gate Help Topics** option.
- Click  (if displayed in the GUI).
- Press **F1** for instant help with the active (current) GUI.

3.8.2 Help Window

Figure 15 shows the e*Gate Help window that first opens when you access the online Help system. It displays both examples of the Enterprise Manager's Help window; with Tabs pane hidden and with the Tabs pane showing.

Figure 15 Enterprise Manager Help Window

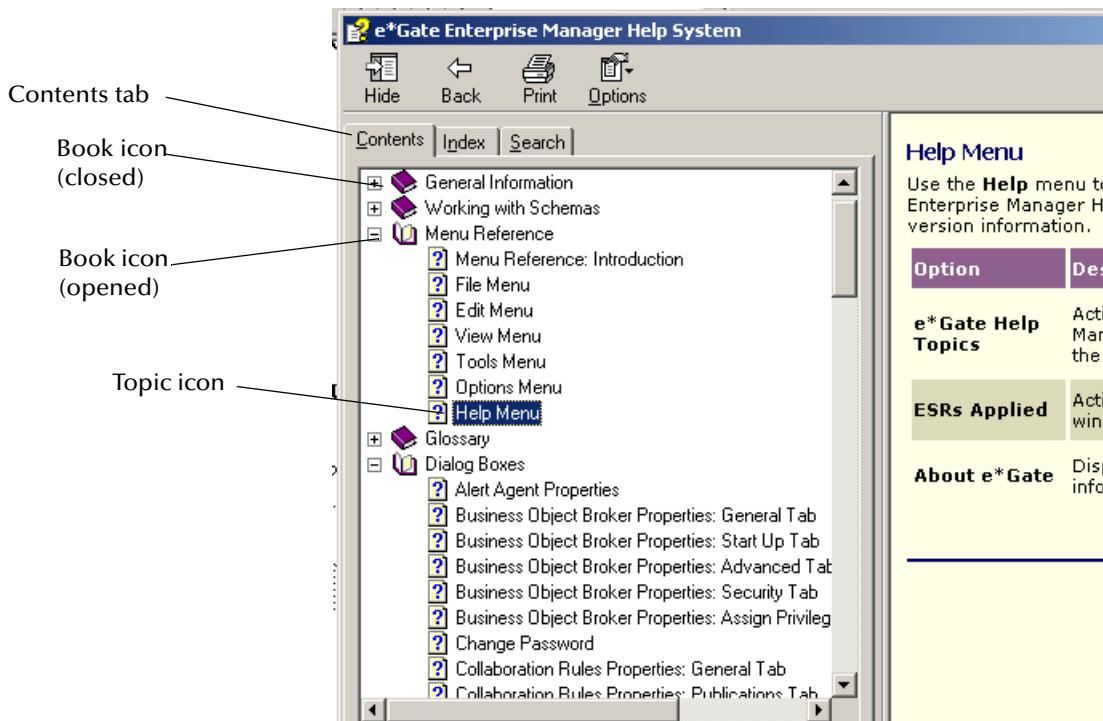


GUI Features

The Help window contains the following components:

- **Tabs** allow you to use the online Help's organization features.
- **Books** hold the chapters in the current online Help system. These books contain topics organized into a step-by-step browse sequence.
- **Topics** display the current Help topic and allow you to browse topics.
- **Contents Tree**, displayed within the Tabs pane (see Figure 16), shows book and topic icons in a graphical form similar to Windows Explorer.

Figure 16 Contents Tree



To use the Contents Tree

- To open a book, double-click a book icon. The book's topics appear as topic icons under the book icon.
- To display a topic, click a topic icon. The topic appears in the Topics pane.

On Entry and Exit

The online Help system shown in [Figure 15 on page 75](#) defaults to the "Enterprise Manager" topic. When you first open a Help window for a system, only the Topics pane displays. Afterward, when you open the same Help window, the panes appear as they were left just before the last exit.

Note: This section uses the online Help for the Enterprise Manager as an example of how to use all the e*Gate online Help systems.

3.8.3 Online Help Features

To best utilize the e*Gate online Help, learn to use the following features:

- **Tab Operation:** Tabs in the Tabs pane allow you to use the Help Index to search for key words, to bookmark topics, and to display (or redisplay) the Contents Tree.
- **Toolbar:** You can access additional online Help features by selecting one of the following buttons at the top of the Help window: **Show/Hide, Back, Forward, Home, or Print**. See “[Toolbar Buttons](#)” on page 78 for details.
- **Printing Help:** You can print hard copies of any or all the online Help topics. See “[Printing Help](#)” on page 78 for details.

The rest of this section explains these features in detail.

Tab Operation

The Tabs pane displays tabs that access search and contents features. Only the Topics pane appears when you first use an online Help system.

To show the Tabs pane

- Click the  (Show) button.

The Search and Contents tabs are displayed, and the  (Show) button changes to the  (Hide) button.

Note: After you hide the Tabs pane, you can maximize the remaining pane. Select the **Maximize** icon to do so. You must move the Windows task bar first, to be sure the pane takes up the entire screen. If so, once the Help window expands, you can move the task bar back to its original position.

Using Tab Features — Click the appropriate tab to use the desired feature. Table 8 below lists these tabs and their functions.

Table 8 Tabs Pane Operation

Name	Function
Contents	Allows you to display the Contents Tree; if you have displayed the Search, Index , or Favorites dialog boxes, you can redisplay the Contents Tree by clicking this tab. See “ To use the Contents tab ” on page 78 for details.
Index	Allows you to search for an online Help Index entry. See “ To use the Index tab ” on page 78 for details.
Search	Allows you to search by key word, that is, search for a desired word in the topic text. See “ To use the Search tab ” on page 78 for details.

When the Tabs pane first appears, it displays the Contents tree. Afterward, when you open the Help window, it appears as it did before the last exit. For more information on Tabs pane features, see the appropriate Microsoft Windows user’s guide.

To use the Contents tab

When the Contents tab is displayed, you can take any of the following actions:

- **Open or close a book** by double-clicking a book icon.
- **Display a topic** by clicking a topic icon.
- **Open all books** by right-clicking in the Contents tree and choosing **Open All** from the shortcut menu. This action opens all books and lists all the topics they contain.
- **Close all books** by right-clicking in the Contents tree and choosing **Close All** from the shortcut menu. This action closes all books.

To use the Index tab

- 1 Type the desired key word to look up in the Index.
- 2 Click **Display**. A list of topics related to the index entry appear.
- 3 In the **Topics Found** dialog box, double-click the topic you want to display.

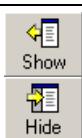
To use the Search tab

- 1 Type the word in which you want to search.
- 2 Click **List Topics**. All topics containing the searched-for word are listed.
- 3 In the **Select Topic to display** pane, double-click the topic you want to display. The selected topic displays; if **Search Highlight** is On, the found words are highlighted.

Toolbar Buttons

Table 9 lists toolbar buttons in the Help window along with their names and functions.

Table 9 Help Window Buttons

Button	Name	Function
	Show Tabs Hide Tabs	The Show Tabs and Hide Tabs buttons act as a toggle. To display the Tabs pane, click the Show Tabs button. To close the Tabs pane, click the Hide Tabs button.
	Back	The Back button takes you to the previous topic in the online Help.
	Print	The Print button allows you to print online Help topics.
	Options	The Options button provides access to the Show Tabs, Hide Tabs, Back, and Print functions noted above, as well as Forward, Home, Refresh, and Internet Options functions. The Search Highlight On/Off switch allows you to control the behavior of the Search tab — that is, whether the topic will display the searched-for word or phrase with highlighting.

Printing Help

If you want hard-copy versions of any online Help topics, you can print all or any part of a book as explained in this section.

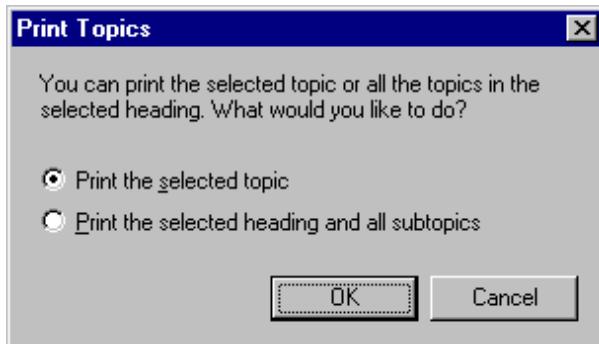
Note: See the appropriate Windows user guide and/or printer manual for details on how to configure your printer and how to use the Windows Print dialog box.

To print a single topic

- 1 With the Contents Tree displayed, place the mouse pointer on the desired Topic icon.
- 2 Click the right mouse button to display a shortcut menu.
- 3 Choose Print.

The following Print Topics dialog box (Figure 17 below) appears:

Figure 17 Print Topics Dialog Box



- 4 Click Print the selected topic, then click OK (see Figure 17 above).

The Windows Print dialog box appears.

- 5 Select the desired Windows print options and click OK to print the selected topic.

To print an entire book

- 1 With the Contents Tree displayed, place the mouse pointer on the desired Book icon.
 - 2 Click the right mouse button to display a shortcut menu.
 - 3 Click Print.
- The Print Topics dialog box appears. See Figure 17.
- 4 Click Print the selected heading and all subtopics, then click OK.
- The Windows Print dialog box appears.
- 5 Select the desired Windows print options and click OK to print all topics in the selected book.

Note: The easiest way to print an entire online Help system is by printing every book in the Contents Tree, one at a time.

Also Print Help Topics

Click  in the toolbar or choose **Print** from the shortcut menu in the Tabs/Contents pane to print Help topics. Either action prints the selected book or topic as explained in the previous procedures.

Closing the Help Window

To exit Help

Click the Windows **Help** button in the upper left corner of the window and then select **Close** from the resulting shortcut menu. Performing this action exits the current online Help system.

Setting Up e*Gate

This chapter explains how to use the e*Gate Enterprise Manager features to create and configure the basic components of a working system.

4.1 Overview of e*Gate Setup

Setting up e*Gate requires a step-by-step approach to system design, architecture, and planning for component interaction. Once you have created a basic requirements checklist, system design, and deployment plan, you must build on this foundation by developing your complete e*Gate system. Then use the e*Gate graphical user interfaces (GUIs) to create and set up all the needed system components.

The entire process of creating and configuring a working e*Gate system is called *system development*. Since this guide primarily explains how to use the Enterprise Manager to set up e*Gate system components, we are calling this process *system setup*. For details on e*Gate system design and development, see the *SeeBeyond eBusiness Integration Suite Deployment Guide*.

You must separately create every named component in the e*Gate system. Then, you must configure most of these components before the system can use them correctly. Configuring e*Gate components use the same GUIs as creation. As you set up each component, you can refer to this chapter for an explanation of appropriate creation and configuration procedures. The text also refers you to later chapters and/or additional documents for more information, where necessary.

4.1.1 e*Gate GUIs

In addition to the Enterprise Manager, e*Gate uses the following GUIs (associated with the Enterprise Manager) for setting up, configuring, and editing system components:

- Java Event Type Definition (ETD) Editor; see [Chapter 5](#).
- Monk ETD Editor; see [Chapter 6](#).
- Java Collaboration Rules Editor; see [Chapter 7](#).
- Monk Collaboration Rules Editor; see [Chapter 8](#).
- Collaboration-ID Rules Editor (for backwards-compatibility with e*Gate Version 3.6 only; see the e*Gate online Help)
- e*Way Editor; see [Chapter 9](#).

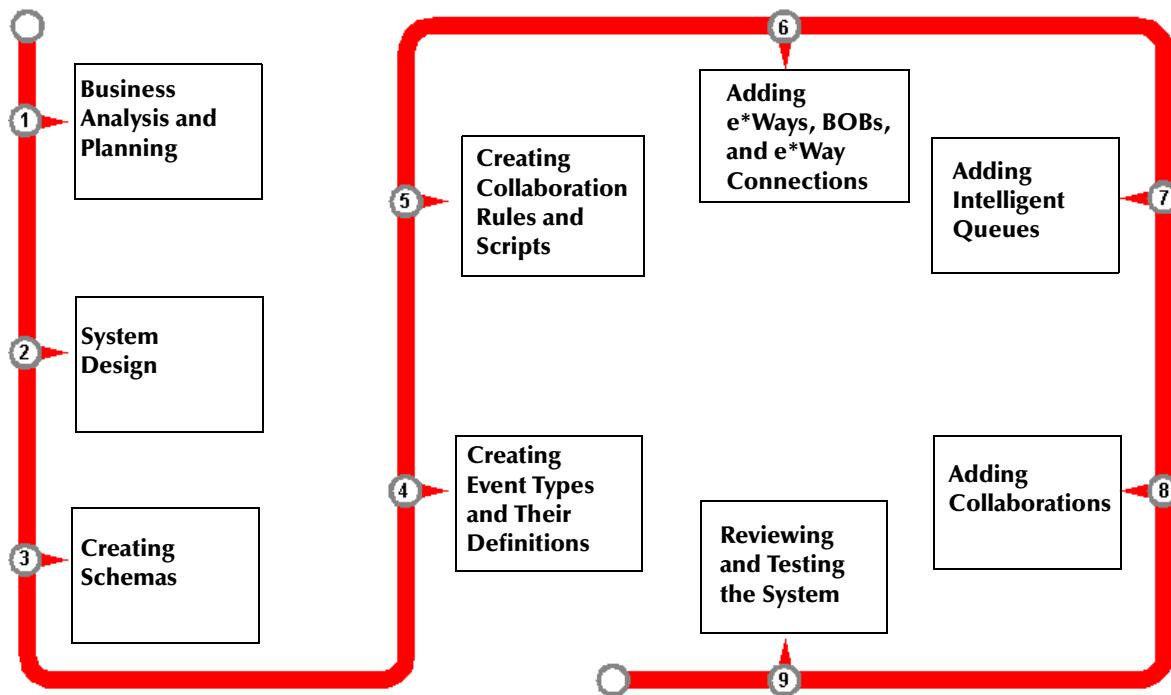
Use the Enterprise Manager to access these GUIs. Each of them has its own window and pane features as well as an online Help system. Also, all the GUIs, including the Enterprise Manager, have properties dialog boxes that aid in the e*Gate component creation, configuration, and editing operations.

This chapter describes these GUI editors, referring you to later chapters and/or other documents for a complete explanation of how to use their specific features.

4.1.2 Setup Steps

Figure 18 shows basic e*Gate setup steps in the form of a road map.

Figure 18 e*Gate Setup Road Map



“Business Analysis and Planning” on page 43 describes the business analysis step shown in Figure 18. In addition, setting up a complete e*Gate system requires all or most of the following general steps:

- [“System Design Components” on page 83](#)
- [“Creating a Schema” on page 86](#)
- [“Creating Event Types and ETDs” on page 95](#)
- [“Creating Collaboration Rules and Scripts” on page 106](#)
- [“Adding e*Ways and BOBs” on page 123](#)
- [“Adding Intelligent Queues” on page 136](#)
- [“Adding Collaborations” on page 142](#)
- [“Reviewing and Testing the System” on page 147](#)

The rest of this chapter explains the basic setup steps, in the order you would use them in setting up a typical e*Gate system. The chapter includes a final section that describes some basic setup review and testing procedures.

4.2 System Design Components

The main work of e*Gate is to route predefined Events (data packets) into and out of external systems, using specialized e*Way Intelligent Adapters. e*Ways are the primary communication links in the e*Gate system.

Since e*Gate is modular and expandable, it can bring together a great number of different external systems, using as many e*Ways as necessary. In setting up e*Gate, you must use its basic modular components to create your own system. The components of an e*Gate system design interact as parts of basic relationships whose details you must configure.

Chapter 2 explained the e*Gate system's basic architecture. For details on e*Gate system design, see the *SeeBeyond eBusiness Integration Suite Deployment Guide*.

This section explains:

- “[Component Data Flow Relationships](#)” on page 83
- “[Component Logical Relationships](#)” on page 85
- “[Data Management Relationships](#)” on page 86

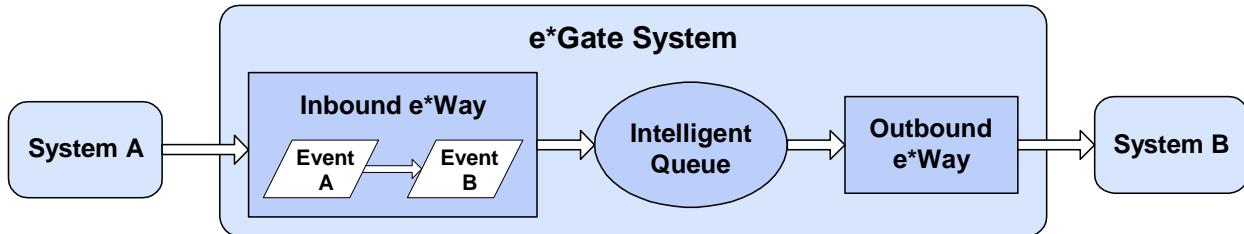
4.2.1 Component Data Flow Relationships

The following e*Gate components control data flow:

- **e*Ways** bring data from outside to inside e*Gate, pass data from inside to outside e*Gate, or both. When connected to e*Way Connections, they can also bring multiple components of data from outside to inside e*Gate at the same time, pass multiple components of data from inside to outside e*Gate, or both at the same time.
- **Business Object Brokers (BOBs)** act as internal e*Ways when connected to IQs. (BOBs are optional).
- **Intelligent Queues (IQs)** store data for later use in the e*Gate system.
- **e*Way Connections** are the encoding of the access information for one particular gateway to an external system (such as a database) or a SeeBeyond JMS IQ Manager. Certain e*Way Connections offer the ability to use XA-compliant calls to external and internal systems that support them (such as Oracle, MQSeries, and SeeBeyond JMS).

In e*Gate, e*Ways have the main responsibility for data transport. Figure 19 shows a simplified system setup from the viewpoint of data flow.

Figure 19 Basic e*Gate Data Flow Relationships



In the sample system shown in Figure 19, e*Gate system data flow takes place in the following basic steps:

- Events flow into e*Gate from an external system (System A) through an inbound e*Way.
- This inbound e*Way (via a Collaboration) transforms Events from Event Type A to Event Type B and places them in an IQ for temporary storage.
- This outbound e*Way takes the Events from the IQ and sends them out of e*Gate to another external system (System B), without changing them.

Note: An e*Way with an e*Way Connection functions the same way as the e*Ways described above, except that it can transfer multiple components of data in both directions at the same time, to and from multiple IQs. Also, any Java-enabled Collaboration can use several e*Way Connection components simultaneously, to transfer data to and from multiple external systems.

See the next section for more information on Collaborations.

4.2.2 Component Logical Relationships

Figure 20 shows basic e*Gate components from the viewpoint of logical relationships.

Figure 20 Basic e*Gate Logical Relationships

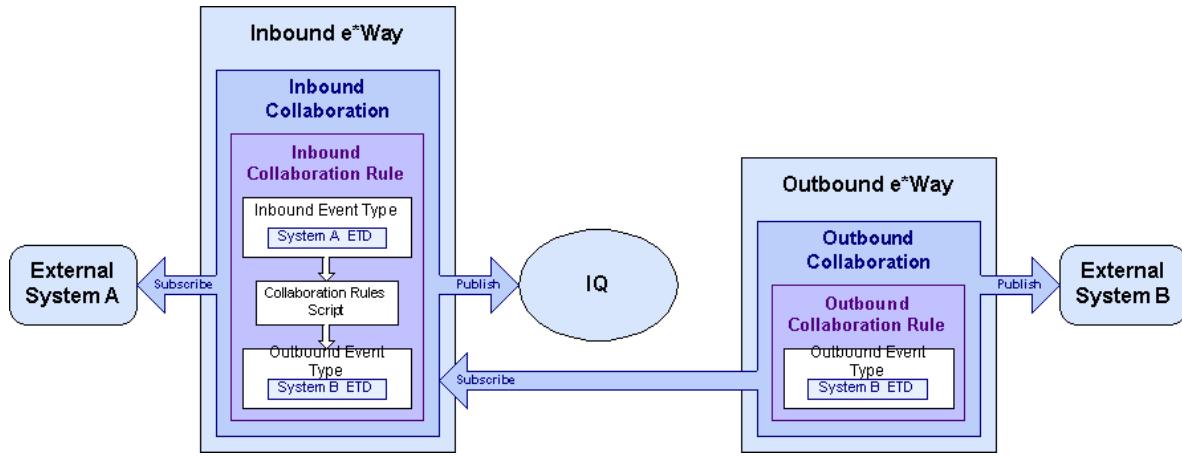


Figure 20 is the same system setup shown in [Figure 19 on page 84](#). The primary components shown in this figure are

- **Event Type Definitions (ETDs)** are defined data structures for a particular type of Event, all of which have specified characteristics in common. They are written in Java or in SeeBeyond's Monk programming language and stored as text files.
- **Collaborations and Collaboration Rules** supply the publish-and-subscribe logic that routes Events through the e*Gate system.
- **Collaboration Scripts** define what Collaborations do with Events, for example, transformation or verification. If necessary, Collaboration scripts (including Collaboration Rules scripts) implement the business logic used to process Events as they move through the e*Gate system.

Logical Routing

Each e*Way contains at least one Collaboration that directs its action. The kind of action depends on the e*Way's Collaboration setup. The Collaborations in [Figure 20 on page 85](#) have the following logical relationships:

- **Publishing** is when an inbound Collaboration publishes to the IQ.
- **Subscribing** is when an outbound Collaboration subscribes to the inbound Collaboration.

The publish and subscribe information in Collaborations establishes the logical routing of e*Gate Events as follows:

- Collaborations can publish or subscribe to external systems.
- Collaborations can also subscribe to other Collaborations and publish to IQs.

Note: Logically speaking, Collaborations never subscribe to IQs.

See “[Collaboration Rules Scripts](#)” on page 107 for more information on e*Gate publish/subscribe (also called pub/sub) logic.

4.2.3 Data Management Relationships

The e*Gate system has the following internal data-management features:

- The **Registry** is the directory store for all e*Gate configuration details.
- The **Registry Service** handles all requests for updates to the Registry and forwards updated files to clients as necessary.
- The **Control Broker** is an e*Gate-generated component that starts and monitors e*Ways and BOBs. A Control Broker must be running on each Participating Host within a schema. There is only one Control Broker per host.
- The **IQ Manager** is an e*Gate component that reorganizes IQs, archives queue information upon request to save disk space, and locks the queues during maintenance.

For an overview of these features, see [“Control Layer” on page 35](#). This chapter describes each of them under the appropriate section as it explains the overall e*Gate setup operation.

- ♦ For more information on data-management relationships and how they operate in e*Gate, see the *e*Gate Integrator System Administration and Operations Guide*.

4.3 Creating a Schema

A schema is a namespace that defines e*Gate system parameters and the relationships between components within the e*Gate system. Schemas can span multiple hosts.

Because all setup and configuration operations take place within an e*Gate schema, you must create a new schema or start an existing one before using the system. A schema is an organization scheme that contains essential system modules and configuration parameters. Schemas store all their configuration information in the e*Gate Registry.

You start and log into the e*Gate Enterprise Manager and then use it to create a new schema on the desired Registry Host.

To create a schema

- 1 Start Enterprise Manager and, on the appropriate Registry Host, log in as Administrator (or another user with equivalent privileges).
- 2 When the **Open Schema on Registry Host: <host name>** dialog box appears, click **New**.

Note: You cannot delete or rename a schema once you have created it.

- 3 Type the desired name in the **New Schema** text box and click **Open**.

The system opens your new schema.

Activating the Host in a New Schema: When you add a new schema, the host attached to its Control Broker is inactive at first. Inactive means this host has not been registered as a Participating Host in the e*Gate Registry.

Note: *The Network host name for an activated Participating Host must match the name of the machine where its Control Broker will be run.*

4.3.1 Control Broker Setup

This section explains how to set up and configure the Control Broker for a schema.

Configuring Control Broker Properties

For the most part, the Control Broker is preconfigured when you create a new schema. To change or activate Control Broker features, open the **Control Broker Properties** dialog box.

This section explains the following tabs:

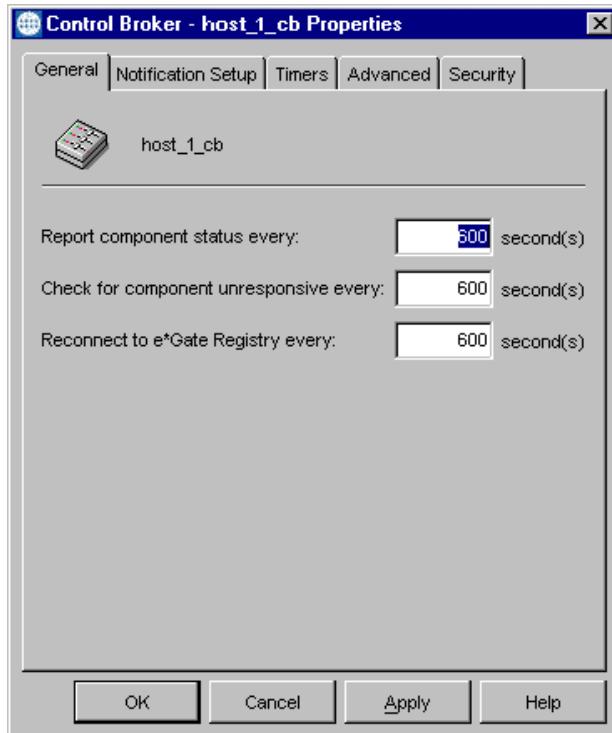
- “[General Tab](#)” on page 88
- “[Notification Setup Tab](#)” on page 88
- “[Timers Tab](#)” on page 90
- “[Advanced Tab](#)” on page 92
- “[Security Tab](#)” on page 93

For more information on Control Brokers and how to operate them, see the *e*Gate Integrator System Administration and Operations Guide*.

General Tab

The **General** tab allows you to reset the times when the Control Broker performs certain tasks. See Figure 21.

Figure 21 Control Broker Properties Dialog Box: General Tab



Use the following text boxes to reset the times that trigger the Control Broker:

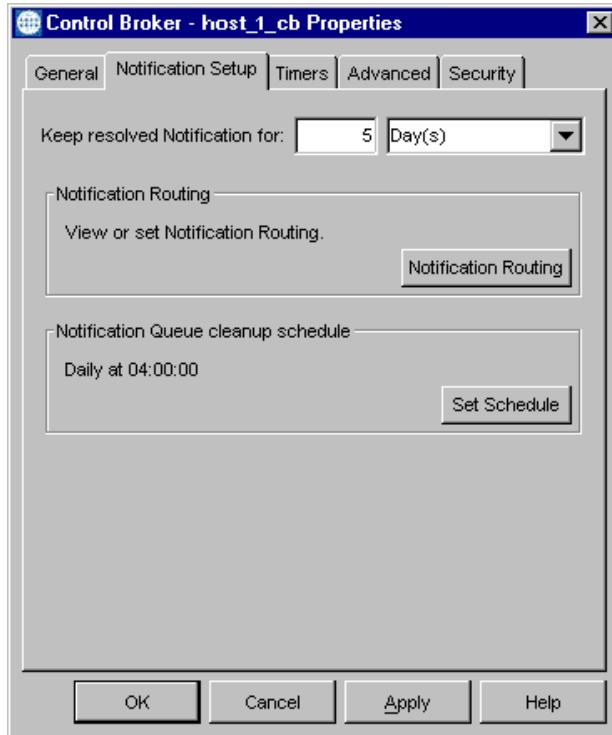
- In the **Report component status every** box, enter the amount of time after which the Control Broker will report the status of the modules it supervises. The default is 600 sec (10 min).
- In the **Check for component unresponsive every** box, enter the amount of time after which the Control Broker will check for unresponsive modules. The default is 600 sec (10 min).
- In the **Reconnect to e*Gate Registry every** box, enter the amount of time after which the Control Broker will reconnect with the registry. The default is 600 sec (10 min).

Notification Setup Tab

The Control Broker's **Notification Setup** tab determines how long to keep resolved notifications, and how frequently to run the cleanup process that deletes expired

notifications. This tab allows you to reset the parameters for retaining notifications and notification routing. See Figure 22.

Figure 22 Control Broker Properties Dialog Box: Notification Setup Tab



To use the Notification Setup Tab features

- In the **Keep resolved Notification for** box, enter the amount of time to keep notifications that have been marked as resolved. (Unresolved notifications are kept indefinitely.) The default is 5 days.

Control Brokers use notifications to track a wide range of Events, ranging from simple status messages to problems that require operator intervention.

Notifications are marked “resolved” when whatever issue caused the notification has been addressed.

Some notifications are marked “resolved” automatically (for example, a notification that arises when a system is unreachable is automatically marked as resolved when communications are re-established). Other notifications must be manually resolved, using the e*Gate Monitor.

Unresolved notifications are stored indefinitely, but resolved notifications can be deleted to save disk space.

- Click **Notification Routing** to edit or view the way notifications are routed. See the *e*Gate Integrator Alert and Log File Reference Guide* and [“Notification Channels” on page 499](#) for more information about notification routing.
- Click **Set Schedule** to determine how frequently the queue of resolved notifications should be cleaned up. The default is daily at 4:00 A.M.

Queue cleanup: There are two factors that determine how the Resolved Notification IQ is cleaned up:

- How long notifications are kept (in other words, when do notifications “expire”).
- How frequently expired notifications are deleted.

The default is to keep resolved notifications for five days, and to run the cleanup daily at 4:00 A.M. Under this plan, the Control Broker begins at 4:00 A.M. to check the Resolved Notification IQ for notifications that were resolved before 4:00 A.M. five days ago, and deletes entries as appropriate.

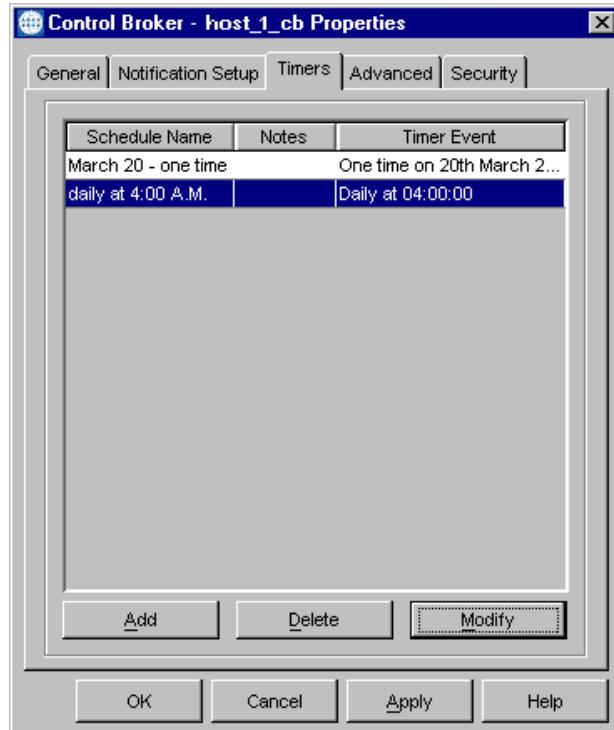
Select an expiration period and cleanup frequency appropriate to the number of resolved notifications you wish to store, and the amount of disk space you need to maintain. For example, to keep an entire week’s worth of notifications, you can set the expiration period to seven days.

Note: Be aware that a cleanup schedule of “every Sunday at 4:00 A.M.” will keep more records online for a longer period than will a schedule of “every day at 4:00 A.M.”

Timers Tab

The **Timers** tab opens the **Timer Events Properties** dialog box, where you configure a Timer Event. Timer Events trigger a signal to the Control Broker to perform certain tasks at predetermined times. You must manually set this function. See Figure 23.

Figure 23 Control Broker Properties Dialog Box: Timers Tab



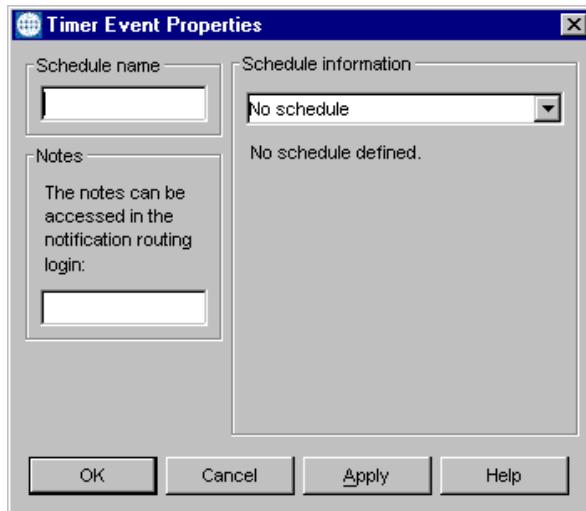
This tab enables you to instruct the Control Broker to send “timer” monitoring Events.

To add a timer

- 1 Click **Add**.

The **Timer Event Properties** dialog box opens. See Figure 24.

Figure 24 Timer Event Properties Dialog Box



- 2 Enter the **Schedule name**, any **Notes**, which are optional, and select a schedule from the **Schedule information** drop-down list.
- 3 Click **OK** to close the **Timer Event Properties** dialog box. The timer schedule you created appears in the **Timer Events Properties** dialog box. Click **OK** to close the **Control Broker Properties** dialog box.

To modify a timer

- 1 Select a timer from the list in **Schedule Name** column on the **Control Broker Properties** dialog box.
- 2 Click **Modify**.
- 3 The **Timer Event Properties** dialog box opens. Modify the schedule name, notes, or schedule information.
- 4 Click **OK** to close the **Timer Event Properties** dialog box, then click **OK** to close the **Control Broker Properties** dialog box.

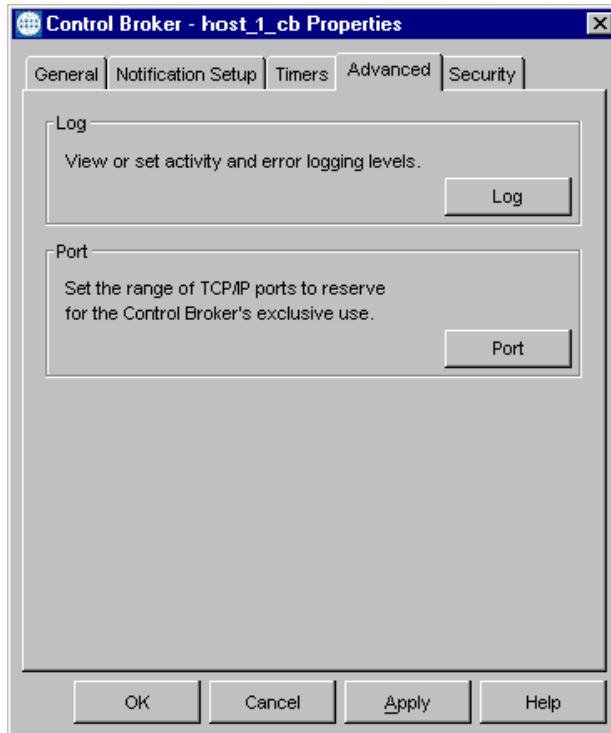
To delete a timer

- 1 Select a timer from the list in **Schedule Name** column on the **Control Broker Properties** dialog box.
- 2 Click **Delete**. The timer schedule is deleted without requesting further confirmation.
- 3 Click **OK** to close the **Control Broker Properties** dialog box.

Advanced Tab

The **Advanced** tab allows you to set the logging level and debug flags of log files, as well as editing the port properties. See Figure 25.

Figure 25 Control Broker Properties Dialog Box: Advanced Tab



To use the Advanced Tab features

- Click **Log** to view or change the logging level for activity or error logs. Log files are text files that contain a record of all actions taken by an e*Way. Use these files to troubleshoot problems in your system. For information on log files, see the *e*Gate Integrator Alert and Log File Reference Guide*; for detailed information on how to use this dialog, see "Activating Logging" in the same manual.

Important: Use logging options freely while developing and debugging schemas, but decrease the level of detail before you migrate the schema to a production environment:
Certain logging options and severity settings cause significant slowing of component performance and overall system throughput.

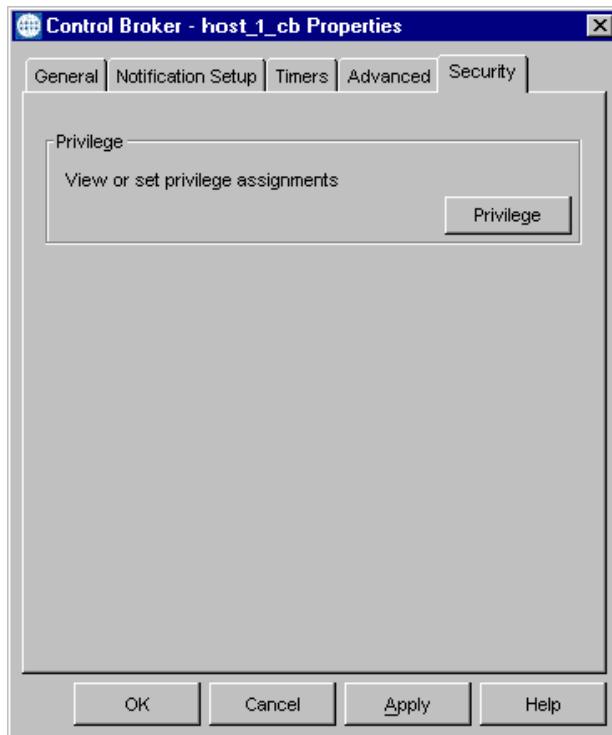
- Click **Port** to view or change the TCP/IP ports used by this Control Broker. This opens the **Port Properties** dialog box, which allows you to select the range of TCP/IP addresses that the Control Broker can use when searching for a port to bind for its exclusive use.

Change these values only if you have reserved the entire default range of ports (5000-5500) for another purpose, or if you otherwise require the Control Broker to use a different port range.

Security Tab

The **Security** tab allows you to assign, change, or delete component privilege levels. See Figure 26.

Figure 26 Control Broker Properties Dialog Box: Security Tab

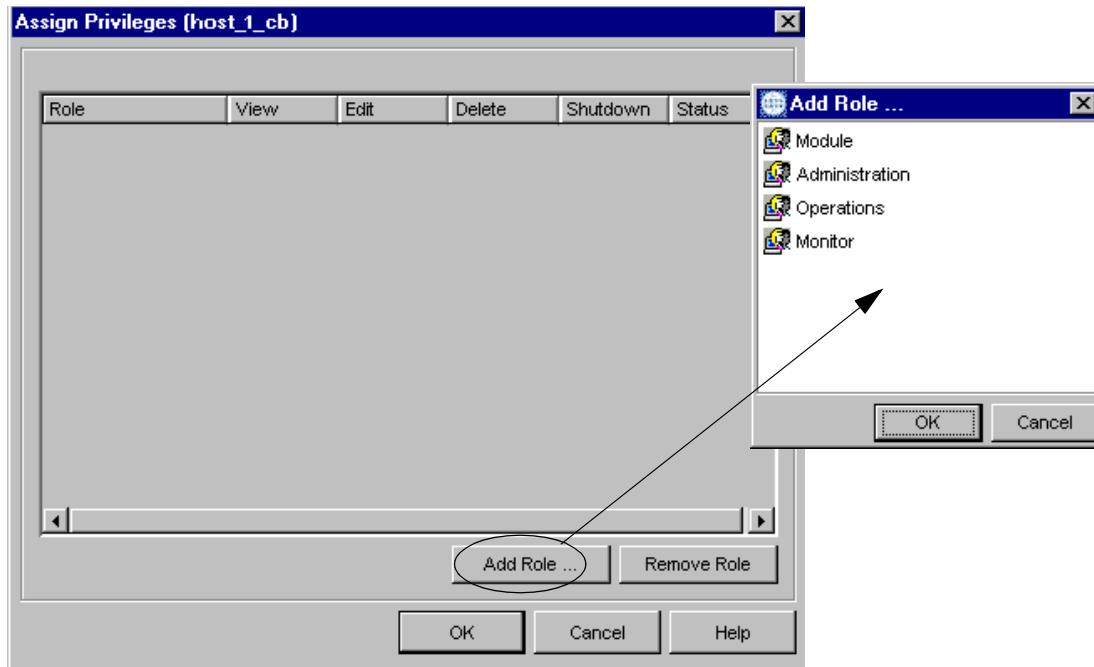


To use the Security tab features

- To assign, change, or delete component privileges to this Control Broker, click the **Privilege** button. The **Assign Privileges** dialog box appears (see [Figure 27 on page 94](#)), listing the Roles that have component privileges for this Control Broker assigned to them. The component privileges applicable to a Control Broker are **View, Edit, Delete, Shutdown, and Status**.

Note: You can only assign **Security** tab privileges if you are an Administrator user. For more information on this feature, see the *e*Gate Integrator System Administration and Operations Guide*.

Figure 27 Assign Privileges Dialog Box and Add Role Selection Box



To assign privileges to a role

- 1 From the **Assign Privileges** dialog box (see Figure 27), click **Add Role**.
- 2 When the **Add Role** dialog box appears, highlight a role whose privileges you intend to set, and click **OK**.
- 3 Your selection appears under the **Role** column on the **Assign Privileges** dialog box. Place a check in the appropriate check boxes (**View**, **Edit**, **Delete**, **Shutdown**, and **Status**) to specify the privileges associated with that role.
- 4 Repeat this procedure as necessary to add additional roles to the list.
- 5 Click **OK** when finished.

To change privileges for a role

- 1 From the **Assign Privileges** dialog box (see Figure 27), place checks in or remove checks from the appropriate check boxes (**View**, **Edit**, **Delete**, **Shutdown**, and **Status**) to re-specify the privileges associated with that role.
- 2 Repeat this procedure as necessary to change other roles in the list.
- 3 Click **OK** when finished.

To remove all privileges from a role

- 1 From the **Assign Privileges** dialog box (see Figure 27), highlight a **Role**.
- 2 Click **Remove Role**.
- 3 Repeat this procedure as necessary to remove other roles in the list.
- 4 Click **OK** when finished.

4.3.2 Host Activation

Before you can run a schema, you must first be sure the host has been activated.

To activate the host

- 1 Type the following text at the command line:

```
stcinstd -rh <host> -rs <schema> -un <username> -up <password> -ss -sa -v
```

- 2 Press ENTER.

Note: In the **stcinstd** command line as shown, the flag **-ss** is optional and means to run the host as a service. The flag **-sa** is also optional and means install service autostart. The optional **-v** flag gives you the “verbose” (extra information) display. Flags not shown (such as **-rp** for registry port) are assumed to use default values.

For more information on using the e*Gate command line, see “[e*Gate Interactive Monitoring](#)” on page 490. For a complete explanation, including the **stcinstd** command, see the *e*Gate Integrator System Administration and Operations Guide*.

4.4 Creating Event Types and ETDs

This section uses the following terminology:

- An **Event** (also called a **message**) is a packet of *data* processed by e*Gate.
- An **Event Type** (also called a **topic**) is a class of Events with common characteristics.
- An **Event Type Definition (ETD)** is a particular data structure that can be stored in a Java-enabled native ETD file (.xsc file) or a Monk ETD file (.ssc file).

Note: In e*Gate, only the first item in the previous list contains data.

ETD Overview

The e*Gate system packages data within Events and categorizes them into Event Types—in other words, classes of Events with a common data structure. What these Events have in common *defines* the Event Type and comprises the ETD. For example, this commonality could be a known number of fields with known characteristics and delimiters.

Important: You must create ETDs before e*Gate can process any data.

This section explains:

- “[Selecting the Event Type Definition Editor](#)” on page 96
- “[Creating Event Types](#)” on page 96
- “[Creating Java-enabled Event Type Definitions](#)” on page 97
- “[Creating Monk Event Type Definitions](#)” on page 103

- “[Assigning Definitions to Monk Event Types](#)” on page 105

4.4.1 Selecting the Event Type Definition Editor

e*Gate offers two ETD editors: The default (Java-enabled) ETD editor allows you to edit **.xsc** files; the Monk ETD editor edits Monk ETD files (**.ssc** files) only.

To set the ETD editor

- 1 In Enterprise Manager, on the **Options** menu, click **Default Editor**.
- 2 In the **Default Collaboration Language** dialog box, choose either **Java** or **Monk** and then click **OK**.

4.4.2 Creating Event Types

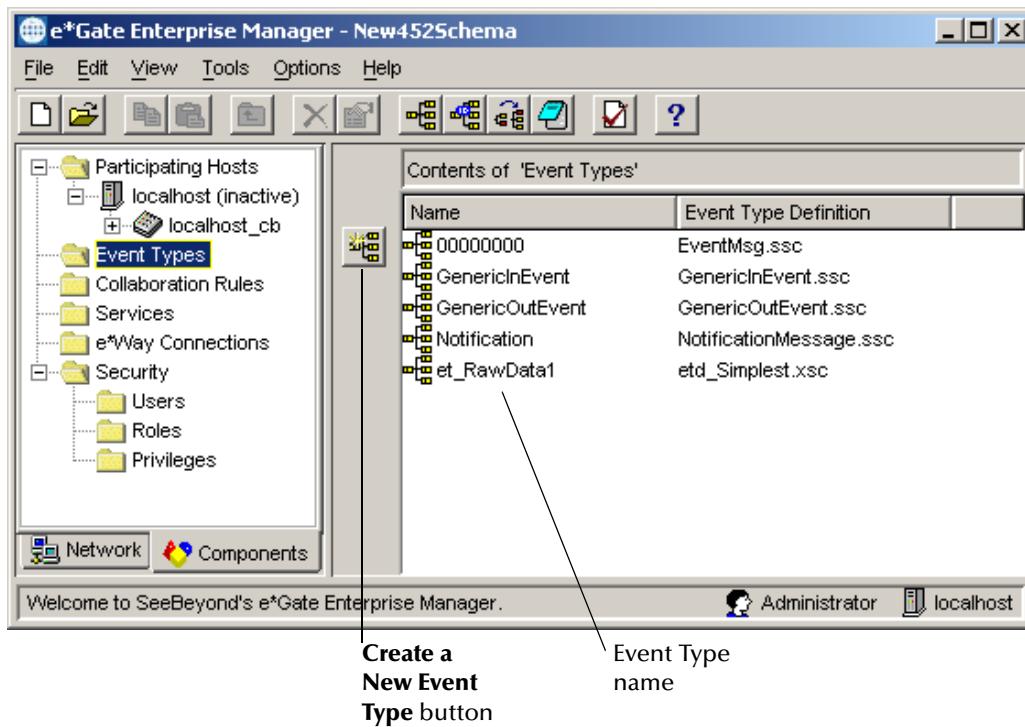
Generally, before you can add and configure ETDs, you must first create Event Types. (But Java Collaborations, for example, do not require ETDs to be tied to Event Types.)

To create Event Types

- 1 In the navigator pane, in the **Components** tab, click the **Event Types** folder.
- 2 On the palette or the **File** menu, click **New**  **.Event Type**.
- 3 In the **New Event Type Component** dialog box, enter a name for a new Event Type and click **Apply** to enter it into the system. Repeat as necessary.

As you create each new Event Type, the Editor pane displays a new Event Type icon with its name. See [Figure 28 on page 97](#).

Figure 28 Enterprise Manager With Event Types



- Click **OK** when you have entered the last Event Type.

You are now ready to create Java or Monk Event Type Definitions; see “[Creating Java-enabled Event Type Definitions](#)” on page 97 or “[Creating Monk Event Type Definitions](#)” on page 103.

4.4.3 Creating Java-enabled Event Type Definitions

You create ETDs using the Event Type Definition Editor. For detailed information about the Event Type Definition Editor, see [Chapter 5](#). e*Gate provides wizards that aid in the creation of your Java-enabled ETDs. These wizards include the following.

- Wizards for building a read-only Java-enabled ETD:
 - ◆ BAPI Wizard
 - ◆ COM/DCOM Wizard
 - ◆ DB Wizard
 - ◆ DTD Wizard
 - ◆ IDoc Wizard
 - ◆ Intranet Flist Wizard
 - ◆ Intranet Opcode Wizard
 - ◆ Jacada Wizard
 - ◆ JDE Wizard

- ◆ Oracle Financials Wizard
- ◆ SAG Wizard
- ◆ SEF Wizard
- ◆ SOAP Wizard
- ◆ XSD Wizard

Note: Depending on the products installed at your site, you may not see all the ETD Builder wizards listed above. The two wizards below, however, are always available.

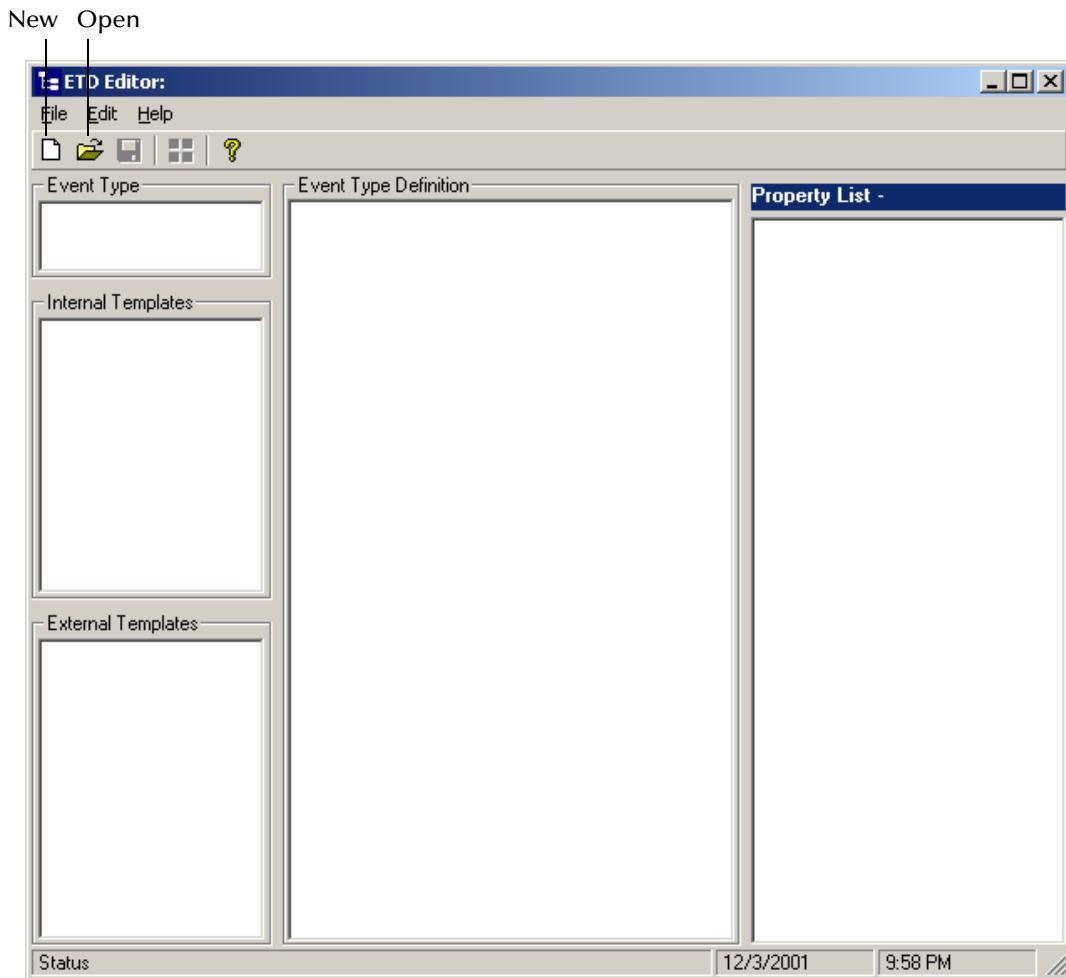
- Wizards for creating a standard read/write Java-enabled ETD:
 - ◆ Standard ETD Wizard—Allows you to create a standard ETD from scratch.
 - ◆ SSC Wizard—Converts a Monk ETD to a Java-enabled standard ETD.

The Java Event Type Definition Editor is divided into five panes:

- The **Event Types** pane lists the name of the ETD that is currently active.
- The **Internal Templates** pane lists the names of the internal templates known to the current ETD.
- The **External Templates** pane lists the names of the external templates known to the current ETD.
- The **Event Type Definition** pane displays the tree of the current ETD.
- The **Properties** pane displays the properties of the highlighted node, field, or method.

These five panes are illustrated in [Figure 29 on page 99](#).

Figure 29 Event Type Definition Editor



The Event Type Definition Editor uses wizards to create ETDs. When ETDs are created, each node is given a name that becomes its own identifier in the ETD tree. See ["About Node Names" on page 174](#) for a list of acceptable node name characters.

To create a Java-enabled ETD

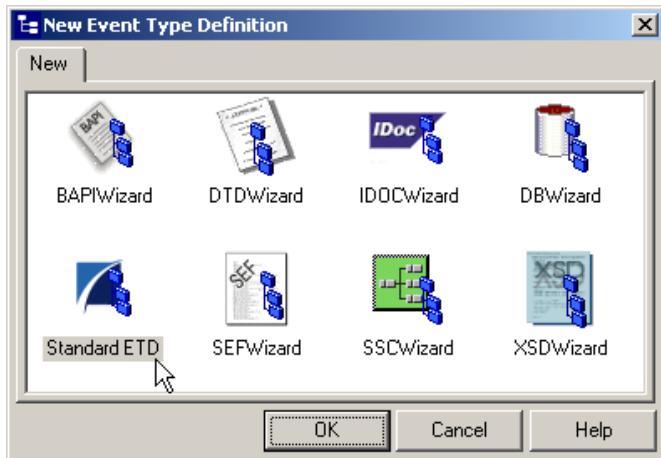
- 1 In Enterprise Manager, on the toolbar or **Tools** menu, click **ETD Editor**.

The Event Type Definition Editor appears, with all panes empty; see [Figure 29 on page 99](#).

- 2 In the ETD Editor, on the toolbar or **File** menu, click **New**.

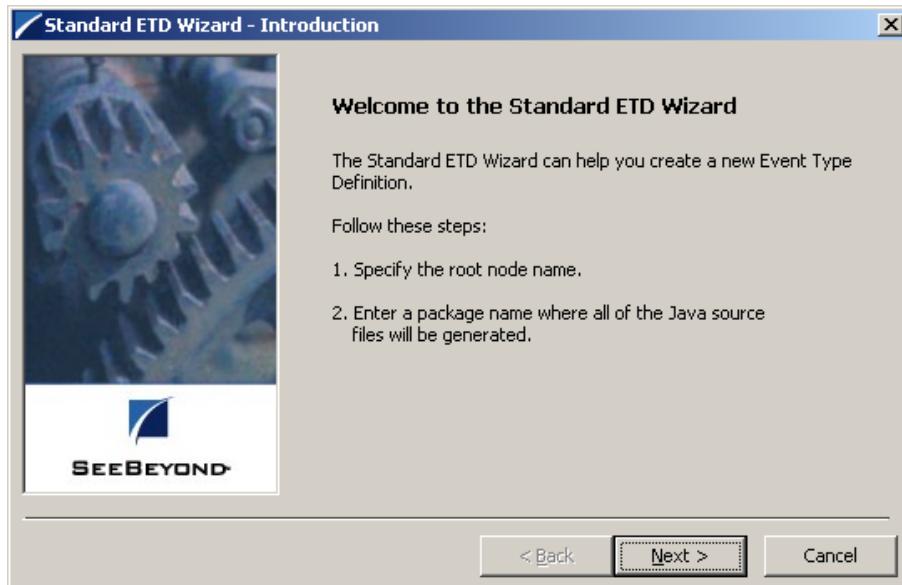
The **New Event Type Definition** dialog box displays icons for the ETD wizards; see [Figure 30 on page 100](#).

Figure 30 New Event Type Definition Window



- 3 Select the appropriate wizard (for example: **Standard ETD**) and click **OK**. The selected wizard opens (see Figure 31); in our example, the **Standard ETD Wizard - Introduction** dialog box.

Figure 31 Example Wizard: Standard ETD



Note: For complete details on how to use all the ETD wizards, see [Chapter 5](#).

- 4 It informs you that you must specify a root node name for the ETD and a package name where all the Java source files are generated and stored. Click **Next**.
- 5 When the **Standard ETD Wizard - Step 1** dialog box opens, enter a **Root Node Name** and a **Package Name** in the text box.
 - ♦ The root node name identifies the ETD within the ETD tree. If necessary, see [“About Node Names” on page 174](#) for rules governing node names.

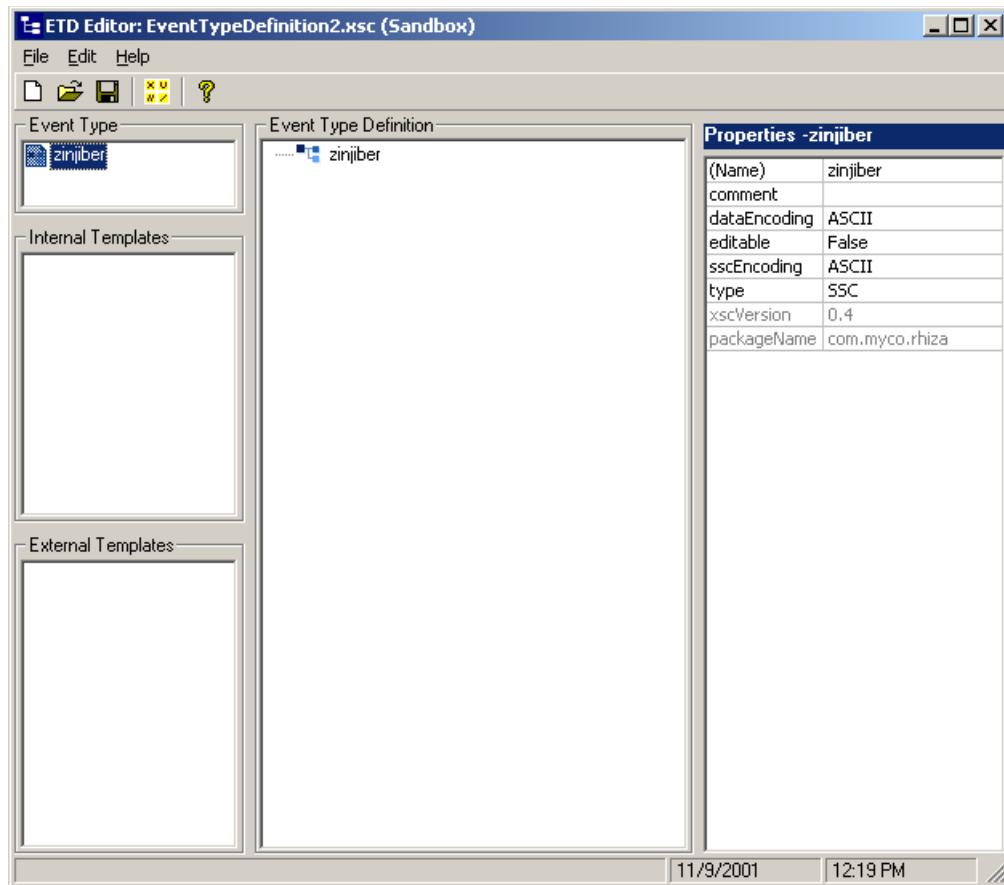
- The package name identifies the location where the ETD Editor will place all the generated Java classes that are associated with this ETD. This name must be a legal Java package name, such as **com.yourco.package**. If necessary, see “[About Package Names](#)” on page 174 for a rules governing package name.

When ready, click **Next**.

- The **Standard ETD Wizard - Step 2** dialog box opens, requesting that you confirm that the **Package Name** you entered is correct. If it is not correct, click **Back** to change the name, or, if correct, click **Finish** to generate the ETD.

The ETD you just created appears in the **Event Type Definition** pane in the **Event Type Definition Editor** dialog box (see Figure 32).

Figure 32 Java Event Type Definition Editor with Sample ETD



To name and save a newly created ETD

- In the ETD Editor window, on the **File** menu, click **Save**.
- When the **Save** dialog box opens, select **etd** as your directory and rename the **.xsc** file from **EventTypeDefinition1.xsc** to an appropriate external filename. The external filename may, but need not, match the internal root node name.

Note: You can save an **.xsc** file even if it is invalid; this allows you to return to it later for further work. You cannot use an ETD until it has been compiled, however.

To compile an ETD

- In the ETD Editor window, on the **File** menu, click **Compile and Save**.

The system saves the ETD Tree and attempts to compile the ETD.

- ♦ If the compile is successful, the status bar is blanked out.
- ♦ If an error is detected—such as a node whose **structure** property is set to **delim** when no value is defined for its **beginDelim** or **endDelim** property—a dialog box displays a text message explaining why the ETD could not be compiled.

After an ETD has been compiled, it resides in your Sandbox. Although *you* can assign it to any Event Type in your Sandbox or in your run-time environment, the ETD is not available to other users until it has been promoted to run time.

To promote an ETD to run time

- In the ETD Editor window, on the **File** menu, click **Promote to Run Time**.

The system prompts you to confirm any unsaved changes and then promotes the ETD *without attempting to validate or compile it*; this allows you to save and share a work in progress. When the promotion is complete, a message displays the names of the promoted files and the contents of the ETD Editor are removed.

To open an existing ETD

- 1 In the ETD Editor window, on the toolbar or **File** menu, click **Open**.
- 2 Highlight the **etd** directory, choose the appropriate file from the list of **.xsc** files, and then click **OK**.

To add elements and fields to an ETD

- In the ETD Editor window, in the **Event Type Definition** pane, right-click any node of the ETD. A shortcut menu appears, containing the following selections:

- ♦ **Add Element**
- ♦ **Add Field**

Depending where you are in the tree structure of your ETD, you can add elements and fields in the following locations in relation to the highlighted node:

- ♦ **Before Selected Node**
- ♦ **After Selected Node**
- ♦ **As Child Node**

When you select a node, the **Properties** pane allows you to view or edit its properties.

Note: For complete details on how to edit ETDs in the Java Event Type Definition Editor, see [Chapter 5](#).

4.4.4 Creating Monk Event Type Definitions

After you have entered all your Event Types, you then create ETDs using the e*Gate Enterprise Manager and ETD Editor.

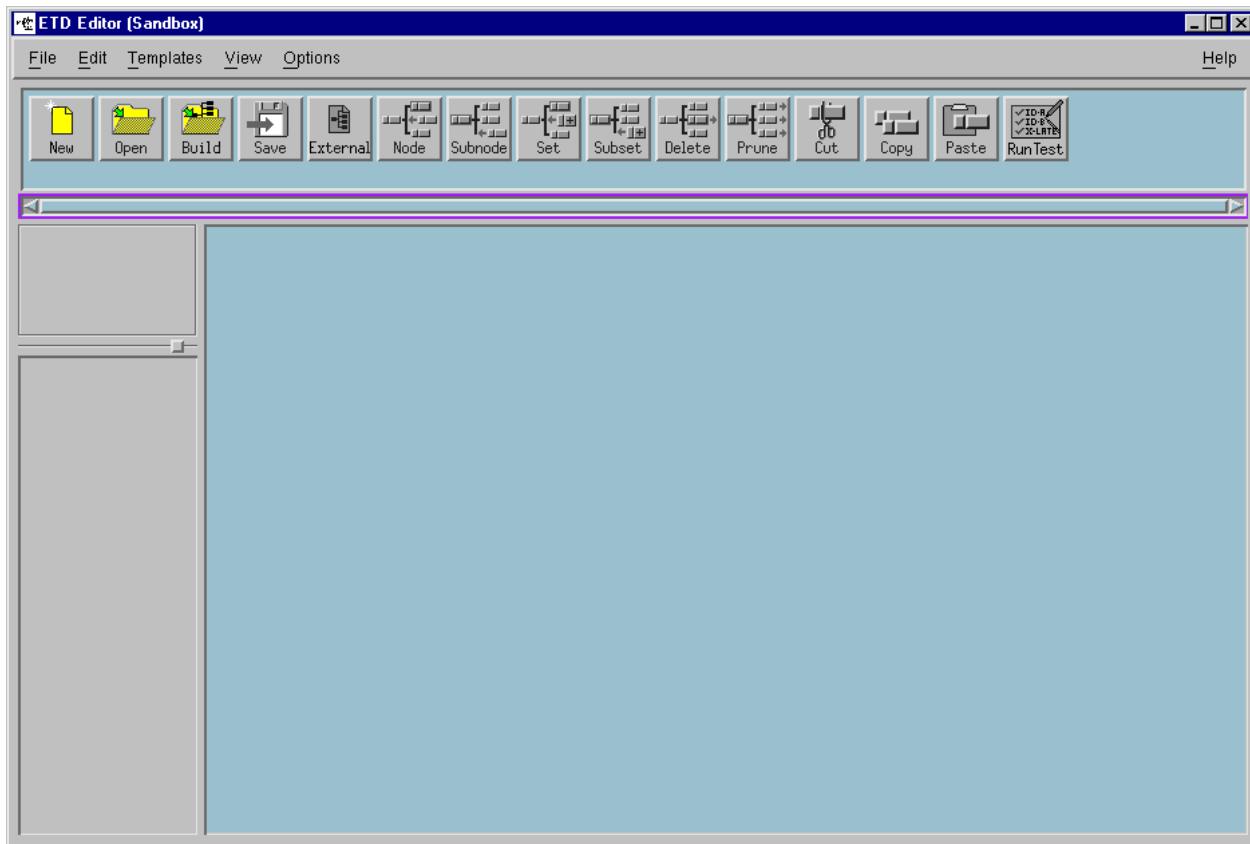
Note: For complete details on how to edit ETDs in the Monk Event Type Definition Editor, see [Chapter 6](#).

To create a Monk ETD

- 1 In Enterprise Manager, on the toolbar, click .

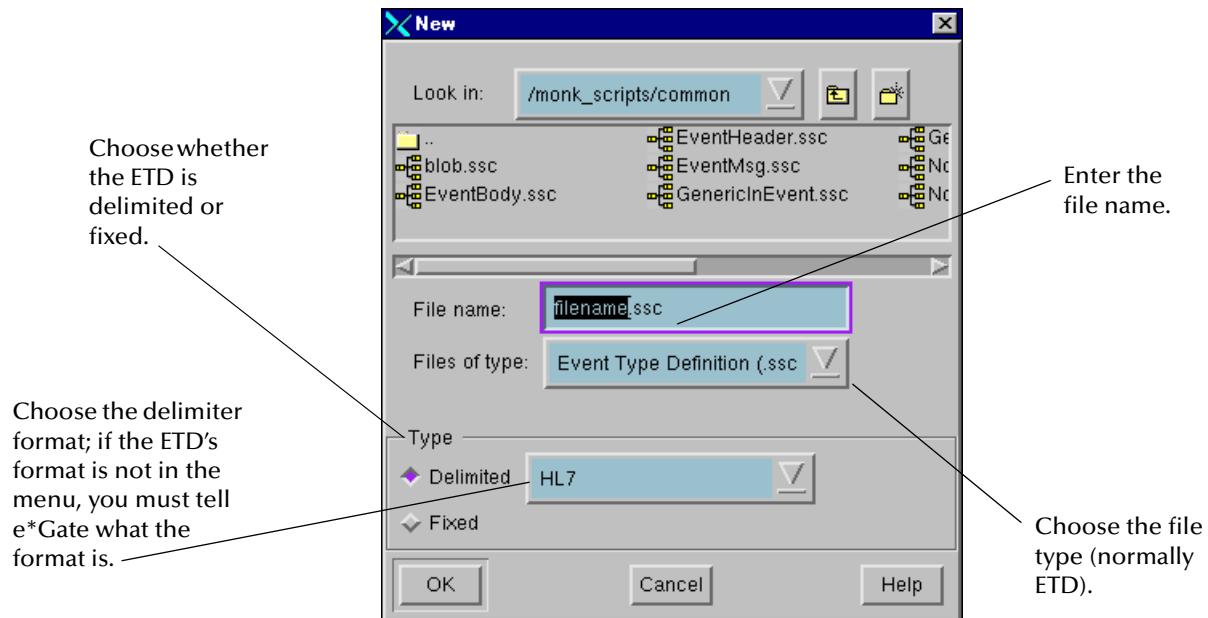
The ETD Editor window appears, with all panes empty. See Figure 33.

Figure 33 ETD Editor Window (New)



- 2 On the toolbar or **File** menu, click .
- 3 When the **New ETD** dialog box appears, enter the desired file name. See [Figure 34 on page 104](#).

Figure 34 New ETD Dialog Box



Note: Be sure to store the ETD file in the suggested folder or one of its subfolders. Do not change the folder name.

- 4 Choose the file type (normally an ETD .ssc file), then click **OK**.
- 5 Using the ETD Editor, finish creating a complete definition for the Event Type as desired. See “[Edit Menu](#)” on page 212 for a complete explanation of how to use the ETD Editor for this operation.

Note: Many e*Ways, for example database e*Ways and SAP, have specialized converters that query the external system and create complex ETDs for you. See the appropriate e*Way user’s guide(s) for details. You can also create your own ETD templates. See “[Working With ETD Templates](#)” on page 254 for details on using ETD templates.

- 6 When you are finished creating the current ETD: On the toolbar or **File** menu, click **Save** to save the .ssc file. e*Gate saves Monk ETD files with this extension.
- 7 On the **File** menu, click **Promote to Run Time**. This action takes the file out of your Sandbox and places it in the system run time.
For an explanation of the Sandbox and run-time states, see “[Sandbox and Run-Time Environments](#)” on page 57.
- 8 When you are finished: On the **File** menu, click **Close** to exit the ETD Editor.
Repeat steps 2 through 7 for each ETD you want to create.

Note: You are now ready to assign ETDs to Event Types (see “[Assigning Definitions to Monk Event Types](#)” on page 105).

4.4.5 Assigning Definitions to Monk Event Types

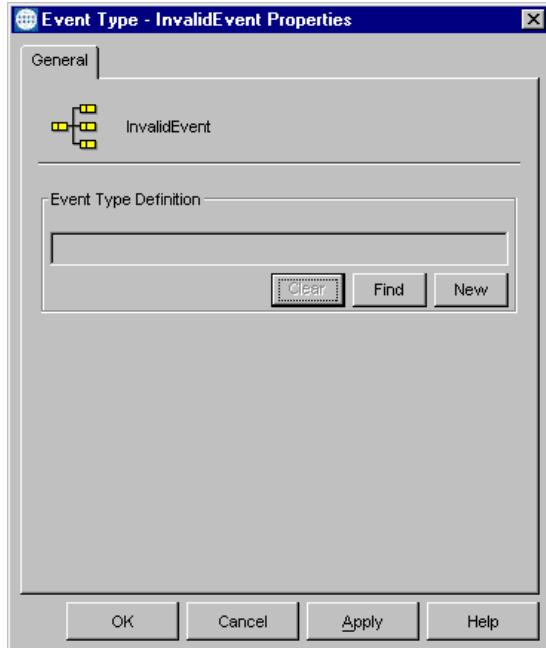
After you have created Monk ETD files, you can assign them to Event Types you have already created.

To assign ETDs to Event Types

- 1 In the Enterprise Manager window, select the **Event Types** folder in the Navigator/Components pane. See [Figure 28 on page 97](#).
- 2 In the Editor pane, select one of the Event Types you created.
- 3 On the toolbar or **Edit** menu, click  **Properties**.

The **Event Type Properties** dialog box appears. See Figure 35.

Figure 35 Event Type Properties Dialog Box



- 4 Under **Event Type Definition**, click **Find**.

The **Event Type Definition Selection** dialog box appears; it is similar to the Windows **Open** dialog box.

Note: Clicking **New** in the **Event Type Properties** dialog box opens the **ETD Editor** window, allowing you to create a new ETD.

- 5 Open the **etd** folder (for Java) or **monk_scripts\common** folder (for Monk), then select the desired file name (**.xsc** for Java, **.ssc** for Monk).
- 6 Click **Select**. The file populates the **Event Type Definition** field.

- 7 To save any work in the properties dialog box, click **Apply** to enter it into the system.
- 8 When finished assigning ETDs to Event Types, click **OK** to close the properties dialog box and apply all the properties.

4.5 Creating Collaboration Rules and Scripts

In the e*Gate system, Events become subject to business logic via the following data operations

- Processing
- Transformation
- Verification

e*Gate uses the following features to govern these operations:

- **Collaboration** is the necessary, configurable component of an e*Way that determines its operation; that is, the logical moving and transformation of Events (see “[Adding Collaborations](#)” on page 142).
- **Collaboration Rules** are the program logic that instructs a Collaboration how to execute the business logic required to support e*Gate’s data transformation and routing.
- **Collaboration service** is the program that defines the structure and operation of a Collaboration Rule’s basic Event-handling processes.
- **Collaboration Rules Script** contains the specific operations (written in Monk) that are used to govern Event-transformation processes within a Collaboration (see “[Collaboration Rules Scripts](#)” on page 107).
- **Business Rules** are the Java source code that creates the output Events that are a result of the Java Collaboration.

Note: Monk is a SeeBeyond programming language designed to handle e*Gate Event processes. For more information on using the Monk language, see the [Monk Developer’s Reference](#).

Collaboration .class files (Java) and Collaboration Rules scripts (Monk) are necessary if you want to have any data transformed and/or verified in some way as it passes through a Collaboration. “[Creating Java Collaborations](#)” on page 112 (Java), “[Creating Monk Collaboration Rules](#)” on page 116 (Monk), and “[Configuring Monk Collaboration Rules](#)” on page 117 (Monk) explain operations with and basic properties of Collaboration Rules components.

This section explains:

- “[Using Collaboration Scripts](#)” on page 107
- “[Java Collaborations](#)” on page 110
- “[Monk Collaborations](#)” on page 116

4.5.1 Using Collaboration Scripts

To define complex Event handling and transformations in a Collaboration and Collaboration Rules, you can use the Monk, Java, or C programming languages to write a Collaboration script, or you can use Collaboration Rules (Monk) or Collaboration (Java) scripts. Once you have written and successfully tested a script, you can then add it to the system's run-time operation.

C-language Scripts

You can write Collaboration scripts in the C programming language if desired, using a text editor. Explaining details of how to write these scripts is beyond the scope of this user's guide. For more information, see the **Gate Integrator Collaboration Services Reference Guide*.

Monk Language Scripts

Table 10 explains the basic types of Collaboration scripts that use the Monk language.

Table 10 Monk Collaboration Script Types

Script Name	Extension	Function
Collaboration Rules script (CRS)	.tsc	Defines Collaboration Rules (specialized Monk programs) that handle a variety of often-used data operations within a Collaboration; written using the Collaboration Rules Editor.
Database access	.dsc	Special Monk scripts used by certain types of e*Ways to communicate with databases — for example, the e*Ways for Oracle, Sybase, and ODBC. Backward compatible.
Monk	.monk	Contains specialized Monk function definitions written in a text file and used to transform business Events; you can also call these functions within Collaboration Rules scripts to perform e*Way-specific operations.
Collaboration-ID Rules	.isc	Helps verify Events within the e*Gate system; written using the Collaboration-ID Rules Editor (for backward compatibility with e*Gate Version 3.6 only; normally you do not need to use this feature or file structure).

Collaboration Rules Scripts

Collaboration Rules scripts are specialized Monk programs that define how Collaborations and Collaboration Rules transform data from input Event Types to output Event Types.

Collaboration Rules Editor

The e*Gate Collaboration Rules Editor feature has tools that help you write Collaboration Rules scripts in the Monk language. The appropriate sections in this chapter explain how to get started with this operation. See [Chapter 8](#) for a complete explanation of how to use the Monk Collaboration Rules Editor feature.

Java Language File Types

Table 11 explains the basic types of Collaboration file types that use the Java language.

Table 11 Java Collaboration File Types

File Name	Extension	Function
collab.xpr	.xpr	Collaboration Rules project file that keeps track of GUI settings and preferences and contains pointers to all files noted below.
collab.xts	.xts	ASCII file that defines the Collaboration Rules layout in the GUI.
collab.class	.class	The class that implements Java Collaborations. After the collab.class files are compiled, they generate .java files.
collab.java	.java	The human-readable Java source code generated by the Java Collaboration Rules Editor; which displays a read-only copy on request.
etd.xsc	.xsc	Stores abstract syntax. Defines Event Types; written by the Event Type Definition Editor (see Chapter 6 for a complete explanation). Equivalent to the .ssc file in Monk.

Java Files

The Java **.xpr**, **.xts**, and **.java** files are functionally comparable to the **.tsc** script in Monk. They manage Java Collaboration dependencies, store Java Collaboration Rules, and implement Java Collaborations. After they are compiled by a compiler (which must be downloaded from Sun—the compiler is not shipped with e*Gate), along with the **.xsc** files corresponding to Event Type Definitions, they become **.class** files that define Event Types.

Collaboration Editor

The Collaboration Editor allows you to view a graphical representation of the Java Collaboration files you wrote and compiled (using the compiler you downloaded).

Note: When you use drag-and-drop (or Find and Map) to connect one node with another, the Editor draws lines between source nodes and destination nodes.

The Collaboration Editor has the capability to accept a DTD or XML schema from a vendor that is XML compliant, or from a standards body, and build Collaborations between these Events. You are also able to write Collaborations between XML and other Event types, as well as Collaborations that transform XML messages (such as I2 TradeMatrix Purchase Orders or RosettaNet Purchase Orders) to/from SAP IDOC formats (such as POCRD01 IDOC).

See the appropriate sections in this chapter for an explanation on how to get started using the Editor, and [Chapter 7](#) for a complete explanation of how to use the Collaboration Editor feature.

Collaboration Rules Properties

All Collaboration Rules components determine the following basic properties for Collaborations:

- A **Subscription** defines Event Types the Collaboration expects as input.
- A **Publication** defines Event Types the Collaboration produces as output.

For more information on Collaboration components and an explanation of how to create them, see [“Adding Collaborations” on page 142](#).

Collaboration Services and Types

You must associate a Service with each Collaboration Rules component. The default e*Gate Collaboration Service types are:

- **Pass Through** is for Collaboration Rules that allow data to flow through without change. For Java Collaborations, there is also a PassThrough class in **STCLibrary**\.
- **Monk** is for Collaboration Rules with Collaboration Rules scripts written in the Monk language.
- **Monk ID** is for e*Gate Version 3.6 only. Substitute the **Monk ID** Service, which is backwards compatible to DataGate, for the **Monk** Service if you are using that version; otherwise, you do not need to use it.
- **Java** is for Collaboration Rules with scripts written in the Java language.
- **C** is for Collaboration Rules with scripts written in the C language.
- **Route Table** operates in conjunction with the Monk ID Service, for the same purpose.

Note: See the *e*Gate Integrator Collaboration Services Reference Guide* for complete information on these Services.

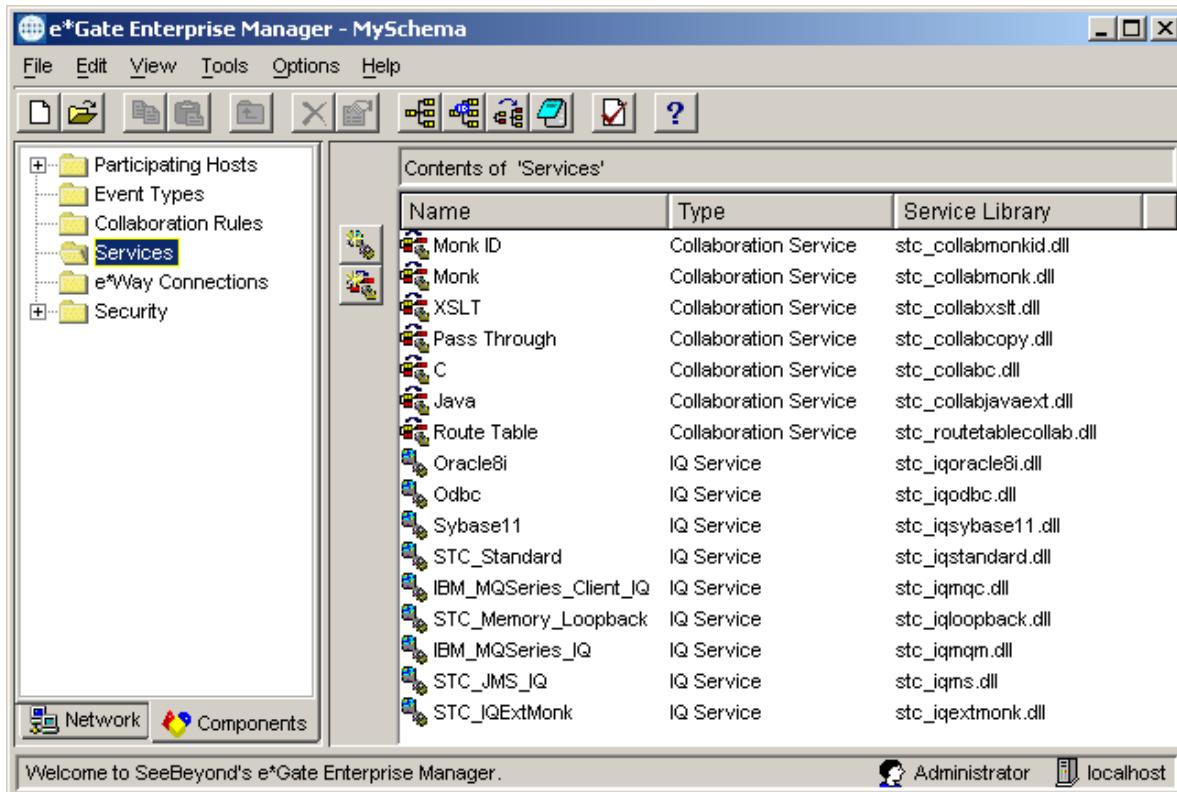
Your e*Gate installation can have additional Collaboration Services available, depending on the add-ons you use and the external systems.

Basic Data Handling: No matter how you set up a Collaboration Rules component, data either flows through it unchanged (using either the Pass Through Service or the **STCJavaPassThrough** class) or changed (Java, Monk, or another service). The Service you use and the way you configure it determine the flow.

To list available Services in your system

Click the **Services** folder in the Enterprise Manager window’s Navigator/Components pane. The Services list appears in the Editor pane. [Figure 36 on page 110](#) shows an example.

Figure 36 Enterprise Manager Window with Services



In addition, the **Services** folder also contains IQ Services. See the following references for details on e*Gate services and the features shown in Figure 36:

- For **Collaboration Services**, see the *e*Gate Integrator Collaboration Services Reference Guide*.
- For **IQ Services**, see the *e*Gate Integrator Intelligent Queue Service Reference Guide* and the *SeeBeyond JMS Intelligent Queue User's Guide*.

See “[Adding Intelligent Queues](#)” on page 136 for information on how to add and configure IQs in your e*Gate system as well as more information on IQ Services.

You can have additional specialized services available, depending on your system needs. See the *e*Gate Integrator Collaboration Services Reference Guide* for details.

4.5.2 Java Collaborations

This section explains how to create and configure Java Collaborations. For additional information on the Java Collaboration Rules Editor, see [Chapter 7](#).

The Java Collaboration Rules Editor is divided into six panes (see [Figure 37 on page 111](#)):

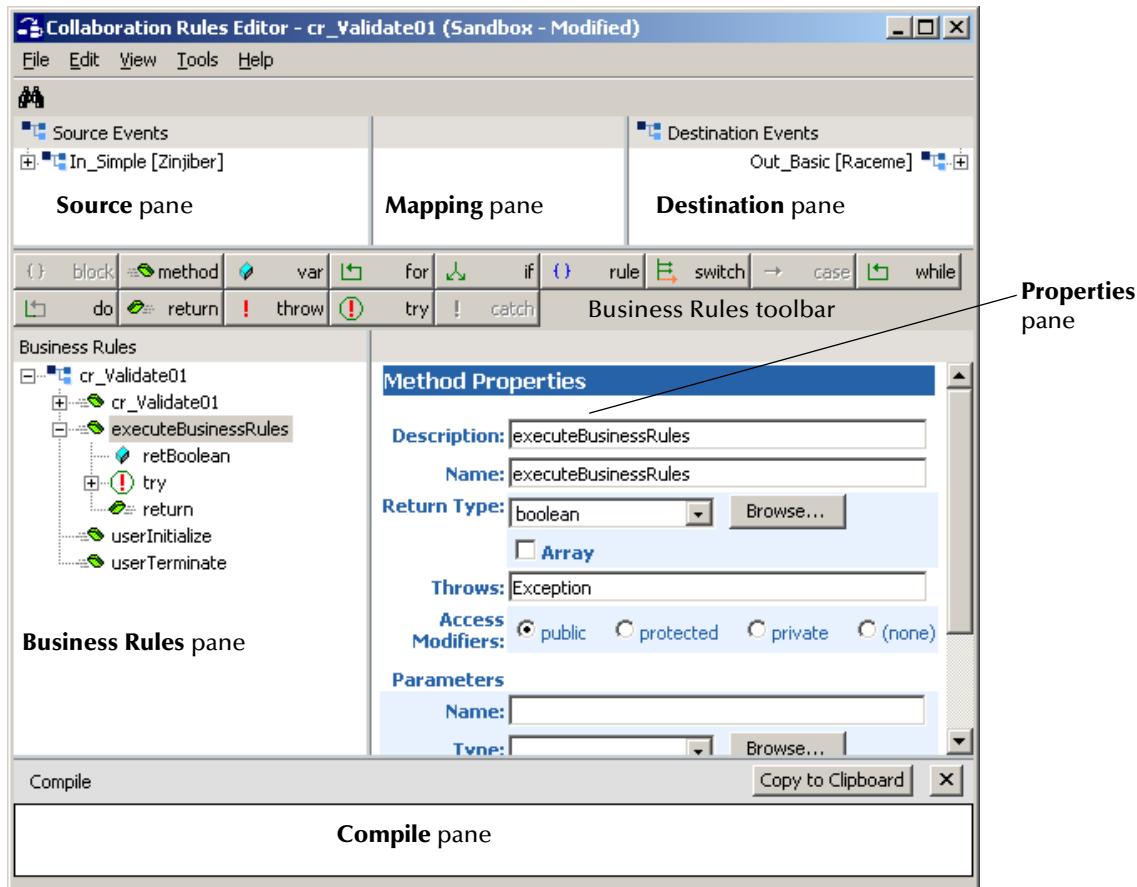
- The **Source** pane lists the ETDs that correspond to the Events that are subscribed to by the Collaboration.

- The **Mapping** pane displays the relationship between the Source and Destination ETDs as a series of lines that connect the associated nodes.
- The **Destination** pane lists the ETDs that correspond to the Events that are published by the Collaboration.
- The **Business Rules** pane displays a graphical depiction of the Java source code that creates the output Events that are a result of the Collaboration. Much of the code displayed in this pane is automatically created by the Collaboration Rules Editor.

Note: The `executeBusinessRules()` method is the primary placeholder for programming that users can add.

- The **Properties** pane displays information about the rule that is selected in the **Business Rules** pane. This is also the area where you edit the selected rule.
- The **Compile** pane displays any errors that occurred as a result of a compilation of a Java Collaboration. Whenever you make any changes to the rules, you must compile the Java Collaboration. On the **View** menu, you can check or uncheck **Display Output** to display or hide the Compile pane.

Figure 37 Java Collaboration Rules Editor



Creating Java Collaborations

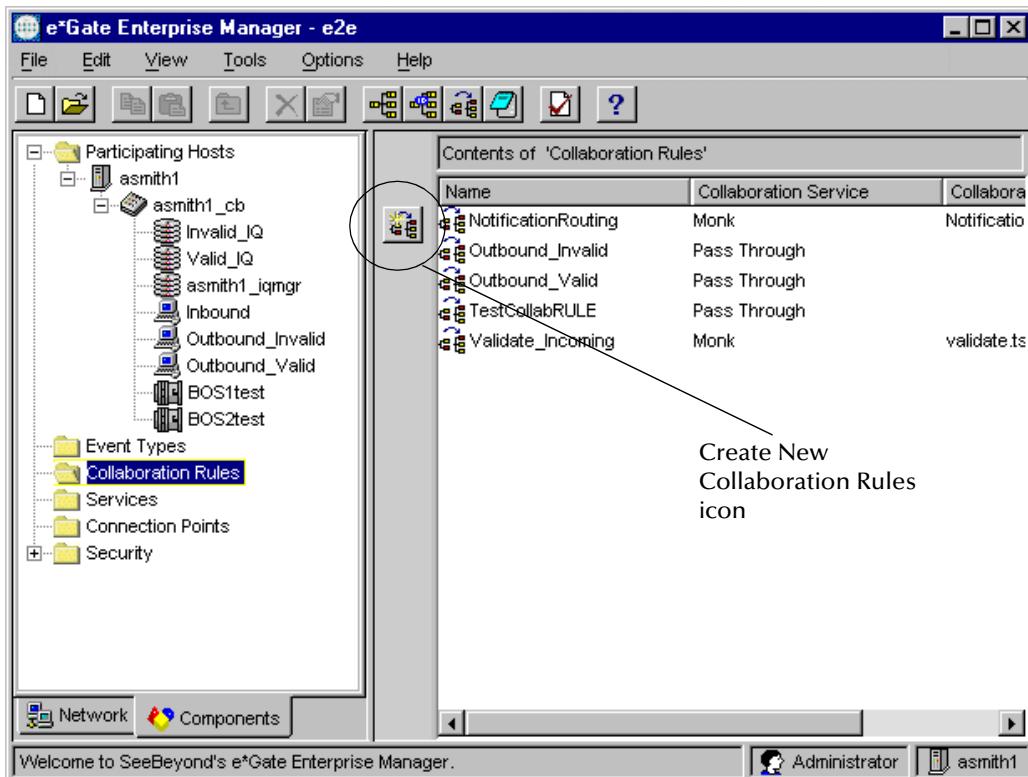
The Java Collaboration Rules Editor is the e*Gate tool for creating Java Collaborations. Unlike Monk Collaborations, Java Collaborations allow you to have multiple ETDs on the Source and Destination sides of a Collaboration. You are able to freely collaborate between these ETDs.

Note: If you want data to flow unchanged from one component to another, you can use the `STCJavaPassThrough.class` in `collaboration_rules\STCLibrary\`.

To create the Java Collaboration

- 1 Select the **Collaboration Rules** folder in the Components pane of the e*Gate Enterprise Manager.
- 2 Click the **Create New Collaboration Rules** icon. See Figure 38.

Figure 38 e*Gate Enterprise Manager: Create New Collaboration Rules



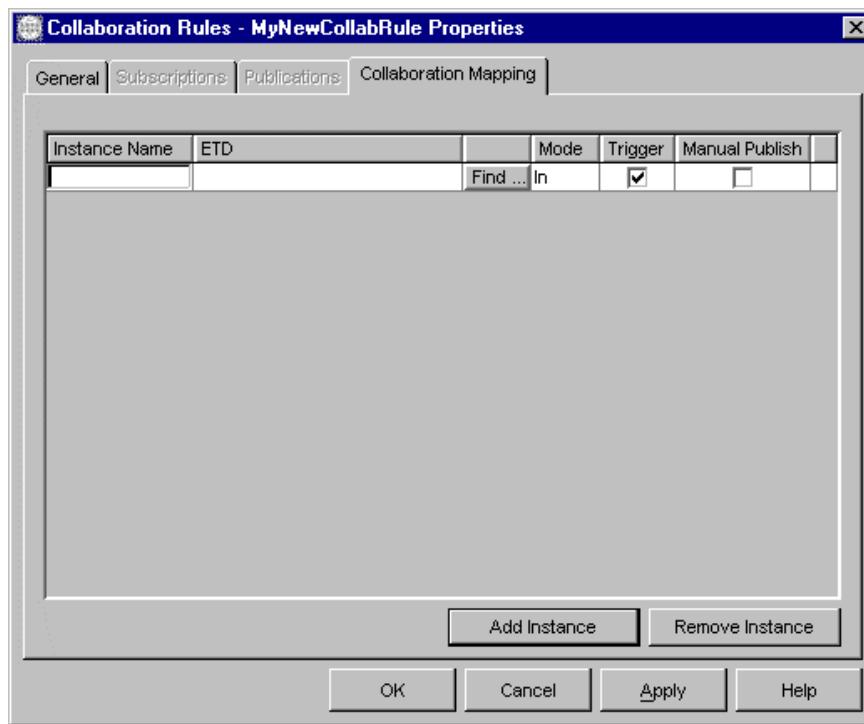
- 3 Type a name for the new Collaboration Rules component in the **Name** text box and then click **OK**.
- 4 Open the properties dialog box for your new Collaboration Rules component name. Make sure the name is highlighted in the Editor pane and then, on the toolbar or **Edit** menu, click **Properties**.

- 5 Optionally, if you want this Collaboration Rule to be Pass Through rule using Java—in other words, transporting but not transforming the data—do the following.
 - A In the Collaboration Rules area, click **Find**.
 - B In the Selection dialog box, navigate to the **collaboration_rules\STCLibrary** folder and click **STCJavaPassThrough.class**.
 - C Click **Select**.

The selection is registered in both the Collaboration Rules area and the Initialization File area. Nothing further is necessary; skip steps 6 through 15.

- 6 Select the **Collaboration Mapping** tab.
- 7 Click **Add Instance** to add a row. See Figure 39.

Figure 39 Collaboration Rules Properties: Collaboration Mapping Tab



- 8 Enter a name for this instance in the **Instance Name** text box.

If you will be creating ETDs on the fly, this name is used as the default root node name for the ETD in the Collaboration you are creating.
- 9 Select the **Mode** for this Event Type instance.
 - ♦ **In** corresponds to source Event Type instance in the Collaboration, and is the default.
 - ♦ **Out** corresponds to destination Event Type instance in the Collaboration.
 - ♦ **In/Out** corresponds to using the same Event Type instance as both a source and destination.

- 10 Choose whether this Event Type instance should trigger the execution of the Collaboration, as follows:
 - ♦ If the **Trigger** check box is activated (the default), receipt of this Event Type by the Collaboration triggers its execution.
 - ♦ If the **Trigger** check box is not activated, the receipt of this Event Type does not trigger the Collaboration's execution.

Note: *At least one of the ETD instances used by the Collaboration must be checked as the trigger. All nontriggered Events must be actively instantiated in the Collaboration by means of calling the `receive()` method.*

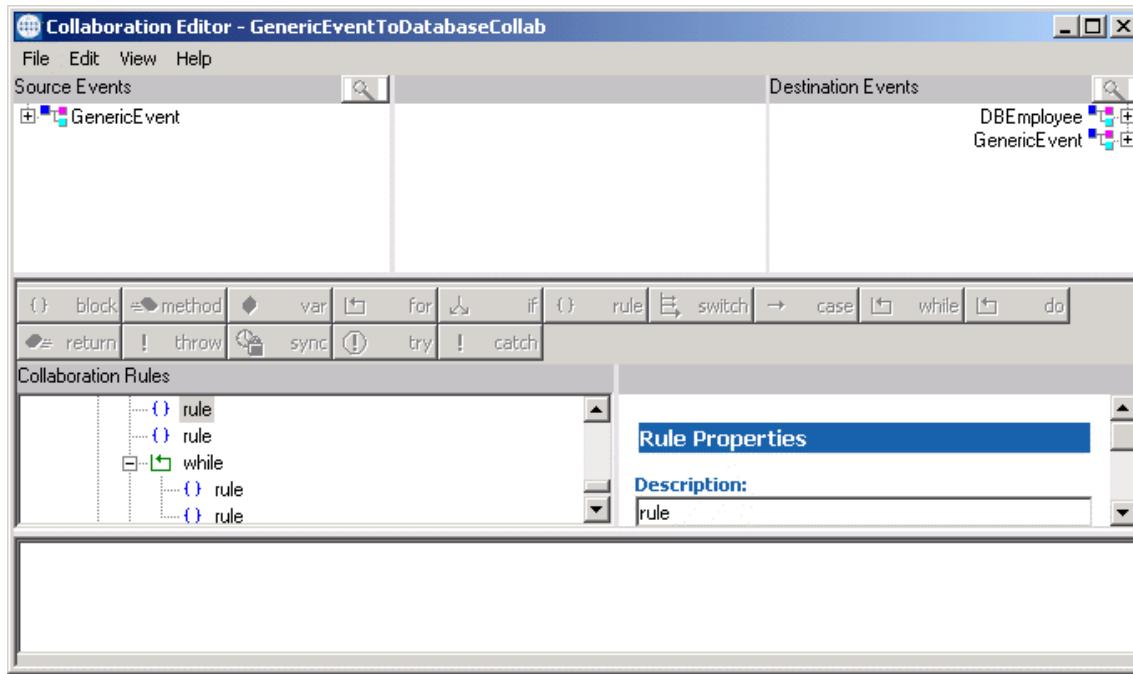
- 11 Choose whether to automatically generate an Event for this Event Type instance when the Collaboration finishes its execution as follows:
 - ♦ If the **Manual Publish** check box is not activated (the default), then an Event is generated for this ETD when the Collaboration finishes its execution.
 - ♦ If the **Manual Publish** check box is marked, then no Event is generated for this ETD even if the ETD instance mode is **Out** or **In/Out** and the ETD has data placed in it by the Collaboration.

Note: *You can still publish an Event based on this Event Type instance even if **Manual Publish** is activated—simply use the `send()` method in your Collaboration Rules.*

- 12 Click **Apply**, then select the **General** tab.
- 13 Under **Collaboration Rules**, click **New**.
- 14 Enter the **Collaboration Class Name** and whether or not to enable **Event Linking and Sequencing (ELS)**.
Enabling ELS adds the ELS methods to a skeleton Collaboration to which you add your custom Java Business Rules. For more information on ELS, see [Chapter 12](#).
- 15 Click **OK**.

The Editor window opens; see Figure 40. The skeleton Java Collaboration is created based on the information entered on the **Collaboration Mapping** tab. This information is displayed by the Editor for this Collaboration.

Figure 40 Java Collaboration Editor with Sample Collaboration



Note: If you double-click an .xsc file, it opens in the Java Collaboration Rules Editor even if the default editor is set to Monk.

You are now ready to add the custom Business Rules to the Collaboration.

Adding Custom Business Rules to the Collaboration

Custom business rules must your newly created Collaboration, which is a collection of Java source code. The rules you add in the Java Collaboration Editor are implemented under the **executeBusinessRules()** method.

To add custom rules to the Java Collaboration

- 1 Optionally, click an appropriate node in the Business Rules pane and then click a button on the Business Rules toolbar to add a rule fragment. For details, see ["Business Rules Toolbar" on page 269](#).
- 2 Optionally, use drag-and-drop or **Find and Map** to copy the data in a node of a source Event Type instance to a node in a destination Event Type instance. For details, see ["Dragging and Dropping Fields" on page 277](#).
- 3 To make the Java code visible in the Business Rules pane, on the **View** menu, click **Display Code**.

This pane can be expanded to show more of the Java code. See Figure 41.

Figure 41 Expanded Java Code

```
initialize : public void initialize(JCollabController aJCollabController) throws CollabConnException
    (O) rule : this.collabController = aJCollabController;
    (O) rule : _in = new In();
    (O) rule : _out = new Out();
translate : public boolean translate() throws CollabConnException, CollabDataException
terminate : public void terminate()
    (O) rule : getIn().terminate();
    (O) rule : getOut().terminate();
executeBusinessRules : public boolean executeBusinessRules() throws CollabConnException, CollabDataException
    retBoolean : boolean retBoolean = true;
    try :
        catch : catch (Exception e)
            (O) rule : e.printStackTrace(System.err);
            (O) rule : throw new RuntimeException(e.getMessage());
        finally :
            (O) rule : return retBoolean;
```

The highlighted method is where you add your custom business rules.

Note: For complete details on how to create the custom business rules in the Java Collaboration Rules Editor, see [Chapter 7](#).

4.5.3 Monk Collaborations

This section explains how to create and configure Monk Collaboration Rules.

Creating Monk Collaboration Rules

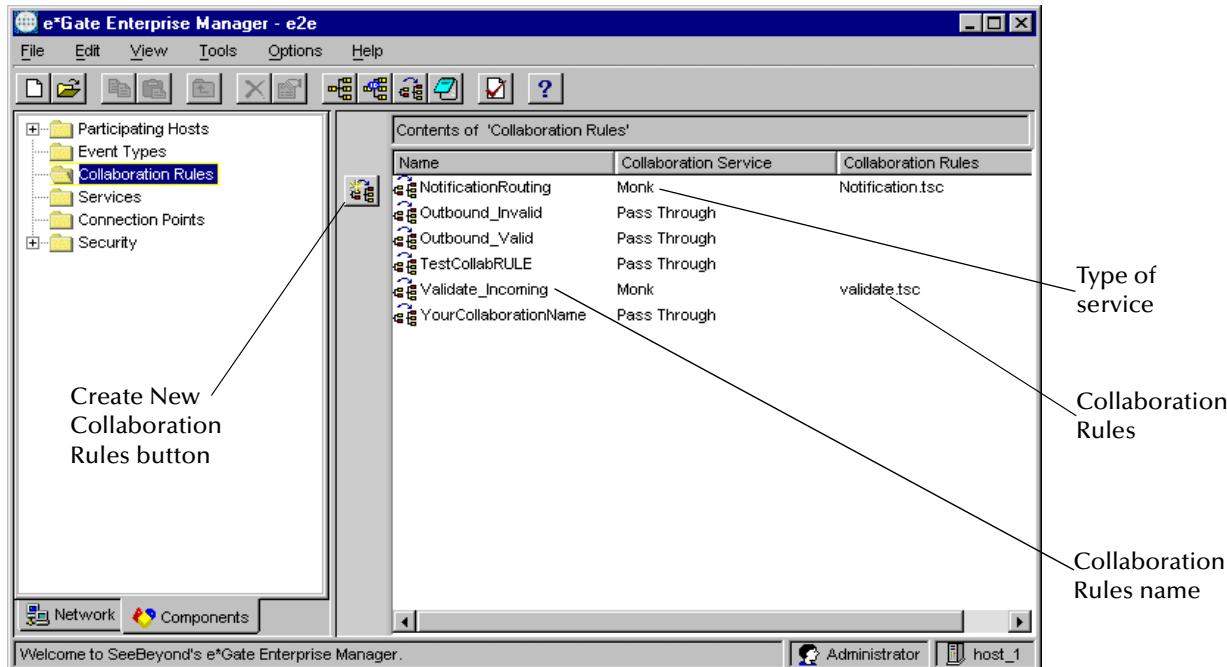
If you want to create Monk Collaboration Rules, you must first create a Collaboration Rules component, then configure it. To create new Collaboration Rules components, use the Enterprise Manager GUI.

To create Collaboration Rules

- 1 In the Navigator/Components pane, select the **Collaboration Rules** folder.
 - 2 On the Palette, click .
- The **New Collaboration Rules Component** dialog box appears.
- 3 Enter the desired name for a new Collaboration Rules component and click **Apply** to enter it into the system.

After you have named each Collaboration Rules component, its name and a Collaboration Rules icon appear in the Editor pane. See Figure 42.

Figure 42 Enterprise Manager with Collaboration Rules Icons



Note: The name of the Collaboration Service does not appear in that column of the Editor pane until after you have configured the Collaboration Rules component. During configuration, you must choose the appropriate Service.

- 4 Name as many additional Collaboration Rules components as desired, clicking **Apply** after naming each one.
- 5 When finished, click **OK** to close the dialog box.

Configuring Monk Collaboration Rules

After you have created your Monk Collaboration Rules components, you must then configure them, using the Enterprise Manager GUI. Once you have configured a Collaboration Rules component, you can either write a new script for it or select an existing script to associate with the component.

Collaboration Rules components start up on command, under conditions you specify, using Monk command parameters. You must enter these commands, using the **Collaboration Rules Properties** dialog box, under the **General** tab.

You can initialize Collaboration Rules in one of the following ways:

- For **Strings**, enter the correct information in the **Initialization string** text box.
- For **Files**, find the file in the **Initialization file** text box. To use this option, you must first create the initialization file, using a text editor.

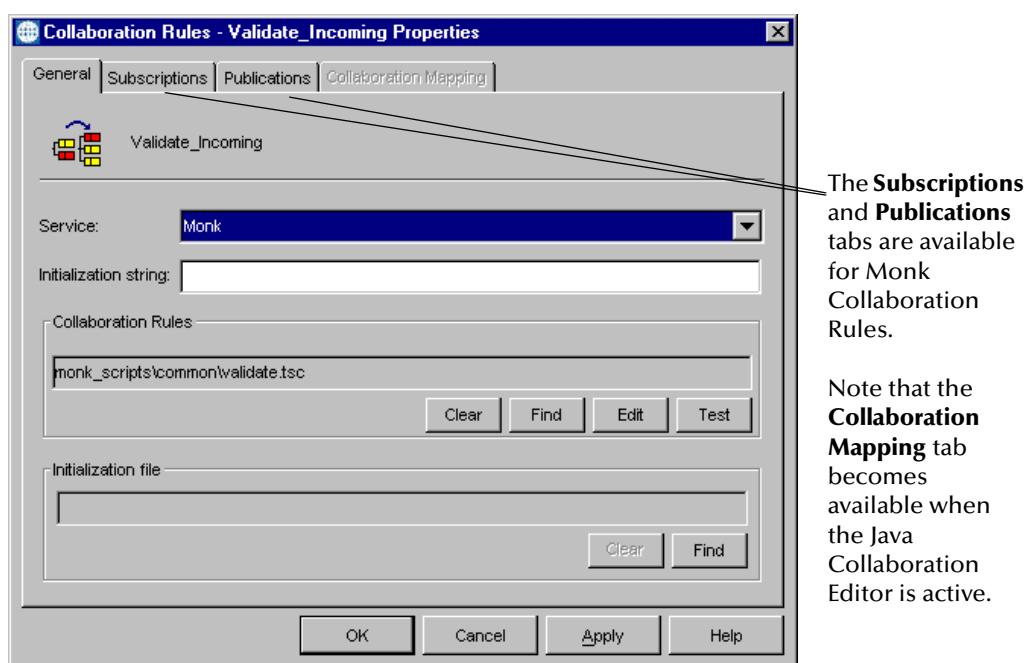
For more information on how to construct these command features, see the *e*Gate Integrator System Administration and Operations Guide*.

To configure Collaboration Rules

- 1 With the **Collaboration Rules** folder open in the Navigator/Components pane, select the desired Collaboration Rules name in the Editor pane.
- 2 On the toolbar or **Edit** menu, click  **Properties**.

The **Collaboration Rules Properties** dialog box appears (see [Figure 43 on page 118](#)), displaying the **General** tab.

Figure 43 Collaboration Rules Properties Dialog Box, General Tab



Note: For more information on the details of information needed to use this properties dialog box, including directory locations, see [Chapter 8](#).

- 3 Select the desired properties for the current Collaboration Rules component as follows:
 - **Service** allows you to choose one of the Collaboration Rules Services: Java, Pass Through, Monk, Monk ID, or another. You must choose a **Service**.
 - **Initialization string** allows you to enter the initialization character string, if necessary, for a script. For information on using initialization strings, see the *e*Gate Integrator Collaboration Services Reference Guide*.
 - **Collaboration Rules** allow you to choose an existing Collaboration Rules script file or create a new one as follows:
 - **Find** displays a Windows **File Selection** dialog box, allowing you to find an existing Collaboration Rules script file.

- ♦ **New** displays the Collaboration Rules Editor window, allowing you to create a new script using the Editor.

Note: After you display a file name for a Collaboration script, the **New** button becomes **Edit**, allowing you to edit the file here if desired.

In addition, you can use the following buttons, if desired:

- ♦ **Clear** removes a selected Collaboration Rules script, allowing you to choose another.
- ♦ **Test** opens the Monk Test Console window, allowing you to test the validity of a Collaboration Rules script. See “[Monk Test Console](#)” on page 149 for details.
- ♦ **Initialization file** allows you to find (or exchange) a file necessary to initialize a Collaboration script.

You must take one of the following steps:

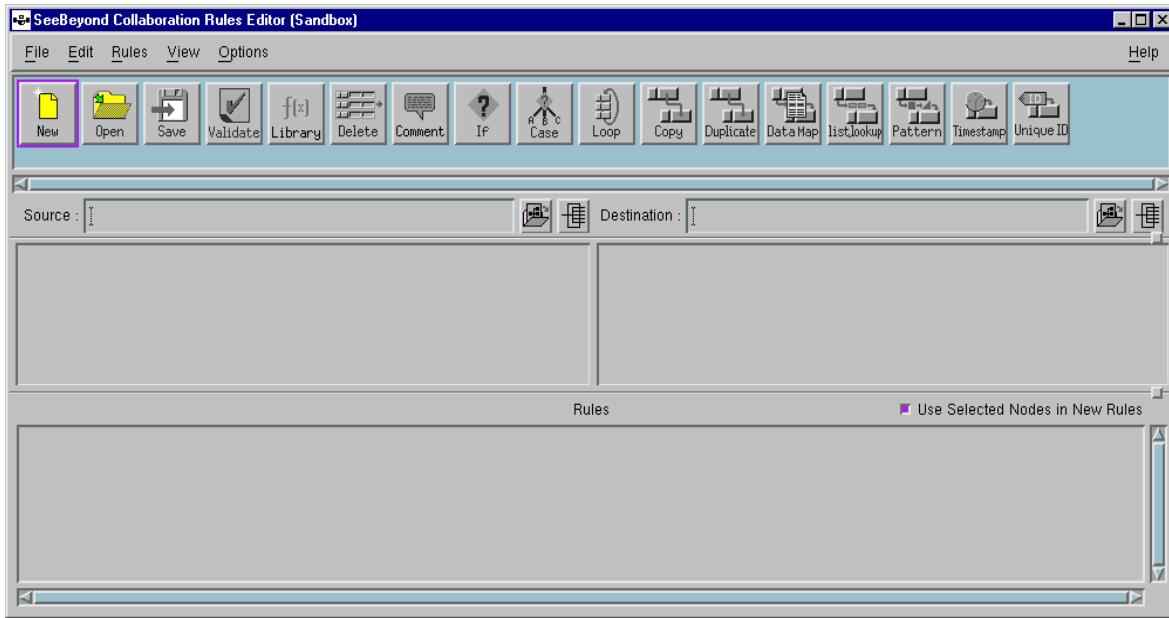
- ♦ If your **Collaboration Rules** script already exists, click **Find** to display a **File Selection** dialog box and choose the desired file. Go on to step 7.

Note: If you have already used a text editor to write a Collaboration script (in Monk or another language) for the current Collaboration Rules component, use this dialog box to select the file containing that script. See [Chapter 8](#) for details on directory locations.

- ♦ If you need to write a new **Collaboration Rules** script, click **New**. The **SeeBeyond Collaboration Rules Editor** window appears. See Figure 44.

Note: You can also open the Editor using the Enterprise Manager toolbar or **Tools** menu: click  **Collab Editor**.

Figure 44 SeeBeyond Collaboration Rules Editor Window (Monk)



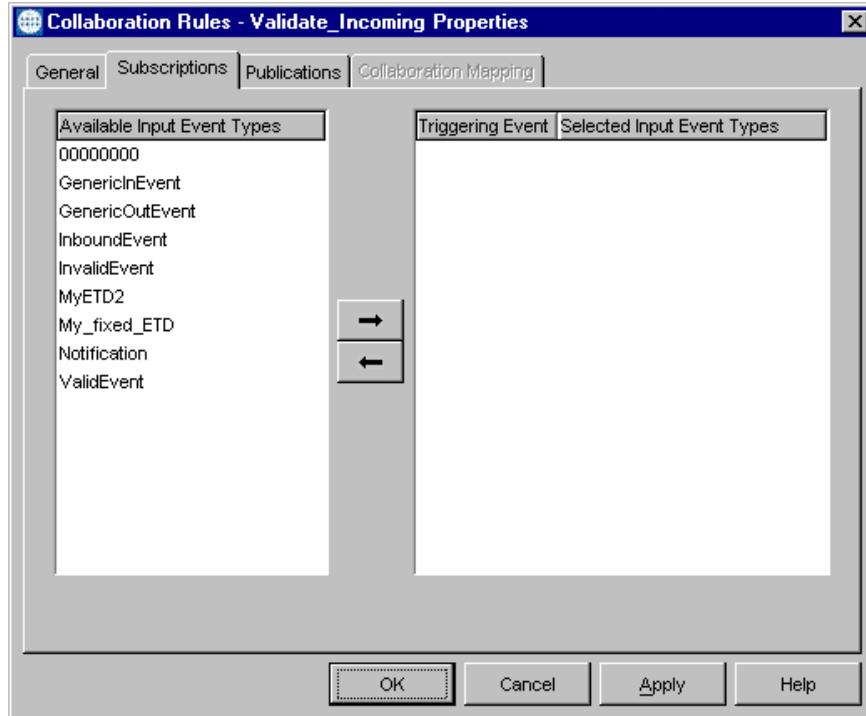
- 4 Using the Collaboration Rules Editor, create the Monk script you need for the current component as desired. See [Chapter 8](#) for a complete explanation of how to use the Collaboration Rules Editor for this operation.

Caution: *If you try to open a file with more than 6500 lines in the Collaboration Rules Editor, you get an error message, and the file does not open. If you need to open a larger file, use a text editor or word processor capable of handling large files.*

- 5 When you are finished creating and/or editing a Collaboration Rules script file:
On the toolbar or File menu, click **Save**, and then click **Close** to close the Collaboration Rules Editor window.
- 6 To save your Collaboration Rules work in this tab of the **Collaboration Rules Properties** dialog box and enter your selections into the system, click **Apply**.
- 7 From the **Collaboration Rules Properties** dialog box, click the **Subscriptions** tab.

The properties dialog box changes (see Figure 45), displaying subscription properties.

Figure 45 Collaboration Rules Properties Dialog Box, Subscriptions Tab



- 8 Choose an input (subscribed) Event Type for the current Collaboration Rules component by selecting a desired Event Type in the **Available Input Event Types** pane, then clicking on the right-pointing arrow. The Event Type's name appears in the right pane. Repeat this step as necessary.

When you move an Event Type into the **Selected Input Event Types** column (right pane), a check box appears next to the Event Type's name, in the **Triggering Event** column.

- 9 A check appears in the **Triggering Event** check box, marking the Event Type as "required" when the subscribing Collaboration begins processing this Event Type. If it is not required to begin processing this Event Type immediately upon the Collaboration's receipt, deselect the check box.

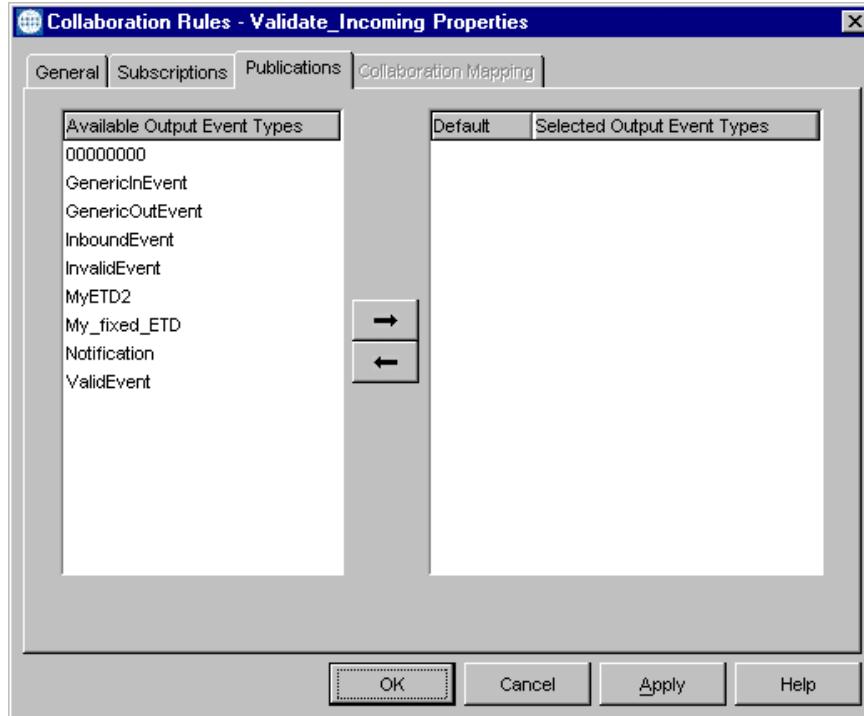
Note: You can process nonrequired events using the command line by calling *iq-get*. See the **Monk Developer's Reference** for more information, and note that this is not supported by JMS IQs. If more than one Event Type is marked required, the Collaboration begins execution immediately upon receipt of any of the required Event Types.

- 10 To save your Collaboration Rules work in this tab of the **Collaboration Rules Properties** dialog box and enter your selections into the system, click **Apply**.

- 11 Click the **Publications** tab.

The properties dialog box changes (see Figure 46), displaying publication properties.

Figure 46 Collaboration Rules Properties Dialog Box, Publications Tab



- 12 Choose one or more output (published) Event Types for the current Collaboration Rules component by selecting a desired Event Type in the **Available Output Event Types** pane then clicking on the right-pointing arrow. The Event Type appears in the right pane. Repeat this step as necessary.

When you move an Event Type into the **Selected Output Event Types** column, an option button appears next to the Event Type's name, in the **Default** column.

- 13 If more than one Event is defined as publishable, use the **Default** button to select the Event Type the current Collaboration Rules component returns. The default publication does not require an explicit call but occurs automatically by the normal execution of the Collaboration Rules.
- 14 To save your Collaboration Rules work in this tab of the **Collaboration Rules Properties** dialog box and enter your selections into the system, click **Apply**.
- 15 When finished configuring your Collaboration Rules component, click **OK** to close the properties dialog box, apply all the properties, and return to the e*Gate Enterprise Manager window.

Repeat the appropriate steps in the previous procedure until you have finished configuring all your Collaboration Rules.

Note: *The Collaboration-ID Rules Editor is a feature that enables compatibility with e*Gate Version 3.6 only. Normally, you do not need to use this GUI or its features. If necessary, see the e*Gate online Help for more information on this feature.*

4.6 Adding e*Ways and BOBs

In the e*Gate system, e*Ways are the most commonly used components for transporting and transforming data. They always interface with at least one external system, and Multi-Mode e*Ways can use e*Way Connections to interface with many external systems as well as with SeeBeyond JMS IQ Managers. See the specific *e*Way Intelligent Adapter User's Guide* for e*Way Connection information relating to your e*Way. BOBs are optional; a BOB can function as a type of internal e*Way to balance the load between IQs.

Since e*Ways and BOBs have similar functions, this section primarily covers e*Ways, but also mentions BOBs. In e*Gate you define all of these components in essentially the same way. For detailed information on how to define and edit all of these components, see [Chapter 9](#).

This section explains the basic procedures in creating and configuring e*Ways in your e*Gate system.

This section explains:

- [“e*Way Operation” on page 123](#)
- [“Before Creating an e*Way” on page 125](#)
- [“Creating e*Ways” on page 127](#)
- [“Configuring e*Ways” on page 128](#)
- [“Adding Business Object Brokers” on page 130](#)
- [“Adding Multi-Mode e*Ways” on page 131](#)

4.6.1 e*Way Operation

e*Ways have the following characteristics:

- **Adapting:** e*Ways face in two directions as they must interact with and adapt to external systems as well as communicate with e*Gate.
- **Transporting:** Acting as “smart” gateways, e*Ways direct the flow of data in and out of e*Gate.
- **Collaborating:** Inbound and outbound e*Gate Collaborations reside in e*Ways and form the core of their operation. They determine:
 - ♦ The routing (publishing/subscribing) of the Events they handle

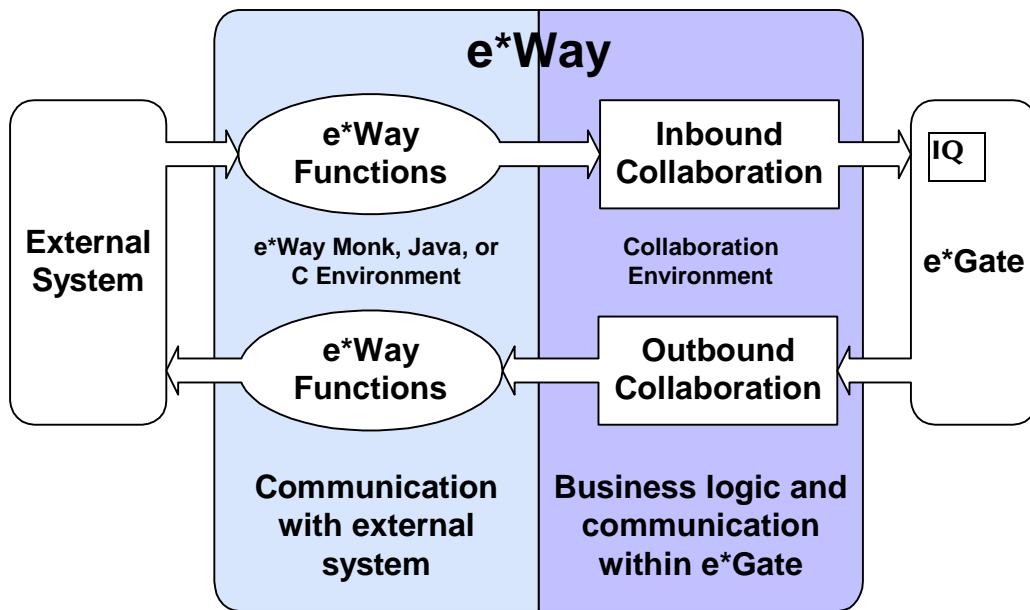
- Any transformation of data as it passes through the e*Way

In e*Gate, e*Ways interact with Collaborations as follows:

- Every e*Way requires at least one Collaboration, but it can have more than one.
- Every e*Way Collaboration that publishes internal e*Gate Events requires at least one IQ.
- Any e*Way Collaboration that only publishes to an external system does *not* require an IQ.

See “[System Design Components](#)” on page 83 for information on how e*Ways interact with Collaborations. Figure 47 shows a basic diagram of how an e*Way operates, using an e*Way with two Collaborations, an inbound and outbound.

Figure 47 e*Way Operation



Starting e*Ways: By default, none of the e*Gate components starts on its own. When you configure an e*Way, you can set it to start automatically, that is, at the same time as the Control Broker that controls it.

Note: See “[Adding Intelligent Queues](#)” on page 136 for more information on IQs.
See “[Adding Collaborations](#)” on page 142 for more information on Collaborations.

Once started, e*Ways operate as follows:

- They transform Events according to specific commands or simply move them through the system unchanged.
- They pass along Events as they are configured to do, and Events not immediately routed remain in IQs until requested.

4.6.2 Before Creating an e*Way

The Enterprise Manager allows you to create and configure e*Ways. Before starting, open all the levels in the **Participating Hosts** folder in the Navigator/Components pane.

To open all Participating Hosts' levels

- 1 Double-click the **Participating Hosts** folder in the Components Tree. The folder opens, displaying any contained hosts' icons. There must be at least one.

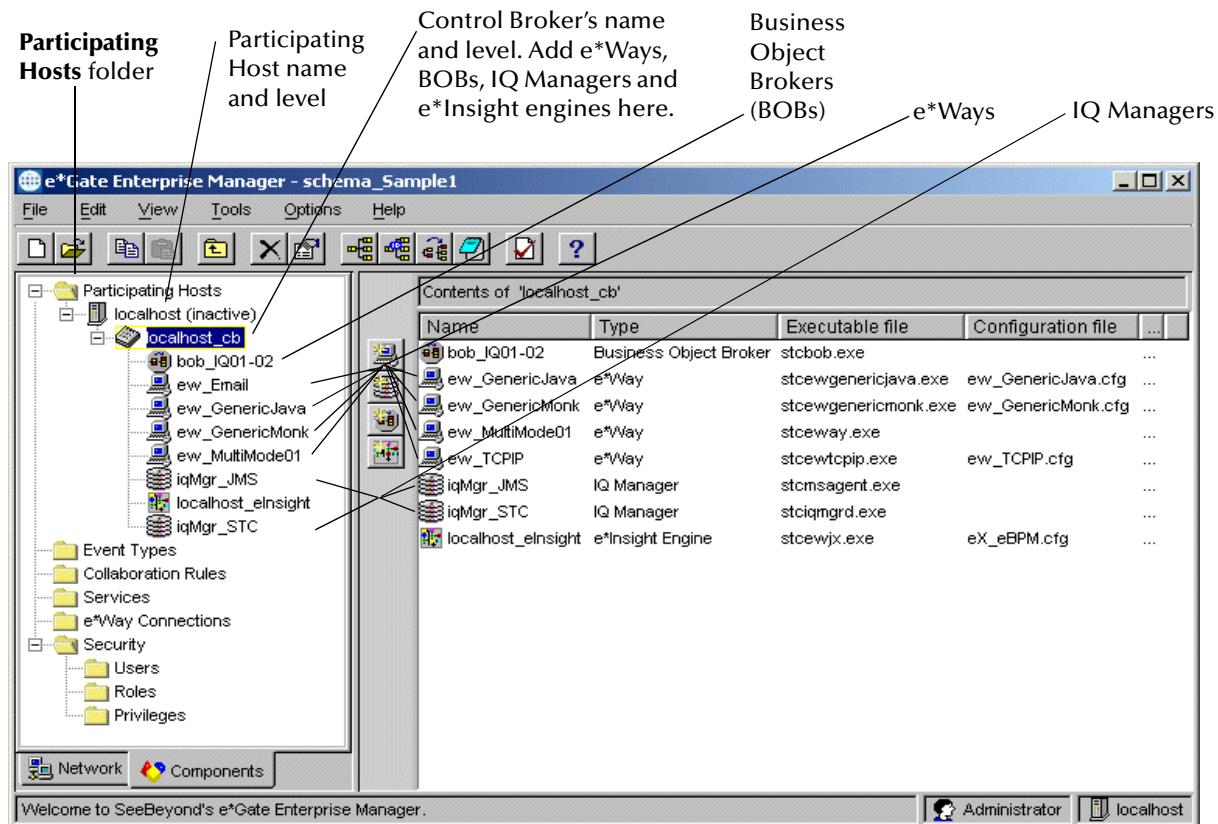
Note: *The contents of the selected Navigator pane component display in the Editor pane, for each level. Also, the Palette buttons are different for each level.*

- 2 Open the desired Participating Host in the same way. The folder displays the contained Control Broker's icon. There can only be one.
- 3 Open the Control Broker in the same way. The folder displays any contained IQ Managers (and IQs, if they have been created). There must be at least one IQ Manager.

Note: *For information on IQs and IQ Managers, see “[Adding Intelligent Queues](#)” on [page 136](#).*

Figure 48 on page 126 shows the Enterprise Manager window with all the Component Tree levels under the **Participating Hosts** folder open.

Figure 48 Contents of Participating Hosts Folder



Create and configure e*Ways, using features contained in the Control Broker level of the navigator tree.

Note: The e*Gate system names Control Brokers, using the convention,

<Control Broker name>_cb

See Figure 48 above for an example of this naming convention. By default, the system names the Control Broker after its Participating Host.

Explanation of Tree Levels

The **Participating Hosts** folder in the navigator tree contains different levels, allowing you to add the following components:

- Participating Hosts (see “[Adding New Participating Hosts](#)” on page 67)
- Control Brokers (see [Chapter 10](#))
- e*Ways and BOBs (see “[Creating e*Ways](#)” on page 127 and “[Adding Business Object Brokers](#)” on page 130)
- IQs and IQ Managers (see “[Adding Intelligent Queues](#)” on page 136)
- Collaborations (see “[Adding Collaborations](#)” on page 142)

Move From Higher to Lower

The components in the previous list display in this tree-layered setup because of their hierarchical relationship to one another. Components at higher levels of the Tree manage those at lower levels. You *must* add new components in the Tree by starting at the higher levels and moving downward.

Control Brokers

Control Brokers serve important system management functions in e*Gate as explained under “[Control Layer](#)” on page 35. You configure e*Way and BOB components at the Control Broker’s level in the Components Tree because Control Brokers directly manage the operation of these components.

System Files and e*Ways

Each e*Way depends on necessary component system files for its operation as follows:

- **Executable File:** Determines the mechanisms it uses to communicate with external systems. For example, the simplest e*Way is called *file-based* because it only reads and writes to a file. The name of its executable file is **stcewfile.exe**, and the system stores it in the following directory:

`\egate\Server\registry\repository\default`

Note: The e*Gate system comes supplied with basic executables like **stcewfile.exe**; see the **Standard e*Way Intelligent Adapters User’s Guide**. If you need specialized executables or files, see the appropriate SeeBeyond user’s guide for details.

- **Configuration File:** Contains exact parameters and values that vary based on the the e*Way’s operation requirements. You already have templates of e*Way configuration files available for you in e*Gate, but you have to take additional configuration steps using the e*Way Configuration Editor.

In these respects, BOBs operate in the same manner as e*Ways. For a more detailed explanation of e*Way executable and configuration files as well as how to create, configure, and edit e*Way (and BOB) requirements, see [Chapter 9](#). Also, make sure to follow the instructions in the user’s guide for the specific e*Way you are configuring.

4.6.3 Creating e*Ways

Use Enterprise Manager to create e*Ways. You can create them in any order.

To create new e*Ways

- 1 In the Navigator/Components pane, open the components tree as explained under “[To open all Participating Hosts’ levels](#)” on page 125.
- 2 Select the Control Broker.
- 3 On the Palette, click . The **New e*Way Component** dialog box appears.

- 4 Enter the desired name for the new e*Way and click **Apply** to enter it into the system. The new name and an e*Way icon appear in both panes.
- 5 Name additional e*Ways as needed. Click **Apply** after you name each one.
- 6 When you are finished, click **OK** to close the dialog box.

4.6.4 Configuring e*Ways

To finish adding e*Ways to your e*Gate system, you must configure them using the Enterprise Manager GUI.

To configure e*Ways

- 1 In Enterprise Manager, select one of the e*Ways you have already named.

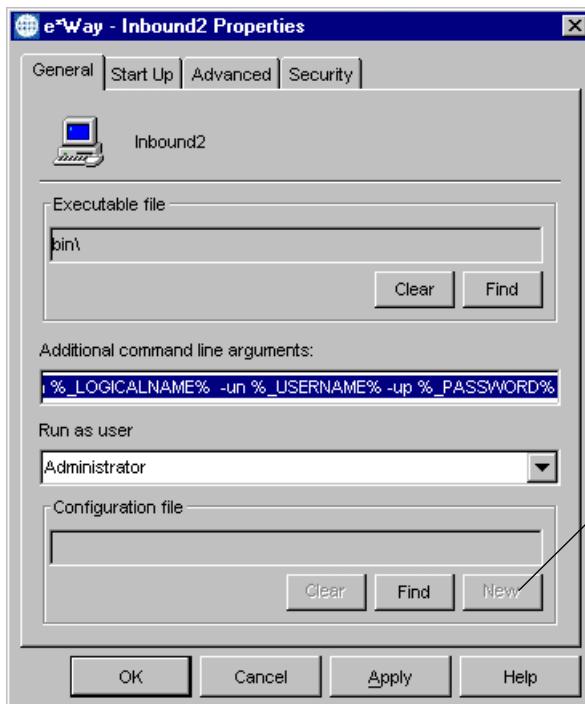
Note: You can select any e*Way since you do not have to configure them in any order.

- 2 On the toolbar or **Edit** menu, click  **Properties**.

The e*Way Properties dialog box (see Figure 49) opens to the **General** tab.

Note: You can only assign **Security** tab privileges if you are an Administrator user. If you are not logged in as "Administrator" you will not see the **Security** tab on this properties dialog box. For more information on this feature, see "["Security Tab" on page 93](#) or the [e*Gate Integrator System Administration and Operations Guide](#).

Figure 49 e*Way Properties Dialog Box, General Tab



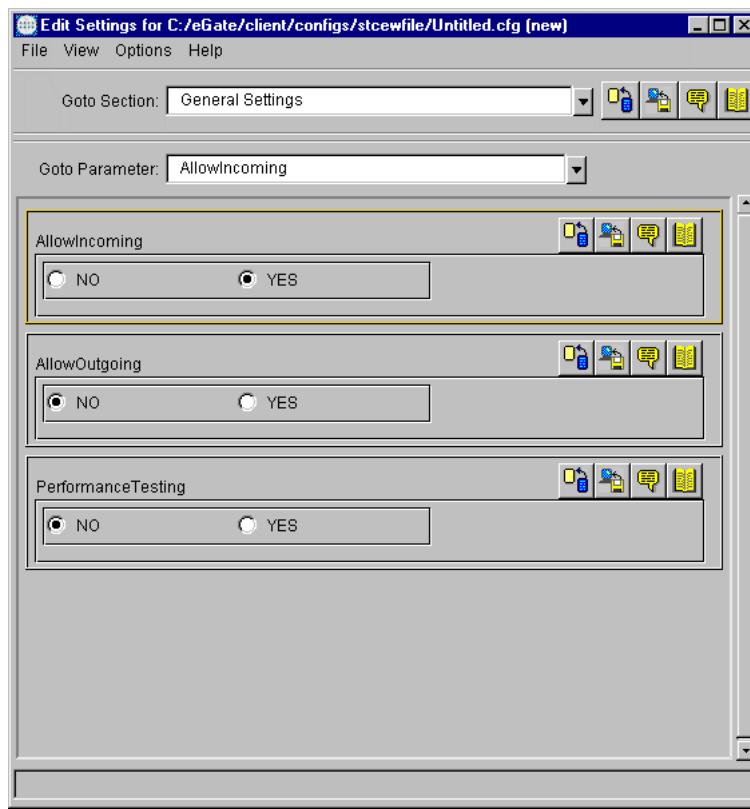
The **New** button becomes available after you choose the executable file.

- 3 Under **Executable file**, click **Find**. A **File Selection** dialog box appears.
- 4 Select the desired executable file and click **Select**. The dialog box disappears, and the chosen file name appears in the properties dialog box under **Executable file**.
For a complete list of available e*Way executable files and their functions, see [Chapter 9](#).

Note: Only enter *Additional command line arguments* if they are required by the current e*Way. See the appropriate e*Way user's guide for details.

- 5 Under **Run as user**, choose the appropriate user name, if necessary.
- 6 Under **Configuration File**, do one of the following actions:
 - If you already have a configuration file available, click **Find** to display a **File Selection** dialog box then select the desired file and click **Select**.
 - If you need to create a new configuration file, click **New** to display the e*Way Configuration Editor; see Figure 50. Use the Editor to configure the desired file.

Figure 50 e*Way Editor Window (New)



See [Chapter 9](#) for more information on how to use the e*Way Editor feature. See the appropriate e*Way Intelligent Adapter user's guide for details on how to use the e*Way Editor to configure a specific type of e*Way.

- 7 Continuing from the **e*Way Properties** dialog box, click the **Start Up** tab. The **e*Way Properties** dialog box changes (see [Figure 198 on page 455](#)), displaying the start-up options.
- 8 If desired, select **Start automatically**. This action configures the current e*Way to start whenever e*Gate activates, eliminating the need to start it manually.

Caution: To determine whether you want an e*Way to start automatically, see the instructions in the user's guide for that e*Way. The e*Way user's guide also gives additional information on specific e*Way configuration options.

- 9 To save any e*Way configuration, click **Apply** to enter it into the system.

Note: The **e*Way Properties** dialog box contains more features under the **Start Up** tab (see [Chapter 9](#)) and another tab, **Advanced** (see the [e*Gate Integrator Alert and Log File Reference Guide](#)) for details on these additional settings. You only need to change the additional features controlled under these tabs if you do not want to accept their displayed system defaults.

- 10 When finished configuring the current e*Way, click **OK** to close the properties dialog box.

4.6.5 Adding Business Object Brokers

You can add BOBs to your e*Gate system, using the Enterprise Manager GUI and procedures similar to those used for e*Ways. BOBs are an optional feature in e*Gate.

To create new BOBs

- 1 In the Navigator/Components pane, open the Components Tree to show the Control Broker's level as explained under "[To open all Participating Hosts' levels](#)" [on page 125](#).
 - 2 On the Palette, click .
- The **New Business Object Broker Component** dialog box appears.
- 3 Enter the desired name for a new BOB and click **Apply** to enter it into the system. The new name and a BOB icon appear in both panes.
 - 4 Name additional BOBs as needed, clicking **Apply** after you name each one.
 - 5 When finished, click **OK** to close the dialog box.

To configure BOBs

- 1 In the Navigator/Components or Editor pane, select one of the BOBs you have already named. You can select any BOB since you do not have to configure them in a specific order.
- 2 On the toolbar or **Edit** menu, click  **Properties**.

The **Business Object Broker Properties** dialog box appears, displaying the **General** tab. This feature configures the same properties as the **e*Way Properties** dialog box. See [Figure 49 on page 128](#).

- 3 Configure BOBs in the same way as you do e*Ways.
- 4 When finished, click **OK** to close the properties dialog box.

4.6.6 Adding Multi-Mode e*Ways

Multi-Mode e*Ways were added to the e*Gate system at Release 4.5. A Multi-Mode e*Way is a multi-threaded component used to route and transform data within e*Gate. Unlike BOBs or traditional e*Ways, Multi-Mode e*Ways can use multiple simultaneous e*Way Connections to communicate with several external systems, as well as IQs or JMS IQ Managers.

For detailed information on how to define and edit a Multi-Mode e*Way, see [Chapter 9](#). This section lists the basic steps for creating and configuring a Multi-Mode e*Way in your e*Gate system.

Before Creating a Multi-Mode e*Way

The Enterprise Manager allows you to create and configure Multi-Mode e*Ways. Before starting, open all the levels in the **Participating Hosts** folder in the Navigator/Components pane.

To open all Participating Hosts' levels

- 1 Double-click the **Participating Hosts** folder in the Components Tree. The folder opens, displaying any contained hosts' icons. There must be at least one.

Note: *The contents of the selected Navigator pane component display in the Editor pane, for each level. Also, the Palette buttons are different for each level.*

- 2 Open the desired Participating Host in the same way. The folder displays the contained Control Broker's icon. There can only be one.
- 3 Open the Control Broker in the same way. The folder displays any contained IQ Managers (and IQs, if they have been created). There must be at least one IQ Manager.

Note: *For information on IQs and IQ Managers, see “[Adding Intelligent Queues](#)” on [page 136](#).*

[Figure 48 on page 126](#) shows the Enterprise Manager window with all the Component Tree levels under the **Participating Hosts** folder open.

Creating and Configuring a Multi-Mode e*Way

You add a Multi-Mode e*Way to your e*Gate system using Enterprise Manager.

Caution: *While you are running a Multi-Mode e*Way, do not use the same machine to edit any ETD involved in the e*Way—if you do, you will be unable to save the ETD changes and unable to use normal methods to halt either the Editor or the e*Way.*

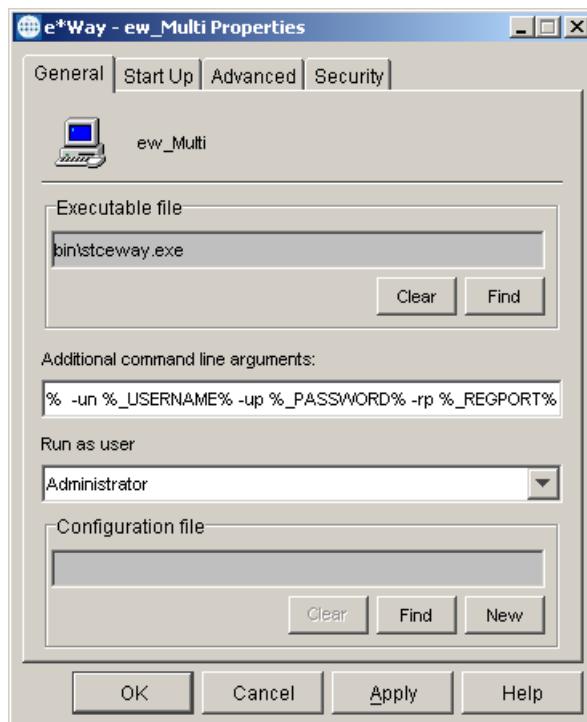
Before you begin: On the **Options** menu, point to **Options** and click **Java** to ensure that the correct editors are used.

To create and configure a new Multi-Mode e*Way

- 1 Select the Navigator's **Components** tab, open the Participating Host on which you want to create the Multi-Mode e*Way, and then open the Control Broker.
- 2 On the Palette, click on the icon **Create a New e*Way**.
- 3 Enter the name of the new e*Way in the **Name** text field and click **OK**.
- 4 Select the new component (if it is not already selected); then, on the toolbar or **Edit** menu, click  **Properties**. See Figure 51.

Note: For an in-depth description of what type of information the various fields require, see "[Multi-Mode e*Way](#)" on page 464.

Figure 51 Multi-Mode e*Way Properties Dialog Box

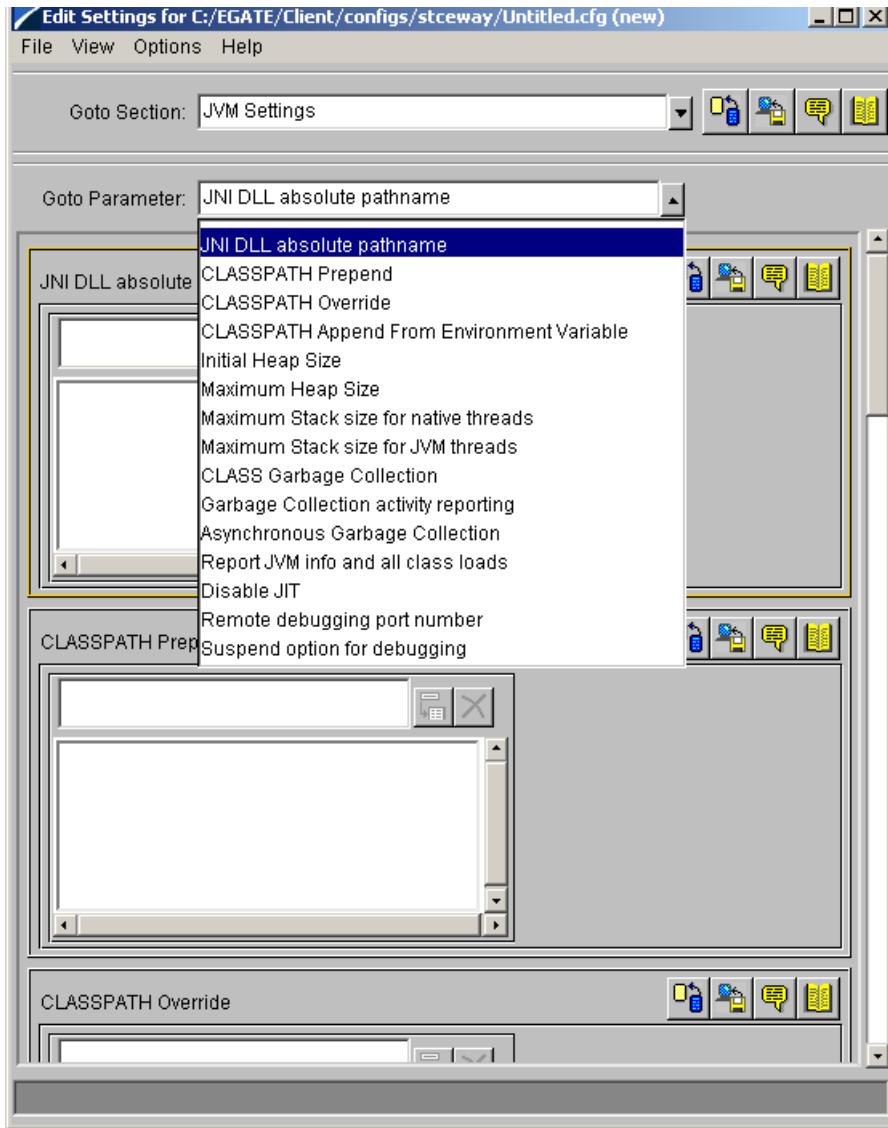


Note: You can only assign **Security** tab privileges if you are an Administrator. If you lack Administrator privileges, you will not see the **Security** tab on this properties dialog box. For more information on this feature, see "["Security Tab"](#) on page 93 or the *e*Gate Integrator System Administration and Operations Guide*.

- 5 Under the **Configuration file** text box, click **New**.

The **Edit Settings** dialog box is displayed. This is where you set the configuration parameters for the Multi-Mode e*Way configuration file. These parameters control the basic Java Virtual Machine (JVM) settings. See Figure 52.

Figure 52 Edit Settings Dialog Box – Multi-Mode e*Way



- 6 Set the configuration parameters for this configuration file. For general information regarding what data to enter in these fields, see ["JVM Settings" on page 482](#).
- 7 After entering the parameters: On the **File** menu, click **Close**; then, in the **Save As** dialog box, click **Save**.
- 8 When you return to the **e*Way Properties** dialog box, click **OK** to save your changes and close the dialog box.

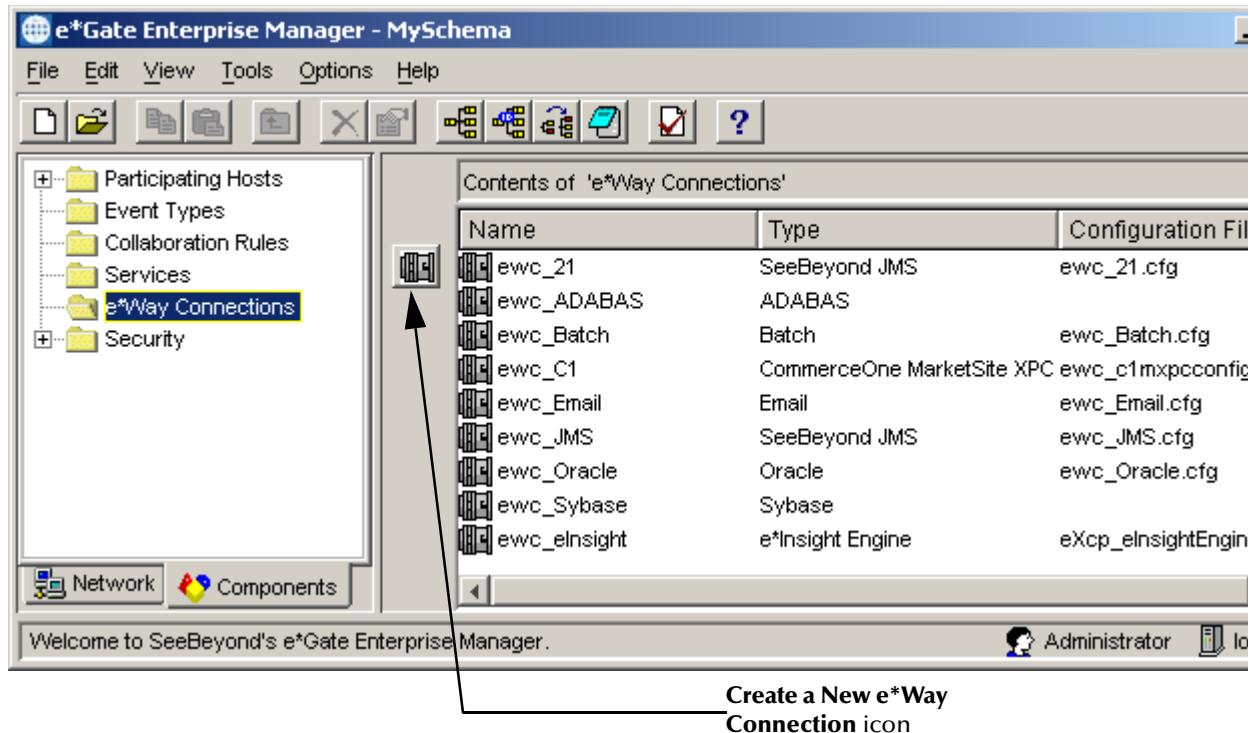
4.6.7 Adding e*Way Connections

An e*Way Connection is the encoding of the access information for one particular external connection. In terms of content, it is similar to an e*Way configuration file, in defining enough information to be able to log in to the particular system.

To create and configure e*Way Connections

- 1 Select the **e*Way Connections** folder in the Components pane of the Enterprise Manager. See Figure 53.

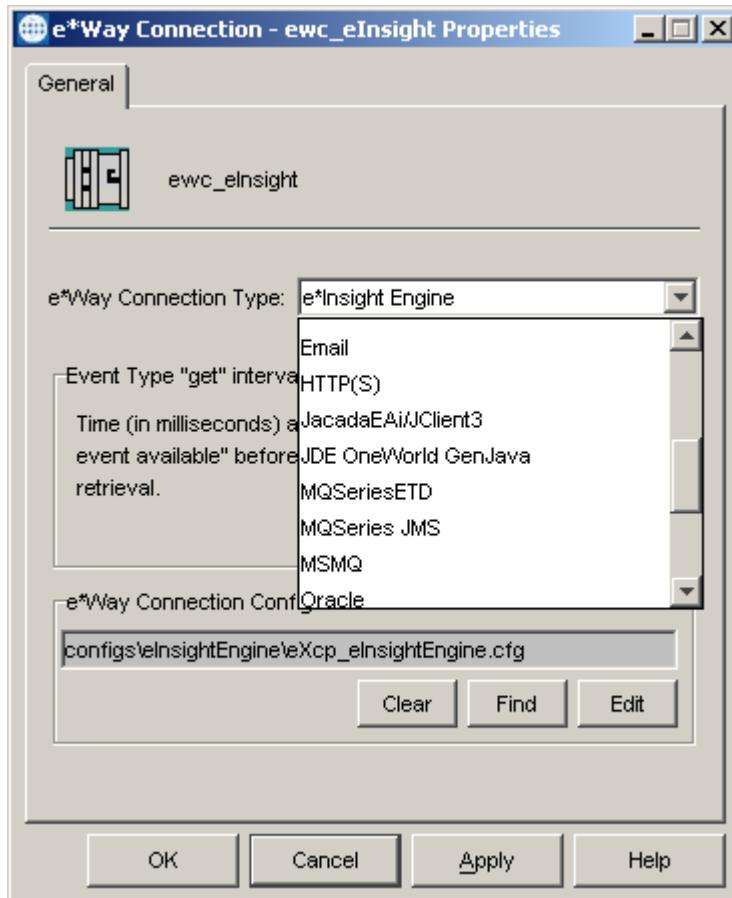
Figure 53 *Gate Enterprise Manager: **Create New e*Way Connection**



- 2 On the palette, click the **Create a New e*Way Connection** icon.
- 3 Type a name for the new e*Way Connection in the **Name** text box, then click **OK**.
- 4 Open the properties dialog box for your new e*Way Connection. Make sure the new e*Way Connection name is highlighted in the Editor pane and then, on the toolbar or **Edit** menu, click **Properties**.

The **e*Way Connection** properties dialog box opens. See Figure 54.

Figure 54 e*Way Connection Properties Dialog Box



- 5 From the **e*Way Connection Type** list, choose the appropriate selection.

Important: Consult the appropriate user's guide for explanations on the various sections and configuration parameters required for the different types of e*Way Connections.

- 6 Enter how long you want to wait before performing another "get" operation when the IQ is empty in the **Event Type "get" interval** text box. Especially for SeeBeyond JMS IQs, you should consult the user's guide to be sure you are using values appropriate for your needs.
- 7 Under the **e*Way Connection Configuration File** pane, do one of the following:
 - A To use an existing configuration file, click **Find**, navigate to the folder that contains the file, and then select the configuration file you want.
 - B To create a new configuration file from scratch, click **Clear** (if necessary to de-reference an existing configuration file), and then click **New**.
- 8 Make the appropriate selections in the **Edit Setting for <directory><filename>.cfg** dialog box.

Important: The information entered in the fields on the **e*Way Connection Properties** dialog box is specific to the type of external system you are connecting to. Make sure you

see the appropriate e*Way user's guide (or the *SeeBeyond JMS IQ User's Guide*) for specifics about what information should be entered in each field.

- 9 On the **File** menu, click **Save As**.

The **Save As** dialog box lists the directory where the e*Way Connection will be stored along with the file name for the e*Way Connection. The default is the name you originally set. When you are satisfied this information is correct, click **Save**.

- 10 Close the **Edit Settings for <directory><eWay-Connection-name>.cfg** dialog box.
- 11 In the **e*Way <name of e*Way Connection> Properties** dialog box, click **OK**.

4.7 Adding Intelligent Queues

IQs are components that provide nonvolatile storage for Events within the e*Gate system as they pass from one component to another. When setting up IQs:

- Unless you are using SeeBeyond JMS IQ Manager, you must create at least one IQ per schema for published Events within the e*Gate system. Create more, depending on the needs of your system.
- All BOBs require at least one IQ.
- e*Ways that publish Events externally do not need IQs.
- IQ Services provide the mechanism for storing Events as they move among IQs and Collaborations. They also handle the low-level implementation of data exchange, for example, system calls to initialize or reorganize a database.
- Each schema must have an IQ Manager before you can add any IQs to it. See “[IQ Managers](#)” on page 137.

Intelligent Queuing: IQs are *intelligent* in that they are more than just a holding tank for Events. They actively record information about the current state of Events. See [Figure 47 on page 124](#) for a diagram of how IQs fit into the e*Gate/e*Way setup. IQs operate with BOBs and Multi-Mode e*Ways in the same way as they do with e*Ways.

For more information on how to add and configure IQs and IQ Managers, see the *e*Gate Integrator System Administration and Operations Guide*. See the *e*Gate Integrator Intelligent Queue Services Reference Guide* and the *SeeBeyond JMS Intelligent Queue User's Guide* for complete information on working with IQs.

IQ Services

You use an **IQ Properties** dialog box to configure IQs in your system (see [Figure 55 on page 138](#)). This properties dialog box offers you a list of the IQ Services available in your system. You can choose one of these Services to configure with an IQ, or you can choose **Undefined**. The IQ Services available to you depend on the IQs installed in your e*Gate system. Three IQ Services are always available:

- **STC_JMS_IQ**—SeeBeyond JMS IQ Service. Service library: **stc_iqms.dll**. This is the only service library possible for IQs managed by a SeeBeyond JMS IQ Manager (in other words, IQs under an IQ Manager whose executable is **stcmsagent.exe**).

- **STC_Standard**—See Beyond Standard IQ Service. Service library: **stc_iqstandard.dll**.
- **STC_Memory_Loopback**—See Beyond Memory Loopback IQ Service. Service library: **stciqloopback.dll**.

You could require another type of Service with a given IQ (or **Undefined**), depending on the e*Way with which the IQ operates. See the appropriate e*Way user's guide for details. For more information on IQ Services, see the *e*Gate Integrator Intelligent Queue Service Reference Guide* and the *SeeBeyond JMS Intelligent Queue User's Guide*.

This section explains:

- [“IQ Managers” on page 137](#)
- [“Working With IQs” on page 137](#)
- [“Attaching IQs” on page 142](#)

4.7.1 IQ Managers

In e*Gate, IQ Managers manage interfaces between IQs and e*Ways. They also do routine operations to oversee IQs: for example, reorganizing them, archiving queue information, and locking queues for maintenance. IQs in your system require at least one IQ Manager.

To create IQ Managers

- 1 In the Navigator/Components pane, open the Components Tree as explained under [“To open all Participating Hosts’ levels” on page 125](#).
- 2 Select the Control Broker’s level.
- 3 On the Palette, click . The **New IQ Manager Component** dialog box appears.
- 4 Enter the desired name for a new IQ Manager and click **Apply** to enter it into the system. The new name and an IQ Manager icon appear in both panes.
- 5 Name as many additional IQ Managers as you want, clicking **Apply** after you name each one.
- 6 When you are finished, click **OK** to close the dialog box.

For more information on IQ and IQ Manager operation, configuration, and maintenance, see the *e*Gate Integrator Intelligent Queue Services Reference Guide* and the *SeeBeyond JMS Intelligent Queue User's Guide*. To configure IQ Managers to attach IQs, see [“Attaching IQs” on page 142](#).

4.7.2 Working With IQs

Add IQs using the Enterprise Manager window. To add them, you must create and configure them.

To create IQs

- 1 In the Navigator/Components pane, open the Components Tree as explained under [“To open all Participating Hosts’ levels” on page 125](#).

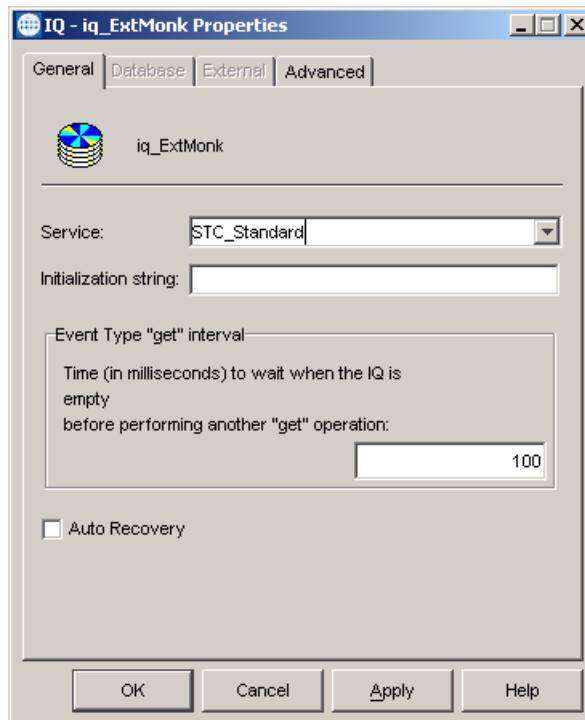
- 2 Select the IQ Manager's level.
- 3 On the Palette, click . The **New IQ Component** dialog box appears.
- 4 Enter the desired name for a new IQ and click **Apply** to enter it into the system. The new name and an IQ icon appear in both panes.
- 5 Name additional IQs as needed, clicking **Apply** after you name each one.
- 6 When you are finished, click **OK** to close the dialog box.

To configure IQs

- 1 In either the Navigator/Components or Editor pane, select one of the IQs you have already named. You can select any IQ since you do not have to configure them in any order.
- 2 On the toolbar or **Edit** menu, click  **Properties**.

The **IQ Properties** dialog box appears, displaying the **General** tab. For example, Figure 55 shows the general properties of the STC_Standard IQ.

Figure 55 STC_Standard IQ Properties Dialog Box, **General** Tab



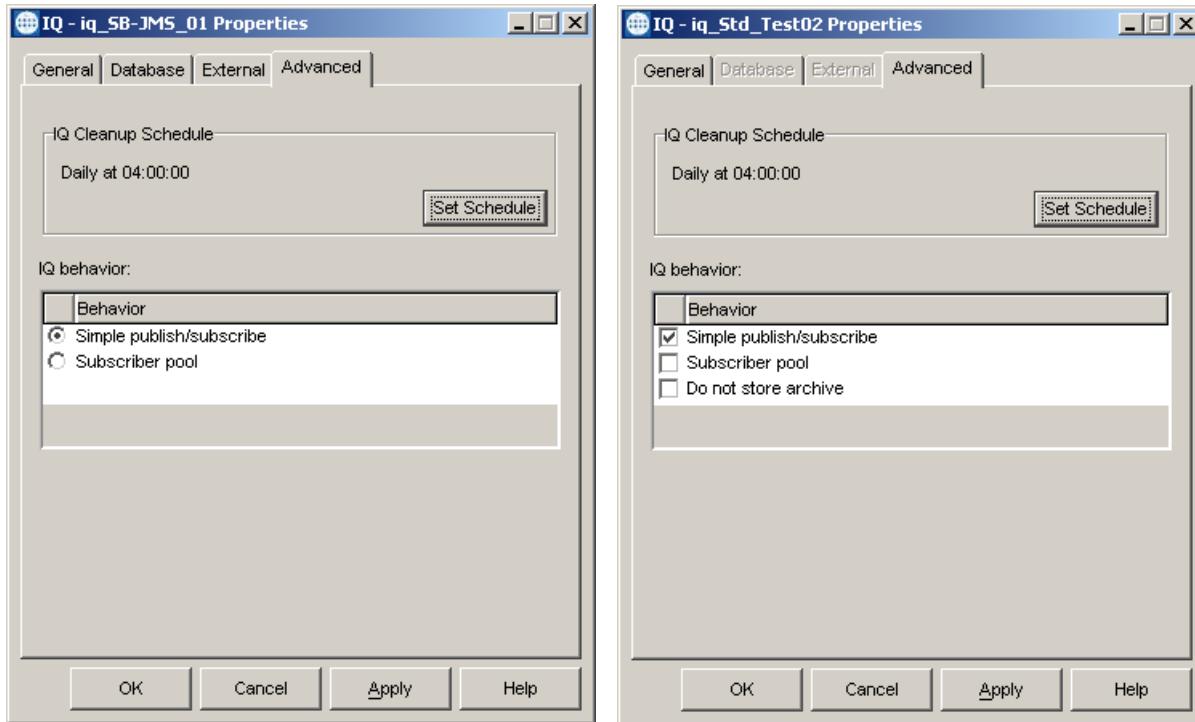
Note that the **External** or **Database** tabs are active only if the **External Service** or **Database** check box was selected on the **IQ Service Properties** dialog box. For more information about these tabs, see the *e*Gate Integrator Intelligent Queue Services Reference Guide*.

- 3 As necessary, enter the following parameters under the **General** tab in the properties dialog box:
 - **Service:** For IQs managed by the SeeBeyond JMS IQ Manager, the service must be **STC_JMS_IQ**; for IQs managed by the SeeBeyond Standard IQ Manager, choose one of the available service. Most systems using the Standard IQ Manager use its default service, **STC_Standard** (SeeBeyond Standard IQ Service); the service defines the IQ (see "[IQ Services](#)" on page 136).

- **Initialization String:** An optional command string available for the STC_Standard only. An entry here is optional. For information on valid parameters, see the *e*Gate Integrator Intelligent Queue Services Reference Guide*.
 - **Event Type “get” interval:** Enter the appropriate interval, the amount of time (in milliseconds) between **get** operations when no Events are stored in the IQ.
- 4 When finished setting the **General** properties, click **Apply** to enter your settings into the system.
 - 5 Click the **Advanced** tab.

The properties dialog box changes (see [Figure 56 on page 139](#)), displaying advanced properties.

Figure 56 IQ Properties Dialog Box, Advanced Tab



- 6 As necessary, enter the following parameters under the **Advanced** tab in the properties dialog box:
 - **IQ Cleanup Schedule:** Click **Set Schedule** to enter a cleanup schedule other than the default. This action opens a dialog box, allowing you to enter a new time and interval.
 - **IQ behavior:** Choose one of the following options
 - **Simple publish/subscribe**
 - **Subscriber pool**
 - For IQs that do not use the STC_JMS_IQ service only: If appropriate, select the **Do not store archive** check box.

Note: *Subscriber pooling allows you to change an Event's status based on the activities of any one of a number of available subscribers. For more information, see either the **JMS IQ User's Guide** (for the SeeBeyond JMS implementation) or the **SeeBeyond eBusiness Integration Suite Deployment Guide**. The **e*Gate Integrator System Administration and Operations Guide** has additional information on some configuration parameters (under the explanation of the **stciqutil** command).*

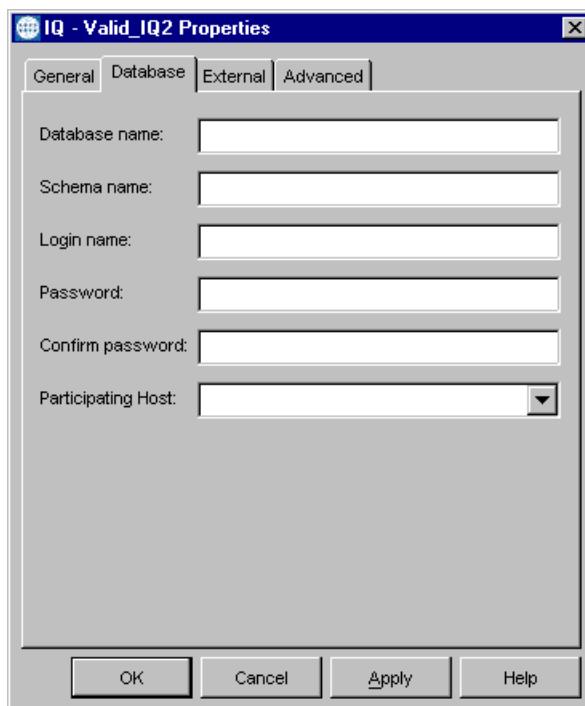
- 7 When finished setting the **Advanced** properties, click **Apply** to enter your settings into the system.

Note: *You only need to configure database parameters if the current IQ is using a database. For the **Database** tab to be active, the **Database** check box must be selected on the **IQ Service Properties** dialog box. For more information about these tabs, see the **e*Gate Integrator Intelligent Queue Services Reference Guide**.*

- 8 Click the **Database** tab.

Use the **IQ Properties** dialog box, **Database** tab, to configure database properties if necessary. See Figure 57.

Figure 57 IQ Properties Dialog Box, Database Tab



- 9 As necessary, enter the following parameters under the **Database** tab in the properties dialog box:
 - ♦ **Database name:** Enter the name of the database the IQ is using.
 - ♦ **Schema name:** Enter the name of the schema that contains the current IQ.

- ♦ **Login name:** Enter the login name for the Participating Host that contains the current IQ.
- ♦ **Password:** Enter the password associated with the current login name given above.
- ♦ **Confirm password:** Enter the same password again; you must type it in the same way as it was in the **Password** text box.
- ♦ **Participating Host:** Select the name of the Participating Host that contains the current IQ.

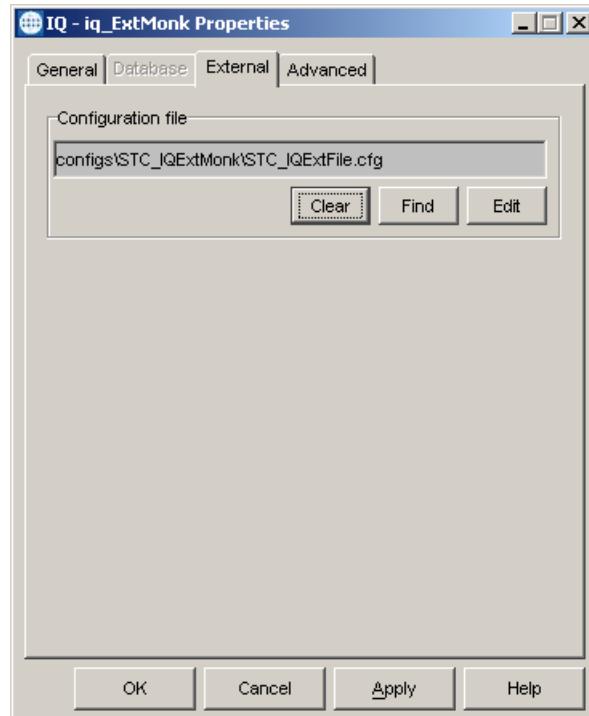
For more information on IQs and the databases they use in the e*Gate system, see the *e*Gate Integrator Intelligent Queue Service Reference Guide*.

Note: You only need to configure database parameters if the current IQ is using a database. For the **External** tab to be active, the **External Service** check box must be selected on the **IQ Service Properties** dialog box. For more information about these tabs, see the *e*Gate Integrator Intelligent Queue Services Reference Guide*.

10 Click the **External** tab.

Use the **IQ Properties** dialog box, **External** tab, to extend the capabilities of e*Gate to allow for transactional behavior between external systems and e*Gate IQs, which in turn allows for guaranteed delivery and nonduplication of Events. See Figure 58.

Figure 58 IQ Properties Dialog Box, External Tab



11 If there are **.def** files in the **configs/<iqservicename>** directory, the **Find** button is active. Use it to locate the desired **.def** file. The **New** button is active when you are

able to create a new configuration file; when a configuration file has already been defined, this button becomes the **Edit** button instead.

Note: For more information on the behavior of this configuration file, see [Chapter 9](#), as this configuration file functions just like an e*Way configuration file.

- 12 To save any IQ configuration, click **Apply** to enter it into the system.
- 13 When finished, click **OK** to close the properties dialog box.

4.7.3 Attaching IQs

By default, none of the e*Gate components starts on its own. When an IQ Manager starts, it then attaches its component IQs. When you configure an IQ Manager, you set it to start automatically, that is, at the same time as the e*Gate system.

To configure the IQ Manager to start automatically

- 1 In the Navigator, select the IQ Manager.
- 2 On the toolbar or **Edit** menu, click  **Properties**.
- 3 Click the **Start Up** tab, check the **Start automatically** box, and click **OK**.

4.8 Adding Collaborations

After you have added the necessary e*Ways, BOBs, and IQs to your e*Gate system, you can then add its Collaborations. A Collaboration is the component within an e*Way or BOB that performs data transformation and routing. If required, these components transform data by executing Collaboration Rules (see [“Creating Collaboration Rules and Scripts” on page 106](#)).

You must associate a Collaboration Rules component with each Collaboration. Collaborations receive and process Event Types to forward the output to other e*Gate components. This is a two-sided process:

- The **Subscriber** is the half of the Collaboration that continually checks for and receives Events of known types (Event Types).
- The **Publisher** is the half of the Collaboration that places processed Events in an IQ under a specific Event Type name.

This section explains:

- [“Collaboration Setup” on page 143](#)
- [“Creating Collaborations” on page 143](#)
- [“Configuring Collaborations” on page 144](#)
- [“Troubleshooting Collaborations” on page 147](#)

4.8.1 Collaboration Setup

When you set up Collaborations, use the following rules:

- You cannot assign the same Collaboration to more than one component (e*Way or BOB).
- You must associate one and only one Collaboration Rules component with each Collaboration.
- Configure Collaborations in publication order. In other words: Begin configuring the inbound Collaborations first, and then configure the Collaborations that the previous ones publish to, and so on.
- Collaborations can subscribe to: other specified Collaborations; e*Way Connections; all Collaborations <ANY>; or external systems <EXTERNAL>. But Collaborations can *never* to IQs.
- Collaborations can publish to: IQs; e*Way Connections; or external systems.

4.8.2 Creating Collaborations

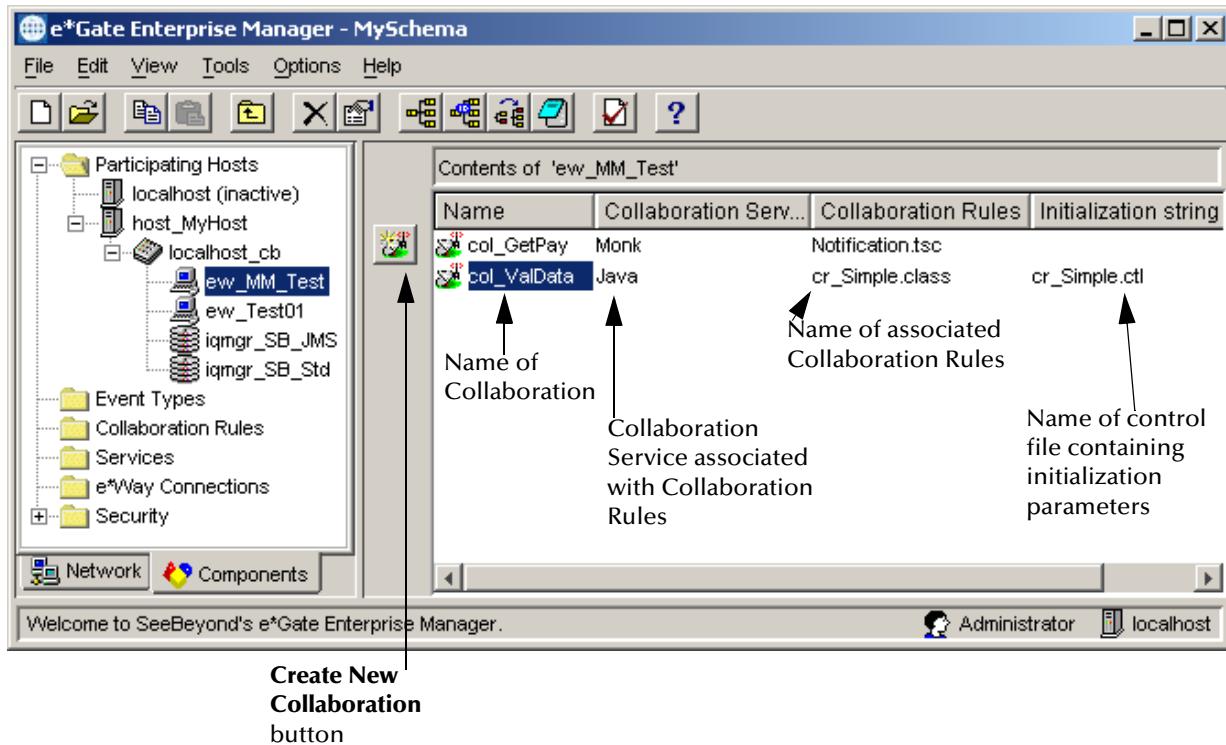
As with e*Ways, BOBs, and IQs, to add Collaborations you must first create them and then configure them. You can create Collaborations in any order, using the Enterprise Manager GUI.

To create Collaborations

- 1 In the Navigator/Components pane, open the Components Tree as explained under ["To open all Participating Hosts' levels" on page 125](#), to display the Control Broker level.
- 2 Select the specific e*Way or BOB where you want to assign the current Collaboration in the Navigator/Components pane.
- 3 On the Palette, click . The **New Collaboration Component** dialog box appears.
- 4 Enter the desired name for a new Collaboration and click **Apply** to enter it into the system.

After you have named each Collaboration, its name and a Collaboration icon appear in the Editor pane. See Figure 59.

Figure 59 Enterprise Manager with Collaboration



Note: The **Collaboration Service**, **Collaboration Rules**, and **Initialization string** columns contain information about the Collaboration Rules you associate with the current Collaboration. This information appears here after you configure the Collaboration.

- 5 Name as many additional Collaborations as you want, clicking **Apply** after you name each one.
- 6 When finished, click **OK** to close the dialog box.

4.8.3 Configuring Collaborations

Configure Collaborations in publication order as explained under “[Collaboration Setup](#)” on page 143. To do this operation, use the Enterprise Manager GUI.

Note: Configuring Collaborations in publication order is not mandatory, but it is recommended for ease of setup.

To configure Collaborations

- 1 Make sure the Navigator/Components and Editor panes display the Control Broker level of the Components Tree.
- 2 In the Navigator/Components pane, select the e*Way or BOB that contains the desired Collaboration.

3 In the Editor pane, select the desired Collaboration.

4 On the toolbar or **Edit** menu, click  **Properties**.

The **Collaboration Properties** dialog box appears, displaying the **General** (the dialog box's only) tab. See [Figure 60 on page 146](#) for an example of the **Collaboration Properties** dialog box with elements selected.

5 Select the desired properties for the current Collaboration as follows:

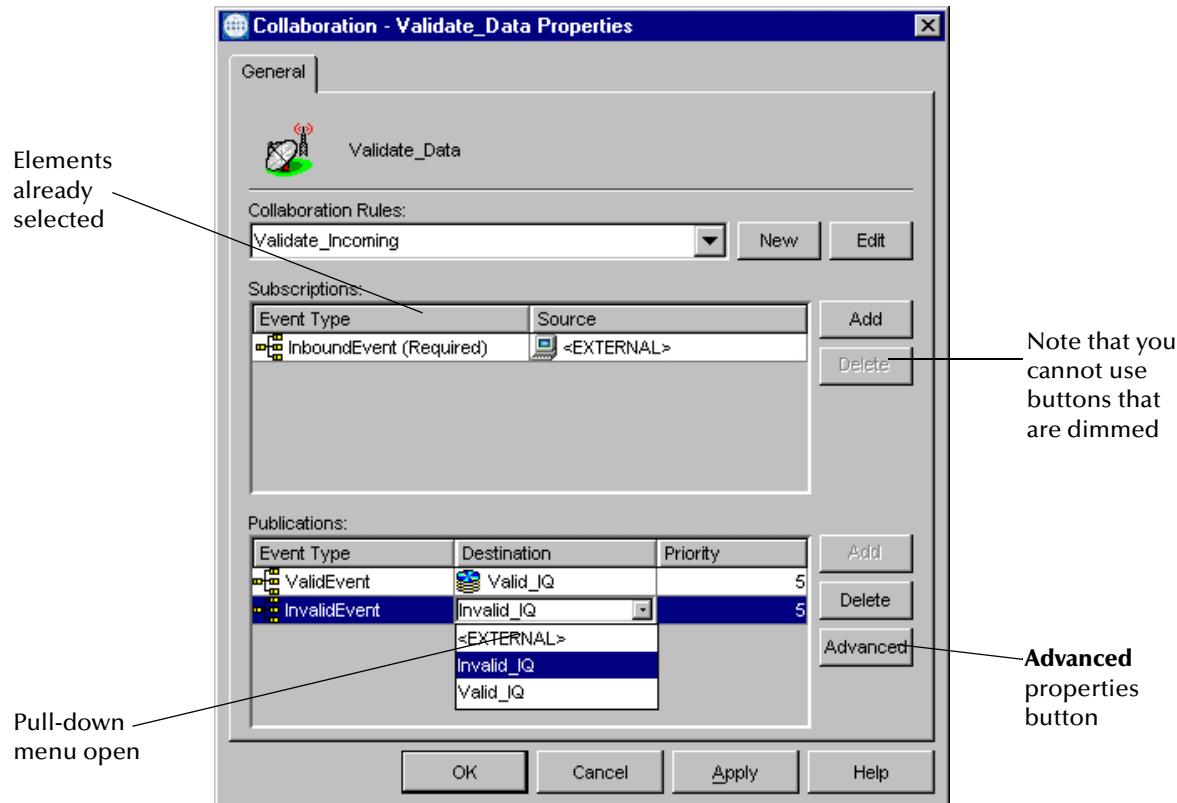
- ♦ For **Collaboration Rules**, select the desired Collaboration Rules component (see [“Creating Collaboration Rules and Scripts” on page 106](#)) for the current Collaboration. The **New** and **Edit** buttons allow you to create a new Collaboration Rules component here or edit an existing one, using the Collaboration Rules Editor.

Note: See [Chapter 7](#) and [Chapter 8](#) for explanations of how to use the Java and Monk Collaboration Rules Editor GUIs.

- ♦ Under **Subscriptions**, select the subscription properties for the current Collaboration as follows:
 - ♦ Click **Add** to make the **Event Type** and **Source** columns active pull-down menus.
 - ♦ Use the menus to choose the Event Type you want the Collaboration to subscribe to and the source of that Event Type.
- ♦ Under **Publications**, select the publication properties for the current Collaboration as follows:
 - ♦ Click **Add** to make the **Event Type**, **Destination**, and **Priority** columns active pull-down menus.
 - ♦ Use the menus to choose the Event Type you want the Collaboration to publish, the destination for that Event Type, and a number representing the priority (the default is 5; for complete information on Event Type priority, see [“About Event Type Priority” on page 146](#)).

Figure 60 shows an example of the **Collaboration Properties** dialog box with elements selected.

Figure 60 Collaboration Properties Dialog Box with Selections



- Click **Advanced** to open an additional properties dialog box, allowing you to configure recovery, audit, and expiration properties for the current publication setup.

- To save any Collaboration configuration, click **Apply** to enter it into the system.
- When finished configuring the current Collaboration, click **OK** to close the properties dialog box.

About Event Type Priority

By default, all Event Types are assigned the same priority, 5; this causes the destination IQ or external system processes all Events in the order they were received. However:

- If Events of a certain Event Type are urgent and you want them to “go to the head of the line” for rush processing, set Event Type Priority to a positive integer less than 5.
- If Events of a certain Event Type can be handled with low urgency, waiting until high-urgency and normal-urgency Events have been processed, set the Event Type Priority to any integer greater than 5. Priority 999,999,999 is the least urgent.

All Events of the same Event Type have the same priority. The rules for retrieving and processing prioritized Events are as follows:

- Events of priority 1 are retrieved before all others.
- Events of priority n are retrieved only after retrieving all Events of priority $m < n$.
- Events of the same priority are retrieved in the order in which they were received.

4.8.4 Troubleshooting Collaborations

If you find you are unable to configure e*Gate components quickly and easily, for example, if you cannot publish to the correct IQ or subscribe to the correct Event, one of two things can have happened:

- You have not created all the necessary components. For example, if you are configuring a Collaboration and cannot publish to the desired IQ, check to see whether you have created that IQ.
- You are not configuring the Collaborations in “downstream” or publication-first order. A Collaboration cannot subscribe to an Event Type that is not yet being published, except for <ANY> Event Type. For the best results, always create publishers *before* you create subscribers.

You can avoid the need to subscribe in publication-first order by configuring subscribers to subscribe to <ANY> Event Type. Check your system design before you use this option. Use this feature (or not) depending on the intent of your overall configuration.

For more information on troubleshooting your system, see the *e*Gate Integrator Alert and Log File Reference Guide*.

4.9 Reviewing and Testing the System

Before you use your new e*Gate system in production, make sure you run it in test situations to ensure smooth and correct system operation. This section offers some basic advice for how to begin the review and testing operation. For complete troubleshooting procedures, see the *e*Gate Integrator Alert and Log File Reference Guide*.

To review your configuration, go through the entire schema and make sure that *every* component is configured correctly, especially Collaborations and IQs, which are not immediately evident in the Navigator pane.

Most e*Gate system components require configuration after creation and do not operate until they are configured. Your e*Gate system does not run correctly until *all* components are configured.

To start your schema

- 1 Type the following text at the command line:

```
stccb -rh host -rs schema -un username -up password  
-ln control broker name
```

- 2 Press ENTER.

Note: For more information on the command line, see the *e*Gate Integrator System Administration and Operations Guide*.

Starting the Control Broker for a schema puts the schema into operation. To confirm that the schema is running, start the e*Gate Monitor and open the same schema.

If information on schema operation appears in the e*Gate Monitor window, the schema is up and running.

For information on how to set up and configure the Control Broker for a schema, see “[Control Broker Setup](#)” on page 87. For more information on the e*Gate Monitor feature, see [Chapter 10](#). For a complete explanation of how to use the e*Gate command line, see the *e*Gate Integrator Alert and Log File Reference Guide*.

4.9.1 Post System Setup Troubleshooting

If there appears to be a problem, do the following actions:

- Double-check to make sure you have created and configured all the components as directed in this chapter.
- Use the e*Gate Monitor to confirm that all the components are running; manually start any that did not start correctly (see [Chapter 10](#) for more information on the e*Gate Monitor). For more information on starting components, see the *e*Gate Integrator System Administration and Operations Guide*.
- If a component starts successfully but halts immediately, the most likely cause is faulty configuration. Check the following components:
 - ◆ Does each e*Way have an executable and a configuration file defined?
 - ◆ Does each Event Type have an ETD file assigned?
 - ◆ Does the appropriate Collaboration Rules component have the corresponding Monk service assigned (if necessary), and the correct Collaboration Rules files both created and assigned?
 - ◆ If you are not using Collaboration Rules, do the Monk, Java, and/or C language scripts you have created test out correctly and have the appropriate Collaboration Service(s) assigned?
 - ◆ Are the IQ Managers running, and did you configure all IQs correctly?
- If all the components start and stay running, the most likely problem is that the Collaboration Rules script used in one or more e*Way Collaborations is not working correctly. Use the Monk Test Console (see the next section) to check that you have created these files in the right way.
- Check the appropriate log files. Log files are text files that contain a record of all actions taken by an e*Way (or other module). For complete information on troubleshooting using e*Gate system log files, see the *e*Gate Integrator Alert and Log File Reference Guide*.

For more information on system testing and troubleshooting after development, setup, and transition to production, see the *Deployment Guide*.

4.9.2 Java Interactive Debugger

The e*Gate Monitor provides access to an in-schema debugging tool for Multi-Mode e*Ways that allows you to control execution, set and clear breakpoints, go to a specific statement, step through code, stop in a specific class or method, break on a specific

exception, and so forth. It can debug multiple Collaborations running in different threads. For complete information, see “[e*Gate Java Debugger](#)” on page 505.

4.9.3 Monk Test Console

The Monk Test Console enables you to test Monk functions and Collaboration Rules scripts directly.

Note: Using this feature correctly can require some knowledge of Monk programming. See the [Monk Developer’s Reference](#) for details.

To access the Monk Test Console

In Enterprise Manager, on the **Tools** menu, click  **Monk Test Console**. The Monk Test Console window appears.

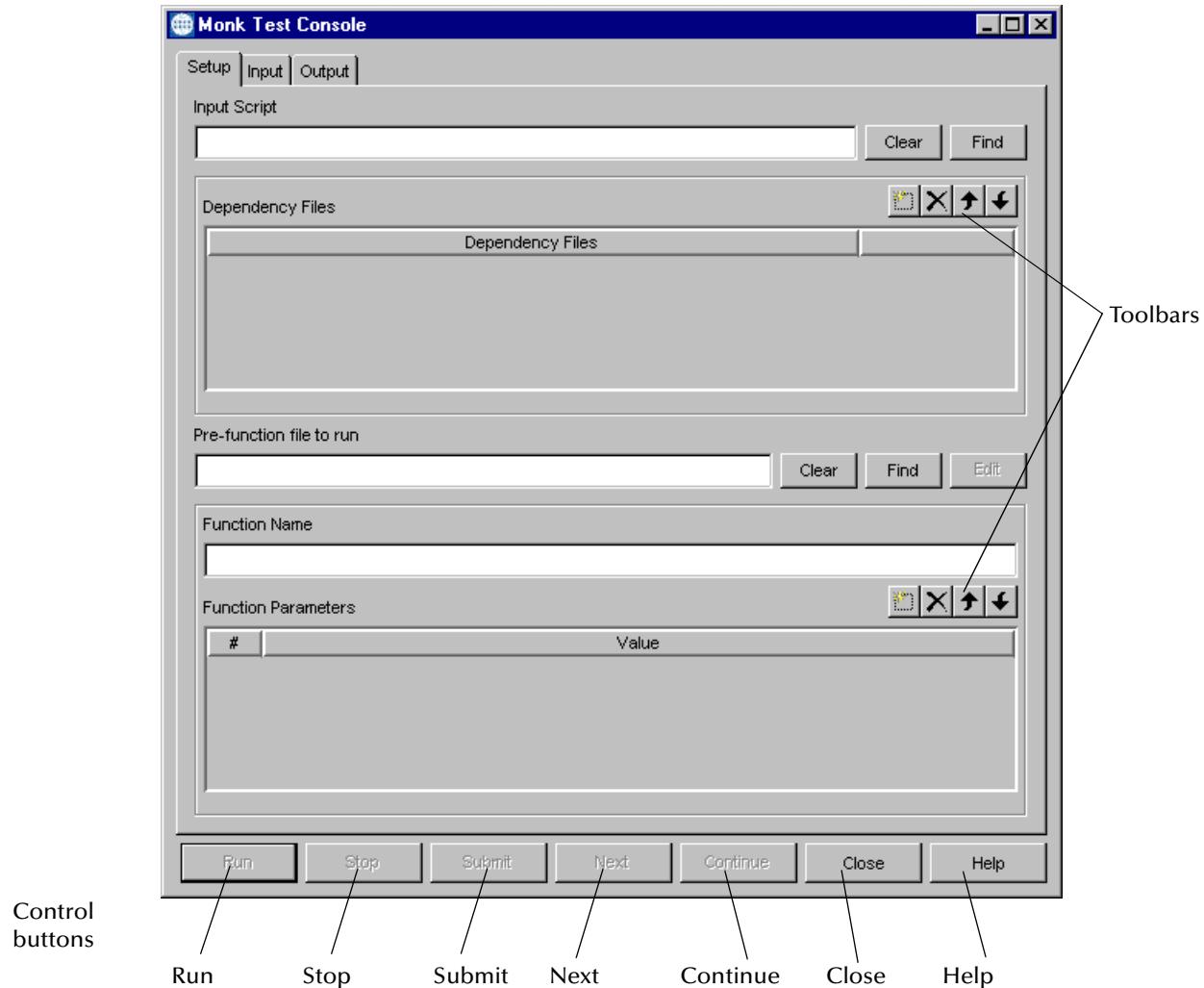
Monk Test Console Tabs

The Monk Test Console window contains the following tabs:

- **Setup:** Specifies the name of the file that contains the script or function to be tested.
- **Input:** Specifies the source of any required input data.
- **Output:** Displays the output of the script/function you are testing.

By default, the **Setup** tab window displays first. Figure 61 below through [Figure 63 on page 152](#) show the Monk Test Console window, using one figure for each of the three window tabs.

Figure 61 Monk Test Console, Setup Tab



Note: Names appear on the control buttons when their functions become available.

Figure 62 Monk Test Console, Input Tab

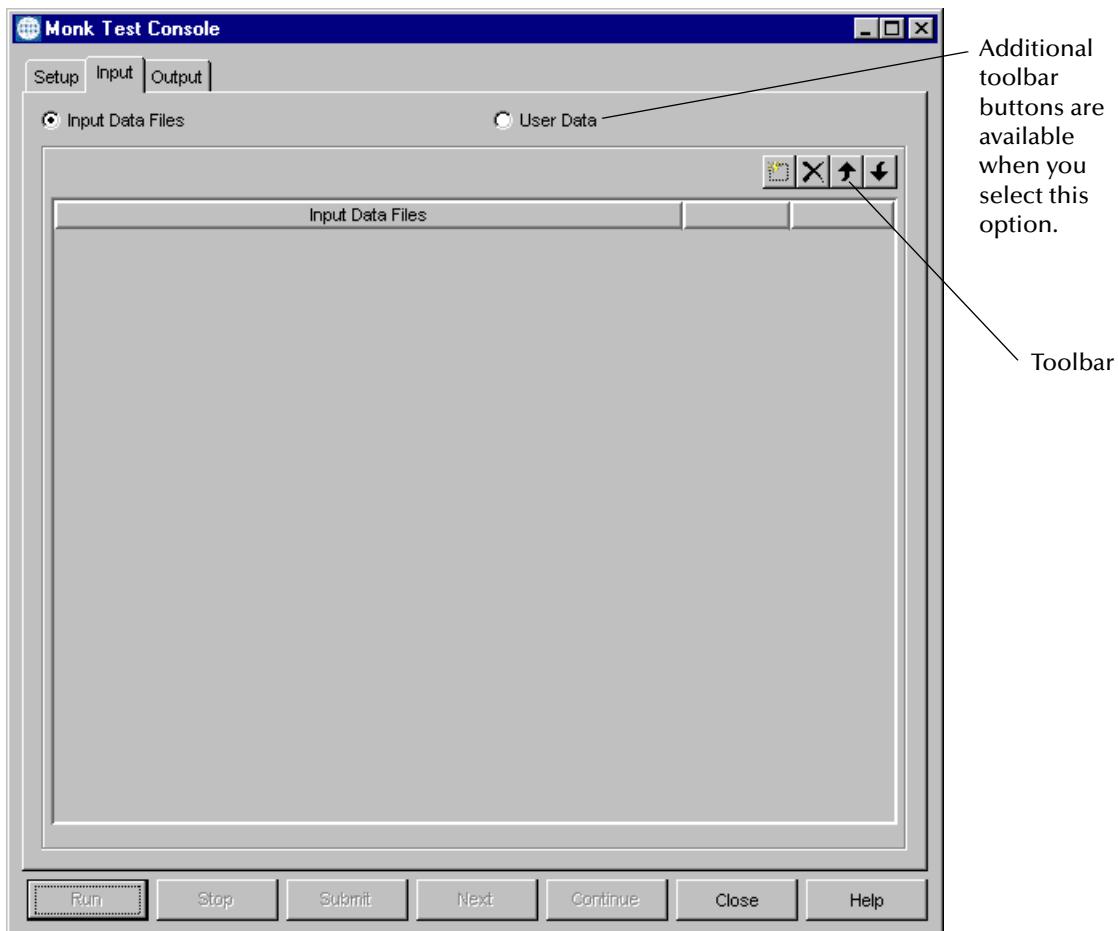
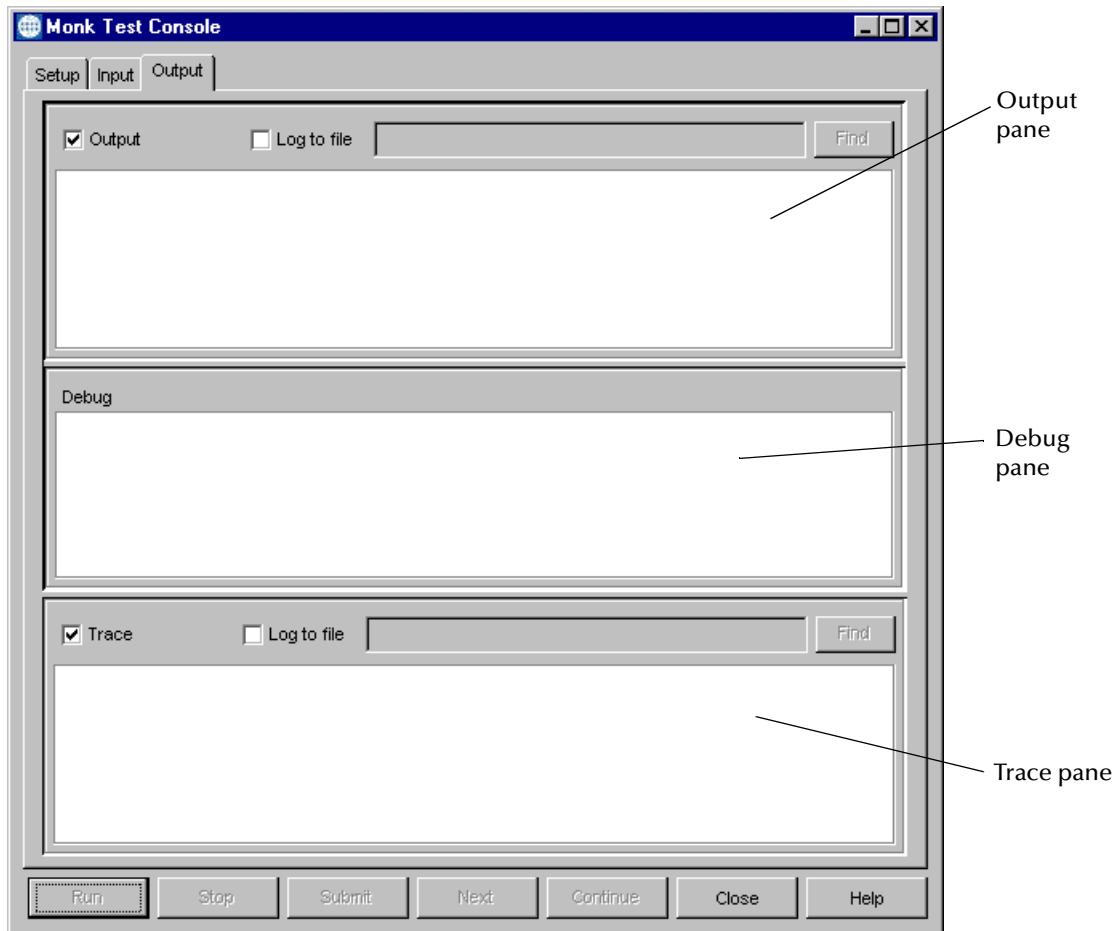


Figure 63 Monk Test Console, Output Tab



Basic Controls

The Monk Test Console window contains the control buttons (bottom row) explained in Table 12.

Table 12 Monk Test Console Control Buttons

Button	Function
Run	Runs the named function from the specified script.
Stop	Stops the function's execution.
Submit	Submits the text in the Output tab's Debug pane to the Monk engine for immediate execution. Results will appear in the Output pane. This control is available only if a (break) function is active within the script being tested.
Next	After a (break) function is called, executes the next line of instructions in the script. This control is available only if a (break) function is active within the script being tested.

Table 12 Monk Test Console Control Buttons (Continued)

Button	Function
Continue	After a (break) function is called, resumes execution until the next (break) function or until the function ends. This control is available only if a (break) function is active within the script being tested.
Close	Closes the Monk Test Console.
Help	Opens the Help window and runs the Help system.

The **Setup** and **Input** tabs (when **Input Data Files** is selected) contain four additional toolbar buttons explained in Table 13.

Table 13 Monk Test Console Toolbar

Button	Function
	Adds a file to the list. Click Find (displays a Select a <Type> File dialog box) to select the file from the Registry or click Edit to edit the file in a text editor. Editing is available from the Input tab only.
	Deletes the selected file from the current list.
	Moves the selected file one position up in the current list.
	Moves the selected file one position down in the current list.

The **Input** tab has two additional buttons that become available when you select the **User Data** option. These buttons are explained in Table 14.

Table 14 Monk Test Console Input/User Data Buttons

Button	Function
	Opens a text file and inserts its contents into the editable area.
	Saves the file to the local system (not the Registry). If you want to commit the saved file to the Registry, you must do so manually.

Setup Features

Use the Monk Test Console **Setup** tab features as follows:

- **Input Script:** Click **Find** to select the Collaboration Rules file to test, or click **Clear** to clear the file name selection. The file being tested *must* be located in the e*Gate Registry.

Find the needed files in the following folder:

\monk_scripts\common

Use files with the extensions **.ssc** and **.tsc**. Highlight the file and click **Select**.

- **Dependency Files:** Use the toolbar buttons to add any files that the file being tested depends on to function correctly: for example, other Collaboration Rules files, files containing Monk functions, or text files. All dependent files *must* be located in the e*Gate Registry. See [Table 13 on page 153](#) for information on how to add a file to this list.
- **Pre-function file to run:** There can be functions that you must run before the file is tested. In such cases, under this option, do one of the following actions:

- **Find:** Click to select the file that contains these functions.
- **Edit:** Click to edit the file in Notepad (the default external editor).
- **Clear:** Click to clear the file name selection.

Find the needed files in the following folder:

\monk_scripts\common

Use files with the extensions **.ssc** and **.tsc**. Highlight the file and click **Select**.

Function Name: Enter the name of the Monk function to test. e*Gate defaults assume that the name of the function is the same as the Collaboration Rules file name. If the function name is different, enter the correct function name.

Function Parameters: If the function requires any additional arguments or parameters, use this option as follows:

- 1 Add a new field either by clicking  or by pressing ENTER.
- 2 Double-click the field to edit it. You must enter all parameters inside double quotation marks ("").

Note: Use the toolbar buttons (see [Table 13 on page 153](#)) to manipulate these elements.

- 3 Press ENTER or select a different field to return to the selection mode.

After Finishing Setup

After you have entered all information required under the **Setup** tab, click the **Input** tab to define the input for the file or function. Also, you can go to the **Output** tab to run the test and to view its results.

Input Features

The **Input** tab selects the source of input for the function specified under the **Setup** tab. To use the Monk Test Console **Input** tab features, do either of the following actions:

- Select **Input Data Files** to add data files to the test schema. Then, use the toolbar buttons (see [Table 13 on page 153](#)) to add the files as follows:
 - ◆ **Find** selects a file from the Registry.
 - ◆ **Edit** modifies the file in the default text editor.
- Select **User Data** to enter your input manually.

When you are finished with the previous operation, select the **Output** tab to run the test and view the results.

Output Features

The **Output** tab displays the output of the script/function specified under the **Setup** tab, using the data specified under the **Input** tab. The **Output** tab has the following panes:

- **Output** displays the output of the script/function being run (**stdout** messages).
- **Debug** enables you to enter information manually during script execution once the script executes a (**break**) function. You can enter multiple lines of Monk code within the Debug pane. To submit the contents of the Debug pane to the Monk engine, click **Submit**.
- **Trace** displays error messages and other warnings (**stderr** messages).

For detailed information on what constitutes appropriate test outputs, see [Chapter 8](#).

To execute the script/function specified under the Setup tab

Click **Run**, then use the appropriate toolbar buttons. See [Table 13 on page 153](#) for an explanation of these buttons.

Note: *Run becomes active after you populate the Input Script field with the script you are going to test.*

To log the contents of either the Output or Trace panes

Do one of the following actions:

- Click the **Log to File** check box directly above the pane whose contents you want to log then specify the path and name of the file in the nearby text box.

- Click **Find** to browse for and select a specific file.

If you have already run the test, you must run it again to send the output to the specified file.

Note: If you use the (*display*) Monk function in your script, the output of the function appears in the **Output** pane. If you use the (*display-error*) Monk function, the output of the function appears in the **Trace** pane.

For more information on the Monk language, see the *Monk Developer's Reference*.

Event Type Definitions (ETDs)

5.1 About This Chapter

The e*Gate system allows you to create Event Type Definitions (ETDs) in either the Java or Monk programming languages. For details about how e*Gate processes Monk Event Types and ETDs, see [Chapter 6 “Monk Event Type Definition Editor” on page 204](#).

This chapter consists of the following sections:

- A brief overview of ETDs—what an Event Type Definition is and how it fits into the e*Gate system.
- A detailed description of the ETD Editor—its features, its graphical user interface (GUI), and the files it uses and creates.
- Basic procedures that tell you how to:
 - ◆ Create a new standard ETD.
 - ◆ Convert a Monk ETD to a Java-enabled standard ETD.
 - ◆ Build a new ETD by converting a file from a source outside e*Gate.
 - ◆ Open, save, and rename an ETD.
 - ◆ Use internal and external templates.
 - ◆ Compile an ETD.
 - ◆ Troubleshoot errors and warnings from the compiler.
 - ◆ Test an ETD.
 - ◆ Promote the ETD from Sandbox to run time.
 - ◆ Work with global and local delimiters.
- Explanations of the properties of each parent node and field of the standard ETD.

Note: For an explanation of the methods associated with the standard ETD, see [Chapter 7 “Java Collaboration Rules” on page 260](#).

5.2 Learning About ETDs

ETDs contain the formats for all Events (packets of data) that are transported and transformed by the e*Gate system. These Events include incoming and outgoing data as well as all data transformed and/or transported within e*Gate.

You can create specific ETDs for Event Types that pass through the e*Gate environment, including standard formats like HL7, X12, UN/EDIFACT, or proprietary formats. Once you know all the kinds of ETDs you need for your system, you can create them according to e*Gate's requirements, or you can use a Java ETD Library add-on.

Note: e*Gate add-on features contain ready-made templates you can use in configuring your system. For add-ons, see the appropriate user's guide for information.

5.2.1 What Is an ETD?

An ETD is a structural representation of an Event; that is, the blueprint of an Event. ETDs have a treelike structure, and are composed of elements called *nodes*.

Terminology: Parent, Child, Sibling, Root, and Leaf Nodes

Any subnode of a given node is called a child node, and the given node, in turn, is the child's parent. Sibling nodes are nodes on the same level under the same parent node. A descendant is the child of [a child of [...]] a child; an ancestor is the parent of [a parent of [...]] a parent.

A root node has no parent; a leaf node has no children. Fields and methods are always leaf nodes.

ETDs can be used to represent the following formats:

- Fixed or Delimited (or any combination of both)
- Objects available via APIs
- Database tables, stored procedures, or prepared statements

In a fixed-length ETD, the length of the data will always be the same. The position of data is described by byte offset and length.

In a delimited ETD, the length of the data varies, and it is *not* described by byte offset and length. Information is separated by a pre-determined delimiter setup within the properties of the ETD. There can be multiple delimiters in an Event, each representing an additional level of parsing.

An ETD has the following structure:

- The root node is the highest node in the tree structure. This node represents the entire Event. It may have one or more child notes, but can never have sibling nodes or be repeating. The properties of the root node cannot be edited (except its name).
- A subnode is the child of exactly one root node or subnode, and can itself be a parent node and/or a sibling node. Parsing instructions for each node are defined in the subnodes. Subnode properties can be edited in the Properties pane of the ETD Editor.

5.2.2 How Does e*Gate Use ETDs?

ETDs serve the following important functions in the e*Gate system:

- e*Gate uses ETDs to parse, validate, and (if necessary) transform Events.
- Event Types and ETDs also contain the instructions e*Gate uses to identify Events. You base ETDs on individual Event Type specifications, and they become the foundation of e*Gate data processing.
- A major advantage of ETDs is their reusability. If there are formats that recur in many of your Events, you can create definitions for those formats and use them as templates in other ETDs.

5.2.3 Java-Enabled ETDs

Using Java-Enabled ETDs (.xsc files), you create and modify Java Collaboration Rules using a simple GUI to manipulate the data structures using standard methods in a widely known language. This in turn unlocks the ability to interface with Java-based or Java-enabled subsystems such as the SeeBeyond JMS IQ Manager.

5.2.4 Monk ETDs

ETDs that are purely Monk and are thus not Java-enabled—in other words, .ssc files—are still supported; see [Chapter 6 Monk Event Type Definition Editor](#) on page 204.

5.3 ETD Editor Overview

The ETD Editor is the graphical user interface (GUI) for creating and modifying Java-enabled ETDs.

Caution: *While you are running a Multi-Mode e*Way, do not use the same machine to edit any ETD involved in the e*Way—if you do, you will be unable to save the ETD changes and unable to use normal methods to halt either the Editor or the e*Way.*

5.3.1 Feature Overview

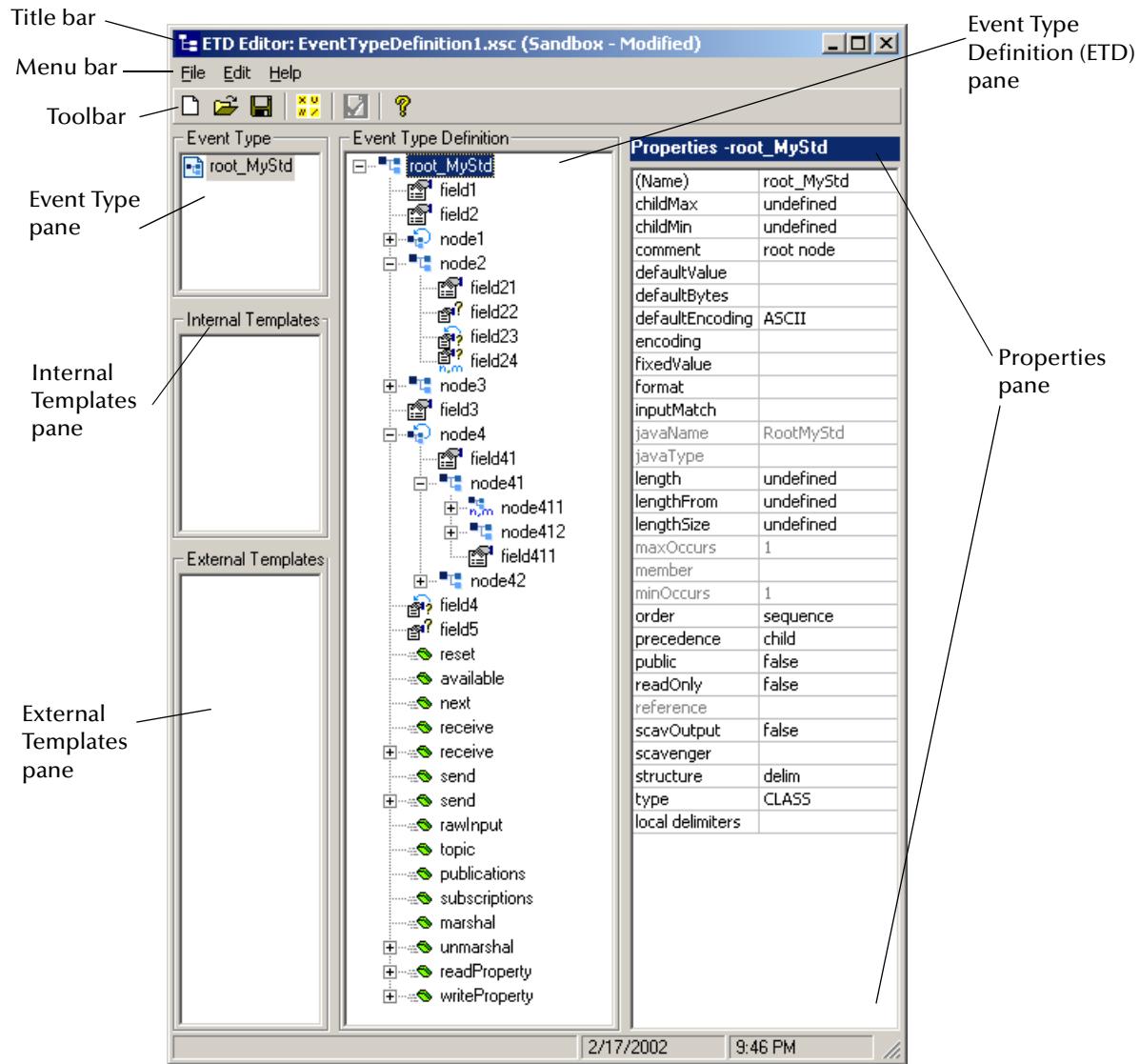
The ETD Editor provides the following features:

- ETD Builder wizards that automate creation of Java-enabled ETD (.xsc files).
- Ability to work with Java-enabled ETDs and to convert Monk ETDs (.ssc files).
- Ability to see all standard ETD methods and their properties.
- Side-branching tree layout that allows you to simultaneously view both the entire tree structure and the properties of the nodes, fields, and methods in the structure.
- Simple one-click creation of internal and external (read-only) templates.
- Ability to drag-and-drop templates or nodes.

5.3.2 GUI Overview

In addition to the main menu and the accompanying toolbar, the ETD Editor has five panes—the Event Type, Internal Templates, External Templates, Event Type Definition, and Properties panes. Figure 64 shows the names and locations of the different areas of the GUI.

Figure 64 ETD Editor GUI Map



ETD Editor GUI Areas

Title Bar

Displays the name of the **.xsc** file associated with the current ETD. When the ETD has unsaved changes to it, the title bar displays **(Sandbox - Modified)** after the file name.

Menu Bar

Contains menu commands; see [Main Menu](#) on page 162.

Toolbar

Contains shortcuts for certain menu commands; see [Toolbar](#) on page 163.

Event Type Pane

Displays the name of the current Event Type. For standard ETDs, this is always the same as the name of the root node.

Internal Templates Pane

Displays the names of all internal templates currently known to the current Event Type. You can right-click this pane add, delete, or rename internal templates, and you can drag internal templates under nodes in the ETD pane.

External Templates Pane

Displays the names of all external templates currently known to the current Event Type. You can right-click this pane to import or delete external templates, and you can drag external templates under nodes in the ETD pane.

Event Type Definition (ETD) Pane

Displays the tree structure for the Event Type or currently selected template. You can right-click a node in this pane to add, delete, or rename elements or fields, and you can expand or collapse a parent node.

Properties Pane

Displays property names and values for the currently selected element, field, or method. You can edit values by clicking the rightmost cell and then either entering a value or selecting from a drop-down list.

5.3.3 Main Menu

Table 15 describes the commands available from the main menu of the ETD Editor.

Table 15 ETD Editor Menus and Commands (Java)

Menu	Command	Description
File	New 	Opens the New Event Type Definition dialog box, presenting you with a choice of ETD Builder wizards you can use to build a new ETD. See “ Building Java-Enabled ETDs ” on page 164.
	Open 	Opens the Open dialog box, which allows you to locate and select any .xsc file that has been saved to either your Sandbox or the run-time environment for your schema. See “ Opening, Saving, and Renaming ETDs ” on page 175.
	Open From Default Schema	Opens the Open dialog box, allowing you to navigate to and select any .xsc file contained in the default schema, which resides in the Registry repository. See “ Opening, Saving, and Renaming ETDs ” on page 175.
	Save 	Regenerates the .xsc file for the current ETD without compiling it.
	Save As	Opens the Save dialog box, allowing you to generate a new .xsc files for the current ETD and save it under a different name without compiling it.
	Compile And Save	Runs the Java compiler and, if the compile is successful, creates an .ssc file for the current ETD. See “ Compiling an ETD ” on page 182.
	Run Test 	Opens the Test Dialog dialog box, allowing you to test how well the ETD parses data from a file you specify. See “ Validating an ETD ” on page 183.
	Promote to Run Time	Moves the current ETD from the sandbox environment to the run-time environment. See “ Promoting to Run Time ” on page 185.
Edit	Close	Closes the Editor.
	Delimiters 	Opens the Global Delimiters dialog box, allowing you to add, delete, and modify the properties of the delimiters used in this ETD. See “ Global and Local Delimiters ” on page 186.
Help	Java Properties	Opens the Java Properties dialog box, allowing you view the class, code editability, and .jar file name and to view or edit the package name and comments for this ETD. See “ Viewing and Editing Java Properties ” on page 176.
	Contents 	Opens the Help browser with the Contents tab showing.
	About	Displays the copyright, version, and build information, a copy of the license agreement, and a command button (System Info) that allows you view system parameters.

5.3.4 Toolbar

The buttons on the toolbar are shortcuts for commands that can be found on the main menu. Table 16 gives a brief description of what each button does.

Table 16 Toolbar Buttons

Button	Description
	Opens the New Event Type Definition dialog box, presenting you with a choice of ETD Builder wizards you can use to build a new ETD. Same as the New command on the File menu.
	Opens the Open dialog box, allowing you to navigate to and select an .xsc file associated with the Java Collaboration Rule you want to edit. Same as the Open command on the File menu.
	Regenerates the .xsc file for the current ETD. Same as the Save command on the File menu.
	Opens the Global Delimiters dialog box, allowing you to add, delete, and modify the properties of the delimiters used in this ETD. Same as the Delimiters command on the Edit menu.
	Opens the Test Dialog dialog box, allowing you to test how well the ETD parses data from a file you specify. Same as the Run Test command on the File menu.
	Opens the Help browser with the Contents tab showing. Same as the Contents command on the Help menu.

5.4 Before Using the ETD Editor

Before you start using the ETD Editor, consider the following Event Type properties:

- **Specifications:** All your Event Type specifications must be complete and correct before the Event Types you define can pass through the e*Gate environment.
- **Level of Detail:** In order to identify and translate Events, the e*Gate system, at a minimum, requires you to define ETDs at the root-node level. However, you must define most Events down to the node level assigned to the data field. This practice allows you to specify any system-required Event identification and/or transformation instructions needed later on.
- **Amount of Detail:** When you define an Event at a particular node level, it is desirable to define that level completely. If you do not define elements of any Event Type adequately, the system is unable to parse Events represented by that type, and these Events fail to pass through the e*Gate network correctly.

Important: *The Java ETD Editor does not validate your Events completely. It is up to you to make sure that you build ETD Trees using valid Event element types and in accordance with your predefined Event Type specifications.*

5.5 Building Java-Enabled ETDs

About ETD Types

All Java-enabled ETDs are of file type **.xsc** and can be displayed in the ETD Editor. However, the Editor distinguishes between standard ETDs and imported ETDs:

- **Standard ETDs** are native to e*Gate. You can create a standard ETD from scratch using the Standard ETD Wizard, or by converting an existing Monk ETD (**.ssc** file) using the SSC Wizard. Standard ETDs are read/write—you can add or delete nodes or edit their properties, and you can work with internal and external templates. For any standard ETD, the value of its **type** property is **SSC**.
- **Imported ETDs** are based on formats or standards external to e*Gate, such as **.dtd** (Document Type Definition) files or **.xsd** (XML Schema Definition) files, including proprietary formats for SAP BAPI and IDoc. This type of ETD can only be built from the appropriate ETD library or using the appropriate ETD Builder wizard, and the resulting ETD is read-only—you can view its structure and properties, but (with some rare exceptions), you cannot modify it in any way. The **type** property of an imported ETD must be something other than **SSC**; examples include **EMAIL**, **HTTP**, **FTPFile**, **X12-4020**, **c1mxpcconfig**, and so forth. As an example, a simple XSD-based ETD is illustrated in [Figure 77 on page 184](#). For information on a particular ETD library, refer to the user's guide for that library.

Note: *ETDs can only reference templates of their own type—a standard ETD can only reference a standard ETD, a DB ETD can only reference a DB ETD, and so forth.*

The term **Complex ETD**, or **API-based ETD**, is sometimes used to refer to non-native ETDs that have extensive APIs for communicating with external systems; by contrast, standard ETDs and simple imported ETDs are called “messageable” or “marshallable” because they can be “marshalled” or flattened into a flat nonhierarchical structure.

Package Names, Node Names, and .jar File Names

When an ETD Builder wizard requires you to enter a package name, it is prompting you for the argument to be passed to the **package** declaration in the Java source code. Packages help you organize your programs, and they prevent conflicts between names in other ETDs, Collaborations, or even other vendor or custom Java objects.

When you compile and save your ETD, you create a single **.jar** file (Java archive file) whose file name is the same as the **.xsc** file name. The **.jar** file contains **.java** (Java source code) and **.class** (executable Java bytestream) files whose names correspond to the root node names in the ETD; if the ETD contains no internal templates, it will have only one root node name. Root node names can *never* contain multibyte characters, even if used on a multibyte operating systems.

Also see [About Package Names](#) on page 174 and [Viewing and Editing Java Properties](#) on page 176.

5.5.1 Starting the ETD Editor

To start the ETD Editor

- 1 In Enterprise Manager, open the schema whose ETDs you want to add or modify.
- 2 On the **Options** menu, click **Default Editor** and, if necessary, set it to **Java**.
- 3 On the main toolbar or **Tools** menu, click **ETD Editor** .

The ETD Editor opens but is blank, since no Event Type or ETD has been defined.

5.5.2 Creating a New Standard ETD

To create a new Java-enabled ETD from scratch, use the Standard ETD Wizard.

To create a Java-enabled ETD using the Standard ETD Wizard

- 1 On the toolbar or **File** menu of the ETD Editor, click **New** .

The **New Event Type Definition** dialog box displays the ETD Builder wizards; see Figure 65.

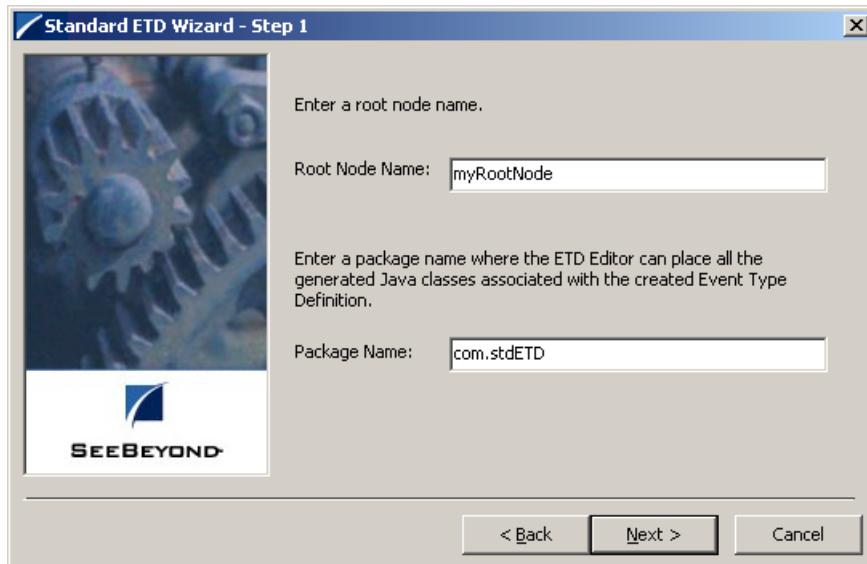
Figure 65 New Event Type Definition Dialog Box



Note: The contents of this dialog box depend on the products you have installed.

- 2 Double-click the Standard ETD Wizard, read the **Standard ETD Wizard - Introduction** to be sure you meet the requirements, and then click **Next** to continue with **Standard ETD Wizard - Step 1**; see Figure 66.

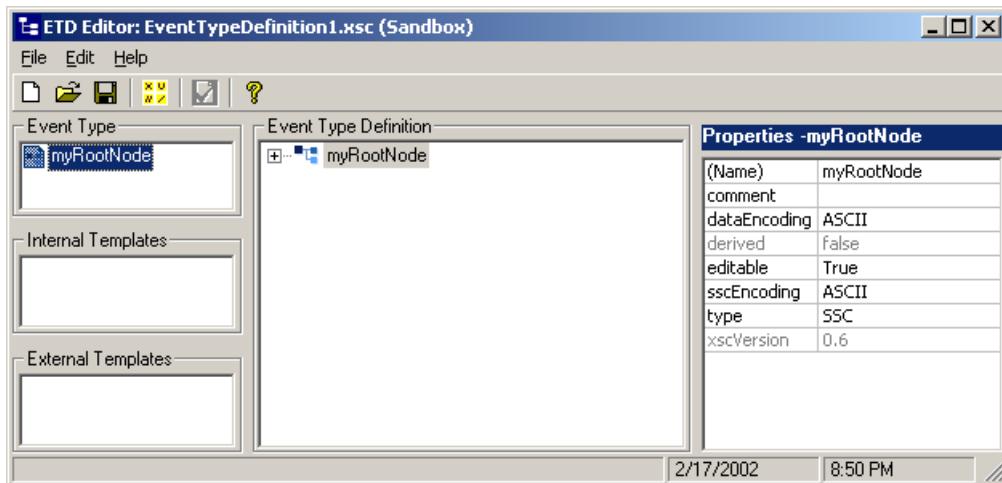
Figure 66 Standard ETD Wizard - Step 1



- 3 Enter a root node name and package name for the container in which the wizard will place the generated Java classes, and then click **Next**. For information on node names and package names, see [Working With Java-Enabled ETDs](#) on page 174.
- 4 Review the information for accuracy, and then click **Finish**.

The Editor creates a new ETD with the specified root and package names. See Figure 67.

Figure 67 Newly Created Standard ETD



You can now modify this ETD as needed. For example, you can set global and local delimiters, add elements and fields, edit the properties of elements and fields, add and manipulate internal templates, and add external templates.

5.5.3 Converting a Monk ETD to a Java-enabled Standard ETD

To take advantage of Java features, such as the SeeBeyond JMS messaging subsystem, you can convert existing Monk ETDs (.ssc files) to Java-enabled ETDs (.xsc files) using the SSC Wizard.

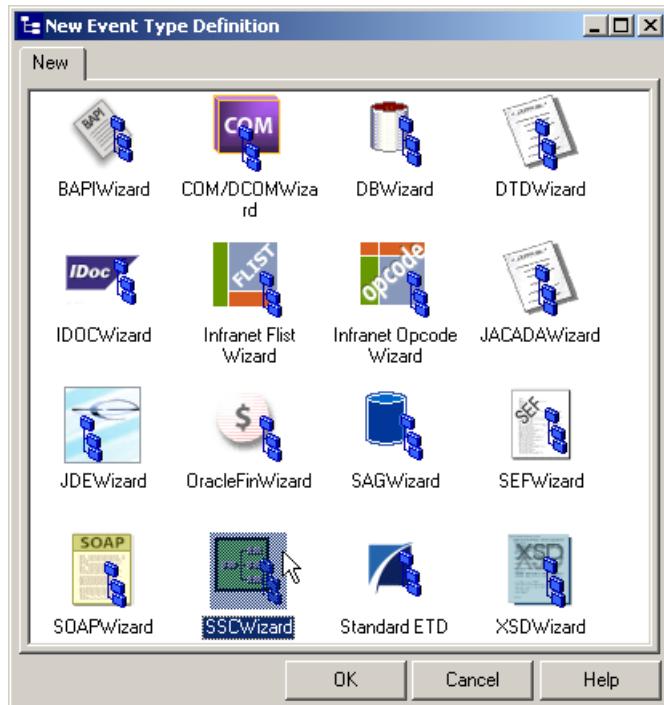
Important: *The following limitations apply to .ssc files being converted to .xsc files:*

- Encoded length for fixed nodes of SSC-based ETDs is currently not supported.
- The longest fully qualified path+nodename cannot exceed 255 characters.
A “fully qualified path” means *packagename + iterated{path+filename}*.
For example, you cannot convert an SSC that has more than six levels if it uses 40-character names at each level. (When an ETD references an internal template, the template’s internal does count against the 255 total, but its node name does).
- The wizard cannot directly import .ssc files that use external templates until all such templates have been compiled and exist as .xsc files in the **etd** folder.
For nested templates, compile the lowest (terminal) templates first and iterate upwards through the hierarchy.
- You cannot use a non-.ssc-based .xsc file as an internal template in an .xsc file that was converted from an .ssc.

To create a Java-enabled ETD using the SSC Wizard

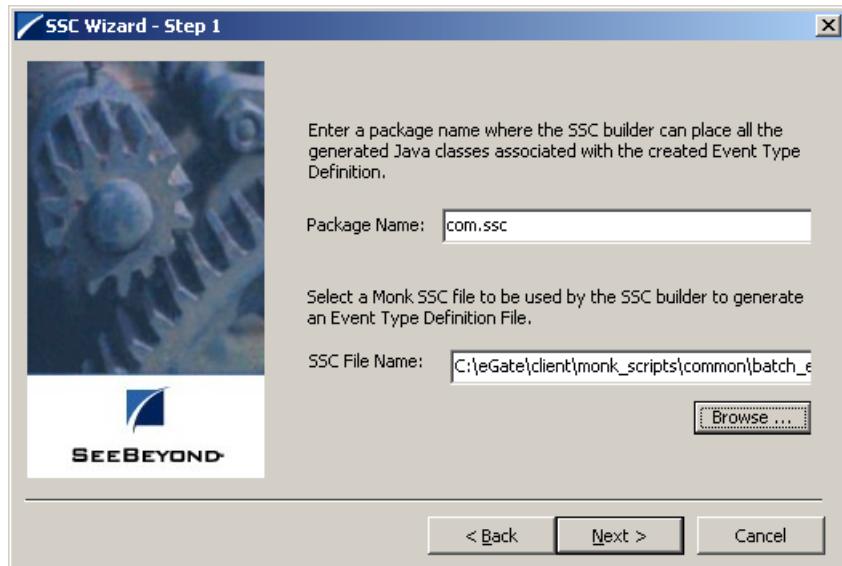
- 1 On the toolbar or **File** menu of the ETD Editor, click **New**  to display the ETD Builder wizards. See Figure 68.

Figure 68 ETD Builder Wizards



- 2** Scroll down to the SSC Wizard and double-click it; read the **SSC Wizard - Introduction** to be sure you meet the requirements; and then click **Next** to continue with **SSC Wizard - Step 1**. See Figure 69.

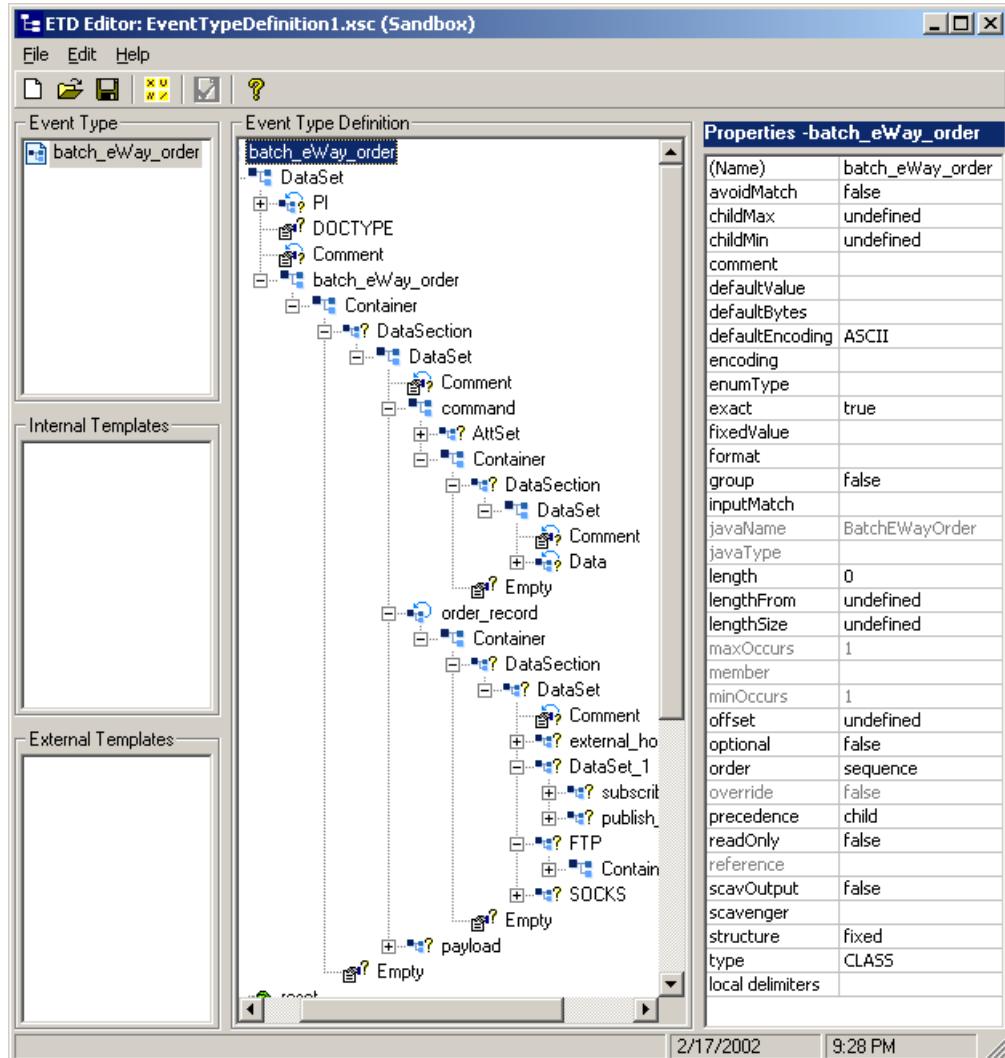
Figure 69 SSC Wizard - Step 1



- 3** Enter a package name for the container in which the wizard will place the generated Java classes. For information on package names, see [About Package Names](#) on page 174.
- 4** Enter the file name (or click **Browse** to locate and select the file name) of the Monk **.ssc** file you want to convert to a Java-enabled **.xsc** file.
- 5** Click **Next**, review the information for accuracy, and then click **Finish**.

The Editor displays the converted ETD; see Figure 70.

Figure 70 Result of Converting a Monk ETD Using the SSC Wizard



You can now modify this ETD as needed. For example, you can set global delimiters, add elements and fields, edit the properties of elements and fields, add and manipulate internal templates, and add external templates. Compiling the ETD generates .xsc, .ssc, and .jar files containing the changes you made using the Editor.

5.5.4 Building an Imported ETD

When you build a new ETD by converting a file from a source outside e*Gate, the ETD you create is read-only (with rare exceptions). You can use read-only ETDs as source and/or destination Event Types in Collaboration Rules just as you can a standard ETD, and you can use read-only ETDs as external templates for a standard ETD, but you cannot make or save any changes to the structure or elements of a read-only ETD.

You use an ETD Builder wizard to create an ETD (*.xsc file) that captures all data in the original data source. Depending on the external source, however, the topology of the mapping may be distorted—for example, when building an ETD from a DTD that has no clearly defined root, the wizard tries to optimize the use of internal templates, and may not necessarily retain the root name supplied to it.

For information on using the ETD Builder wizards for any of the following, refer to the corresponding e*Way User's Guide, ETD Library User's Guide, or Toolkit. For example:

- BAPI Wizard—See the *e*Way Intelligent Adapter for SAP (BAPI) User's Guide*.
- COM/DCOM Wizard—See the *e*Way Intelligent Adapter for COM/DCOM User's Guide*.
- DB Wizard—See the appropriate e*Way User's Guide for the applicable database management system (Oracle, Sybase, DB2 Universal Database, and so on).
- DTD Wizard—See the *XML Toolkit*.
- IDoc Wizard—See the *e*Way Intelligent Adapter for SAP (ALE) User's Guide* and the *e*Way Intelligent Adapter for SAP (EDI) User's Guide*.
- Infranet Flist Wizard—See the *e*Way Intelligent Adapter for Portal User's Guide*.
- Infranet Opcode Wizard—See the *e*Way Intelligent Adapter for Portal User's Guide*.
- Jacada Wizard—See the *e*Way Intelligent Adapter for Jacada Enterprise/Access User's Guide*.
- JDE Wizard—See the *e*Way Intelligent Adapter for JDE OneWorld GenJava User's Guide*.
- OracleFin Wizard—See the *e*Way Intelligent Adapter for Oracle Financials User's Guide*.
- SAG Wizard—See the *e*Way Intelligent Adapter for ADABAS Natural User's Guide*.
- SOAP Wizard—See the *SOAP e*Way Intelligent Adapter User's Guide*.
- XSD Wizard—See the *XML Toolkit*.

Using the SEF Wizard

The Standard Exchange Format (SEF) is an open standard maintained by the Foresight Corporation and used to exchange EDI implementation guidelines in a machine-readable form. SEF documentation can be obtained through the following URL:

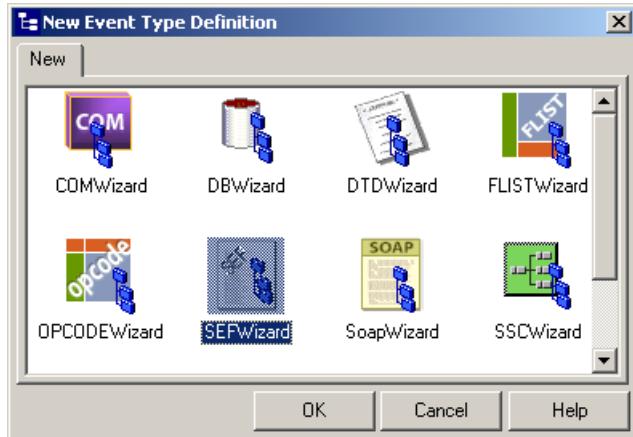
<http://www.foresightcorp.com>

To create a read-only Java-enabled ETD using the SEF Wizard

- 1 On the toolbar or **File** menu of the ETD Editor, click **New** .

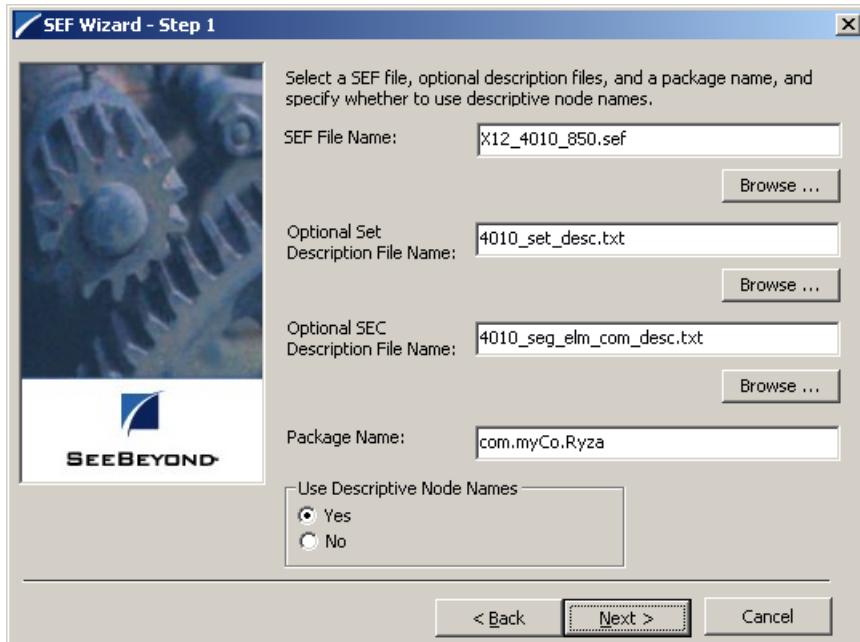
The **New Event Type Definition** dialog box displays the ETD Builder wizards. See Figure 71.

Figure 71 ETD Builder Wizards



- 2 Scroll down to the SEF Wizard and double-click it, read the **SEF Wizard Introduction** dialog box to be sure you meet the requirements, and then click **Next** to continue with **SEF Wizard - Step 1**; see Figure 72.

Figure 72 SEF Wizard - Step 1



- 3 Enter the file name (or click **Browse** to navigate to and select the file name) of the the .sef file you want to convert to an ETD.

- 4 Optionally, enter the file name (or click **Browse** to navigate to and select the file name) of the Set Description file you want to associate with the ETD.

This file would contain a description of the transaction sets, mapping three-digit codes to meaningful phrases.

- 5 Optionally, enter the file name (or click **Browse** to navigate to and select the file name) of the SEC Description file you want to associate with the ETD.

This file would contain a description of the Segments, Elements, and Composites that make up a transaction

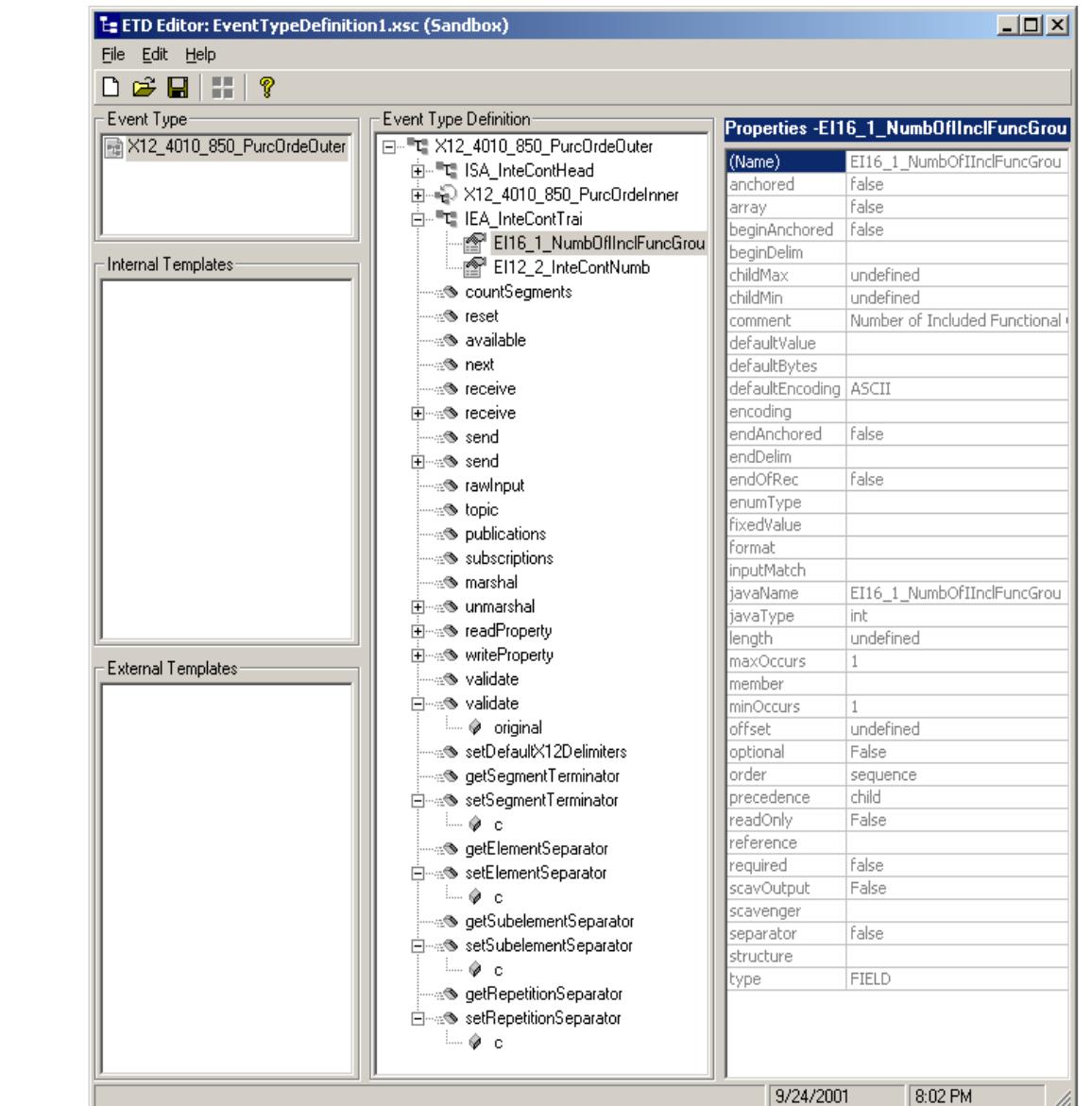
- 6 For the files specified in steps 4 or 5, if you want the ETD nodes named using the descriptive fields, keep the **Yes** option button setting; otherwise, click **No**.

If you click **No** but you specified file names in steps 4 or 5, then the set descriptions or SEC descriptions are preserved as **comment** properties of the individual nodes.

- 7 Click **Next**, review the information for accuracy, and then click **Finish**.

The Editor displays the converted ETD. See Figure 73.

Figure 73 Result of Converting an SEF File Using the SEF Wizard



5.6 Working With Java-Enabled ETDs

About Package Names

Package names are case-sensitive and must consist only of alphanumeric characters (a-z, A-Z, 0-9) and optional dot delimiters. This restriction applies even on multibyte operating systems. Each delimited segment must begin with an alphabetic character (a-z, A-Z), and no delimited segment can be a Java keyword. For example, **com.stc.x12** and **com.stc.X12** are distinct legal package names; **com.0abc**, **\$temp**, **com.stc.else**, and **_abc** are all illegal package names.

Upper-ASCII characters x'80' through x'9A' [ÇüéâäåçêëèïìÄÅÉæÆôöòùÿÖÜ] and x'A0' through x'A7' [áíóúñÑao] are also permitted in package names. The package name must be different from any of the node names.

About Node Names

When you create ETDs, you give each node a name that becomes its own identifier in the ETD Tree. This name becomes the label for the node in the ETD Tree. A node name is a *Java identifier*: It cannot be a Java keyword or BooleanLiteral, its first character must be a *Java letter*, and each of its zero or more subsequent characters must be a *Java letter-or-digit* (any letter or digit drawn from the entire Unicode character set, although ETDs whose node names contain multibyte characters can only be compiled on a multibyte operating system). Also, a node name cannot be a DOS device name (such as CON, PRN, AUX, CLOCK\$, NUL, COM1, COM2, ..., COM9, LPT1, ..., LPT9).

A node name cannot be the same as the package name, and cannot match the name of any of its sibling nodes. (All root nodes within an Event Type—in other words, the root nodes of the Event Type and each of its internal and external templates—are siblings.)

Node names are case sensitive—in other words, a node named **myNode** is different from a node named **MyNode**. Also, although multibyte characters are permissible in node names, such .xsc files can only be compiled in a multibyte operating system.

Create each name for your ease of reference. For your convenience, do not use more than 40 characters in a single node name. It is desirable to make each root node name unique within a schema. This ensures that you do not confuse one ETD with another. If you want to access nodes by name, you must give unique names to all the nodes within a given level.

Note: Use naming conventions that you can easily recognize later on. For example, if you name an Event Type with **et_Sample** or an ETD as **etd_Sample**, both the Event Type and the ETD are clearly marked as belonging to the Event named **Sample**.

5.6.1 Basic ETD Procedures

Caution: Before using a Participating Host to open any ETD, shut down all of that Host's components (BOBs or e*Ways) that might be using the ETD. Trying to edit an ETD on the same machine that is using the ETD in a running e*Gate module can destabilize both the Editor and the module.

Opening, Saving, and Renaming ETDs

You can open an existing ETD (.xsc file) even if it has not yet been compiled.

- Use the **Open** menu command or toolbar button to locate and open an ETD that already exists in your Sandbox or run-time environment. This operation loads the ETD and associated files into your Sandbox.
- Use the **Open from Default Schema** menu command to open an ETD that exists in the **default** schema (located on the Server in the Registry repository). This allows you to browse the **default** version of the ETD in *read-only* mode. To create a local copy for editing or promotion to run time, you must first save it to a location in your own Sandbox.

Note: The Editor opens only one ETD at a time. If you have an ETD open and try to open another, you are prompted to save or discard unsaved changes in the current ETD.

To open an ETD that resides in your Sandbox or run-time environment

- 1 On the toolbar or **File** menu, click **Open** .
- 2 In the **Open** dialog box, locate the .xsc file you want to load, and then click **Open**.

To open an ETD that resides in the *default* schema

- 1 On the **File** menu, click **Open From Default Schema**.
- 2 In the **Open** dialog box, locate the .xsc file you want to load, and then click **Open**.

The **default** schema's ETD is displayed in *read-only* mode. You cannot edit, compile, or promote this ETD unless you first save a local copy of it to your Sandbox.

To save an ETD under a new name or location in your Sandbox

- 1 On the **File** menu, click **Save As**.
- 2 In the **Save** dialog box, navigate to the location where you want to save this .xsc file, enter the file name you want (do not change the file type), and click **Save**.

To save your work in progress on an ETD

- On the toolbar or **File** menu, click **Save** .

Viewing and Editing Java Properties

ETDs have the following Java properties:

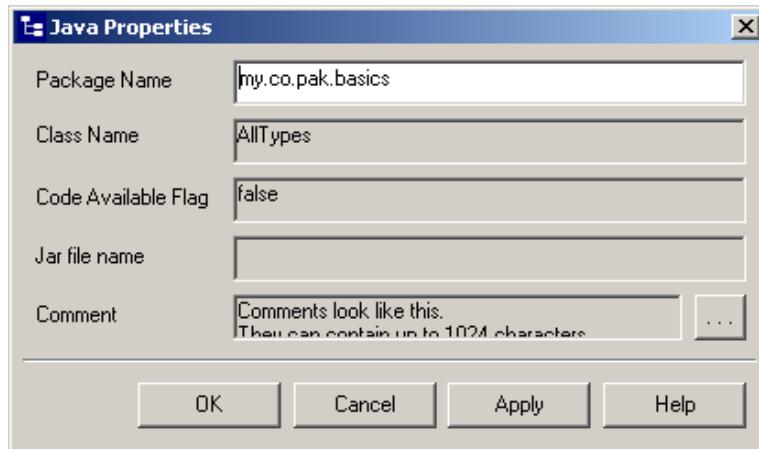
- **.jar** file name—When you compile and save your ETD, you create a single **.jar** file (Java archive file) whose file name is the same as the **.xsc** file name. The **.jar** file contains **.java** (Java source code) and **.class** (executable Java bytestream) files whose names correspond to the root node names in the ETD.
- package name—When you create an ETD, you must supply a package name; this becomes the **package** declaration in the Java source code, and helps prevent conflicts between names in other ETDs, Collaborations, or even other vendor or custom Java objects.
- class name—The **.class** name is the same as the file name under which you saved the ETD.
- code available flag—Indicates whether the **.xsc** file is available for use. When an editable ETD is undergoing modification, its code remains unavailable (code=**false**) until the ETD is recompiled or the changes are canceled.
- Java-property comments—You can associate a Java properties comment with the ETD; note that this is a comment on the Java properties, and is not the same as the comment associated with the ETD itself or with the ETD's root node.

To view the Java properties of an ETD or to edit its comment field

- 1 On the **Edit** menu, click **Java Properties**.

The **Java Properties** dialog box appears. See Figure 74.

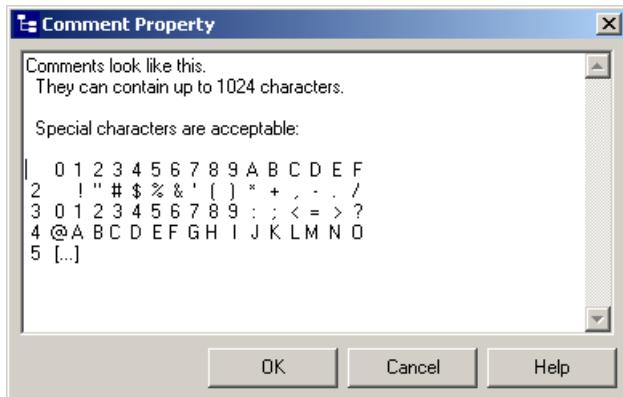
Figure 74 The **Java Properties** Dialog Box



- 2 To view the entirety of a multi-line comment, or to modify it, click the ... button.

The **Comment Property** dialog appears. See Figure 75.

Figure 75 The **Comment Property** Dialog Box



Working with Elements and Fields

The **Event Type Definition** pane provides a tree-like graphical of the elements (parent nodes) and fields (leaf nodes) in your ETD and allows you to add, delete, move, and modify the properties of the various nodes.

To expand or collapse an element (parent node)

- To expand a collapsed parent node, click . All immediate children of the selected parent node are displayed directly beneath it.
- To collapse an expanded parent node, click . All descendants of the selected parent node are hidden.

To add an element (parent node) or field (leaf node) before or after a sibling node

- Right-click a node and, on the shortcut menu, do one of the following:
 - ♦ To add a parent node as a sibling, point to **Add Element** and then click either **Before Selected Node** or **After Selected Node**.
 - ♦ To add a leaf node as a sibling, point to **Add Field** and then click either **Before Selected Node** or **After Selected Node**.

Note: You cannot add a sibling to the root node.

To add an element or field inside a parent node

- Right-click a parent node and, on the shortcut menu, do one of the following:
 - ♦ To add a parent node, point to **Add Element** and then click **As Child Node**.
 - ♦ To add a leaf node, point to **Add Field** and then click **As Child Node**.

To move an existing node inside, before, or after another element or field

- Click the node, drag it towards the target node, and do one of the following:
 - ♦ To move the selected node inside an element, drop the node onto the target element.

- ♦ To move the selected node immediately before another element or field, hold down the SHIFT key as you release the mouse button.
- ♦ To move the selected node immediately after another element or field, hold down the CTRL key as you release the mouse button.

The node—with its descendants, if it has any—moves to the indicated destination. There is no Undo.

Tip: *On the keyboard, the SHIFT key is above the CTRL key; SHIFT-drag-and-drop means “move before” while CTRL-drag-and-drop means “move after.”*

To rename an element or field

Do one of the following:

- Triple-click the node and then enter the new name.
- Click the node, pause, click it again, and then enter the new name.
- Right-click the node, click **Rename** on the shortcut menu, and enter the new name.

Note: *To undo the **Rename** operation, press ESCAPE before taking any other action.*

To delete an element or field

- Right-click the node, and, on the shortcut menu, click **Delete**.

Note: *You cannot undo a **Delete** operation.*

To delete all descendants of an element (parent node)

- Right-click the node, and, on the shortcut menu, click **Delete All Child Nodes**.

Note: *You cannot undo a **Delete All Child Nodes** operation.*

To specify that a node need not contain data

- 1 Select the node.
- 2 In the Properties pane, change the value of the **minOccurs** property to 0.

The icon for the node has a question mark added to it—for example, or —to signal that it might or might not contain data. See Table 17.

When the ETD is compiled, a **boolean** method named **has<MyNode>()** is generated; this method lets you query whether the node contains data in the instance being processed. If the value of the **maxOccurs** property is 1, a **void** method named **omit<MyNode>()** is also generated; calling the **omit...()** method forces the corresponding **has...()** to **false**, allowing you to skip the node even if it contains data in the instance being processed.

To specify that a node is repeating

- Select the node and, in the Properties pane, change the value of **maxOccurs** to one of the following:
 - ♦ -1. This indicates that the node repetition has no upper bound.

- ♦ an integer greater than 1 but not less than the current value of **minOccurs**.
If necessary, adjust **minOccurs** to a lower value before you set **maxOccurs**.

The icon for the node changes to signal the type of repetition it can have. See Table 17.

When the ETD is compiled, an **int** method named **count<MyNode>()** is generated; this method lets you query how many copies occur in the instance being processed.

Table 17 Node Icons for Java ETDs

Icon	minOccurs	maxOccurs	Meaning
	min=1	max=1	Default setting: A field (leaf node) that must occur exactly once, with no repetitions.
	min=1	max=1	Default setting: An element (parent node) that must occur exactly once, with no repetitions.
	min=0	max=1	Leaf node that might or might not occur once, but does not repeat.
	min=0	max=1	Parent node that might or might not occur once, but does not repeat.
	min=0	max=-1	Repeating leaf node that might or might not occur once, with unlimited repetitions.
	min=0	max=-1	Repeating parent node that might or might not occur once, with unlimited repetitions.
	min=0	max>1	Repeating leaf node that might or might not occur once, with limited repetitions.
	min=0	max>1	Repeating parent node that might or might not occur once, with limited repetitions.
	0<min<max	max>1	Repeating leaf node that must occur at least once, with limited repetitions.
	0<min<max	max>1	Repeating parent node that must occur at least once, with limited repetitions.
	min=1	max=-1	Repeating leaf node that must occur at least once, with unlimited repetitions.
	min=1	max=-1	Repeating parent node that must occur at least once, with unlimited repetitions.
	1<min	max=-1	Repeating leaf node that must occur at least twice, with unlimited repetitions.
	1<min	max=-1	Repeating parent node that must occur at least twice, with unlimited repetitions.

Using Templates

About internal templates

If your ETD has different areas with identical subtrees, you can decrease effort and increase maintainability by using internal templates instead of re-creating the subtree each new place it is used. After you have created an internal template, you can drag it under any node of the ETD tree of either the Event Type or another internal template. The new node is an alias for the template, and can be assigned any valid node name; the **member** property of the new node contains the name of the internal template.

The Editor allows you to temporarily create circular dependencies—for example, you can have a node of Template1 contain an instance of Template2 even while a node of Template2 contains an instance of Template1—but such an ETD cannot be compiled.

About external templates

You can use the external template feature to have your ETD reference a frozen copy of another ETD of the same type. For example, a standard ETD can reference one or more standard ETDs, and an imported ETD can reference another ETD of its own type. For information on a particular Java ETD library, refer to the user's guide for that library. An external template is a pointer to an .xsc file that resides outside the current ETD; the **reference** property of the new node contains the path and file name of the external template.

Note: *For the external template itself, all elements and properties are read-only. However, when the template is used as an **instance** within an ETD or internal template, the instance properties can be edited.*

To create and name a new internal template

- 1 Right-click the Internal Templates pane; on the shortcut menu, click **New Template**. A new internal template appears, with a generic name like **InternalTemplate1**. The value of the template's **type** property is **CLASS**.
- 2 In the Properties pane, change the **(Name)** property to a valid node name. See "[About Node Names](#)" on page 174.

To create a new external template

- 1 Right-click the External Templates pane; on the shortcut menu, click **Import External Reference**.
- 2 In the **Open** dialog box, navigate to the .xsc file you want to import and click **Open**. A new external template appears; you can view all its properties and members, but cannot make changes. The value of the template's **type** property is **CLASS**.

To work with an internal template

- 1 In the Internal Templates pane, click the template and edit its root node properties as needed. Such properties include its name, delimiters, comments, default values, and so forth; see [Properties of Standard ETD Parent Node Elements \(type=CLASS\)](#) on page 192.

- 2 Within the internal template, create elements and fields (parent and leaf nodes) and edit their properties as needed. An internal template can contain external templates or other internal templates.
- 3 To add the internal template to the ETD or to another internal template, open the destination and then drag the template to the correct parent node. See “[To place and edit an instance of a template” on page 181.](#)

Note: You can continue to add, delete, and modify elements and fields within an internal template even after it is instantiated within the ETD or other internal template; the changes you make are propagated to all instances.

To place and edit an instance of a template

- 1 In the left-hand pane, click the name of the ETD or internal template that will contain the new template instance; then, in the Event Type Definition pane, locate the destination node.
- 2 Drag the template towards the destination node and then do one of the following:
 - ♦ To move the template inside an element, drop the template onto the destination element.
 - ♦ To move the template node immediately before another element or field, hold down the SHIFT key as you release the mouse button.
 - ♦ To move the template node immediately after another element or field, hold down the CTRL key as you release the mouse button.

Tip: On the keyboard, the SHIFT key is above the CTRL key; SHIFT-drag-and-drop means “move before” while CTRL-drag-and-drop means “move after.”

An instance of the template appears under, before, or after the selected node; the value of the instance’s **type** property is **REFERENCE**, and the value of its **member** property is the template name.

- 3 As needed, edit the leaf-node properties of this new instance; see [Properties of Standard ETD Field Nodes \(type=FIELD\)](#) on page 197. Compared to field nodes, a template instance has additional properties (such as **member** and **reference**), and many more of its properties are read-only.

To delete an existing template

Before you begin: If the template is used as a node in the main ETD or in any other internal template, you must first find and delete all instances where it is used.

After the ETD is free of all references to [templates with references to [...]] the template, do the following.

- 1 In the Internal Templates or External Templates pane, right-click the template.
- 2 On the shortcut menu, click **Delete**.

To rename an existing internal template

- In the Internal Templates pane, do one of the following:
 - ♦ Triple-click the template name and then enter a new valid node name.

- ♦ Select the template name and then, in the Properties pane, change its **(Name)** property to a valid node name.

Compiling an ETD

After you have created the tree structure of your ETD and modified the properties of its elements and fields, you must compile it.

- On the **File** menu, click **Compile and Save**.

The standard Java compiler is invoked: If no problems are found, no message is returned. Otherwise, error messages are presented to you in the **Error** dialog box.

For a partial list of the most frequently encountered compile errors, see Table 18.

Table 18 Common Errors When Compiling an ETD

Error	Probable cause	Steps to resolve
leaf-node "Element1" has no Java type (274)	The Event Type and/or an Internal Template has one or more parent nodes that lack any children.	Find all childless parent nodes and either give them children fields or delete them.
reference to undefined template "iTemp1" from node "iTemp11" in [top] -> int_Temp2	An instance of an internal template was dragged into a node of one of the ETD trees, but the template itself was later renamed or deleted from the Internal Templates pane.	Delete all remaining ETD instances of templates that have been deleted; correctly rename all ETD instances of templates that were renamed after being instantiated.
circular local template dependency: [top] -> intl_Temp2 -> intl_Temp1 -> int_Temp2	Internal Template #1 references Internal Template #2, which [references template #3, [...] which] references Internal Template #1.	Analyze the circular chain and delete all template instances that are inappropriate.

Note: Whenever you make a change to an ETD, you must recompile it before the change will be noticed by any Collaboration whose Event Type instances use the modified ETD. After a modified ETD has been recompiled, you must reload it into any Collaboration that references it and then recompile the Collaboration.

5.6.2 Validating an ETD

After your ETD has compiled without error, you can validate it against representative data files. Use the ETD **Test Dialog** to parse the data using the current ETD structure. If the parsing is unsuccessful, error messages are shown in the Trace Data pane; if it is successful, the results are shown in the Test Tree pane.

Figure 76 shows the results of parsing a representative data file named **juicer6.xml**.

Figure 76 ETD Editor – **Test Dialog** Showing Parsed Data

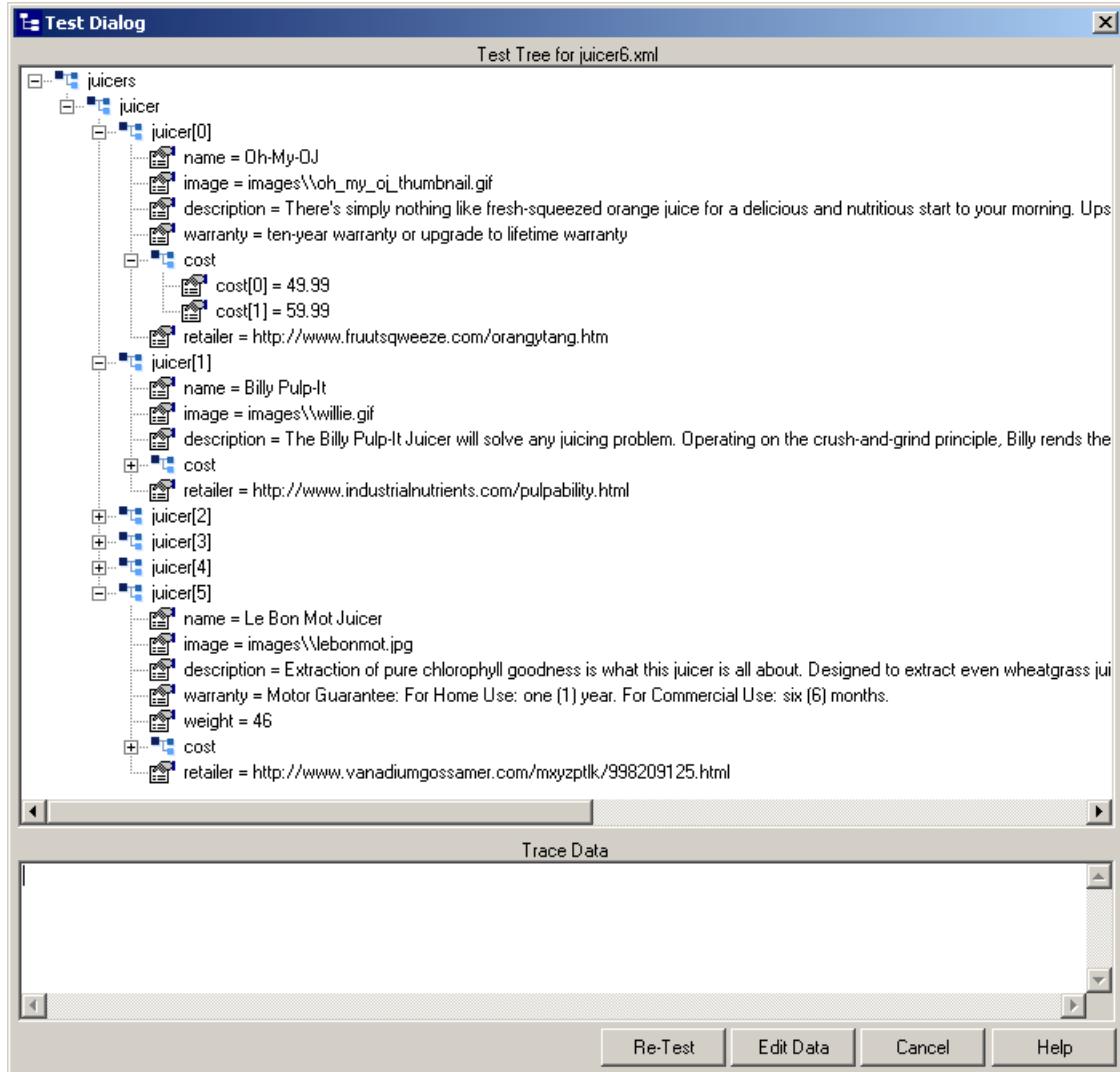
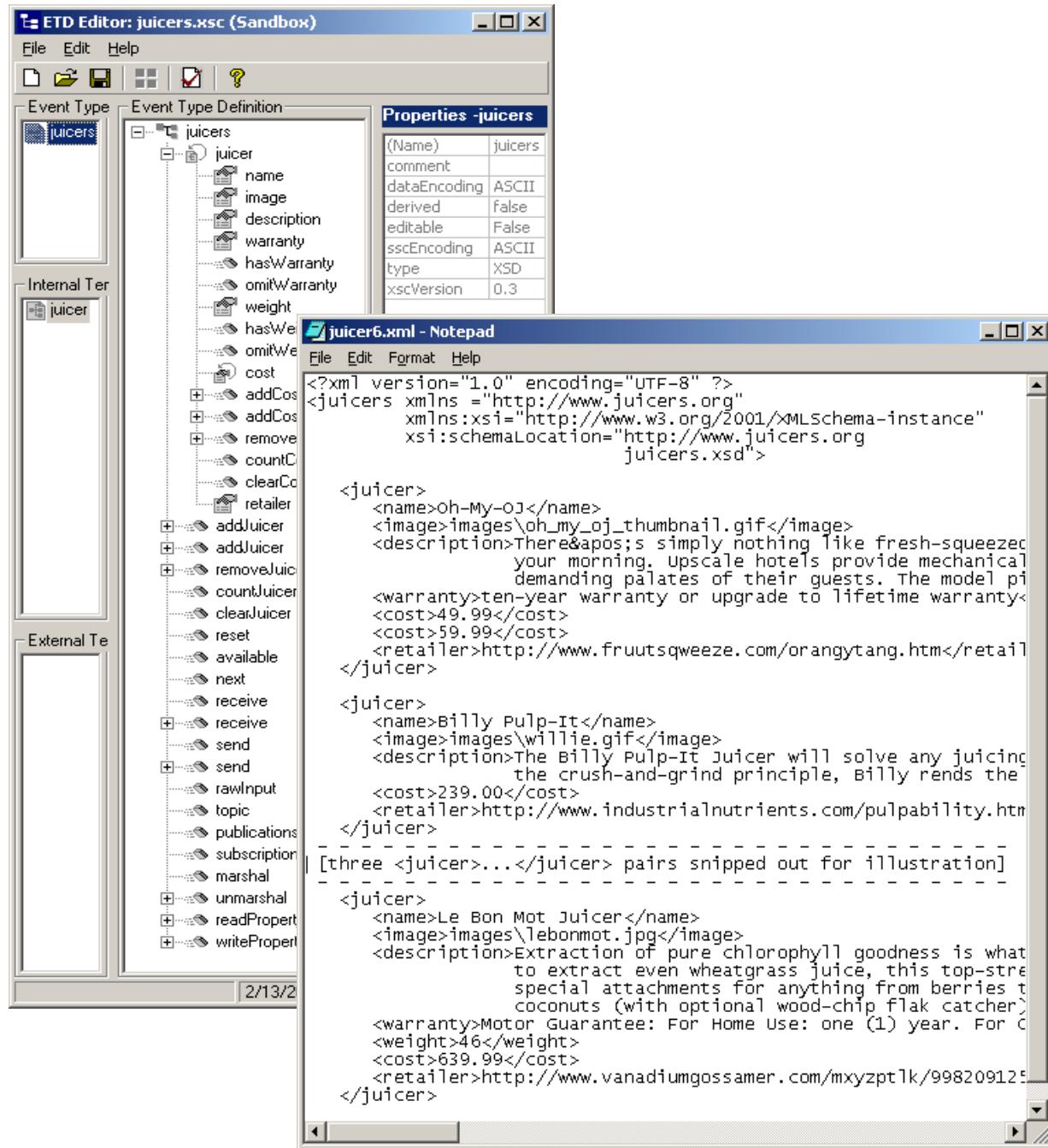


Figure 77 shows the sample ETD next to the data file it parsed:

- The ETD, **juicers.xsc** (of type XSD), contains one repeating node, **juicer**, which is an instance of internal template **juicer.xsc**. The nodes in **juicer.xsc** are XML tags like `<name>`, `<image>`, `<description>`, `<warranty>`, `<cost>`, and `<retailer>` that classify data extracted from web pages that advertise juicers for home or commercial use.
- The test data file, **juicer6.xml**, contains data harvested from six such web pages.

Figure 77 Sample ETD and Sample Test Data



Benefits of the ETD Tester

Making a practice of validating your ETDs against sample data has several benefits:

- You will gain an understanding of how the data will appear to the e*Gate system, allowing you to correct mistakes or false starts and to generally tailor the ETD to your data needs before putting it into production.
- You can double-check hypothetical and worst-case scenarios in advance, rather than waiting to diagnose and debug them after the fact.

- If you validate against historical data files, you can gain a better understanding of the data, including typical cases, special cases, and unusual conditions. This can help you to create more efficient tools, such as internal and external templates.
- You can easily create a suite of test data files for cross-comparisons.

To test an ETD against sample data

- 1 On the toolbar or **File** menu of the ETD Editor, click **Run Test** .

Note: *The ETD file must be compiled and saved before it can be tested. Java-enabled ETDs that were built using versions of e*Gate earlier than 4.5.2 are ineligible for testing.*

- 2 In the **Open** dialog box, locate and select a sample data file to be parsed. Successful results are displayed in the Test Tree pane. See [Figure 76 on page 183](#).
- 3 To troubleshoot errors or validate the data with alterations, you can click **Edit Data** to open the file in a text editor. See [Figure 77 on page 184](#). After editing the data file, you can overwrite the original file or else save it under a different name or location.
- 4 To locate and parse another data file, click **Re-Test**.

5.6.3 Promoting to Run Time

After your ETD has compiled without error and you have tested it thoroughly in the sandbox environment, you are ready to promote it.

To promote an ETD from sandbox to run time

- On the **File** menu, click **Promote**.

A dialog box tells you which files have been successfully promoted; see Figure 78. The Editor remains open, but no Event Type is current.

Figure 78 Successful Promotion of an ETD



5.6.4 Global and Local Delimiters

Delimited data streams have characters or strings that causes each chunk of data to be separated from the previous and the next. In this sentence, the individual words are delimited by the blank spaces that separate them. Each sentence in this paragraph is delimited from the next by a period. In the same way, each paragraph on this page is delimited by vertical blank space.

ETD nodes are generally either fixed-length or delimited. When parsing an Event based on a fixed-length ETD node, the system sees each of its data components as having a predefined length. When parsing an Event based on a delimited ETD node, the system uses its preassigned special characters to determine the length of its various data components.

About Global Delimiters

In Java-enabled ETDs, each node and field has a **structure** property that determines whether it is delimited, fixed length, or other. Since a child node does not inherit the fixed-length or delimited character of its parent, global delimiters would seem to be unnecessary. However, because the ETD structure is strictly hierarchical, each child node has a strictly fixed rank—child of the root, grandchild of the root, third-level, fourth-level, and so on—and so a global scheme can be used to impose a default on all delimited nodes throughout the entire hierarchy. This is the purpose of *global delimiters*. See “[Using Global Delimiters](#)” on page 187.

About Global Delimiter Levels

When you set global begin and end delimiters for a particular level, and when you set the properties for that level, you are instructing the system to use those characters and those properties by default every time it needs to parse a delimited node at that particular level. Although each node can establish its own local delimiters that override the default for its level (see below), the children and grandchildren of a custom-delimited parent node are nonetheless subject to the global delimiters if they are set for the next lower levels.

About Local Delimiters and Delimiter Groups

You can establish a special set of delimiters and properties scoped to the particular node —leaf node or parent node (even the root node)—where they are set, overriding whatever delimiters and delimiter level properties are prescribed by the global delimiter settings for that level. These delimiters and properties are not inherited by a parent node’s child nodes or [great...]grandchild nodes.

About Local Delimiter Groups

When there are several begin delimiters and several end delimiters, sometimes they can be pooled: in other words, all begin delimiters can match any end delimiters. In other circumstances, however, a particular begin delimiter requires a particular end delimiter. In this case, you want to set up *local delimiter groups* to specify the matching rules. See “[Using Local Delimiters](#)” on page 188.

Using Global Delimiters

To add a global delimiter level and modify its properties

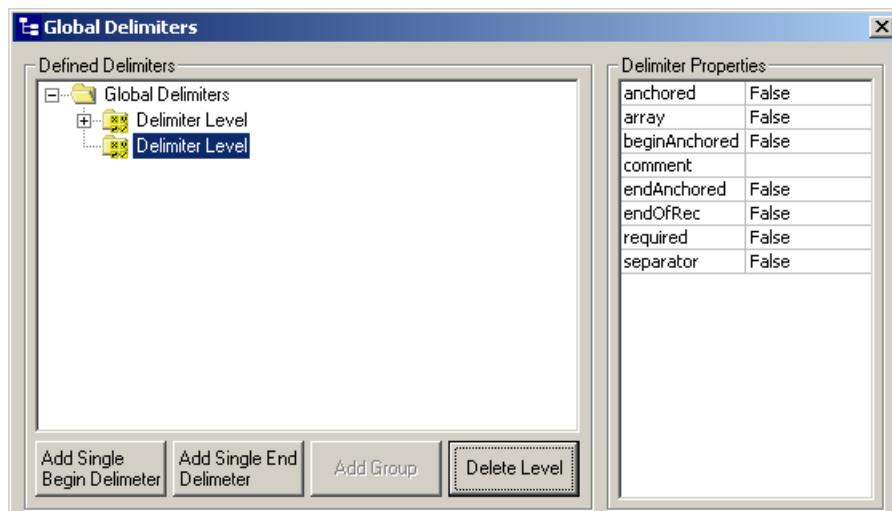
- 1 On the toolbar or **Edit** menu, click **Delimiters** .

The **Global Delimiters** dialog box appears.

- 2 To add a new global delimiter level, click the **Global Delimiters** label (if necessary), and then click **Add Level**.

The new delimiter level is appended below any existing levels. See Figure 79.

Figure 79 Global Delimiters Dialog Box - Delimiter Level



- 3 To modify an existing delimiter level, select its entry in the **Defined Delimiters** pane and make changes to the right-hand column of the **Delimiter Properties** pane. If you have other changes to make, click **Apply**.

For complete information on the meanings of the various delimiter properties, see ["Properties of Delimiters" on page 201](#).

Note: If you make a mistake, you can press **ESC** or click **Cancel**  to close the dialog box and discard all unapplied changes.

- 4 When you are satisfied with your changes, you can either add repeat steps 2 and 3 to add and modify more delimiter levels, or you can use the [procedure on page 187](#) below to add delimiters to the current level.

To add or modify begin/end delimiters for a global delimiter level

- 1 In the **Global Delimiters** dialog box, select the global delimiter level whose delimiters you want to add or modify.

- 2 Do one of the following:

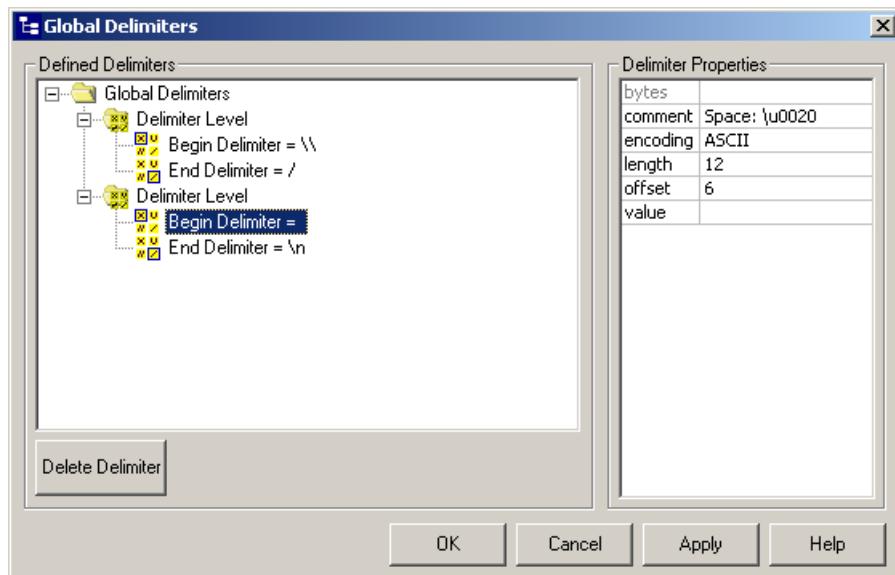
- ♦ To add a begin delimiter (which can be a string of many characters), click **Add Single Begin Delimiter**, and then add or modify its properties.

- To add an end delimiter (which can be a string of many characters), click **Add Single End Delimiter**, and then add or modify its properties.
- To modify the properties of an existing delimiter, select its entry in the **Defined Delimiters** pane, and then make changes to the right-hand column of the **Delimiter Properties** pane.

Figure 79 illustrates some common special characters being used as delimiters: At level 1, the begin delimiter is a \ (backslash character), which is input via the escape sequence \\ . Level 2 is delimited at the start by the blank space character (Unicode \u0020\—notice the comment) and at end by the newline character, input via the escape sequence \\n .

For complete information, see “[Properties of Delimiters](#)” on page 201.

Figure 80 Setting Global Begin and End Delimiters



- 3 If you make a mistake and want to abandon your current changes, press ESC or click **Cancel** ✕ to close the dialog box; or, if you are satisfied with your current changes and have other changes to make, click **Apply**; or, if you are satisfied with all your changes, click **OK**.

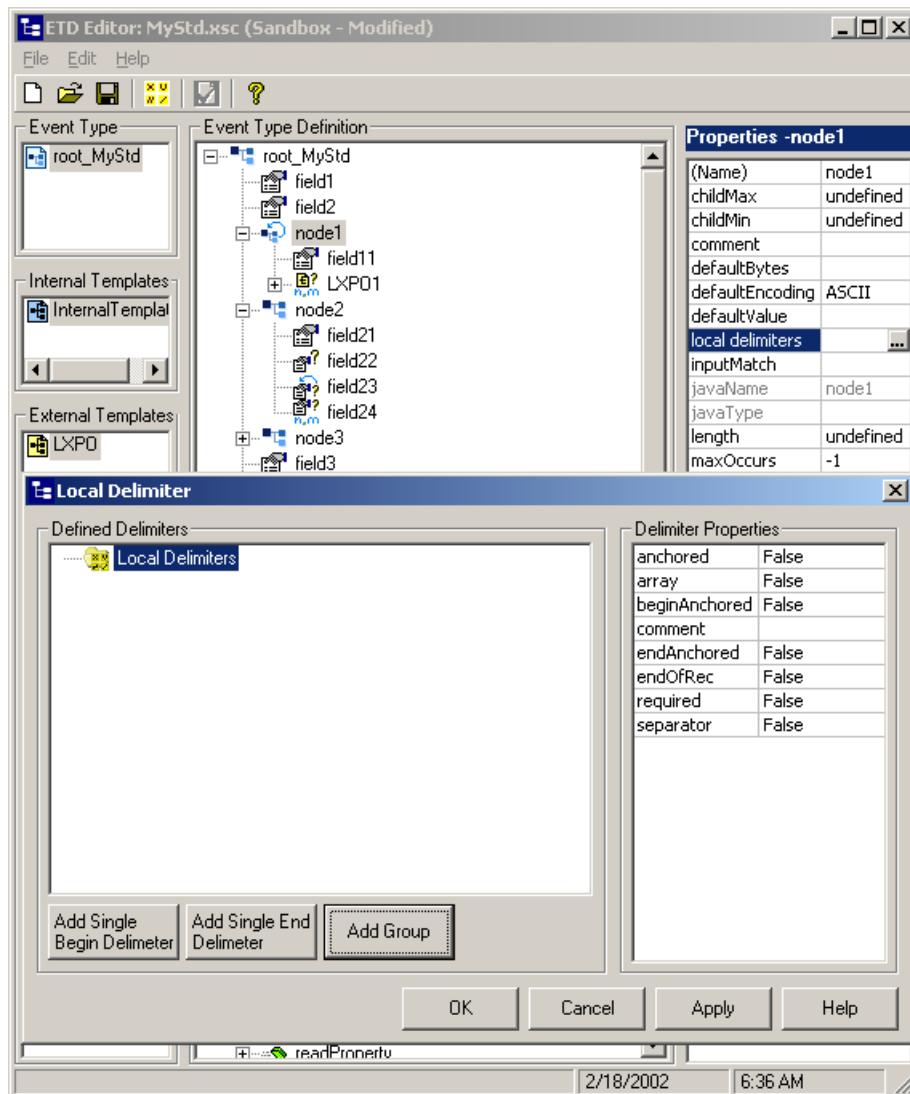
Using Local Delimiters

To add, delete, or modify the local delimiters of a delimited node

- 1 In the Event Type Definition (center) pane, highlight the node whose delimiters you want to edit.
- 2 In the Properties pane, in the **local delimiters** property, click the ... button.

The **Local Delimiters** dialog box appears. See Figure 81.

Figure 81 Local Delimiters Dialog Box - Local Delimiters Level



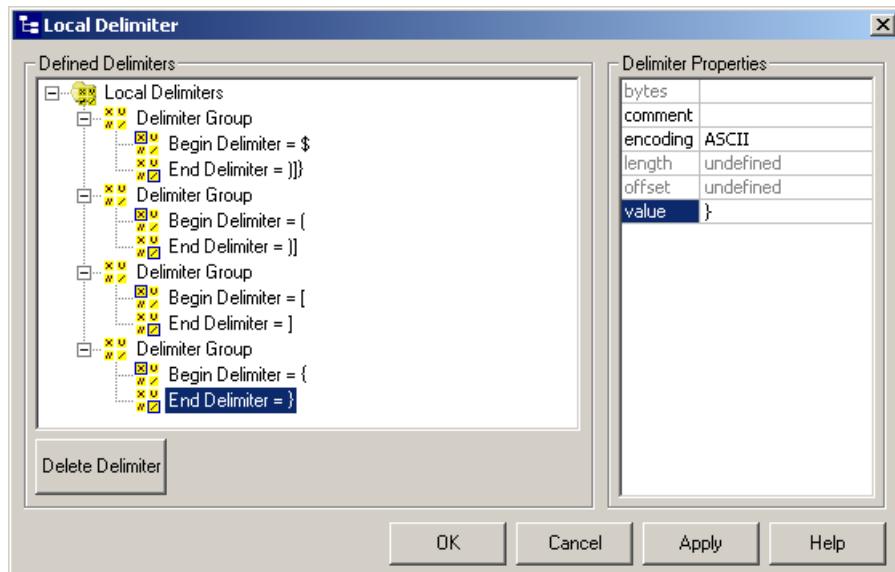
Notice the similarity to the Global Delimiters dialog box when Delimiter Level is selected (see [Figure 79 on page 187](#)). The difference between the two is that on a global level, you can set delimiters and properties for many levels, whereas at the local level, you are only setting delimiters and properties for the current level; however, at the local level, you can set up delimiter groups.

- 3 Do one of the following:
 - To add a begin delimiter (which can be a string of many characters), click **Add Single Begin Delimiter**, and then add or modify its properties.
 - To add an end delimiter (which can be a string of many characters), click **Add Single End Delimiter**, and then add or modify its properties.
 - To modify the properties of an existing delimiter, select its entry in the **Defined Delimiters** pane, and then make changes to the right-hand column of the **Delimiter Properties** pane.

- ♦ To add a delimiter group, click the **Local Delimiters** label (if necessary), and then click **Add Group**. Repeat as needed.

An example of a local delimiter group is shown in Figure 82.

Figure 82 Example of a Local Delimiter Group



In the example shown above:

- ♦ The begin delimiter \$ can take any of the following as an end delimiter:)] or }
 - ♦ The begin delimiter (can take either of the following as an end delimiter:) or]
 - ♦ The begin delimiter [can only take the following as an end delimiter:]
 - ♦ The begin delimiter { can only take the following as an end delimiter: }
- 4 If you make a mistake and want to abandon your current changes, press ESC or click **Cancel** to close the dialog box; or, if you are satisfied with your current changes and have other changes to make, click **Apply**; or, if you are satisfied with all your changes, click **OK**.

5.7 Standard ETD Properties

The ETD Editor allows you to see the properties of each node, field, and method. This section lists and explains the node and field properties for the Standard ETD; for explanations of the Standard ETD methods, see [Chapter 7 “Java Collaboration Rules” on page 260](#).

Note: *Nodes, field, and method properties for imported ETDs are not covered in this book. For a list of the corresponding documentation for imported ETDs, see [Building an Imported ETD](#) on page 170.*

5.7.1 Event Type Properties

Table 19 lists and explains the properties of the Event Type instance itself, for Events based on the Standard Event Type Definition.

Table 19 Properties of the Standard Event Type

Property Name	Type	Explanation	Permitted, Default, and Special Values
(Name)	java.lang.String	Provides an image of the (Name) property of the root node. Must be a Java identifier, different from the value of packageName , and must not be a DOS device name like PRN or COM1.	For permissible values, see “About Node Names” on page 174 .
comment	java.lang.String	Free text, up to 1024 characters long. You can use \u as an escape character to specify Unicode characters.	Default: (empty). Preset to: (empty).
dataEncoding	java.lang.String	Allows you to use data whose character encoding is different from the encoding of the ETD as a whole.	Default: (current value of the sscEncoding property for this Event Type). Preset to: ASCII
derived	boolean	Whether this ETD was created by deriving it from an imported ETD. (Derived and imported ETDs are not covered in this guide.)	Must be: false By definition, Standard ETDs are never Derived ETDs.
editable	boolean	Whether or not the fields of this Event Type can be saved with modifications.	Default: true Preset to: true
sscEncoding	java.lang.String	Character encoding to set as default for the current ETD.	Default: ASCII Preset to: ASCII
type	java.lang.String	For the Standard ETD, this is SSC by definition.	Must be: SSC
xscVersion	java.lang.String	A mandatory identifier used internally to set parsing rules.	Must be: 0.6 (for this version)

5.7.2 Properties of Root and Element Nodes (Parent Nodes)

Table 20 lists and explains the properties of parent nodes (including root nodes of the Event Type itself and any templates it contains) in a Standard ETD. For methods associated with these elements, see “[Methods for Elements and Fields of ETDs](#)” on page 328.

Note: Here, a parent node is defined as a node whose *type* property has the value CLASS.

Table 20 Properties of Standard ETD Parent Node Elements (type=CLASS)

Property Name	Type	Explanation	Permitted, Default, and Special Values
(Name)	java.lang.String	A descriptive string you enter to identify this parent element.	For permissible values, see “ About Node Names ” on page 174. Preset to: Element< n > (except for the root node).
childMax	int	Specifies an upper limit for the number of children nodes directly under this element. Corresponds to the Monk NofN modifier; see max-rep in the <i>Monk Developer’s Reference</i> .	Nonnegative integer, or -1. ▪ -1 means “unbounded”. Default: (undefined). Preset to: undefined
childMin	int	Specifies a lower limit for the number of children nodes directly under this element. Corresponds to the Monk NofN modifier; see max-rep in the <i>Monk Developer’s Reference</i> .	Nonnegative integer. Default: (undefined). Preset to: undefined
comment	java.lang.String	Holds free text, up to 1,024 characters long. You can use \u as an escape character to specify Unicode characters.	Default: (undefined). Preset to: (empty).
defaultBytes	java.lang.String	If present, overrides the current setting of the defaultValue property for this element.	Default: (current value of the defaultValue property for this node). Preset to: (empty).
defaultEncoding	java.lang.String	Specifies the character encoding to use for the value of defaultValue (or for the value of defaultBytes if it is set).	Preset to: ASCII .
defaultValue	java.lang.String	Specifies a value to be given to this node if it is marshalled but contains no data. You can use \u as an escape character to specify Unicode characters.	Default: (undefined). Preset to: (empty). Examples: ▪ (no middle initial) ▪ 000-00-0000 ▪ no data

Table 20 Properties of Standard ETD Parent Node Elements (type=CLASS) (Continued)

Property Name	Type	Explanation	Permitted, Default, and Special Values
encoding	java.lang.String	(not applicable for parent elements)	Preset to: (empty).
enumType	java.lang.String	(not applicable for Standard ETDs)	Preset to: (empty).
fixedValue	java.lang.String	(not applicable for Standard ETDs)	Preset to: (empty).
format	java.lang.String	(not applicable)	Preset to: (empty).
inputMatch	java.lang.String	Specifies a Monk regular expression to be applied against the data contained within this parent element.	Preset to: (empty). See “ Creating Delimited ETDs ” on page 217.
javaName	java.lang.String	Contains the original name for this element, unless explicitly modified later by the user.	Default: (the original value of the (Name) property for this node, after forcing any leading lowercase alphabetic character to uppercase).
javaType	java.lang.String	Specifies the Java data type for this element.	Must be one of the following: <ul style="list-style-type: none"> ▪ boolean ▪ byte ▪ byte[] ▪ char ▪ double ▪ float ▪ int ▪ java.lang.String ▪ long ▪ short Default: java.lang.String Preset to: java.lang.String
length	int	Specifies the length of the data stream in this element. Applies to delimited elements only.	Nonnegative integer. Default for delimited nodes: 0 Preset for delimited nodes: 0 Default and preset value for fixed nodes: undefined
lengthFrom	int	(not applicable for Standard ETDs)	Nonnegative integer. Preset to: undefined
lengthSize	int	(not applicable for Standard ETDs)	Positive integer. Preset to: undefined
local delimiters		Opens the Local Delimiter dialog box. For details, see “ Global and Local Delimiters ” on page 186.	

Table 20 Properties of Standard ETD Parent Node Elements (type=CLASS) (Continued)

Property Name	Type	Explanation	Permitted, Default, and Special Values
maxOccurs	int	Specifies an upper limit on the number of times this parent element can occur. For the equivalent in Monk, see Table 32 on page 230 . Any element whose maxOccurs value is unbounded or greater than its minOccurs value has an associated count() method; see "count_MyNode_0" on page 329 .	Nonnegative integer, or -1: <ul style="list-style-type: none"> ▪ -1 means unbounded. Default: 1 Preset to: 1
member	java.lang.String	Applies only to a node that is an alias for an internal template. When applicable, contains the name of the internal template.	Preset to: (empty)
minOccurs	int	Specifies a lower limit to the range of the number of times this parent element can occur. For the equivalent in Monk, see Table 32 on page 230 . Any element whose minOccurs value is 0 has an associated has() method; see "has_MyNode_0" on page 331 .	Preset to: 1
offset	int	Specifies the start position of this parent element relative to its own parent. (Note that root nodes do not have this property.)	Preset to: undefined
optional	boolean	When set to true , specifies that this parent element can occur zero times, even if minOccurs is greater than 0. (Note that root nodes do not have this property.)	Preset to: false
order	java.lang.String	Specifies whether data in this node's children can appear in any order, or if they must be presented in sequence.	Must be one of the following: <ul style="list-style-type: none"> ▪ any ▪ sequence Preset to: sequence

Table 20 Properties of Standard ETD Parent Node Elements (type=CLASS) (Continued)

Property Name	Type	Explanation	Permitted, Default, and Special Values
precedence	java.lang.String	When an end-delimiter is used at more than one node level, specifies whether the delimiter terminates the current=lowest node (child) or the highest node (parent) having the same delimiter.	<p>Must be one of the following:</p> <ul style="list-style-type: none"> ▪ child ▪ parent <p>Preset to: child</p> <p>Example: If your global begin-delimiters and end-delimiters are set as follows:</p> <ul style="list-style-type: none"> ▪ Level 1: < > ▪ Level 2: { } ▪ Level 3: < > ▪ Level 4: { } ▪ Level 5: < > <p>and if a > character is encountered at level 5, then:</p> <ul style="list-style-type: none"> ▪ A setting of child terminates level 5 ▪ A setting of parent terminates level 3
public	boolean	(not applicable for Standard ETDs)	Preset to: false
readOnly	boolean	(not applicable for Standard ETDs)	Preset to: false
reference	java.lang.String	For Standard ETDs, applies only to a node that is an alias for an external template. When applicable, contains the path and name of the external template.	
scavOutput	boolean	When set to true , specifies that exactly one instance of the first character specified in the scavenger property should be prepended to the element.	Preset to: false
scavenger	java.lang.String	Specifies the characters to be stripped out if they appear at the start of this element.	<p>Preset to: (empty)</p> <p>Examples:</p> <ul style="list-style-type: none"> ▪ \u0020 (blank space) ▪ \u0009 (tab) ▪ \u000D (carriage-return)
structure	java.lang.String	Specifies the method for determining the boundary of the data in this element.	<p>Must be one of the following:</p> <ul style="list-style-type: none"> ▪ array ▪ delim ▪ fixed ▪ set <p>Default: (none)</p> <p>Preset to: delim</p>

Table 20 Properties of Standard ETD Parent Node Elements (type=CLASS) (Continued)

Property Name	Type	Explanation	Permitted, Default, and Special Values
type	java.lang.String	<p>By definition, all parent nodes (including the root node) in a Standard ETD are of type CLASS.</p> <ul style="list-style-type: none"> ▪ For properties of <i>field</i> nodes in a Standard ETD, see Table 21 on page 197. ▪ For properties of other ETDs, refer to the user's guide for the corresponding e*Way, ETD Builder, or toolkit. 	<p>Default: (none) Must be: CLASS</p>

5.7.3 Properties of Field and Reference Nodes (Leaf Nodes)

Table 21 lists and explains the properties of field and reference nodes in a Standard ETD. For methods associated with these fields, see “[Methods for Elements and Fields of ETDs](#)” on page 328.

Note: A field node is defined as a node whose **type** property has the value **FIELD**; a reference node, such as a template instance, has a type of **REFERENCE** and a few additional properties but is otherwise almost identical a field node. In a Standard ETD, a field node is equivalent to a leaf node: Every field node has exactly one parent node and zero children.

Table 21 Properties of Standard ETD Field Nodes (type=**FIELD**)

Property Name	Type	Explanation	Permitted and Default Values
(Name)		Allows you to enter or modify a descriptive name to identify this field.	For permissible values, see “ About Node Names ” on page 174. Preset to: Field<n>
childMax	int	(not applicable for leaf nodes)	Preset to: undefined
childMin	int	(not applicable for leaf nodes)	Preset to: undefined
comment	java.lang.String	Contains free text, up to 4,095 characters long. You can use \u as an escape character to specify Unicode characters.	Preset to: (empty).
defaultBytes	java.lang.String	If present, overrides the current setting of the defaultValue property for this field.	Preset to: (empty).
defaultEncoding	java.lang.String	Specifies the character encoding to use for the value of defaultValue (or for the value of defaultBytes if it is set).	Preset to: ASCII .
defaultValue	java.lang.String	Specifies a value to be given to this field if it is marshalled but contains no data. You can use \u as an escape character to specify Unicode characters.	Preset to: (empty).
local delimiters		Opens the Local Delimiter dialog box. For details, see “ Global and Local Delimiters ” on page 186.	

Table 21 Properties of Standard ETD Field Nodes (type=FIELD) (Continued)

Property Name	Type	Explanation	Permitted and Default Values
encoding	java.lang.String	If present, overrides the default encoding for this field. (The default encoding for a field can be learned using the getUnmarshalEncoding() and getMarshalEncoding() methods). This allows you to specify a nondefault character encoding for this particular field.	Default: (current value of etd.dataEncoding — in other words, the dataEncoding property for the Event Type). Preset to: (empty).
format	java.lang.String	Specifies a directive for parsing and rendering data in this field, using the Java DecimalFormat class. For example: <ul style="list-style-type: none"> ▪ ; separates positive from negative representations ▪ 0 indicates a required digit ▪ # indicates an optional digit ▪ . indicates a decimal point ▪ - indicates a minus sign ▪ , indicates a grouping separator (such as thousands) ▪ E separates mantissa from exponent in scientific notation ▪ ' quotes a special character For complete information, see java.sun.com/j2se/1.3/docs/api/java/text/DecimalFormat.html	Default: #;# Preset to: (empty). Examples: <ul style="list-style-type: none"> ▪ '\$###,##0.00;('\$###,##0.00) uses accounting convention for negative numbers ▪ 0.00E0;-0.00E0 uses scientific notation ▪ ##0.0E0;-##0.0E0 uses engineering notation ▪ ## o'clock uses a special character
inputMatch	java.lang.String	Specifies a Monk regular expression to be applied against the data contained within this field.	Preset to: (empty). See “ Creating Delimited ETDs ” on page 217.
javaName	java.lang.String	Contains the original name for this field, unless the javaName property has been explicitly modified later by the user.	Default: (the original value of the (Name) property for this node, after forcing any leading lowercase alphabetic character to uppercase).

Table 21 Properties of Standard ETD Field Nodes (type=FIELD) (Continued)

Property Name	Type	Explanation	Permitted and Default Values
javaType	java.lang.String	Specifies the Java data type for this field.	Must be one of the following: <ul style="list-style-type: none"> ▪ boolean ▪ byte ▪ byte[] ▪ char ▪ double ▪ float ▪ int ▪ java.lang.String ▪ long ▪ short Default: java.lang.String Preset to: java.lang.String
length	int	For fixed-length fields (only), Specifies the length of the field.	Nonnegative integer. Default for delimited fields: 0 Preset for delimited fields: 0 Default and preset value for fixed-length fields: undefined
maxOccurs	int	Specifies an upper limit to the range of the number of times this field can occur. For the equivalent in Monk, see Table 32 on page 230 . Any field whose maxOccurs value is unbounded or greater than its minOccurs value has an associated count() method; see "count_MyNode_0" on page 329 .	Positive integer, or -1: <ul style="list-style-type: none"> ▪ -1 means unbounded. Default: 1 Preset to: 1
member	java.lang.String	<i>Applies only to a node that is an instance of an internal template.</i>	<i>Not applicable to field nodes.</i> For instances of internal templates, contains the name of the internal template itself.
minOccurs	int	Specifies a lower limit to the range of the number of times this field can occur. For the equivalent in Monk, see Table 32 on page 230 . Any field whose minOccurs value is 0 has an associated has() method; see "has_MyNode_0" on page 331 .	Preset to: 1
offset	int	Specifies the start position of this field relative to its parent.	Preset to: undefined

Table 21 Properties of Standard ETD Field Nodes (type=FIELD) (Continued)

Property Name	Type	Explanation	Permitted and Default Values
optional	boolean	When set to true , specifies that this field can occur zero times (even if minOccurs is greater than 0).	Preset to: false
order	java.lang.String	Specifies whether data in this field can appear in any order, or if it must be presented in sequence.	Must be one of the following: <ul style="list-style-type: none"> ▪ any ▪ sequence Preset to: sequence
precedence	java.lang.String	When an end-delimiter is used at more than one node level, specifies whether the delimiter terminates the current-lowest node (child) or the highest node (parent) having the same delimiter.	Must be one of the following: <ul style="list-style-type: none"> ▪ child ▪ parent Preset to: child Example: If your global begin-delimiters and end-delimiters are set as follows: <ul style="list-style-type: none"> ▪ Level 1: < > ▪ Level 2: { } ▪ Level 3: < > ▪ Level 4: { } ▪ Level 5: < > and if a > character is encountered at level 5, then: <ul style="list-style-type: none"> ▪ A setting of child terminates level 5 ▪ A setting of parent terminates level 3
readOnly	boolean	(not applicable for Standard ETDs)	Preset to: false
reference	java.lang.String	<i>Applies only to a node that is an instance of an external template.</i>	<i>Not applicable to field nodes.</i> For instances of external template, contains the relative path and file name of the external template.
scavOutput	boolean	When set to true , specifies that one instance of the first character specified in the scavenger property should be prepended to the field. scavOutput is meaningful only if the scavenger property is set.	Preset to: false
scavenger	java.lang.String	Specifies the characters to be stripped out if they appear at the start of this field.	Preset to: (empty) Examples: <ul style="list-style-type: none"> ▪ \u0020 (blank space) ▪ \u0009 (tab) ▪ \u000D (carriage-return)

Table 21 Properties of Standard ETD Field Nodes (type=**FIELD**) (Continued)

Property Name	Type	Explanation	Permitted and Default Values
structure	java.lang.String	Specifies the method for determining the boundary of the data in this field.	Must be one of the following: <ul style="list-style-type: none"> ▪ array ▪ delim ▪ fixed ▪ set Default: (none) Preset to: delim
type	java.lang.String	By definition, a field node in a Standard ETD is of type FIELD . <ul style="list-style-type: none"> ▪ For properties of parent elements in a Standard ETD, see Table 20 on page 192. ▪ For properties of other ETDs, refer to the user's guide for the corresponding e*Way, ETD Builder, or toolkit. 	Must be: FIELD A value of REFERENCE indicates that this node is an instance of an internal or external template.

5.7.4 Properties of Delimiters

Table 22 lists and explains the properties of global and local delimiters and delimiter levels. These properties apply only to delimited elements or fields—in other words, elements and fields whose **structure** property is set to **delim**. For more information, see “[Global and Local Delimiters” on page 186](#).

Table 22 Properties of Global and Local Delimiters

Property Name	Type	Explanation	Permitted, Default, and Special Values
anchored	boolean	Applies to a Delimiter Level only. When set to true , indicates that the delimiters must be the starting and ending characters of this element.	Preset to: false
array	boolean	Applies to a Delimiter Level only. When set to true , indicates that if an array node appears at this level, the delimiter at this level should be used; but if no array node appears at this level, this level should be skipped and the non-array delimiter at the next lower level should be used.	Preset to: false
beginAnchored	boolean	Applies to a Delimiter Level only. When set to true , indicates that the begin delimiter must occur at the current location of the Event data.	Preset to: false

Table 22 Properties of Global and Local Delimiters (Continued)

Property Name	Type	Explanation	Permitted, Default, and Special Values
bytes		Applies to a Begin/End Delimiter only. The byte sequence that corresponds to the value of this delimiter.	
comment	java.lang.String	Holds free text, up to 1,024 characters long. You can use \u as an escape character to specify Unicode characters.	Default: (undefined). Preset to: (empty).
encoding	java.lang.String	Applies to a Begin/End Delimiter only. The character encoding used for this delimiter's value .	Preset to: ASCII
endAnchored	boolean	Applies to a Delimiter Level only. When set to true , indicates that the end delimiter must occur at the current location of the Event data.	Preset to: false
endOfRec	boolean	Applies to a Delimiter Level only. When set to true , causes the final end-delimiter character to be appended to the end of the data.	Preset to: false
length	int	Applies to a Begin/End Delimiter only. Specifies the length of this delimiter.	Nonnegative integer. Default: undefined Preset to: undefined
offset	int	Applies to a Delimiter Level only. Specifies the start position of this delimiter.	Preset to: undefined
required	boolean	Applies to a Delimiter Level only. When set to true , specifies that delimiters must be specified locally (and thus not inherited from the settings in the Global Delimiters dialog box).	Preset to: false
separator	boolean	Applies to a Delimiter Level only. When set to true , specifies that a field repetition character is required. See "Specifying HL7 Repeating Fields" on page 231 .	Preset to: false
value	java.lang.String	Applies to a Begin/End Delimiter only. Specifies the character or characters to be used as a delimiter.	Default: (none set) To specify a \ (backslash), use \\. You can specify any character as \u####, where #### is the character's Unicode representation.

Chapter 6

Monk Event Type Definition Editor

This chapter explains how to use the Monk Event Type Definition (ETD) Editor feature to create, add, and define Monk ETDs (**.ssc** files) in the e*Gate system.

6.1 Monk ETD Editor Overview

Monk ETDs created using this SeeBeyond ETD Editor are written in the Monk programming language.

You can create and modify these ETDs using the Monk ETD Editor graphical user interface (GUI). This feature helps you to make sure that your ETDs conform to the specific requirements of the e*Gate system.

The Monk ETD Editor contains a specialized GUI called the ETD Tree that allows you to create complete definitions for your Event Types, using a branched setup. The ETD Tree lets you place Event nodes into a hierarchical arrangement that represents ETD structures in a convenient, vertical and graphical format.

This chapter explains how to use the Monk ETD Editor under the following sections:

- [“Getting Started” on page 204](#)
- [“ETD Editor Window” on page 208](#)
- [“Edit Menu” on page 212](#)
- [“Basic ETD Operations” on page 242](#)
- [“Working With ETD Templates” on page 254](#)

6.2 Getting Started

You create the definitions of your Event Types by representing them in the ETD Editor window as a set of interrelated data components (nodes) in the ETD Tree.

Note: As an option, you can set up ETDs using a text editor and Monk, SeeBeyond’s programming language. If you prefer this option, see the **Monk Developer’s Reference**.

6.2.1 ETD Creation and Nodes

When you create an ETD, you build an Event's generic components, for example, data segments and fields. The ETD Editor calls these components nodes, and represents them graphically.

Working with Nodes

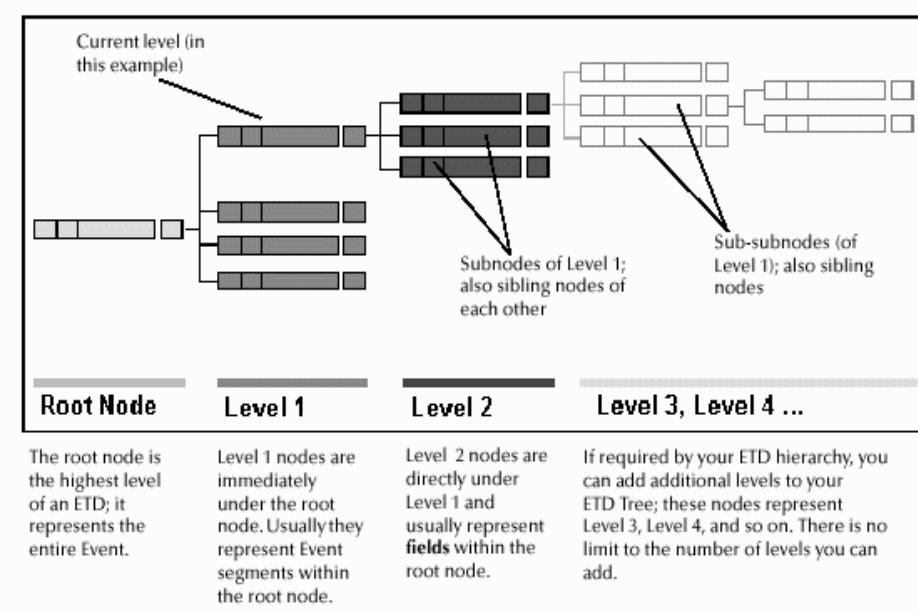
Each ETD has a single root node that represents the entire Event. You can add additional nodes to more fully describe the organization of data within an Event Type. For each node in an ETD, you must specify the following types of information:

- **General:** Basic node information, for example, the node's name and whether it is delimited or fixed.
- **Content:** Input data, output data, and scavenger characters the node contains.
- **Repetition:** Node data order and repetition characteristics.
- **Delimiters:** Information describing the node's delimiter properties, if present.

Node-level Structure

[Figure 83 on page 205](#) shows the node-level structure of the ETD Tree as represented in the ETD Editor window. Keep in mind that the terms "subnode" and "sub-subnode" are relative to the current working level and can apply to any level. In Figure 83, Level 1 is the current level.

Figure 83 ETD Editor Tree



Parent and Child Nodes

Any subnode of a given node is called a child node, and the given node, in turn, is the child's parent. Sibling nodes are nodes on the same level under the same parent node.

See [Figure 84 on page 208](#) for an illustration of this node structure in the ETD Editor window.

Naming Nodes

When you create ETDs, you give each node a name that becomes its own identifier in the ETD Tree. Table 23 shows the acceptable characters for node names.

Table 23 Acceptable Node-name Characters

Description	Character(s)
All letters	A through Z a through z
All digits	0 through 9
Plus sign	+
Dash	-
Asterisk	*
Slash (forward)	/
Equal sign	=
Exclamation mark	!
Question mark	?
Dollar sign	\$
Underscore	_
Ampersand	&
Caret	^
(Japanese only)	All kanji characters, using shift-JIS (S-JIS) encoding

Note: The name of a node is the source of the node's label in the ETD Tree. Create each name for your ease of reference. Use naming conventions that you can easily recognize later on.

Rules for Node Naming

Name nodes according to the following conventions:

- **First Character:** Can be any character except
 - ◆ Digits (0 through 9)
 - ◆ Plus signs (+)
 - ◆ Dashes (-)
- **Case:** Node-name interpretation is case sensitive; that is, the system would see "My_Node" and "my_node" as two separate names.
- **Number of Characters:** For your convenience, do not use more than 40 characters in a single node name.

- **Uniqueness:** It is desirable to make each root node name unique within a schema. This ensures that you do not confuse one ETD with another. If you want to access nodes by name, you must give unique names to all the nodes within a given level.
- **Total length:** For very deep ETDs (with twenty or more levels), restrict the number of characters per node name even further to ensure that the total fully qualified path of the node—that is, all ancestor nodes concatenated with the leaf node—does not exceed 1000 characters.

6.2.2 Before Using the ETD Editor

Before you start using the ETD Editor, consider the following Event Type properties:

- **Specifications:** All your Event Type specifications must be complete and correct before the Event Types you define can pass through the e*Gate environment. An ETD is required for all e*Gate-processed Events.
- **External Templates:** You must list all of the Event Types that would be appropriate for use as external templates. If there are certain formats that recur throughout any Event Type, then make sure to incorporate those formats into templates you can reuse as you create more ETDs. This operation ensures a consistent format and saves you time. See “[Using External Templates](#)” on page 254 for details.
- **Level of Detail:** In order to identify and translate Events, the e*Gate system, at a minimum, requires you to define ETDs at the root-node level. However, you must define most Events down to the node level assigned to the data field. This practice allows you to specify any system-required Event identification and/or transformation instructions needed later on.
- **Amount of Detail:** When you define an Event at a particular node level, it is desirable to define that level completely. If you do not define elements of any Event Type adequately, the system is unable to parse Events represented by that type, and these Events fail to pass through the e*Gate network correctly.

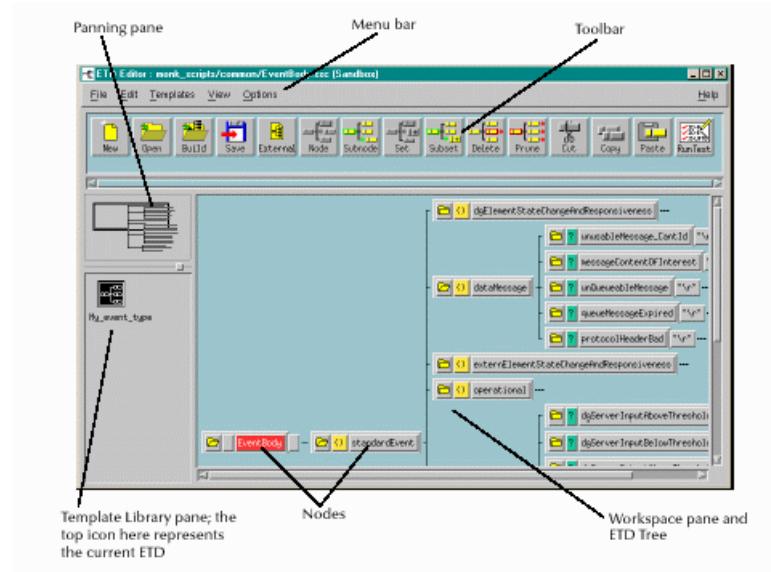
See the sections under “[Edit Menu](#)” on page 212 for complete details on the correct Event Type properties that you must define for e*Gate.

Caution: *The ETD Editor does not validate your Events completely. It is up to you to make sure that you build ETD Trees using valid Event element types and in accordance with your predefined Event Type specifications.*

6.3 ETD Editor Window

Figure 84 below shows an example of the ETD Editor window with a sample ETD Tree setup displayed.

Figure 84 ETD Editor and Tree Structure



The window above has the following major sections:

- **Menu bar:** contains the ETD Editor menus.
- **Toolbar:** contains buttons that allow you easy access to often-used features.
- **Workspace pane:** provides your primary working area where you build the ETD Tree and node picture of your ETDs.
- **Panning pane:** shows a thumbnail overview diagram of the current ETD Tree's structure.
- **Template Library pane:** displays a button for the current ETD and each template in the Template Library; see [Using External Templates](#) on page 254.

To access the ETD Editor window

From the e*Gate Enterprise Manager window, click the **ETD Editor** command button (or, on the **Tools** menu, choose **ETD Editor**). The ETD Editor window first appears empty, with no graphic ETD Tree setups displayed.

The rest of this section explains the basic e*Gate GUI features of the ETD Editor.

6.3.1 Toolbar

Table 17 shows the toolbar buttons and the functionality of each.

Table 24 Toolbar Buttons and Functions

Button	Name	Function
	New	To create a new ETD, click the New button. The New ETD dialog box appears; this is where you define the initial properties of your new ETD, including its type and file name.
	Open	Allows you to open an ETD file. The Open ETD dialog box appears. Select the desired ETD to open. The e*Gate Editor displays the selected ETD in the Workspace pane.
	Build	Builds an ETD for you, based on the parameters you provide: that is, sample input data, delimiters, and the number of levels to include in the definition. See Table 25 on page 211 for more details.
	Save	Allows you to save the current ETD. The ETD is saved under the name assigned to it in the New ETD dialog box.
	External	Allows you to include an external template in your ETD. See Table 27 on page 213 for more details.
	Node	Allows you to add a node on the same level of the ETD Tree hierarchy as the one you have currently selected. See Add Node in Table 26 on page 212 for more details.
	Subnode	Allows you to add a node on level below the node you have currently selected. See Table 26 on page 212 for more details.

Button	Name	Function (Continued)
	Set	Allows you to add a node set (placeholder) that is on the same level of the ETD Tree hierarchy as the one you have currently selected. See Table 26 on page 212 for more details.
	Subset	Allows you to add a node set (placeholder) that is one level below the node you have currently selected. See Add Subset in Table 26 on page 212 for more details.
	Delete	Allows you to delete a node and all its subnodes.
	Prune	Allows you to delete subnodes belonging to a node. See Table 26 on page 212 for more details.
	Cut	Allows you to delete a node and its associated subnodes and temporarily place them in an internal buffer. See Table 26 on page 212 for more details.
	Copy	Allows you to temporarily place a selected node and its associated nodes in an internal buffer. See Table 25 on page 211 for more details.
	Paste	Allows you to paste the contents of an internal buffer into your ETD Tree. See Table 26 on page 212 for more details.
	Run Test	Allows you to test your ETD against sample input data and tells you whether your ETD works. See Table 25 on page 211 for more details.

6.3.2 Menu Bar

The Monk ETD Editor has the following menus:

- **File**
- **Edit**
- **Templates**

- **View**
- **Options**
- **Help**

The next section explains the commands available under each of these menus.

File Menu

Table 25 below lists and describes the **File** menu commands.

Table 25 ETD Editor File Menu

Command	Function
New	Opens the New ETD dialog box, where you define basic details about your new ETD file, including the file name (extension must be .ssc). See “ Creating ETD Files ” on page 215 for details.
Open	Displays the Open ETD dialog box, allowing you to choose an ETD to display. See “ Opening ETDs ” on page 242 for details.
Build	Builds an ETD for you, based on the parameters you provide, that is, sample input data, delimiters, and the number of levels to include in the definition. See “ Using the Build Tool ” on page 242 for details.
Save and Edit Using External Editor	Allows you to open, edit, and save an ETD file, using a file editor outside the e*Gate system.
Reload from Local Machine	Allows you to reload an ETD file into e*Gate from a local system.
Save	Saves the open ETD under its current file name.
Save As	Allows you to save your ETD to a new name. Click Save to complete the operation. See “ Saving ETDs Under New Names ” on page 246 for details.
Export to DTD	Opens an Export To dialog box that allows you to save the current ETD file as a DTD (XML-type) file with a .dtd extension. See SeeBeyond’s <i>XML Toolkit</i> for details. NOTE: This feature is only available with the XML Converter add-on.
Promote to Run Time	Allows you to move the current ETD file out of a Sandbox (nonoperating) state and into a state ready for operation within the system.
Remove	Allows you to delete the current version of the ETD file from your personal Sandbox folder.
Default Delimiters	Allows you to specify the default delimiters used to parse Event elements. See “ Defining Default Delimiters ” on page 217 for details.
Run Test	Tests your ETD against sample input data and tells you whether your ETD works. See “ Testing ETD Files ” on page 247 for details.
Main Comment	Opens a dialog box where you can enter notes about your ETD. See “ Creating ETD Comments ” on page 250 for details.
Close	Closes the ETD Editor window and exits the feature.

Edit Menu

Table 26 below lists and describes the **Edit** menu commands.

Table 26 ETD Editor Edit Menu

Command	Function
Add Node	Inserts a node in the ETD Tree on the same level as the currently selected node. See “ Adding Delimited-ETD Nodes ” on page 228 or “ Adding Fixed-ETD Nodes ” on page 235 for details.
Add Subnode	Inserts a node in the ETD Tree one level below the currently selected node. the inserted node is associated with the selected node. See “ Adding Delimited-ETD Nodes ” on page 228 or “ Adding Fixed-ETD Nodes ” on page 235 for details.
Add Set	Inserts a node set placeholder in the ETD Tree on the same level as the currently selected node. Node sets consist of multiple nodes that are order independent and/or repeat. See “ Adding Node Sets ” on page 239 for details.
Add Subset	Inserts a node set placeholder in the ETD Tree one level below the currently selected node or node set. The inserted set is associated with the selected node. See “ Adding Node Subsets ” on page 241 for details.
Delete	Allows you, in one step, to delete the selected node and all of its associated subnodes. You cannot delete a root node. See “ Deleting ETDs ” on page 253 for details.
Prune	Deletes all subnodes associated with the selected node. You can use this function to delete all nodes below the root node. See “ Pruning ETDs ” on page 252 for details.
Cut	Deletes the selected node and all associated subnodes, temporarily placing them in an internal buffer. See “ Using Cut, Copy, and Paste ” on page 251 for details.
Copy	Temporarily places a selected node and its associated subnodes in an internal buffer. You cannot cut, copy, or paste between two ETD Editor windows. See “ Using Cut, Copy, and Paste ” on page 251 for details.
Paste	Pastes anything into a selected location in your ETD Tree that you last placed in an internal buffer with the Cut or Copy command. See “ Using Cut, Copy, and Paste ” on page 251 for details.
Find	Allows you to search through your ETD for a particular node. You search by node name. See “ Finding ETD Nodes ” on page 250 for details.

Templates Menu

Table 27 below lists and describes the **Templates** menu commands.

Table 27 ETD Editor Templates Menu

Command	Function
New Internal Template	Allows you to create, from scratch, an internal template to use in your ETD. See “ Using Internal Templates ” on page 257 for details
External Template	Allows you to access an external template to use in the current ETD. See “ Using External Templates ” on page 254 for details.
Delete Template	Allows you to delete an internal or external template from your ETD. Removes the template’s button from the Template Library pane.
Templify	Allows you to convert an ETD or portion of an ETD to an internal template.
Resolve	For external templates, breaks the link between the selected instance of an external template in your ETD and its source file. See “ Breaking Template Links ” on page 257 for a description of how to use the Resolve command with an external template. For internal templates, breaks the link between the selected instance of an internal template in your ETD and its source. For internal templates, the Resolve command is the opposite of Templify . To use this, select the internal template’s root node and then choose the Resolve command.

View Menu

Table 28 below lists and describes the ETD Editor **View** menu commands.

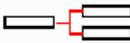
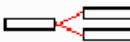
Table 28 ETD Editor Templates Menu

Command	Function
Float Toolbar	Opens a floating version of the toolbar that you can drag to a convenient location in the ETD Editor window.
Float Event Type Definition Pan Window	Opens a floating version of the panning tool. The panning tool lets you change your view of the ETD if it is too big to fit in the Workspace pane. Grab the frame in the panning tool and move it around to pan across your ETD.
Node Properties	Displays the Node Properties dialog box containing the properties of the currently selected node. For delimited notes, see “ Adding Delimited-ETD Nodes ” on page 228 for details. For fixed nodes, see “ Adding Fixed-ETD Nodes ” on page 235 for details.
Collapse	Closes all branches in your ETD Tree below the selected node.
Expand	Explodes the ETD Tree one level so that you see all branches of subnodes belonging to a node.
Expand All	Explodes the ETD Tree on all levels so that you see all branches of nodes, subnodes, and sub-subnodes in the ETD.

Options menu

Table 29 below lists and describes the **Options** menu commands.

Table 29 ETD Editor Options Menu

Command	Function
Horizontal Layout	Configures your ETD Tree in a left-to-right hierarchical structure.
Vertical Layout	Configures your ETD Tree in a top-to-bottom hierarchical structure.
Manhattan Style	Configures the line connections ETD Tree components as follows: 
Traditional Style	Configures the lines connections ETD Tree components as follows: 
Insert After	Inserts a sibling node or node set below the currently selected node.
Insert Before	Inserts a sibling node or node set above the currently selected node.

Help Menu

Table 30 lists and describes the ETD Editor **Help** menu commands.

Table 30 ETD Editor Help Menu

Command	Function
e*Gate Help Topics	Allows you to access the online Help system for the ETD Editor GUI.
About e*Gate	Displays basic information about the current e*Gate software version.

6.4 Creating and Building ETDs

You must create an ETD for every Event Type that passes through the e*Gate system. Use the ETD Editor to create the following basic kinds of ETDs:

- **Delimited:** Variable-length Events, whose data is separated by delimiters.
- **Fixed:** Constant-length Events, with no delimiters.

You create an ETD file in the same way for both kinds. However, specifications for building the two Event Types are different. You can also create node set placeholders in any ETD Tree.

This section explains:

- “[Creating ETD Files](#)” on page 215
- “[Building Delimited ETDs](#)” on page 217
- “[Building Fixed ETDs](#)” on page 232
- “[Adding Node Sets](#)” on page 239

6.4.1 Creating ETD Files

Initial steps for creating ETDs are the same for delimited and fixed ETDs, that is, you must first create the ETD file.

Naming Files

When naming ETD files, use the same naming conventions as you do for nodes (see “[Naming Nodes](#)” on page 206). The only exception is the maximum number of allowable characters, which is 256, including the path as shown in the **New ETD** dialog box (see [Figure 85 on page 216](#)).

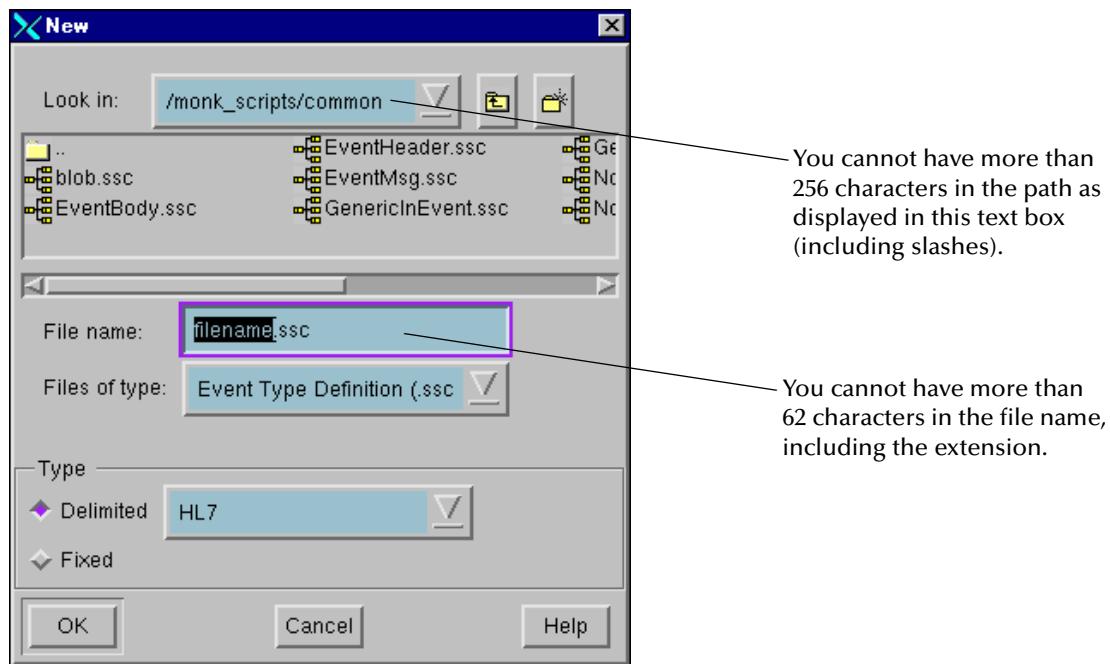
SeeBeyond recommends that you give the ETD file a name suitable for its root node. For example, if you want to call the root node “New_Data_1,” then name the ETD file **New_Data_1.ssc**.

To create ETD files

- 1 From the ETD Editor window, click the **New** command button (or, on the **File** menu, choose **New**). The **New ETD** dialog box appears (see [Figure 85 on page 216](#)).

Note: *If you are already using this window with a different ETD displayed, choosing the **New** command allows you to start creating another ETD. The system warns you if you have not saved your current ETD.*

Figure 85 ETD Dialog Box



- 2 In the **Name** box, enter a file name for the ETD.

The file extension, **.ssc**, is the default file extension for ETD files. The system automatically adds it to your file name, so you do not must specify it. *Do not change this extension.*

Note: For system file-naming conventions, see “[Naming System Elements](#)” on [page 45](#).

- 3 For **Files of Type**, leave the box set to **Event Type Definition**.
- 4 Select the file **Type** appropriate for your ETD, **Delimited** or **Fixed** as follows:
 - **Variable-length Events:** Select the **Delimited** file type then select the delimited Event Type out of the following delimiter options: **HL7**, **X12**, **EDIFACT**, or **Other**.

Note: If you select the **Other** option, you must define default delimiters for your ETD. See “[Defining Default Delimiters](#)” on [page 217](#) for additional procedures.

- **Fixed-length Events:** Select **Fixed** and go on to the next step.
- 5 Click **OK** to close the **New ETD** dialog box.

An icon appears in the Template Library pane. This icon represents your new ETD.

Delimited and Fixed ETDs

Specifications for ETDs are different, depending on whether the corresponding data files are delimited or fixed. See the following sections for further details:

- [“Building Delimited ETDs” on page 217](#)

- “[Building Fixed ETDs](#)” on page 232

Table 23 below lists additional options for building both delimited and fixed ETDs, along with references to sections in this user’s guide that explain these options.

Table 31 Options for Delimited and Fixed ETDs

Option	Purpose	Details Contained In
Creating node sets	Define nodes that as a group, are order-independent or repeat.	“Adding Node Sets” on page 239
Referencing external templates in your ETD	Reference standard ETDs from the current ETD to help you build and maintain it more easily.	“Using External Templates” on page 254
Creating an internal template	Create a subset of an ETD that can be used as a repeatable pattern to build other subsets within the same ETD.	“Using Internal Templates” on page 257
Referencing internal templates in your ETDs	Use an internal template you have created in the current ETD.	“Referencing Internal Templates in ETDs” on page 258

6.4.2 Building Delimited ETDs

If your ETD represents a variable-length Event requiring delimiters, you must create a delimited ETD. This section explains how to do this operation.

Creating Delimited ETDs

Create your new ETD file as explained under [“Creating ETD Files” on page 215](#). To do this operation, use the ETD Editor window and the **New ETD** dialog box.

Defining Default Delimiters

Default delimiters have the following basic characteristics:

- When you create a new ETD file, the **New ETD** dialog box asks you to specify the default delimiters you want to use. For example, if you select the **HL7** option, the system uses standard HL7 default delimiters in your ETD. *You must assign a set of default delimiters to all delimited ETDs.*
- If you select the **Other** option in the **New ETD** dialog box, you must define your own default delimiters. For details on how to use the **Other** option, see [“Defining Default Delimiters” on page 217](#) and follow the procedures under that section.
- When you define default delimiters, you are specifying the delimiters that apply to an entire ETD. You define default delimiters in node levels; that is, Level 1 delimiters apply to the first level of nodes below the root node, Level 2 delimiters apply to the subnodes immediately below the Level 1 nodes, and so on. Delimiters do *not* apply to the root node.
- In addition, you can set the default delimiter for each level in the ETD Tree hierarchy.

Array Attribute: When you select the **array** attribute within a level, the delimiters shown in the **Set Delimiters** dialog box (Figure 86) are actually the repetition delimiters for the previous level. See step 6 in the [procedure on page 220](#).

Also, see the following references:

- For details on how to extract delimiters from an input Event, see [“Extracting Input Delimiters” on page 247](#).
- For details on delimiter syntax, see [“Delimiter Syntax” on page 222](#). Define default delimiters, using the ETD Editor window.
- For details on how to extract delimiters from particular locations in an inbound Event, see [“Extracting Input Delimiters” on page 247](#).

Note: If you are using the HL7 standard delimiters, the repetition field separator is, by default, a tilde (~). You can change this character if your HL7 Events use a different repetition separator. The HL7 repetition separator divides multiple occurrences of a field. See [“Specifying HL7 Repeating Fields” on page 231](#) for details about how to use this separator in an ETD.

To save any dialog box entries before you are finished

Click **Apply** to save your entries into the system and continue working in the dialog box.

To change a set of dialog box entries

Use one of the following buttons:

- ♦ **Delete Level:** Click to delete the selected level.
- ♦ **Restore Delimiters:** Click to restore the previous set of saved delimiter settings.

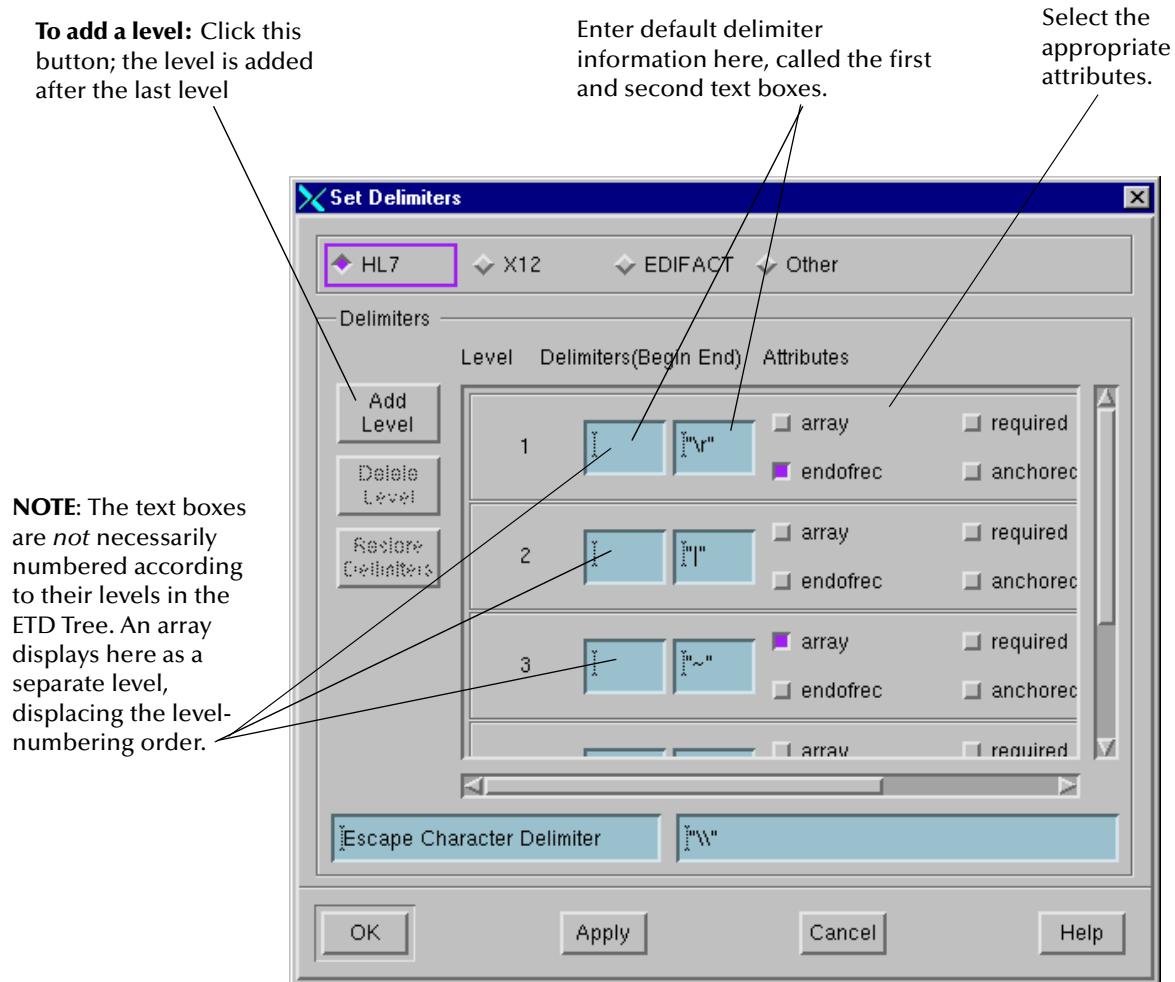
See [Figure 87 on page 220](#).

To define default delimiters

- 1 On the **File** menu, click **Default Delimiters**.

The **Set Delimiters** dialog box appears (see Figure 86 below).

Figure 86 Set Delimiters Dialog Box



2 Select one of the following default-delimiter options:

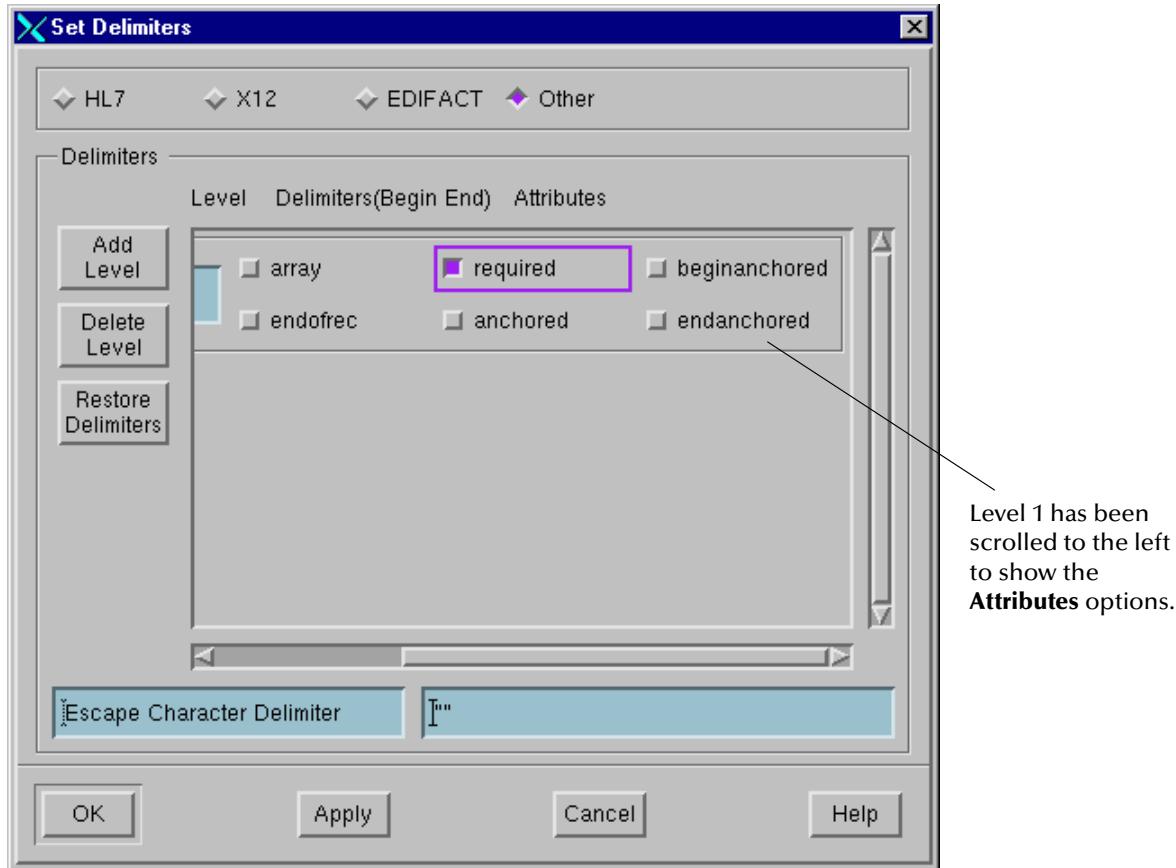
- **HL7:** Displays the standard set of delimiters for HL7 Events.
- **X12:** Displays the standard set of delimiters for X12 Events.
- **EDIFACT:** Displays the standard set of delimiters for EDIFACT Events.
- **Other:** Use to define your own set of delimiters or add to a set of standard set. For correct syntax, see ["Delimiter Syntax" on page 222](#).

3 Take one of the following steps:

- **For Standard Options:** If you select one of the standard options in the previous list, the set of delimiters for that standard displays in the dialog box. If you do not want to change these options, you are now finished. Skip to step 14.
- **For Other Option:** If you chose **Other** from the previous list, the **Set Delimiters** dialog box changes (see [Figure 87 on page 220](#)). You must go on to the next step.

- 4 Click **Add Level**, if you are creating or adding delimiters to add your first level of delimiters (under the **Other** option). See Figure 87 below.

Figure 87 Set Delimiters Dialog Box, Other Selected



Note: You can add a maximum of 255 levels of delimiters.

- 5 Enter the appropriate information for the new level in the text boxes (see [Figure 86 on page 219](#)) as follows:

- ♦ **First Text Box:** Enter the beginning delimiter in double quotation marks (for example, “^”). If there is no beginning delimiter, leave the box blank.
- ♦ **Second Text Box:** Enter the end delimiter in double quotation marks or enter the byte position to extract the delimiter from. A byte position can be specified either as [n] or [n,m], where n and m are integers and the first byte is “[0].” If you specify a byte position, you cannot specify a beginning delimiter (and you must leave the first box blank).

Note: If you want to use brackets ([]) as delimiters, you must enclose them in double quotation marks (“[]”).

- 6 Select one or more of the following **Attributes** as desired:

- ♦ **End of Record (endofrec):** Marks the end of a record.

- ♦ **Required (required):** Indicates that the delimiter must appear in the node.
- ♦ **Anchored (anchored):** The delimiters must be the first and last characters of this segment of the Event, equivalent to selecting both **beginanchored** and **endanchored**.
- ♦ **Begin Anchored (beginanchored):** The beginning delimiter must be the first characters of this segment of the Event.
- ♦ **End Anchored (endanchored):** The end delimiter must be the final characters of this segment of the Event.

Note: The **array** option does not apply to a nonrepetition delimiter.

- 7 Take one of the following steps:
 - ♦ **No Repetition Delimiters:** If you do not have an associated repetition delimiter, go back and repeat steps 4 through 7 until you are finished then skip to step 14.
 - ♦ **With Repetition Delimiters:** If this delimiter does have an associated repetition delimiter, go on to the next step.
- 8 Click **Add Level** and select the following attribute:
 - ♦ **Array (array):** Only use this option in a level where you add repetition delimiters (delimiters for repeating nodes).
- 9 Enter the appropriate information for the repetition delimiter in the text boxes (see step 5).
- 10 Select any additional attributes as desired (see step 6). Make sure you have selected the **array** attribute.
- 11 Repeat steps 4 through 7 until you are finished. Go on to the next step.
- 12 For the **Escape Character Delimiter**, you can enter any character you want in that text box, between the double quotation marks (if you chose **Other**). If you skip this entry, the default escape character is the backslash (\). See "[To enter an escape character with your own default delimiters](#)" on page 221 for details.
- 13 Click **Apply** at any time to save your entries.
- 14 When you are completely finished, click **OK** to close the dialog box.

To enter an escape character with your own default delimiters

If you want to tell Monk that the next character is not a delimiter, you must escape that character. Do this action by entering an escape character directly before the desired character. The default escape character is the backslash (\).

If you define or add your own set of default delimiters (using the **Other** option), you can enter your own escape character (in double quotation marks) used in your Event. You can also change this text box's label (see [Figure 87 on page 220](#)).

This step is optional. If you do enter your own escape character, keep the following facts in mind:

- ♦ If you are using the HL7 standard delimiters, the default value consists of two backslash characters (\ \). In the HL7 format, this value is used in Events as part of an escape sequence that provides special text formatting instructions.
- ♦ For any other Event Type besides HL7, fill in the escape character used in your Events.

Delimiter Syntax

Use the following syntax when entering delimiters in the **Set Delimiters** dialog box:

- Enclose all delimiters in double quotation marks; to include quotation marks as a delimiter, use

\ "

- When entering a single character, use any character except the backslash (\ \).

- Use a backslash to escape special characters, for example,

\ t

- For hexadecimal values, the syntax is

\ x<value1><value2>

The <valueN> variables can be 0 through 9, a through f, or A through F. For example, the maximum value for a hexadecimal value is \xFF.

- For octal values, the syntax is

\ o<value1><value2><value3>

The <value1> variable can only be 0 through 3, <value2> 0 through 7, and <value3> 0 through 7. For example, the maximum value for an octal value is \o377.

- For multi-character delimiters, the syntax is

"<characters>"

Creating Root Nodes for Delimited ETDs

The root node is always the first element you add in an ETD. To create a root node, use the ETD Editor window.

The root node is at the top of the ETD Tree for a defined Event Type and represents the entire ETD. All nodes you add under a delimited ETD root node are delimited types.

To create delimited ETD root nodes

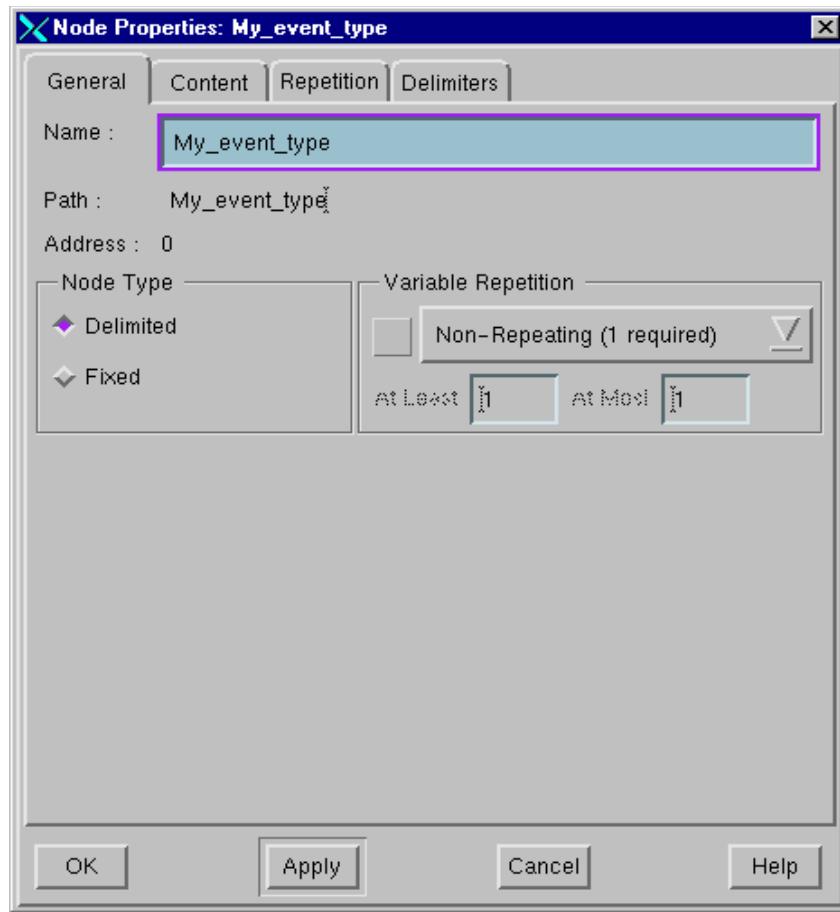
- 1 On the toolbar or **Edit** menu, click  **Add Node**.

The root node appears in the Workspace pane.

- 2 Double-click the root node's label (its third component).

The **Node Properties** dialog box appears, displaying the **General** tab. See [Figure 88 on page 223](#).

Figure 88 Delimited Root Node Properties Dialog Box, General Tab



3 Under the **General** tab, enter the following properties:

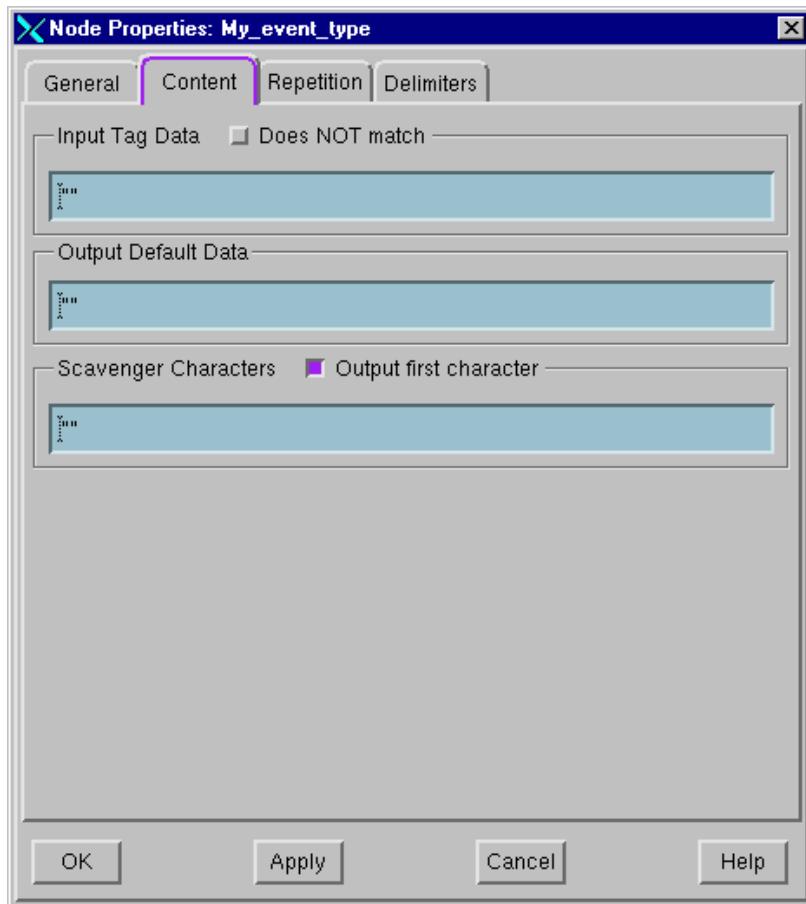
- **Name:** The node's suggested name, automatically inserted in the **Name** text box, although you can change it. See “[Naming Nodes](#)” on page 206 for node-naming rules. The ETD’s icon label in the Template Library pane also shows this name.

Note: Make sure you give the node a meaningful name in relation to the current ETD, for future reference.

- **Path/Address:** Read-only, scrollable fields that display the path and address of the current node in the ETD, starting with the root node.
- **Node Type:** **Fixed** or **Delimited**; by default, **Delimited** is already selected.
- **Variable Repetition:** Options are dimmed, since a root node *cannot* repeat.

4 Click the **Content** tab. See Figure 89.

Figure 89 Delimited Root Node Properties Dialog Box, Content Tab

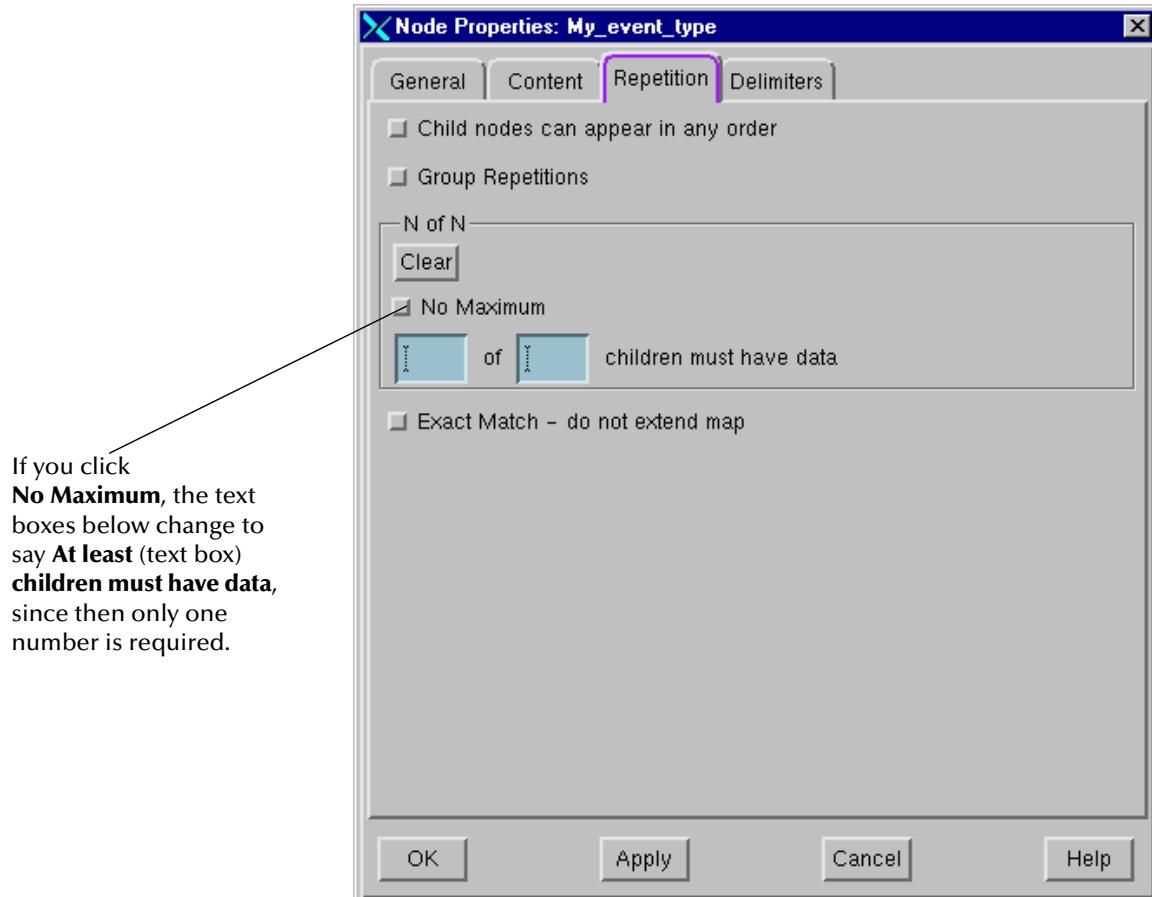


5 Under the **Content** tab, enter the following properties:

- **Input Tag Data:** Enter, in double quotation marks, a Monk regular expression to be applied against the data contained within this node. If the expression does not match, the map fails. For more information about Event mapping, see the “Monk and Event Definitions” section in the *Monk Developer’s Reference*. You can leave this field unused by entering “”.
- **Does NOT match:** Use this option to do a logical NOT on the regular expression specified under **Input Tag Data**.
- **Output Default Data:** Enter, in double quotation marks, the data to be inserted as this node’s contents if the node is otherwise empty. You can leave this field unused by entering “”.
- **Scavenger Characters:** Enter, in double quotation marks, a list of characters to be ignored (in other words, treated as white space). The “scavenger characters” only strip leading characters. Characters that appear within the Event string remain unchanged.
- **Output first character:** Copies the first character in the “scavenger characters” list to the first character of the input data. The remainder of the input data remains unchanged.

- 6 Click the **Repetition** tab (see Figure 90 below). Note that these properties apply to the nodes on levels under the current root node and not to the root node itself.

Figure 90 Delimited Root Node Properties Dialog Box, Repetition Tab



- 7 Under the **Repetition** tab, enter the following properties:

- **Child Nodes Can Appear in Any Order:** Check this box to let the system know whether child nodes can appear in any order in the Event.
- **Group Repetitions:** Check to reorder the data within the child nodes, collating data elements within an Event based upon the **Input Tag Data** pattern specified under the **Content** tab.
- **N of N:** Allows you to require that a certain number of child nodes contain data. Use this feature as follows:
 - To specify a minimum number of nodes, click **No Maximum**, then enter the number of child nodes in **at least** (text box) **children must have data**.
 - To specify a minimum number within a maximum number of nodes (for example, 3 of 5), clear **No Maximum** if necessary, then enter the appropriate numbers.
 - If you do not want to require that a certain number of child nodes contain data, under **N of N**, click **Clear**.

- ♦ **Exact Match - do not extend map:** Click to override Monk's default behavior, that is, to extend the map if Event data or delimiter defaults imply the presence of subnodes. *This option is only available with the root node.*

Using N of N with XML

This feature, introduced by SeeBeyond's XML DTD converter, provides support for XML. Use it to map the children of a desired parent node and validate their data in terms of given requirements.

N of N means at least *N* and not more than *N*. You must provide a minimum and maximum number of repetitions when mapping the children of parent nodes.

Examples

If you enter 3 of 5, the current node is only valid if it meets the following requirements:

- Has three or more child nodes containing data
- Has a total of five or fewer child nodes containing data

You can use **N of N** when the container node in the DTD file is used for matching one out of two cases. **N of N** then forces Monk to match exactly one case because the generated **.ssc** file uses the following syntax:

```
(NofN (1 1))
```

This command instructs Monk to match a minimum of one case and a maximum of one case when mapping the children. The first case is for matching the element that has data and an end tag.

For example, if you enter the following XML data:

```
"<A>
A's Data
</A>"
```

The first case matches

>
A's Data
"

The following XML input, without data, matches the second case,

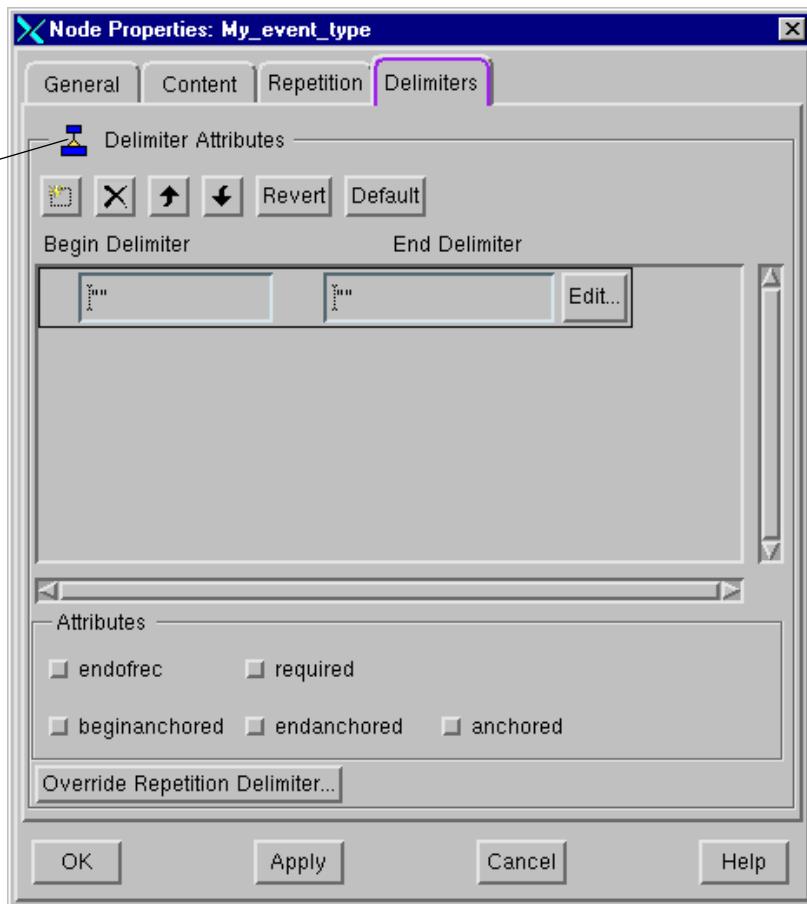
<A />"

For more information, see SeeBeyond's *XML Toolkit*.

8 Click the **Delimiters** tab (see Figure 91 below).

Figure 91 Delimited Root Node Properties Dialog Box, Delimiters Tab

Symbol indicates current values are inherited from the node defaults; if you assign values to override the defaults, this symbol changes to a check mark.



9 Use the **Delimiters** tab to enter the properties that override your preset default delimiters. See ["Defining Default Delimiters" on page 217](#) for details. The buttons under this tab have the following functions:

- ◆ : Allows you to add a begin/end delimiter pair to the current node; click this button then enter the desired pair and click **OK**. Make sure the added delimiter is enclosed within double quotation marks.
- ◆ : Allows you to delete a begin/end delimiter pair from the current node; select the desired pair then click this button.
- ◆ : Allow you to move the selected begin/end delimiter pair one position up or down in the current node's parsing order; select the desired pair then click the appropriate button.
- ◆ **Revert**: Allows you to redisplay the initial values shown in this properties dialog box's tab, when you first opened it (for the current session).

- **Default:** Allows you to return this properties dialog box's tab information to the node's default delimiter values.
 - **Override Repetition Delimiter:** Opens a dialog box similar to this properties dialog box's tab, which allows you to enter a special set of values to change the repetition delimiter. Use this dialog box in the same way as you do the **Delimiters** tab in the delimited **Node Properties** dialog box.
- 10 When you want to save any root-node properties, click **Apply** to enter them into the system.
 - 11 When you are finished with the properties dialog box, click **OK** to close it.
 - 12 Finish building the ETD Tree for the current ETD as follows:
 - If you must add individual delimited Event elements (such as segments) to your definition, go to "[Adding Delimited-ETD Nodes](#)" on page 228.
 - If your ETD consists of a group of Event elements that can occur in any order, then you must add a node set below your root node. Go to "[Adding Node Sets](#)" on page 239 for details.

To change an existing begin/end delimiter pair

- 1 Select the **Edit** button corresponding to the delimiter pair you want to change.
- 2 Make the changes and click **OK**. Be sure the delimiter is enclosed within double quotation marks.

To change attributes for all begin/end delimiter pairs

Use the **Attributes** section of this tab under the delimited **Node Properties** dialog box. See the list under step 6 in the [procedure on page 220](#) for the **Set Delimiters** dialog box, for instructions on how to set these attributes.

Adding Delimited-ETD Nodes

Once you create a delimited ETD root node, all nodes you add to the root node are automatically in a delimited format. This section explains how to create delimited subnodes and define their properties.

Delimited nodes display accordingly in the ETD Tree. Node icons and all icons at levels below them display the default delimiters assigned to their particular Tree level. See Figure 92.

Figure 92 Root Node and Node Icons



If you are creating a delimited ETD, but have not yet added your root node, see "[Creating Root Nodes for Delimited ETDs](#)" on page 222 for details on this operation.

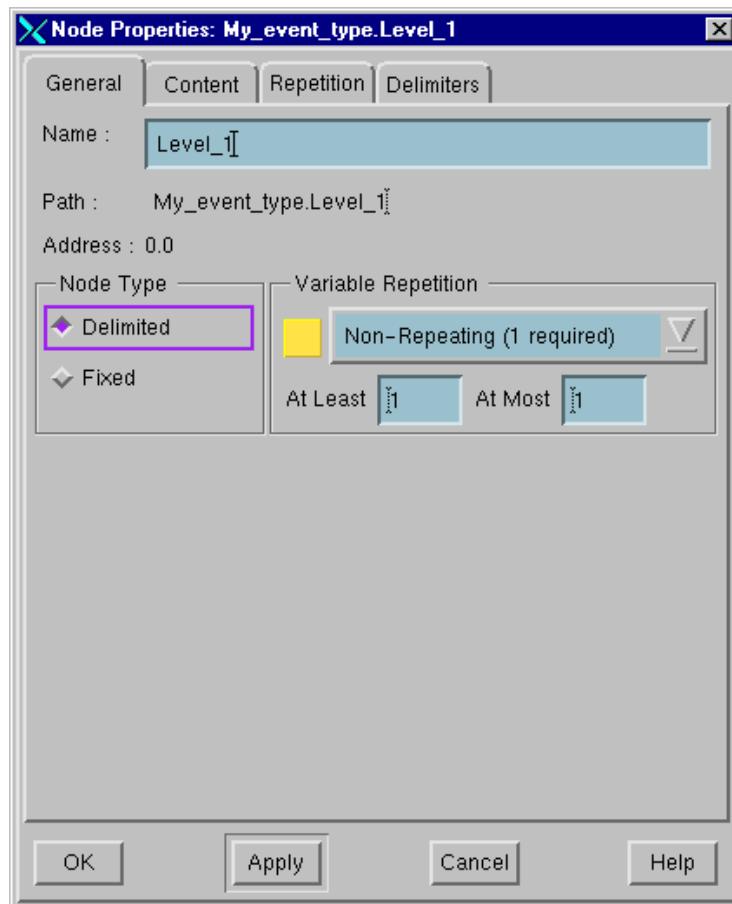
Use these same procedure for each level of nodes, subnodes, and sub-subnodes you must add to your ETD Tree. Use the ETD Editor window to do these operations.

To add delimited-ETD nodes

- 1 In the Workspace pane, select the desired delimited-ETD root node.
- 2 Click the **Add Subnode** command (or, on the **Edit** menu, choose **Add Subnode**). A delimited-ETD node icon appears by the selected root node.
- 3 Double-click the new node's label (its third component).

The **Node Properties** dialog box appears, with the **General** tab elements displayed (see [Figure 93 on page 229](#)).

Figure 93 Delimited Node Properties Dialog Box, General Tab



- 4 Under the **General** tab assign the following properties:
 - **Name:** The node's suggested name is automatically inserted in the **Name** text box, although you can change it. See ["Naming Nodes" on page 206](#) for node-naming rules. The ETD's icon label in the Template Library pane also shows this name.
 - **Path/Address:** Read-only, scrollable fields that display the path and address of the current node in the ETD, starting with the root node.
 - **Node Type:** **Fixed** or **Delimited**; by default, **Delimited** is already selected.

- ♦ **Variable Repetition:** Select the option that matches the number of times the node repeats as shown in Table 32.

Table 32 Delimiter Variable Repetition Options

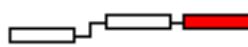
Number of Node Repetitions	Option To Choose	Entries	Symbol Displayed by Node
Only one occurrence	Non-Repeating	1, 1	1
Can occur either once or never	Optional	0, 1	?
Can occur zero or more times	Optional Repeating	0, INF	*
Can occur one or more times	Repeating	1, INF	+
Can occur within a range of repetitions	At Least ... At Most Fill in the repetition range, using the integers <i>n</i> to <i>m</i> For example, if nodes repeat 1 to 5 times, enter 1 and 5 .	<i>n, m</i>	< <i>n-m</i> >

Note: A symbol appears in the node's second component in the ETD Tree. The symbol's format represents the selected repetition option.

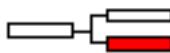
- 5 Click the **Content** tab. Enter properties as instructed under step 5 in the [procedure on page 224](#).
- 6 Click the **Repetition** tab. Enter properties as instructed under step 7 in the [procedure on page 225](#).

Note: This tab in the *Node Properties* dialog box does not contain the *Exact Match - do not extend map* option since it only applies to root nodes.

- 7 Click the **Delimiters** tab. Enter properties as instructed under step 9 in the [procedure on page 227](#).
- 8 When you want to save any node properties, click **Apply** to enter them into the system.
- 9 When you are finished with the properties dialog box, click **OK** to close it.
- 10 You can do one of the following operations to continue building your ETD Tree:
 - ♦ **To Add Subnodes:** Select the node to which you want to add subnodes (nodes to lower levels). Choose the **Edit > Add Subnode** menu command (or click the **Subnode** button). Repeat this step for each subnode you want to add.



- ♦ **To Add Sibling Nodes:** Select the node to which you want to add sibling nodes (nodes on the same level). On the toolbar or **Edit** menu, click **Add Node**. Repeat this step for each sibling node you want to add.



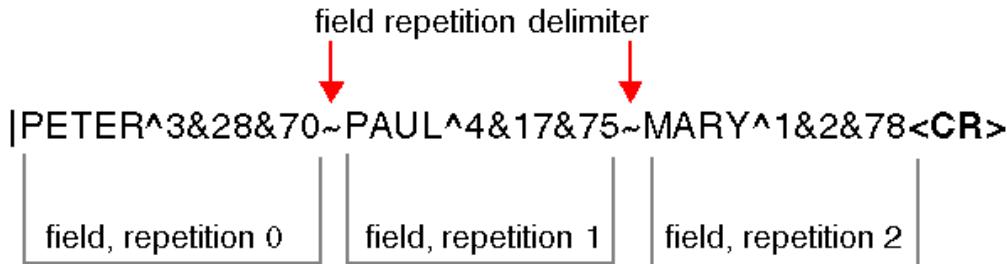
- 11 Repeat steps 3 through 9 in these procedures as necessary, for each subnode and/or sibling node you began adding under step 10. Also if necessary, add sub-subnodes and sibling sub-subnodes in the same way.
- 12 When you are finished building your ETD, save your changes.

Specifying HL7 Repeating Fields

In the HL7 Event Type standard, there is a special delimiter that marks instances of repeating fields. You specify this special character in the **Set Delimiters** dialog box, available in the File menu.

The default field repetition delimiter is a tilde (~). Figure 94 shows a sample HL7 field that uses this special delimiter.

Figure 94 Field Repetition Delimiter Usage



Caution: Only use field repetition delimiters at two levels below the root node (the field level).

Use Node Subsets: The procedure below uses an example to show how to insert a field repetition delimiter in an ETD, using node subsets. See “[Adding Node Sets](#)” on [page 239](#) for details on how to set up node sets and subsets.

Use this feature, for example, if you must set up a segment that contains a repeating field. You must add a set placeholder that represents the repeating field and contains the repetition delimiter.

To insert field repetition delimiters in ETDs

- 1 On the **File** menu, click **Set Delimiters**. The **Set Delimiters** dialog box appears (see [Figure 86 on page 219](#)).
- 2 Choose the HL7 default delimiters.
- 3 Set up a root node, followed by a node that represents the repeating segment.

- 4 Select the node that represents the repeating segment and, on the toolbar or **Edit** menu, click  **Add Subset**.
- A set placeholder node appears below the repeating segment node. The new node represents the repeating field. Give this node the same name as the field.
- 5 Double-click the new set placeholder's label (its third component).
The **Set** properties dialog box appears (see [Figure 100 on page 240](#)).
- 6 The properties you define for this set are the properties of the repeating field. Define the new set's properties as follows:
 - **Name:** Enter the name of the repeating field.
 - **Set Type Options:** The **Ordered Set** option is automatically selected. Do not change it.
 - **Special Character Option:** Select the **Repetition Field Separator** button. This inserts the field repetition delimiter in the ETD hierarchy, so that each instance of the repeating field is separated by this special delimiter. The HL7 default special delimiter is a tilde (~).
 - **Variable Repetition Options:** Select the **Repeating** option.
- 7 Click **OK** to apply the properties and close the dialog box.
- 8 Select the node set placeholder for the repeating node set—in other words, the node that contains the repeating field, and then click the **Subnode** command button (or, on the **Edit** menu, choose **Add Subnode**) to add the first component of the repeating field.
- 9 Repeat step 8 as many times as necessary to define all the components of the repeating field. Define the properties of these nodes.
- 10 When you are finished, save your changes.

6.4.3 Building Fixed ETDs

You must create a fixed ETD if your ETD represents an Event with a fixed length. This section explains this operation.

Creating Fixed ETDs

Create your new ETD file as explained under [“Creating ETD Files” on page 215](#), using the ETD Editor window and the **New ETD** dialog box.

Creating Root Nodes for Fixed ETDs

The root node is always the first element you add in an ETD. The root node is at the top of the ETD Tree for a defined Event Type and controls the purpose for all Events of that type. All nodes you add under a fixed ETD root node are fixed types.

To create fixed-ETD root nodes

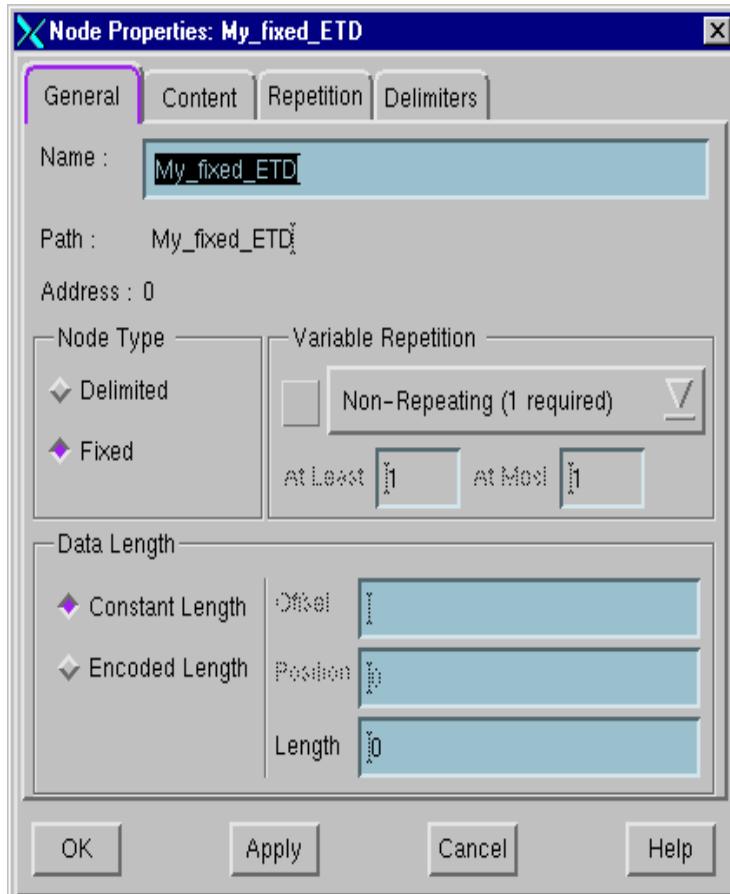
- 1 On the toolbar or **Edit** menu, click  **Add Node**.

A root node icon appears in the Workspace pane.

- 2 Double-click the root node's label (its third component).

The **Node Properties** dialog box appears, displaying the **General** tab. See Figure 95.

Figure 95 Fixed Root Node Properties Dialog Box, General Tab

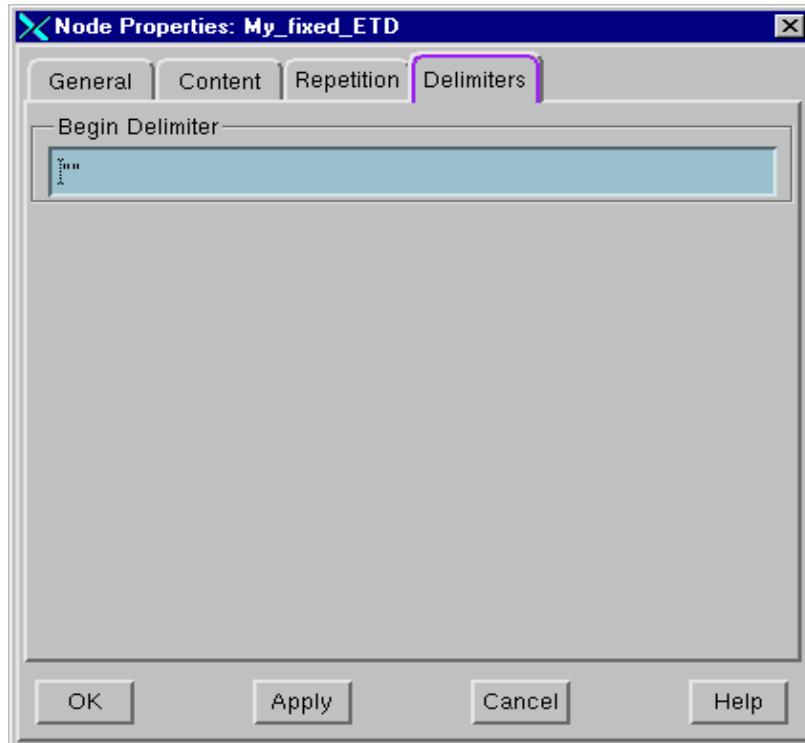


- 3 Under the **General** tab, enter the following properties:

- **Name:** The node's suggested name is automatically inserted in the **Name** text box, although you can change it. See “[Naming Nodes](#)” on page 206 for node-naming rules. The ETD’s icon label in the Template Library pane also shows this name.
- **Path/Address:** Read-only, scrollable fields that display the path and address of the current node in the ETD, starting with the root node.
- **Node Type:** **Fixed** or **Delimited**; if necessary, click **Fixed** to change to the appropriate ETD type.
- **Variable Repetition:** These options are dimmed, since the root node *cannot* repeat.
- **Data Length:** Select whether the node has a **Constant Length** (length never changes) or **Encoded Length** (length contained within Event data), then enter the following parameters:

- **Length:** You can enter the entire Event's length in this text box, but it is only required if you're not going to represent all Event elements in the structure. When all elements are represented in the structure, the system calculates the total Event length by adding up the lengths of Event elements.
 - **Offset:** The text box is dimmed, since the root node represents the entire Event.
 - **Position:** Since there is no offset, there is no offset position.
- 4 Click the **Content** tab. Enter properties as instructed under step 5 in the [procedure on page 224](#).
 - 5 Click the **Repetition** tab. Enter properties as instructed under step 7 in the [procedure on page 225](#).
 - 6 Click the **Delimiters** tab (see Figure 96 below). This feature is optional. If the current ETD has no delimited elements, skip this and the next step.

Figure 96 Fixed Root Node Properties Dialog Box, Delimiters Tab



- 7 If you must use the **Delimiters** tab, enter the string representing a delimiter (after which to search for the node) within the fixed data. The delimiter marks the point from which the offset and length, as specified within the **General** tab for the current node, are measured.
- 8 When you want to save any root-node properties, click **Apply** to enter them into the system.
- 9 When you are finished with the properties dialog box, click **OK** to close it.

10 Finish building the ETD Tree for the current ETD as follows:

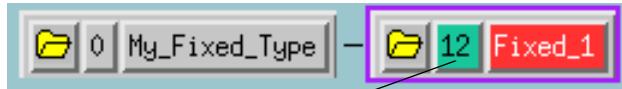
- If you must add individual fixed Event elements (such as segments) to your definition, go to [“Adding Fixed-ETD Nodes” on page 235](#).
- If your ETD consists of a group of Event elements that can occur in any order, then you must add a node set below your root node. Go to [“Adding Node Sets” on page 239](#) for details.

Adding Fixed-ETD Nodes

If you are creating a fixed ETD, but have not yet added its root node, go to [“Creating Root Nodes for Fixed ETDs” on page 232](#).

Once you create a fixed-ETD root node, the nodes you add to the root node are automatically in a fixed-length format. This section explains how to add fixed nodes and define their properties, using the ETD Editor window.

Figure 97 Fixed-ETD Node Icons



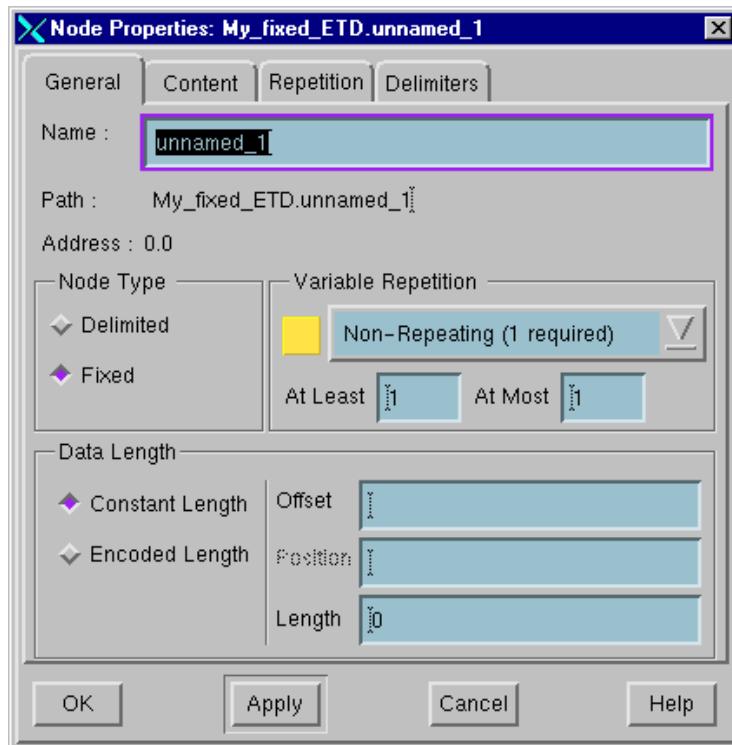
The nodes you add to a fixed ETD root node are also in a fixed format and display the lengths you assign them.

To add fixed-ETD nodes

- 1 In the Workspace pane, select the desired fixed-ETD root node.
- 2 Click the **Subnode** command button (or, on the **Edit** menu, choose **Add Subnode**). A fixed-ETD node icon appears by the selected root node.
- 3 Double-click the new node’s label (its third component).

The **Node Properties** dialog box appears, with the **General** tab elements displayed (see [Figure 98 on page 236](#)).

Figure 98 Fixed Node Properties Dialog Box, General Tab



4 Under the **General** tab, enter the following properties:

- **Name:** The node's suggested name is automatically inserted in the **Name** text box, although you can change it. See “[Naming Nodes](#)” on page 206 for node-naming rules. The ETD's icon label in the Template Library pane also shows this name.
- **Path/Address:** Read-only, scrollable fields that display the path and address of the current node in the ETD, starting with the root node.
- **Node Type:** Already selected, **Fixed**, indicating that the node is in a fixed format.
- **Variable Repetition:** Select the option that matches the number of times the node repeats as shown in [Table 32 on page 230](#).
- **Data Length:** Select whether the node has a **Constant Length** (length never changes) or **Encoded Length** (length is contained within Event data), then enter the following parameters:
 - **Length:** Enter the node's data length, in bytes. Count the length value from 1; if the node's data is 10 bytes long, enter 10 in the **Length** box.
 - **Offset:** This text box is optional. If you don't want to fully specify a fixed ETD, you can use the **Offset** text box to account for unspecified bytes that precede the node you are currently defining. If you don't know the byte offset, or starting location, of a node, count it from byte 0, the beginning of the Event.

- ♦ **Position:** Enter the position of the offset byte.

Note: See “[Specifying Byte Offsets in Fixed ETDs](#)” on page 238 for a detailed explanation about how to use this feature.

- 5 Click the **Content** tab. Enter the properties as instructed under step 5 in the [procedure on page 224](#).

Note: Use the **Input Tag Data** text box to specify a string required to identify an Event element. For example, if a field requires some type of code, you can specify it in the **Input Tag Data** text box.

- 6 Click the **Repetition** tab. Enter properties as instructed under step 7 in the [procedure on page 225](#).

Note: This tab in the **Node Properties** dialog box does not contain the **Exact Match - do not extend map** option since it only applies to root nodes.

- 7 Click the **Delimiters** tab. This feature is optional. If the current ETD has no delimited elements, skip this and the next step.
- 8 If you must use the **Delimiters** tab, enter the string representing a delimiter (after which to search for the node) within the fixed data. The delimiter marks the point from which the offset and length, as specified within the **General** tab for the current node, are measured.
- 9 When you want to save any node properties, click **Apply** to enter them into the system.
- 10 When you are finished with the properties dialog box, click **OK** to close it.

Notice that the new fixed-ETD node now displays in the ETD Tree with the following information:

- ♦ Length, in bytes, displayed in its second component
- ♦ Repetition symbol, if applicable, also displayed in the second component
- ♦ Name appearing in the third component

- 11 Select the node you just added and, on the toolbar or **Edit** menu, click  **Add Node** to add the next node in the ETD Tree.

You can build additional fixed nodes as explained under step 10 in the [procedure on page 230](#) for delimited nodes.

- 12 Set up the node’s properties as explained in steps 3 through 10 in this procedure. Repeat these steps for each node you add to the fixed ETD.
- 13 After you are finish building your ETD: On the toolbar or **File** menu, click  **Save**.

Specifying Byte Offsets in Fixed ETDs

A byte offset is a number that tells how far from the beginning of an Event a particular element is. In e*Gate, you must count byte offsets from byte 0, the beginning of the Event. When you define nodes in a fixed ETD, the system automatically assumes that the nodes immediately follow each other with no gaps in between.

The **Offset** feature in the **Node Properties** dialog box allows you to specify gaps between nodes, while accounting for every byte in a fixed Event. This section presents two examples to explain this feature.

First Example

This example shows how you can use the **Offset** feature to specify a fixed ETD. In this Event, there are the following fixed-length elements: A, B, C, and D. Their lengths are

- Element A = 10 bytes
- Element B = 10 bytes
- Element C = 14 bytes
- Element D = 4 bytes
- **Total length** = 38 bytes

To account for all 38 bytes in the Event, while only creating nodes in the ETD for Elements A and D, specify the following byte offset and length values:

- **Element A:** Byte Offset = 0, Length = 10
- **Element D:** Byte Offset = 34, Length = 4

Remember that byte offsets are counted from 0, which is the first byte location in an Event. Element D starts at byte location 34, because byte locations 0 through 33 are taken up by Elements A (byte locations 0 through 9), B (byte locations 10 through 19), and C (byte locations 20 through 33).

Second Example

What if you only want to specify Element B in your ETD? You still have to account for all 38 bytes in the Event. Specify nodes as follows:

- 1 Create a fixed root node. As you define the root node's properties, enter **38** in the **Node Properties** dialog box's **Length** text box. This tells the system that the entire Event is made up of 38 bytes.
- 2 Under the root node, add a fixed node representing Element B. As you define its properties, specify its offset and length as follows:

Element B: Byte Offset = 10, Length = 10

In the first example, where Elements A and D are specified, you do not define the total Event length in the root node because the last element, Element D, is present in the ETD. As long as the final element in the Event is a part of the ETD, the system knows how long the entire Event is.

In the second example, you specify the total Event length in the root node because the last element, Element D, is not present in the ETD. In this way, you let the system know how long the entire Event is.

6.4.4 Adding Node Sets

To define a group of Event elements that can occur in any order and/or repeat, you must create a node set. A node set is a placeholder in your ETD Tree whose only function is to indicate that the nodes under it make up a group of Event elements.

You can define the following types of group Event elements in the ETD Editor window:

- **Unordered:** Nodes can appear in any order in an Event.
- **Repeating:** A group of nodes repeats in a predefined order.

Placeholders on the Same Level: When you see a node set placeholder in an ETD Tree, remember that its associated nodes, although appearing under it, are actually on the same level of the tree as the node set placeholder to which they are linked.

To add node sets to ETDs

- 1 Select the node to which you want to add the node set.

For example, select the root node if you want to add a group of unordered segment nodes.

- 2 On the toolbar or **Edit** menu, click  **Add Subset**.

A set placeholder appears by the selected node. Set placeholders are visually distinguishable from regular nodes on the ETD Tree because they display one of the following symbols:

- (): Indicates an ordered set (this is the default setting).
- #(): Indicates an unordered set.

For an example, see Figure 99 below.

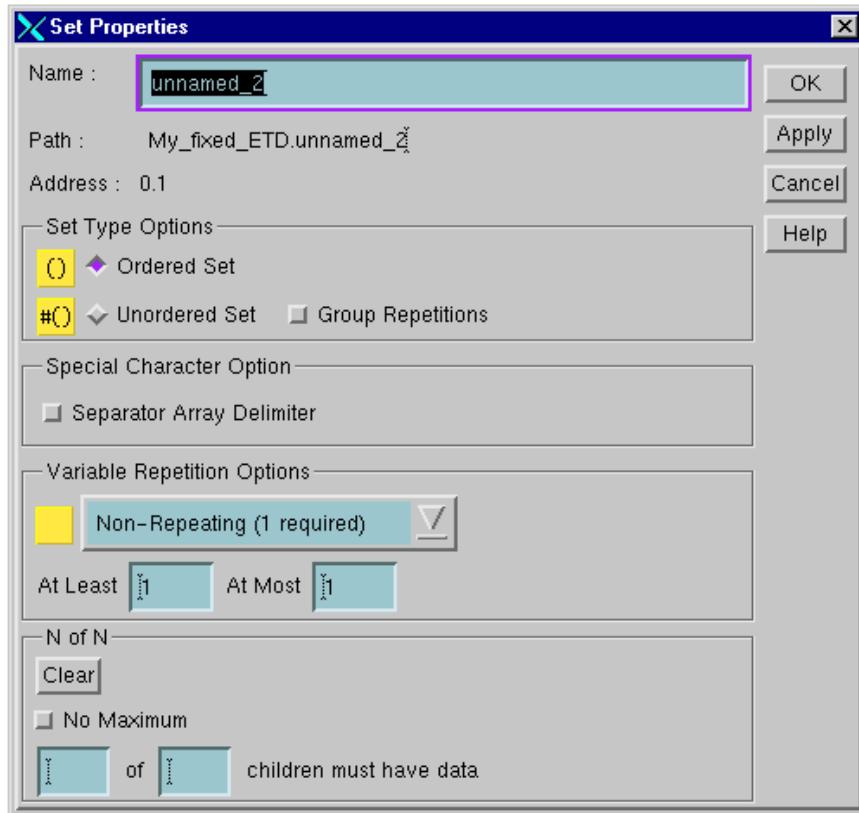
Figure 99 Node Set Placeholder



Note: If you want to add another set placeholder on the same level in the ETD Tree as the node you previously selected, leave the desired node selected. Then, on the toolbar or **Edit** menu, click **Add Set** again and proceed with these instructions.

- 3 Double-click the set placeholder's third component to display the **Set Properties** dialog box. See Figure 100.

Figure 100 Set Properties Dialog Box, Node Set



- 4 Define the node set by assigning the following properties.

- **Name:** The set's suggested name is automatically inserted in the **Name** text box, although you can change it. See [“Naming Nodes” on page 206](#), and also use the node-naming rules for sets.
- **Path/Address:** Read-only, scrollable fields that display the path and address of the current node in the ETD, starting with the root node.
- **Set Type Options:** Use one of the following options:
 - **Ordered Set:** Selected by default; this option indicates the nodes below the current node make up an ordered set. The root node displays in the ETD Tree with the symbol () .
 - **Unordered Set:** This option indicates the nodes below this root node do not have to be in any particular order. The root node displays in the ETD Tree with the symbol #(). Select **Group Repetitions** if this property applies to the current node set.
- **Special Character Option:** Select this option for *delimited* Events only. If creating an HL7 ETD, you must select the **Separator Array Delimiter** box. For details on using this option, see [“Specifying HL7 Repeating Fields” on page 231](#).

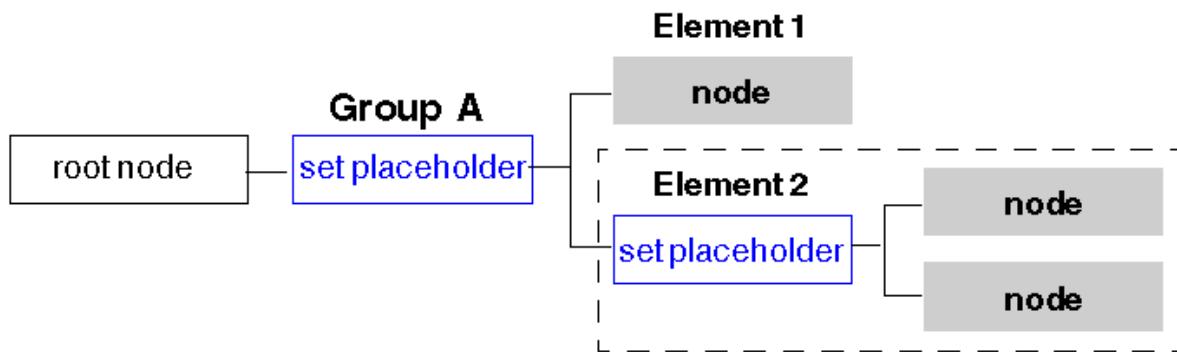
- **Variable Repetition:** Select the option that matches the number of times the node set repeats as shown in [Table 32 on page 230](#).
 - **N of N:** Allows you to require that a certain number of child nodes contain data. See step 7 in the [procedure on page 225](#) for details on this feature. Also, see [“Using N of N with XML” on page 226](#) for XML instructions.
- 5 When you want to save any node-set properties, click **Apply** to enter them into the system.
 - 6 When you are finished with the properties dialog box, click **OK** to close it.
 - 7 Select the set placeholder node you just created and, if necessary, do one or more of the following actions:
 - Add another set placeholder to define a group of Event elements within the set. To do this operation, repeat the previous set of procedures.
 - Add delimited nodes to help complete the set. See [“Adding Delimited-ETD Nodes” on page 228](#) for details.
 - Add fixed nodes to help complete the set. See [“Adding Fixed-ETD Nodes” on page 235](#) for details.

Adding Node Subsets

Sometimes a group of Event elements contains subgroups of elements. These subgroups are called node subsets.

For example, Group A can consist of two repeating elements, Element 1 and Element 2. Element 1 is a single segment. Element 2 is a repeating group made up of two segments. To represent Element 2 in the ETD, you must add a node subset placeholder underneath Group A’s set placeholder. Figure 101 provides a diagram of this example.

Figure 101 Node Subset Diagram



To add node subsets to ETDs

- In the toolbar or **Edit** menu, click  **Add Subset** to add a subset node.
- Configure node subsets in the same way as you do node sets.

6.5 Basic ETD Operations

This section describes basic operations you can do with ETD files, including finding, opening, and saving the files.

This section explains:

- [“Opening ETDs” on page 242](#)
- [“Using the Build Tool” on page 242](#)
- [“Saving ETDs Under New Names” on page 246](#)
- [“Extracting Input Delimiters” on page 247](#)
- [“Testing ETD Files” on page 247](#)
- [“Creating ETD Comments” on page 250](#)
- [“Finding ETD Nodes” on page 250](#)
- [“Editing ETD Files” on page 250](#)

6.5.1 Opening ETDs

Open an existing ETD, using the **Open ETD** dialog box. After you open an ETD file, you can view or edit the file as desired.

To open ETD files

- 1 On the toolbar or **File** menu, click  **Open**.

If you currently have an ETD open and have not saved that file, the system warns you and asks whether you want to save your current file, proceed without saving, or cancel your activity.

Unless you cancel your activity, the **Open ETD** dialog box appears, showing a list of saved ETDs.

- 2 Select the file to open and click **OK**.

The ETD appears in the ETD Editor window’s Workspace pane.

You can also click **Templates** to view and select from a list of **HL7**, **X12**, **EDIFACT**, or **Other** type ETDs.

ETD File Names: By default, e*Gate looks for ETDs as files with the extension **.ssc**. The system’s naming convention for ETD files is the ETD name followed by the extension **.ssc**.

6.5.2 Using the Build Tool

The system can build a delimited ETD for you, when you provide it with certain requirements, including

- Sample data file
- Delimiters to use to parse the Event

- Number of levels to include in the ETD hierarchy

Using Add-ons: If you have an add-on that provides a library converter, you can use the Build tool to generate the desired ETD, such as for XML or a database e*Way. For details on this operation, see the appropriate user's guide for the add-on feature.

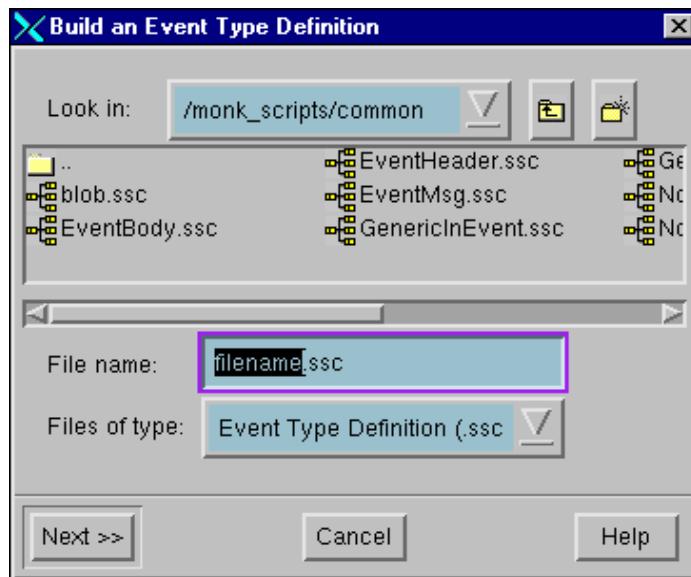
Note: *The Build tool only creates delimited ETDs. Fixed ETDs must be created from scratch. See “Building Fixed ETDs” on page 232 for details about creating fixed ETDs.*

To create an ETD using the Build tool

- 1 Choose File > Build or click on .

The first **Build an Event Type Definition** dialog box, shown in Figure 102 below, appears.

Figure 102 Build an Event Type Definition Dialog Box (Default)

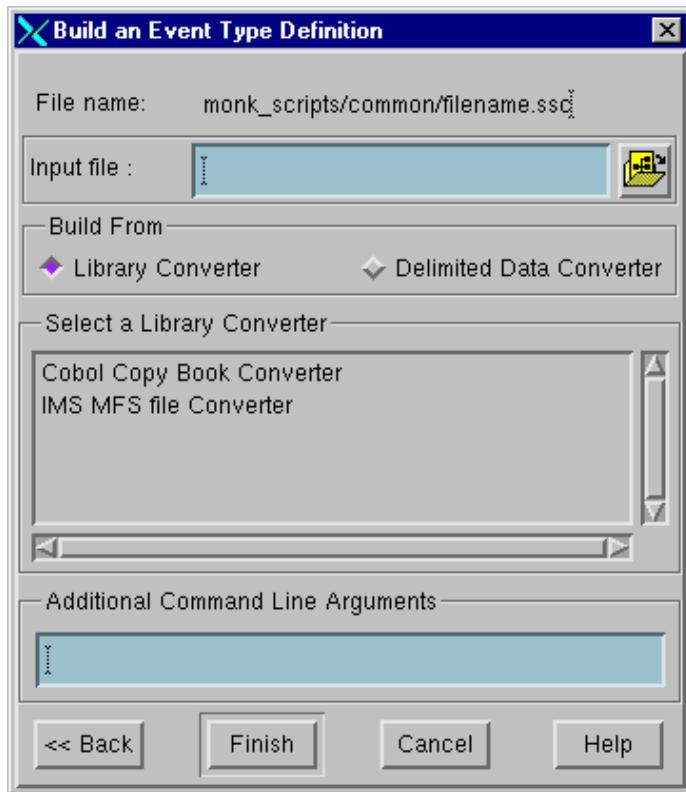


Leave **Files of Type** set to **Event Type Definition (.ssc)**.

- 2 Type the desired ETD file name in the **File name** text box. Do not add a file extension in the Name box; the default ETD file extension, **.ssc**, is automatically appended to your file name once you exit this dialog box.
- 3 Click **Next**.

The **Build an Event Type Definition** dialog box changes to a different display as shown in [Figure 103 on page 244](#).

Figure 103 Build an Event Type Definition Dialog Box (Library)



At any time, you can click **Back** to return to the previous dialog box display.

- 4 To select a sample input data file for the system to use to build the ETD, click to the right of the **Input file** text box.

The **File Selection** dialog box appears, allowing you to browse folders to find the desired file.

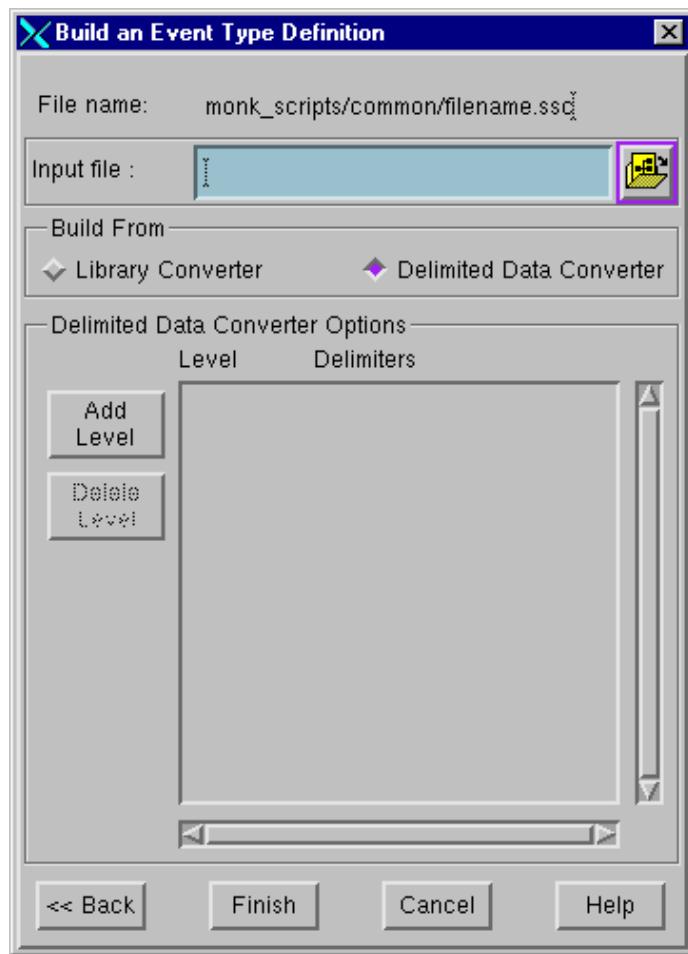
- 5 In the dialog box shown in Figure 103, select the input ETD file you want to use and click **Finish** to close the dialog box.

The ETD file name appears in the **Input file** text box in the **Build an Event Type Definition** dialog box.

- 6 Select the appropriate **Build From** option as follows:

- ♦ **Library Converter:** If there is a special ETD converter (provided with an add-on) you want to use. The dialog box remains the same as shown in Figure 103.
- ♦ **Delimited Data Converter:** If you want to use the ETD Editor's default delimited ETD converter. The default converter creates ETDs, using the delimiters and number of levels you specify. The dialog box changes to a different display as shown in [Figure 104 on page 245](#).

Figure 104 Build an Event Type Definition Dialog Box (Delimited Data)



- 7 Enter the rest of the necessary information in the **Build an Event Type Definition** dialog box as indicated in Table 33 below.

Table 33 Build an Event Type Definition Dialog Box Options

Dialog Box Display	Option	Description	Valid Values
Library Converter Options <i>(see Figure 103 on page 244).</i>	Select a Library Converter	Select a library converter from this list.	The list of converters varies, depending on what is installed at your site.
	Additional Command Line Arguments	Use this text box to enter any special command-line arguments your selected library converter can need.	Specify the desired arguments.

Table 33 Build an Event Type Definition Dialog Box Options (Continued)

Dialog Box Display	Option	Description	Valid Values
Delimited Data Converter Options <i>(see Figure 104 on page 245).</i>	Level	Number of levels to include in the ETD Tree. Add numbered levels using the Add Level button.	Any number (automatically assigned).
	Delimiters	In the text box, enter each delimiter to be used in the ETD. Do not include any spaces between delimiters unless you want to use a space as a delimiter.	Each delimiter must be specified as a single character, with the exception of \r (RETURN). Enclose delimiters in double quotation marks ("").
	Add Level	Click to add a new level of delimiters below the current final level.	N/A
	Delete Level	Click to delete the selected level.	N/A

- 8 When you are finished, click **Finish** to close the dialog box.
After a pause, the ETD Editor window displays the autogenerated ETD.
- 9 When you are finished building your ETD, click the **Save** command button (or, from the **File** menu, choose **Save**).

Note: If you want to see all levels of your ETD and cannot, choose the **View > Expand All** menu option.

6.5.3 Saving ETDs Under New Names

Use this procedure to save an ETD to a file name other than the one you saved it to last.

To save an existing ETD file under a new file name

- 1 Make sure the desired ETD file you want to save to a different name is already open. The current file becomes the basis for the new one.
- 2 On the **File** menu, click **Save As** to display the **Save As Event Type Definition** dialog box (similar to the standard Windows **Save As** dialog box).
The new ETD is saved to the current schema. Notice that in the **File Name** text box, the name text is automatically selected before the default file extension, **.ssc**.
- 3 Type a file name without deselecting the **File Name** text.
- 4 If you want to save the new file as a template, click **Templates**. For more information, see “[Working With ETD Templates](#)” on page 254.

Caution: Do not change the default .ssc file extension. If you do, you cannot find the file later because the ETD Editor's file selection only looks for files with the .ssc extension.

- 5 When you are finished working with the new ETD, click the **Save** command button (or, from the **File** menu, choose **Save**).

6.5.4 Extracting Input Delimiters

To extract a delimiter from a particular byte location in an input Event, use the following syntax in the **Delimiters** text boxes in the **Set Delimiters** dialog box:

[n]

This syntax represents the end byte location of the delimiter you want to extract from the input Event. Remember that in e*Gate, byte locations are counted from 0, the beginning of the Event. For example, if you want to extract a delimiter from the fifth byte of an Event, type [4].

If you want to extract a range of delimiters, type

[m,n]

This expression means, "begin with the *m*th byte and continue up to and including the *n*th byte." For example, [2,4] would extract the delimiters in bytes 3 through 5.

6.5.5 Testing ETD Files

The Editor provides an option that allows you to test a delimited ETD file against sample input data. If the sample data maps to the delimited ETD, then your structure is ready to use to build Event IDs and translations.

It is a good idea to test each delimited ETD before you use it in Event IDs and translations. This practice avoids additional troubleshooting tasks if there is a problem with the structure.

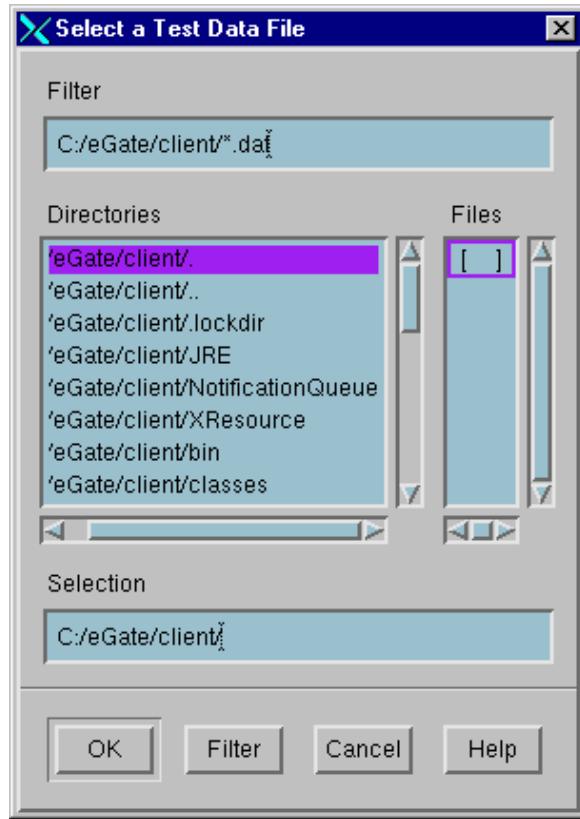
Note: You cannot use the Run Test feature with fixed-ETD files.

To use the Run Test feature

- 1 Open an ETD file in the ETD Editor window.
- 2 Choose **File > Run Test** or click 

The **Select a Test Data File** dialog box appears. By default, the dialog box first shows the contents of the \eGate\client\ directory (see [Figure 105 on page 248](#)).

Figure 105 Select a Test Data File Dialog Box

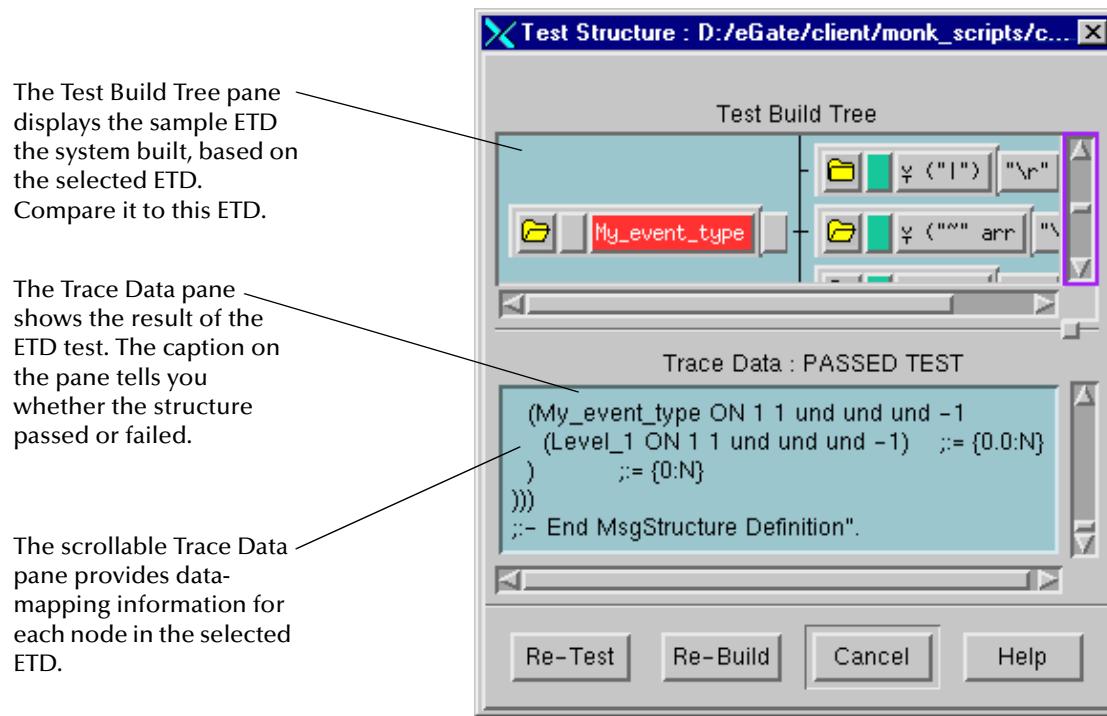


- 3 Scroll through the **Directories** and **Files** lists to find the location of your input data ETD file. Click the **Filter** button to change directories.
- 4 Select the desired input data file in the **Files** column and click **OK**.

The file selection dialog closes, and the system maps the data in the input data file to the currently open ETD. It also builds a sample ETD based on the input data and uses this sample to compare to your selected ETD.

After a pause, the **Test Structure** dialog box appears (see [Figure 106 on page 249](#)).

Figure 106 Test Structure Dialog Box



- 5 To see how the system mapped your ETD, look for the test result in the Trace Data pane at the bottom of the dialog box. Then, look at the sample ETD in the Test Build Tree pane and compare it to your ETD.
The caption above the pane displays either "PASSED TEST" or "FAILED TEST" depending on the test results.
- 6 Take one of the following steps:
 - ♦ If your ETD passed, click **Cancel** to close the **Test Structure** dialog box. You are finished with this procedure.
 - ♦ If your ETD did not pass, scroll through the Trace Data pane and look for error Events. Go to the next step.
- 7 Look at the sample ETD in the Test Build Tree pane, and compare it to your ETD. If there are discrepancies between the two, try modifying your ETD so that it more closely resembles the sample ETD.
- 8 Without closing the **Test Structure** dialog box, go to the ETD Editor window and modify your ETD as needed. Save your modified ETD.
- 9 Return to the **Test Structure** dialog box, and click **Re-Test** to check whether your modified ETD passes the test.
- 10 If you have made changes to your selected ETD file, you can update the sample ETD in the **Test Structure** dialog box by clicking **Re-Build**.
- 11 When you have finished, click **Cancel** to close the dialog box.

6.5.6 Creating ETD Comments

Use this feature to insert an informational comment that applies to the current ETD.

To insert an ETD comment

- 1 On the **File** menu, click **Main Comment**.
The **Comment** dialog box appears.
- 2 Click in the **Comment** dialog box to activate it, and then enter your comments.
There are no syntax restrictions.
- 3 Click **OK** to save your work and exit the dialog box.
The comment you entered is saved with the current ETD.

6.5.7 Finding ETD Nodes

Use the **Find** option to search through your ETD for a particular node.

To search for and find ETD nodes

- 1 To start at the beginning of the ETD, select the root node.
- 2 On the **Edit** menu, click **Find**.
The **Find** dialog box appears.
- 3 In the **Name to Find** text box, type the name of the node you're looking for.
You must enter the complete node name. The ETD Editor does not find a node if you only enter a partial name.
- 4 Use the **Find** dialog box as follows:
 - ♦ Click **Next** to skip to the next occurrence of a node.
 - ♦ Click **Previous** to search backward.
- 5 When you're done, click **Cancel** to close the **Find** dialog box.

6.5.8 Editing ETD Files

Whether you have finished building an ETD file or want to make some changes to a node you have just added, you can edit an ETD Tree at any time. You can cut, copy, and paste portions of an ETD, in addition to changing a node's details and other ETD modifying functions.

Note: *In an ETD, you cannot edit external templates that are linked to their source files. If you must make changes to a referenced external template, you must open the source file and make changes there. If the link between an external template and its source file has been broken, you can edit that template in your ETD. See “[Breaking Template Links](#)” on page 257 for information on the **Resolve** option in the **Templates** menu.*

This section explains:

- “[Moving Nodes](#)” on page 251
- “[Using Cut, Copy, and Paste](#)” on page 251
- “[Pruning ETDs](#)” on page 252
- “[Changing Node Details](#)” on page 253
- “[Modifying Internal Templates](#)” on page 253
- “[Deleting ETDs](#)” on page 253

Moving Nodes

Using your mouse, you can move nodes to new locations in your ETD. You can move a node up or down in the same level of the ETD Tree or up or down to another level of the ETD Tree.

To change the location of a node in an ETD

- 1 Select the node you want to move.
- 2 Click and hold the middle mouse button.

Note: *If your mouse does not have a middle button, hold down both mouse buttons to have the same effect as pressing the middle mouse button.*

- 3 Drag the node to its new location.

As you drag the node towards a spine in the ETD Tree, arrows indicate where the node will be inserted when you release the mouse button. Use these arrows to guide you as you move a node.

- 4 Release the middle mouse button when you see the arrow in the ETD Tree pointing to the correct location.

The node is moved to the new location, along with any subnodes associated with it.

Using Cut, Copy, and Paste

In the ETD Editor, the cut, copy, and paste features operate in a similar way to those of an ordinary text editor. Use the appropriate menu commands or toolbar buttons to carry out these functions.

Note: *Use the paste function to paste any node in the internal buffer into your ETD Tree. You cannot cut, copy, or paste between two ETD Editor windows.*

To cut a node

- 1 Select the node you want to cut. Then click the **Cut** command button (or, on the **Edit** menu, choose **Cut**).

If there are any subnodes associated with the node you cut, those subnodes are also cut. The node you cut is temporarily placed in an internal buffer until the next time you use the **Cut** or **Copy** features.

- 2 Click the **Paste** command button (or, on the **Edit** menu, choose **Paste**) to paste the node in the selected area of the current ETD Tree.

To copy a node

- 1 Select the node you want to copy. Then click the **Copy** command button (or, on the **Edit** menu, choose **Copy**).

If there are any subnodes associated with the node you copied, those subnodes are also copied. The node you cut is temporarily placed in an internal buffer until the next time you use the **Cut** or **Copy** features.

- 2 Click the **Paste** command button (or, on the **Edit** menu, choose **Paste**) to paste the node in the selected area of the current ETD Tree.

To paste into your ETD Tree

- 1 Select the node below which you want to insert the contents of an internal buffer.
- 2 Click the **Paste** command button (or, on the **Edit** menu, choose **Paste**).

The buffer's contents are inserted at the top of the level below the node you selected in step 1. If the buffer is empty then nothing is pasted.

If you must move what you just inserted, select it and use the middle mouse button to drag and drop it to a new location. For details on how to drag and drop nodes, see "[Moving Nodes](#)" on page 251.

Pruning ETDs

Use the pruning function when you want to delete subnodes belonging to a node or nodes belonging to a root node.

To prune ETD subnodes under a node

- 1 Select the node whose subnodes you want to delete.
- 2 Click the **Prune** command button (or, on the **Edit** menu, choose **Prune**).

The subnodes are deleted, leaving the node on the ETD Tree.

To prune ETD nodes under a root node

- 1 Select the root node whose nodes you want to delete.
- 2 Click the **Prune** command button (or, on the **Edit** menu, choose **Prune**).

The nodes are deleted, leaving the root node on the ETD Tree.

Changing Node Details

Once you have an ETD built and want to make changes to node details, you can double-click a node's label to display either the **Node Properties** dialog box or **Set Properties** dialog box and make changes. These GUIs have the following characteristics:

- **Node Properties Dialog Box:** Here you define a node's name, type (delimited or fixed), tag data, and repetition information. For details on how to access and use this properties dialog box for delimited Events, see "[Adding Delimited-ETD Nodes](#)" on page 228. For fixed Events, see "[Adding Fixed-ETD Nodes](#)" on page 235.
- **Set Properties Dialog Box:** Here you define properties for a group of nodes that are order-independent and/or repeat. For details on how to access and use this properties dialog box, see "[Adding Node Sets](#)" on page 239.

Caution: Remember to save the current ETD whenever you are done making changes.

Modifying Internal Templates

This section explains how to edit ETD e*Gate system templates. For a complete explanation of ETD templates, see "[Using Internal Templates](#)" on page 257.

To modify internal templates

- 1 In the Template Library pane, click the icon of the internal template you want to modify.
The template appears in the Workspace pane.
- 2 Make the desired changes to it.
- 3 At the top of the Template Library pane, click the current ETD icon (top icon in the Template Library pane) to redisplay it in the Workspace pane.
The internal template, if already included in the ETD Tree, reflects any changes.
- 4 When you have finished making changes to your template: On the toolbar or **File** menu, click  **Save**.

To remove internal templates from the ETD Tree

- 1 Select the root node of the internal template in the ETD Tree.
- 2 Click the **Delete** command button (or, on the **Edit** menu, click **Delete**).
A confirmation message warns you.
- 3 Click **Yes** to proceed with the deletion or **No** to cancel.

Deleting ETDs

To delete an entire ETD

- 1 Select the root node of the ETD you want to delete.

- 2 Click the **Delete** command button (or, on the **Edit** menu, click **Delete**).
A confirmation message warns you.
- 3 Click **Yes** to proceed with the deletion or **No** to cancel.

Note: A deleted ETD file, although empty, is still in your schema directory. The ETD Editor does not provide a way to delete the file.

To delete an individual node and associated subnodes

- 1 Select the node you want to delete.
 - 2 Click the **Delete** command button (or, on the **Edit** menu, choose **Delete**).
A confirmation message warns you.
 - 3 Click **Yes** to proceed with the deletion or **No** to cancel.
- If you click **Yes**, the node is removed from the ETD Tree, along with any subnodes that are associated with it. This action accomplishes the entire operation in one step.

6.6 Working With ETD Templates

The ETD Editor feature in e*Gate allows you to create and modify both internal and external ETD templates. This section explains in detail how to use this feature.

This section explains:

- [“Using External Templates” on page 254](#)
- [“Using Internal Templates” on page 257](#)

6.6.1 Using External Templates

An external template is an ETD you can reuse to duplicate a given ETD structure in other ETDs. If you include an external template in an ETD, the system brings it in as a referenced file. In such cases, the ETDs made from the template automatically update whenever anyone changes the source file.

This feature becomes an advantage if you use an external template for many different ETD files. By updating the template’s source file, you automatically update the ETDs in which the external template was included; however, the Collaboration that references the current ETD does not change.

Note: You cannot edit external templates that are linked to their source files. If you must make changes to a referenced external template, you must open the source file and make changes there. Also, you can break the external template’s link to its source file by choosing the **Templates > Resolve** menu option (see [“Breaking Template Links” on page 257](#)). This feature allows you to edit an external template in the ETD Tree where it is used.

Including External Templates in ETDs

Any ETD file outside of your current open ETD file can become an external template, including

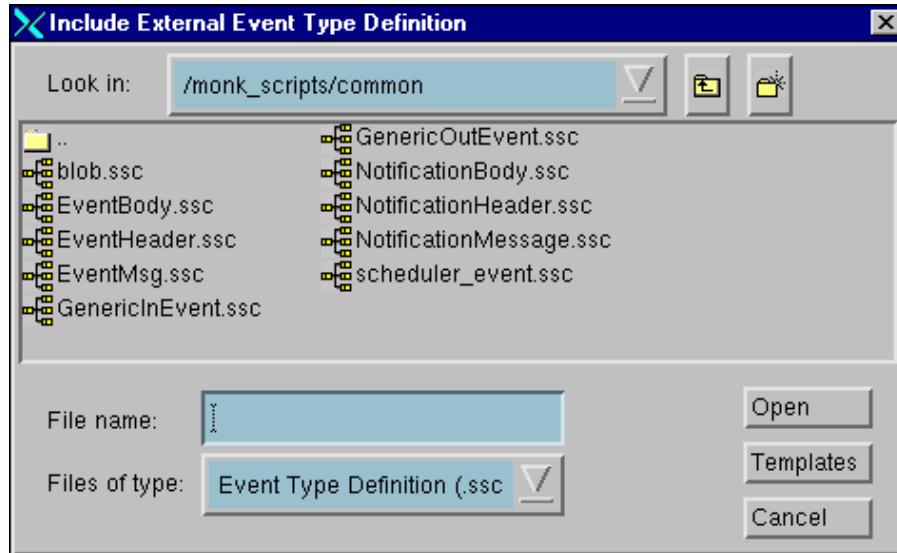
- Another ETD file in the current schema
- A file from the ETD library, which is included with your e*Gate software and consists of standard HL7, X12, EDIFACT, and other ETDs.

To include external templates in ETDs

- 1 Choose the **Templates > External Template** menu option.

The **Include External Event Type Definition** dialog box appears (see [Figure 107 on page 255](#)).

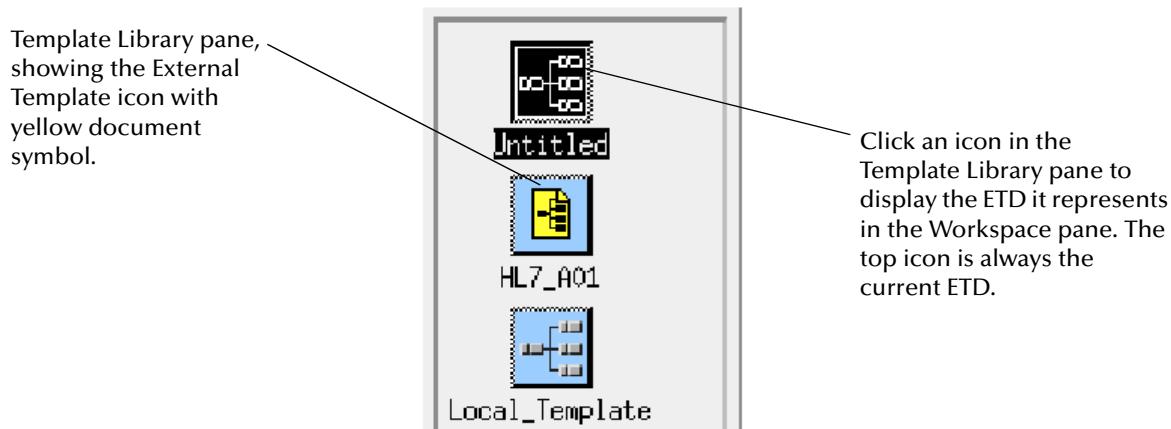
Figure 107 Include External Event Type Definition Dialog Box



- 2 Select the ETD you want to include as an external template.
- 3 You can also click **Templates** to select HL7, X12, EDIFACT, or Other ETDs.
- 4 Click **Open** to choose the desired template.

An icon that represents the external template appears in the Template Library pane (see [Figure 108 on page 256](#)). The new icon is distinguished from other icons by a yellow document symbol. You can click the icon to display the external template structure, but you cannot make changes to it.

Figure 108 ETD External Template Icons



- 5 In the ETD, click the left mouse button on the node or subnode to which you want to attach the external template as a child node.
- 6 Click the middle mouse button on the external template's icon in the Template Library pane.
The external template is added to the node you selected in step 5.
- 7 When you are finished with your template, save your changes.

Caution: *Keep in mind that when you include an external template in your ETD as a referenced file, any changes made to the source file is also made to the copy that is in your ETD. If you do not want an external template in your ETD automatically updated, choose the **Templates > Resolve** menu option (see “[Breaking Template Links](#)” on page 257).*

Changing ETD Repetition Properties

Inside the ETD where you included the external template, you can modify the external template's root node repetition properties, if you want.

To modify external ETD root-node repetition properties

- 1 Double-click the external template's root node label to display the **Node Properties** dialog box, and modify the repetition information.
- 2 Click **OK** to save the new repetition properties and close the **Node Properties** dialog box.

By adjusting the external template's root node repetition properties, you are telling the system how many times you want that (external) ETD to repeat in the current ETD. You can specify different repetition properties for each external template you include in your ETD.

For details about how to specify repetition information, using the **Node Properties** dialog box, see “[Adding Delimited-ETD Nodes](#)” on page 228.

Breaking Template Links

You can break the link between the external template source file and the copy in its associated ETD.

To break a link between an ETD and its template

- 1 Select the external ETD template's root node.
- 2 Choose the **Templates > Resolve** menu option.

Complete this step *only* if you do not want any changes made to the external template source file to update the copy in your ETD, or if you want to edit the external template in the ETD Tree where it is used.

6.6.2 Using Internal Templates

Internal templates provide an easy way to create a portion of an ETD, such as a node and its subnodes, save it, and reuse it as you build an ETD. Use internal templates for any node combination that occurs in multiple places.

Internal templates are different from external templates because they are only available within the ETD in which you create them. The main advantage of an internal template is that if you must update it, you can change the template. All instances of that template in your ETD are automatically updated when you change the template.

See “[Modifying Internal Templates](#)” on page 253 for details on how to update an internal template.

Creating Internal Templates

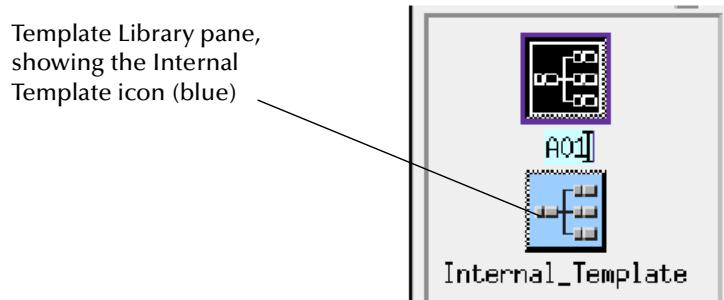
Once you have created internal templates, you can then use them as you build an ETD. See “[Referencing Internal Templates in ETDs](#)” on page 258 for more information. You can create an internal template at any point in the process of creating an ETD.

To create internal templates

- 1 Choose the **Templates > New Internal Template** menu option.

The Internal Template icon appears in the Template Library pane (see Figure 109 below).

Figure 109 Internal Template Icon



- 2 Click the Internal Template icon in the Template Library pane to select it.
The icon turns black when selected and the Workspace pane goes blank so you can create the new internal template in it.
- 3 Build your internal template the same way you build an ETD. For details, see “[Edit Menu](#)” on page 212.

Note: *The internal template icon in the Template Library pane is automatically labeled with the internal template’s root node name.*

- 4 Click the ETD icon at the top of the Template Library pane to redisplay your ETD.
If you already began to build an ETD when you started this procedure, that structure reappears in the Workspace pane.

Converting Existing ETDs

An alternate method of creating internal templates is to convert a portion of an ETD to an internal template.

To convert existing ETDs to internal templates

- 1 In your ETD, select the node you want to convert to an internal template. Note that any subnodes associated with the selected node are also converted to an internal template.
- 2 Choose the **Templates > Templify** menu option.

The selected node and its associated subnodes redisplay with green spines, which means they are part of an internal template. An internal template icon appears in the Template Library pane with the name of the selected node below it.

Referencing Internal Templates in ETDs

Once you have created internal templates, you can use them to build your ETD.

To reference internal templates in ETDs

- 1 If you have not yet built your ETD, you must do so. Follow the procedures explained under “[Edit Menu](#)” on page 212.
- 2 Select the node or subnode you would like attached to the internal template.
- 3 From the Template Library pane, click your middle mouse button on the internal template you want to include in the ETD.
The internal template appears under the node or subnode you selected in step 2.
- 4 Repeat steps 2 and 3 for each internal template you want to add to your ETD.

Changing ETD Repetition Properties

Inside the ETD where you included the internal template, you can modify the internal template’s root node repetition properties.

By adjusting the internal template's root node repetition properties, you are telling the system how many times you want that (internal) ETD to repeat in the current ETD. You can specify different repetition properties for each internal template you include in your ETD.

To modify internal ETD root-node repetition properties

- 1 Double-click the internal template's root node label to display the **Node Properties** dialog box.
- 2 Modify the root node's repetition information as desired.
For details about how to specify repetition information in the **Node Properties** dialog box, see "[Adding Delimited-ETD Nodes](#)" on page 228.
- 3 Click **OK** to save the new repetition properties and close the **Node Properties** dialog box.
- 4 When you are finished with your template, save your changes.

Java Collaboration Rules

To transport and transform data, you need code that specifies where to find the data, how to understand it, and what to do with it. For Java-enabled components, this code consists of methods supplied with the ETDs, used in conjunction with Business Rules—methods and logic you define using the Java Collaboration Rules Editor.

7.1 About This Chapter

This chapter consists of the following sections:

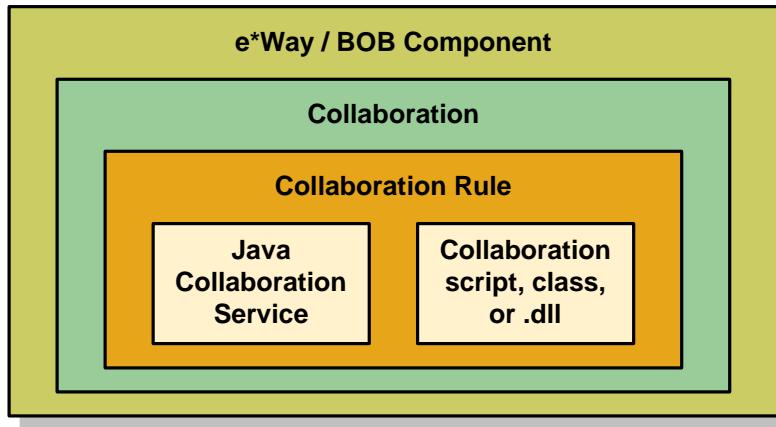
- A brief overview of Java Collaboration Rules—what a Collaboration Rule is and how it fits into the e*Gate system.
- A brief discussion of the methods associated with each node of an ETD and with the ETD itself.
- A detailed description of the Java Collaboration Rules Editor—its features, its graphical user interface (GUI), and the files it uses and creates.
- Basic procedures that tell you how to:
 - ◆ Create a new Java Collaboration Rule.
 - ◆ Work with simple mappings between two standard ETDs.
 - ◆ Use viewing options to give yourself more screen space, display code with the GUI, or view a separate window containing just the generated Java code.
 - ◆ Save, compile, and promote a Java Collaboration Rule.
 - ◆ Troubleshoot errors and warnings from the compiler.
 - ◆ Enable or disable Event Linking and Sequencing (ELS).
 - ◆ Browse for methods in classes and packages supplied by third parties.
 - ◆ Add and modify classpaths and packages.
 - ◆ Create and use your own custom Java methods.
- In-depth explanations of all Business Rules provided by the Java Collaboration Rules Editor, with short sample code segments.

For explanations of the classes and methods you can use in Collaborations between standard ETDs, see “[Java Classes and Methods](#)” on page 542. This appendix also provides instructions on creating and including your own custom methods.

7.2 Learning About Java Collaboration Rules

In the same way that every Event must have an Event Type and the skeleton of an Event Type is its ETD, every e*Way or BOB participates in one or more Collaborations, and the heart of a Collaboration is its Collaboration Rule; see Figure 110 below.

Figure 110 Relationship of Collaboration Rules to e*Gate Components



A Collaboration Rule takes its character from its Collaboration Service—a dynamic-link library (DLL) file that bridges differences between different hosts and operating systems. A Collaboration is called “Java-enabled” when its Collaboration Rule uses the Java Collaboration Service (sometimes abbreviated **JCS**). For complete information on all e*Gate Collaboration Services, refer to the *Collaboration Services Reference Guide*.

A Java Collaboration always goes through the following three phases, at a minimum:

- **userInitialize()**, where the system is set up. Normally, this contains no user code, but see “[Character Encodings in the Java Collaboration Rules Editor](#)” on [page 581](#).
- **executeBusinessRules()**, where the system runs the Business Rules you supply. The **executeBusinessRules()** block is run each time triggered Events are available.
- **userTerminate()**, where the system does final housecleaning after the rules are run.

Collaborations that use Event Linking and Sequencing (ELS, discussed in [Chapter 12](#)) have additional standard methods that allow them to link Events together. ELS-enabled Collaborations can call the Business Rules block—the **executeBusinessRules()** method and all user-written code under it—as often as needed, on a per-group basis.

7.2.1 Files Used by Java Collaboration Rules

Before you can even create a Java Collaboration Rule, you must give it the names of the ETDs (.xsc files) whose data it will transport and transform. The Rule’s classpath includes the package names of the .xsc files used by inbound and outbound Events; you can supply additional packages and classpaths using the Editor. Default initialization settings for the Rule are stored in its .ctl file; you can override some settings at run time.

Editing a Rule requires two other file types: For example, if you create and save a Rule named **cr_MyRule**, the Editor creates **cr_MyRule.xpr** and **cr_MyRule.xls**. An **.xpr** file is a project file to keep track of GUI settings and preferences as well as pointers to other files; an **.xls** file keeps track of the data transformations presented in the GUI. You can save the GUI settings without compiling; this allows you to resume unfinished work.

An actual Rule consists of compiled code: It is a Java object, and thus a **.class** file. When you compile the Rule named **cr_MyRule**, you create files **cr_MyRule.class** and **cr_MyRuleBase.class** as well as the Java source code, **cr_MyRule.java**, and a control file, **cr_MyRule.ctl**.

Caution: *Do not edit any of these files outside of e*Gate; the Java Collaboration Editor requires that they be kept in synch.*

Table 34 Java Collaboration File Types

File Type	Description
.class	Executable bytestream. When the Collaboration Rules file is successfully compiled, the result is two .class files: <CollabRuleName>.class and <CollabRuleName>Base.class .
.ctl	Control file. Contains initialization information for the Collaboration Rule. Used by the e*Gate registry to track dependencies.
.java	Java source code. The Editor generates this file so you can view the Java code it creates and the debugger can display individual lines of source code; however, you cannot modify this file — the Editor discards any changes made outside of it.
.xpr	Project file. Keeps track of Editor settings and options.
.xls	Transformation script. Contains details of the business rules as they are displayed by the Editor.

7.3 Where Do Methods Come From?

The previous chapters' discussion of Event Type Definitions (ETDs) focused entirely on the structural characteristics of their data—hierarchical relationships between nodes, boundaries set by delimiters or fixed record-lengths, and the various properties of each element and node. But ETDs also have a dynamic quality, in that every ETD and every node within the ETD has methods that define what you can do with the data.

Example: What's for Dinner?

The following XML is a DTD for an object called **dinner**.

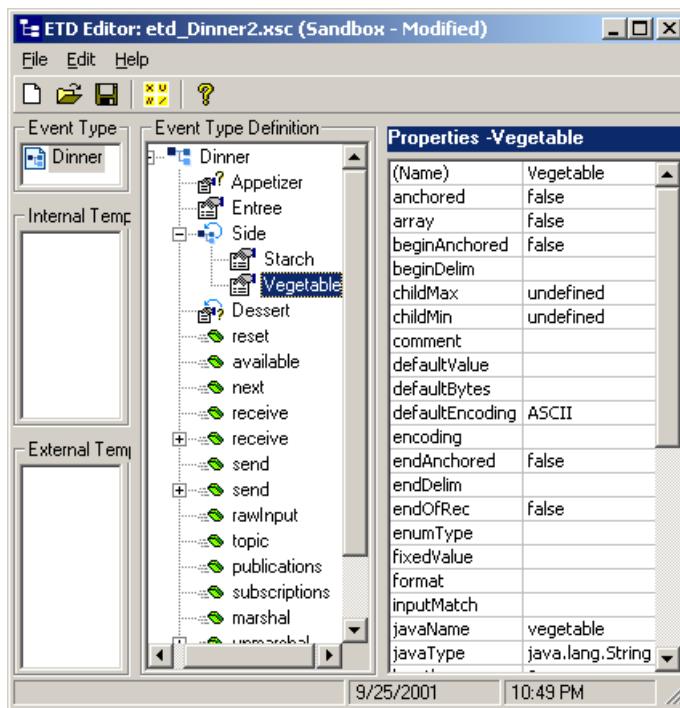
```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT Dinner (Appetizer?, Entree, Side+, Dessert*)>
```

```
<!ELEMENT Appetizer (#PCDATA)>
<!ELEMENT Entree (#PCDATA)>
<!ELEMENT Side (Starch | Vegetable)>
<!ELEMENT Starch (#PCDATA)>
<!ELEMENT Vegetable (#PCDATA)>
<!ELEMENT Dessert (#PCDATA)>
```

In other words, a dinner comprises: zero or one appetizer, exactly one entree, at least one side dish (which must be either a starch or a vegetable), and zero or more desserts.

When you use this **.dtd** file as input to the DTD Wizard and create an ETD, the output looks like Figure 111.

Figure 111 Dinner Seen as an Event Type Definition



Saving the ETD creates a 7550-byte **.xsc** file that contains all the property names and values. It also creates a 22886-byte **.jar** file. How did processing of a 287-byte **.dtd** file result in all this?

The extra size comes from the methods and other code generated by the ETD:

- Every Standard ETD is automatically given methods like **reset()**, **available()**, **next()**, and so on, to allow the Collaboration to work with the ETD in its entirety—for example, to marshal the data in the next Event of this Event Type. For complete information on these methods, see [“Methods for Standard Java-enabled ETDs” on page 333](#). For information on methods that are specific to certain imported ETDs, consult the user’s guide for the appropriate e*Way, ETD Builder, or toolkit.
- Since each parent node and leaf node is capable of having its data read and written, the ETD generates a **getNode()** method and a **setNode()** method for every node except the Event Type root, such as:

- ♦ `java.lang.String getAppetizer()`—retrieves current data in the **appetizer** node. The data type of the return value comes from the **javaType** property.
- ♦ `void setEntree(java.lang.String val)` — puts *val* into the **entree** node. The data type of the input argument *val* comes from the **javaType** property for **entree**.
- ♦ `void setDessert(int i, java.lang.String val)`— for repeating nodes, the `getNode()` and `setNode()` methods are constructed so as to require a counter argument.
- For any node whose **minOccurs** property has a value of **0** (and thus might not be present), the ETD generates a boolean `hasNode()` method, such as:
 - ♦ `boolean hasAppetizer()`
- For any node whose **maxOccurs** value is **unbounded** or greater than its **minOccurs** value, the ETD generates a boolean `countNode()` method for all such nodes, such as:
 - ♦ `int countDessert()`
- For repeating nodes that contain subnodes, the ETD generates different signatures for the different types of `getNode()` and `setNode()` methods that might be needed:
 - ♦ `Side getSide(int i)`
 - ♦ `Side[] getSide()`
 - ♦ `void setSide(int i, Side val)`
 - ♦ `void setSide(int i, java.lang.String val)`
 - ♦ `void setSide(Side [] val)`
 - ♦ `int countSide()`

7.4 Java Collaboration Rules Editor Overview

The Java Collaboration Rules Editor is the GUI for creating and modifying Java Collaboration Rules.

7.4.1 Feature Overview

The Java Collaboration Rules Editor provides the following features:

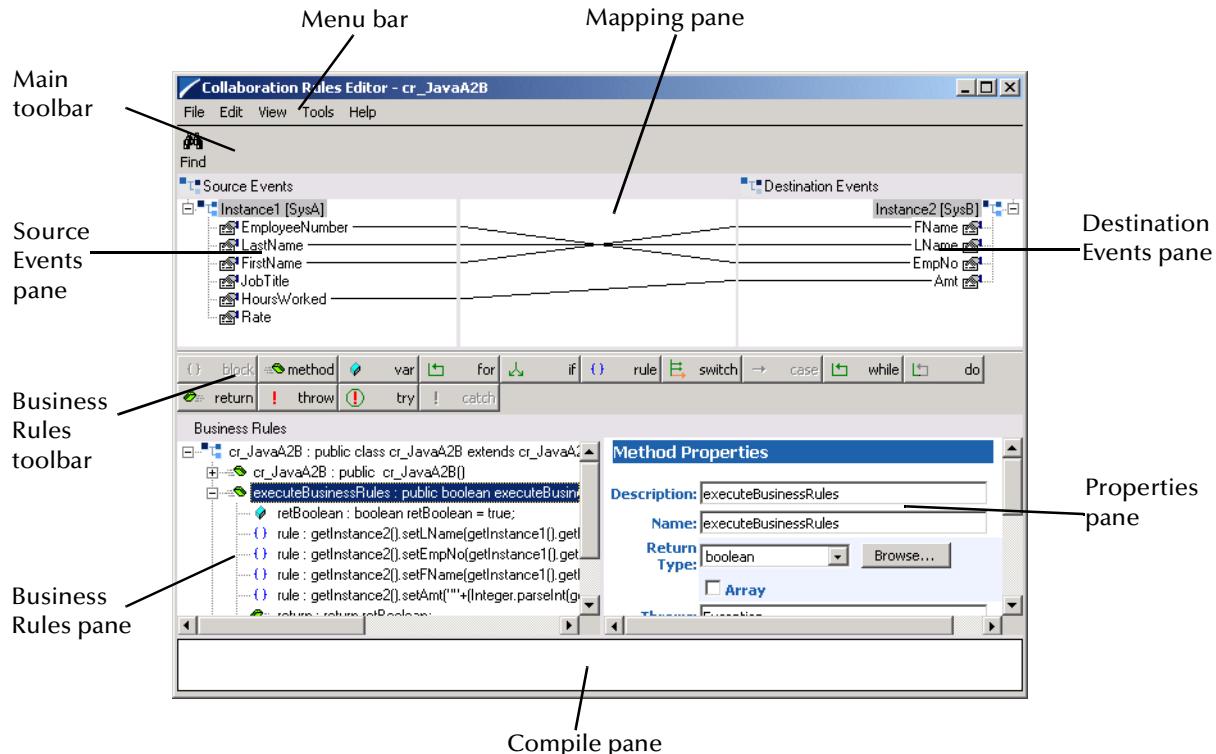
- Multiple source and destination Events. The Java Collaboration environment allows you to have multiple Event Type instances on the Source and Destination panes of the Collaboration and to freely collaborate between them.
- Mouseover tooltips for each field, node, and method of each ETD.
- Right-click (shortcut menu) access to the properties of each field, node, and method.
- Right-click (shortcut menu) access to classes, packages, and methods in **.jar** files and classpaths you have imported.
- Graphical depiction of the relationships between source and destination nodes.
- Self-documenting code using the area provided in the Properties pane.

- Ability to view the raw Java source code from within the Editor.
- Cut, copy, and paste of text as well as multiple nodes in the Business Rules pane.
- Access to Event Linking and Sequencing (ELS) via a simple three-step wizard.

7.4.2 GUI Overview

In addition to the menu bar and the accompanying toolbar, the Java Collaboration Rules Editor GUI has six panes—the Source Events, Mapping, Destination Events, Business Rules, Properties, and Compile panes—and a Business Rules toolbar running horizontally through the center of the GUI, containing buttons used to insert common programming constructs into the Collaboration Rule. Figure 112 shows the names and locations of the different areas of the GUI.

Figure 112 Java Collaboration Rules Editor GUI Map



The Editor GUI Panes

Source Events Pane

Contains the ETDs that correspond to the Event Types that are *subscribed to* by the e*Gate Collaboration Rule. For any field, node, or method, a mouseover (point and wait) displays an informational tooltip, and a right-click displays a shortcut menu that allows you to view the properties of the field, node, or method.

Mapping Pane

Shows the relationship between the Source and Destination ETDs as a series of lines connecting the associated nodes.

Destination Events Pane

Contains the ETDs that correspond to the Event Types that are *published* by the e*Gate Collaboration Rule. As with the Source Events pane, a mouseover on any field, node, or method displays an informational tooltip, and a right-click displays a shortcut menu that allows you to view the element's properties.

Business Rules Pane

A graphical depiction of the Java source code that creates the output Events that are the result of the Collaboration. Much of the code displayed here is generated automatically by the Editor. The `executeBusinessRules()` method is a placeholder for additional code you provide.

Properties Pane

Displays information about the rule selected in the **Business Rules** pane. This is also the area where you edit the selected rule. A mouseover on any property label displays an informational tooltip for that property, and a right-click displays a shortcut menu that allows you to cut, copy, or paste, to Undo the most recent action, or to insert a Java method from any defined class.

Compile Pane

Displays informational and error messages that are output as a result of compiling the Java Collaboration.

7.4.3 Menu Commands

Table 35 below details the commands available from the Java Collaboration Rules Editor menu bar.

Table 35 Java Collaboration Rules Editor Menu Commands

Menu	Command	Description
File	Open	Opens the Open dialog box, allowing you to navigate to and select an .xpr file associated with the Java Collaboration Rule you want to edit.
	Save	Regenerates the .xpr and .xts files for the current Rule, saving your current GUI settings and work-in-progress without compiling.
	Save As	Opens the Save dialog box, allowing you to generate new .xpr and .xts files for the current Rule and save it under a different name without compiling it.
	Compile	Runs the Java compiler and displays informational and error messages in the Compile pane.
	Promote	Moves the current Collaboration Rule from the sandbox environment to the run-time environment.
	Enable ELS	Provides access to the Event Linking and Sequencing (ELS) methods that act as a preprocessor to the standard business rules. For details, see Chapter 12 .
	Enable Marshalling Error Trapping	Provides access to the Marshalling Error Trapping methods: onMarshalException() and onUnmarshalException() .
	Enable Transaction Synchronization	Provides access to the Transaction Synchronization methods: OnPrepare() , OnCommit() , and OnRollback() .
Edit	Exit	Closes the Editor.
	Cut	Removes the selection and copies the contents to the Clipboard.
	Copy	Copies the contents of the selection to the Clipboard.
	Paste	Pastes the contents of the clipboard to the cursor location.
	Delete	Deletes the selection.
	Search and Replace	Opens the Search And Replace dialog box, allowing you to locate text in various property fields and optionally substitute other text. See “Searching and Replacing Within a Collaboration” on page 287 .
	Find and Map	Opens the Find and Map dialog box, allowing you to locate nodes and fields in source and destination Event instances and, optionally, create mapping between a source field and a destination field. See “Using Find and Map” on page 278 .

Table 35 Java Collaboration Rules Editor Menu Commands (Continued)

Menu	Command	Description
View	Display Tool Labels	Displays or hides text labels for the buttons in the Business Rules toolbar.
	Display Code	Displays or hides the code in the Business Rules pane.
	Display Tool Bar	Displays or hides the main toolbar.
	Display Output	Displays or hides the Compile pane. When you compile a Collaboration Rule, Display Output is automatically turned on.
	View Java Code	Opens a View Java Code window that allows you to examine, select, and copy the contents of the Java source text file created by the Editor.
Tools	Options	Opens the Java Classpaths dialog box, allowing you to select (from <eGate>\client\) the Java classpaths that are used to import Java packages into the Java Collaboration Rule. See “ To make external (third-party or custom) .jar or .class files available to your Collaboration ” on page 282.
	Java Imports	Opens the Java Imports dialog box. See “ To view and use methods from an imported class ” on page 284.
	ELS Wizard	Starts the Event Linking and Sequencing Wizard. This command is available only when the Enable ELS command is checked under the File menu. See Chapter 12 .
Help	Contents	Opens the Help browser with the Contents tab showing.
	Index	Opens the Help browser with Index tab showing.
	Search	Opens the Help browser with Search tab showing. Not currently implemented
	About Collaboration Editor	Displays the copyright, version, and build information, a copy of the license agreement, and a command button (System Info) that allows you view system parameters.

7.4.4 Main Toolbar

The buttons on the main toolbar are shortcuts for commands that can be found on the menu bar. Table 36 gives a brief description of what each button does.

Table 36 Main Toolbar Buttons

Button	Description
	Opens the Find and Map dialog box, allowing you to locate nodes and fields in source and destination Event instances and, optionally, create mapping between a source field and a destination field. Same as the Find and Map command under the Edit menu. See “ Using Find and Map ” on page 278.

7.4.5 Business Rules Toolbar

The buttons on the Business Rules toolbar are used to insert common Java programming constructs into the Collaboration. Table 37 below gives a brief description of what each button inserts into the code.

Table 37 Business Rules Toolbar Buttons

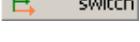
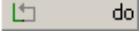
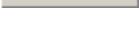
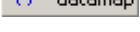
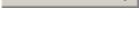
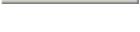
Button	Description
	Adds a rule that groups several Java statements together.
	Adds a rule that is a member function of a Java class.
	Allows you to define a variable.
	Adds a counter delimited loop to your code. The code within the for loop is executed until the counter value is reached.
	Adds a conditional expression statement to code.
	Adds an unspecified expression statement to your code. This allows you to use the methods associated with the ETD of the Event instance.
	Adds a switch statement that begins a set of conditional branches.
	Adds a conditional branch to a switch statement. You must be in a switch rule fragment for this button to be active.
	Adds an expression delimited loop to your code. The code within the while loop is executed as long as the expression is true.

Table 37 Business Rules Toolbar Buttons (Continued)

Button	Description
 do	Adds an expression delimited loop to your code. The code within the do loop is executed at least once and as long as the accompanying while expression is true.
 return	Returns processing to the caller.
 throw	Adds a rule that initiates an exception to your code.
 try	Adds an exception handling block to your code. An internal try...catch already exists in executeBusinessRules() , so that exceptions do not generate compilation errors; however, in userInitialize() and userTerminate() , you must add your own try...catch blocks.
 catch	Adds a catch block to your code. The catch block contains the code to handle exceptions thrown within the try block. This button is inactive when the rule selected is outside of a try block.
 copy	Opens the Copy dialog box, allowing you to add a copy rule to your code. See copy on page 294.
 datamap	Opens the Data Map dialog box, allowing you to add a datamap rule to your code. See datamap on page 296.
 list lookup	Opens the List Lookup dialog box, allowing you to add a listlookup rule to your code. See list lookup on page 307.
 timestamp	Opens the Insert Timestamp dialog box, allowing you to add a timestamp rule to your code. See timestamp on page 318.
 uniqueid	Opens the Insert Unique ID dialog box, allowing you to add a uniqueid rule to your code. See uniqueid on page 322.

7.5 Working With Java Collaboration Rules

This section provides basic information you need to know for working with Java Collaboration Rules:

- [“Creating a New Java Collaboration Rule” on page 271](#)
- [“The Mapping Pane” on page 275](#)
- [“The View Commands” on page 278](#)
- [“Saving, Compiling, and Promoting Collaboration Rules” on page 280](#)
- [“Enabling and Disabling ELS” on page 281](#)
- [“Setting Classpath and Package Options” on page 282](#)
- [“Searching and Replacing Within a Collaboration” on page 287](#)

7.5.1 Creating a New Java Collaboration Rule

You create a Java Collaboration Rule by designating one or more source Events and one or more destination Events and then setting up rules governing the relationship between fields in the Event instances.

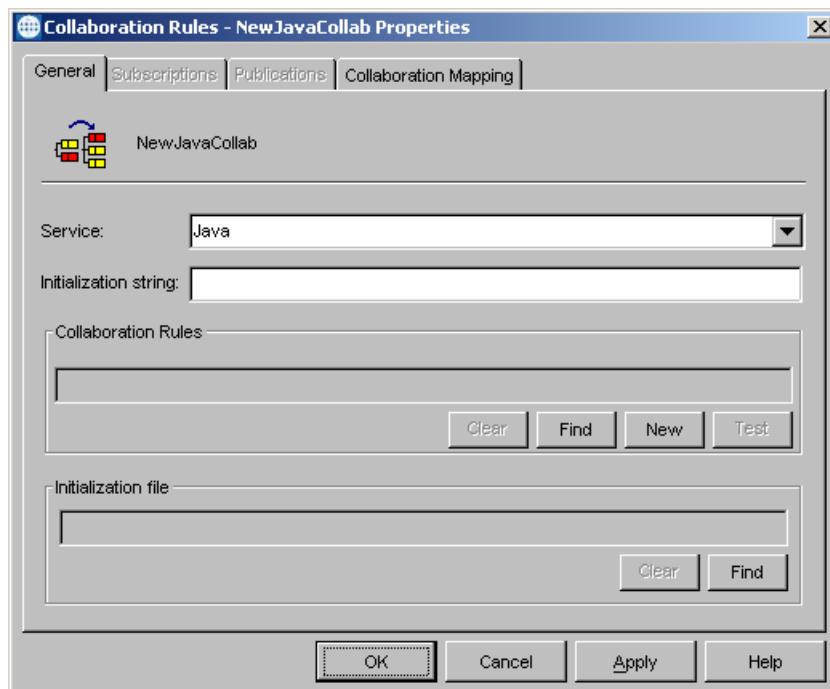
Note: *ETDs for source and destination must exist before creating the Collaboration Rule. For more information on creating the ETDs, see [Chapter 5 Event Type Definitions \(ETDs\)](#) on page 157.*

To create a new Java Collaboration Rule

- 1 In the Navigator (leftmost) pane of Enterprise Manager, select the Components tab if necessary and click the **Collaboration Rules** folder.
- 2 On the **File** menu, point to **New** and click **Collaboration Rules** .
- 3 Enter a name for the Collaboration Rule and click **OK**.
- 4 Double-click the new Collaboration Rule to display its properties.
- 5 In the **Service** list, click **Java**.

The **Collaboration Mapping** tab is activated.

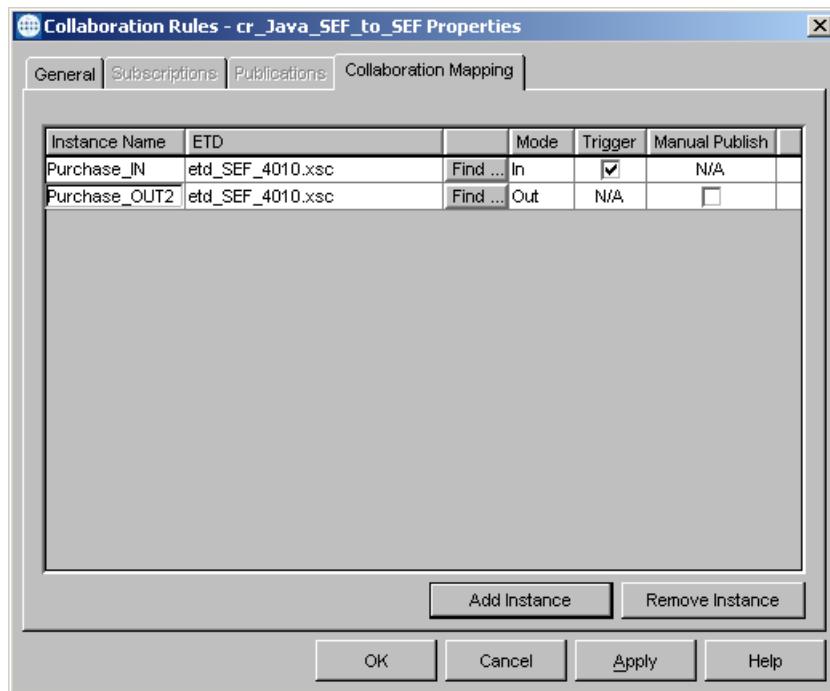
Figure 113 The Collaboration Rule Properties



- 6 Click the **Collaboration Mapping** tab.
- 7 Click **Add Instance** to add a new instance.
- 8 Enter an **Instance Name** for the instance. The **Instance Name** will be used by the Collaboration Rules Editor to identify the source and destination Events.

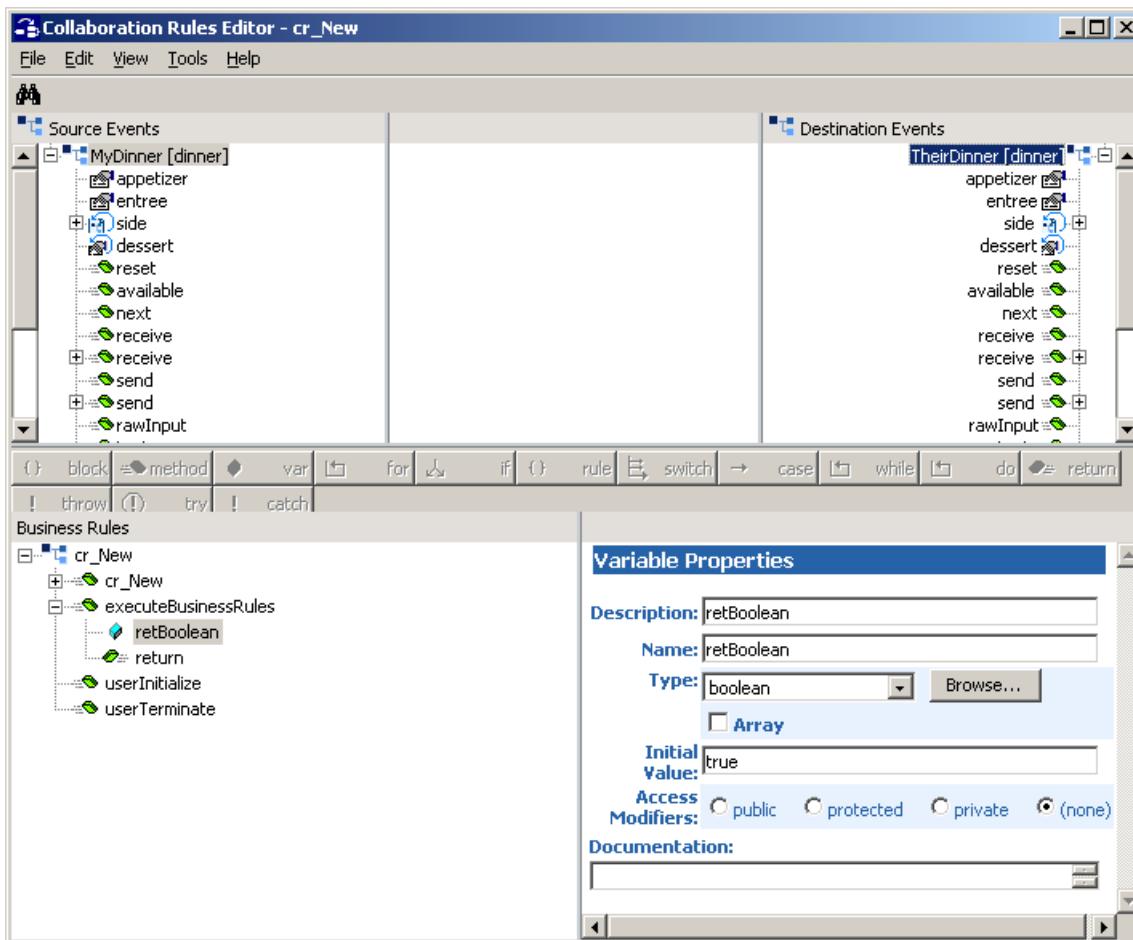
- 9 Click **Find** to display a list of ETD files (.xsc files).
- 10 Select the source ETD and click **Select**.
The name of the ETD is displayed in the **ETD** field.
- 11 In the **Mode** list, click **In** or **In/Out**. For a discussion of In, Out, and In/Out modes, see ["In, Out, and In/Out Event Instances" on page 274](#).
- 12 Optionally, repeat steps 7 through 11 to create additional source Event instances.
- 13 Select the **Trigger** check box for one or more inbound Events; see ["Trigger Check Box" on page 274](#).
- 14 Repeat steps 7 through 10 for each destination instance.
- 15 In the **Mode** list, click **Out** or **In/Out** for each destination instance.
- 16 Optionally, you can select the **Manual Publish** check box for one or more outbound Events; see ["Manual Publish Check Box" on page 274](#).
- 17 Click **Apply** to save the changes.

Figure 114 The Collaboration Mapping Tab



- 18 Click the **General** tab. Optionally, you can enter an initialization string to override certain run-time settings. For details, see the *Collaboration Services Reference Guide* chapter on the Java Collaboration Service.
- 19 Click **New** to create a new Java Collaboration Rule.
The Java Collaboration Rules Editor displays the newly created Rule. The source and destination Events are populated with the ETDs defined in the **Collaboration Mapping** tab of the Collaboration Rules properties. See Figure 115.

Figure 115 The New Collaboration Rule



When you drag a child node into the Rule area of the Rule Properties pane (or into the Result area of the Return Properties pane, or similar), or if you use drag-and-drop or **Find and Map** to map one node to another node, the Editor automatically traces the path of the selected node or nodes upwards to the root and supplies an appropriate **get()** or **set()** method:

- For a node in the Source Events pane, the Editor supplies a **get()** method to fetch data from it: **getSourceRoot().getSourceChild().getSourceGrandChild().[...].getNode()**.
- For a node in the Destination Events pane, the Editor supplies a **set()** method to put data into it: **getDestRoot().getDestChild().getDestGrandChild().[...].setNode()**.

If you drag a repeating node or the child of a repeating node, you are prompted by the **Select Repetition Instance** dialog box to specify which iteration to use.

- For simple one-to-one operations, such as copying the fifth instance of a nine-repetition node to the third instance of a five-repetition node, you can simply enter the appropriate numbers.
- For more complex operations, it can be helpful to set up a counter in a **for** loop; see “**for**” on page 303.

7.5.2 Settings for the Collaboration Rules Properties dialog box

The Collaboration Rules Properties dialog box has only two tabs that are accessible: **General** and **Collaboration Mapping**. (The other, inaccessible, tabs are provided for consistency with Collaboration Rules based on Collaboration Services other than Java.)

General Tab

In this tab, you specify the Collaboration Service (must be **Java** for Java Collaborations). You can optionally also specify one or more flags and values in the Initialization String box, as described the *Collaboration Services Reference Guide*.

In the Collaboration Rules area, you can click **Clear** to remove the current setting, click **Find** to select a different existing Collaboration Rule (.class) file, or click **New** or **Edit** to start the Collaboration Rules Editor.

Collaboration Mapping Tab

In this tab, you click **Add Instance** and supply an instance names and ETD (.xsc file) for each Event Types in your Collaborations, as well as a mode (In, Out, or In/Out).

In, Out, and In/Out Event Instances

Event instances are displayed in the Source Events pane and Destination Events pane according to the mode—**In**, **Out**, or **In/Out**—selected for them in the Collaboration Mapping tab: Event instances set to **In** appear in the Source Events pane only; Event instances set to **Out** appear in the Destination Events pane only; and Event instances set to **In/Out** appear in both Source Events and Destination Events panes.

Note: *You cannot use an In/Out Event with a JMS e*Way Connection. Also, you cannot connect the same In/Out Event to two or more different external sources—for example, if an In/Out Event instance is connected to a certain e*Way Connection, all instances of that Event must be connected to the same e*Way Connection.*

Trigger Check Box

Optionally, you can select the **Trigger** check box for all inbound Events where you want the `executeBusinessRules()` method to run whenever data is available. Clear the **Trigger** check box for any Events where you want the Event to be instantiated only when you explicitly invoke the `receive()` method. At least one inbound Event must be set to **Trigger**—otherwise, `executeBusinessRules()` never runs.

Manual Publish Check Box

Optionally, you can select the **Manual Publish** check box for outbound Events where you want the `send()` method to run only when explicitly called within the business rules. Clear the **Manual Publish** check box for Events where you want the Collaboration to invoke the `send()` method automatically.

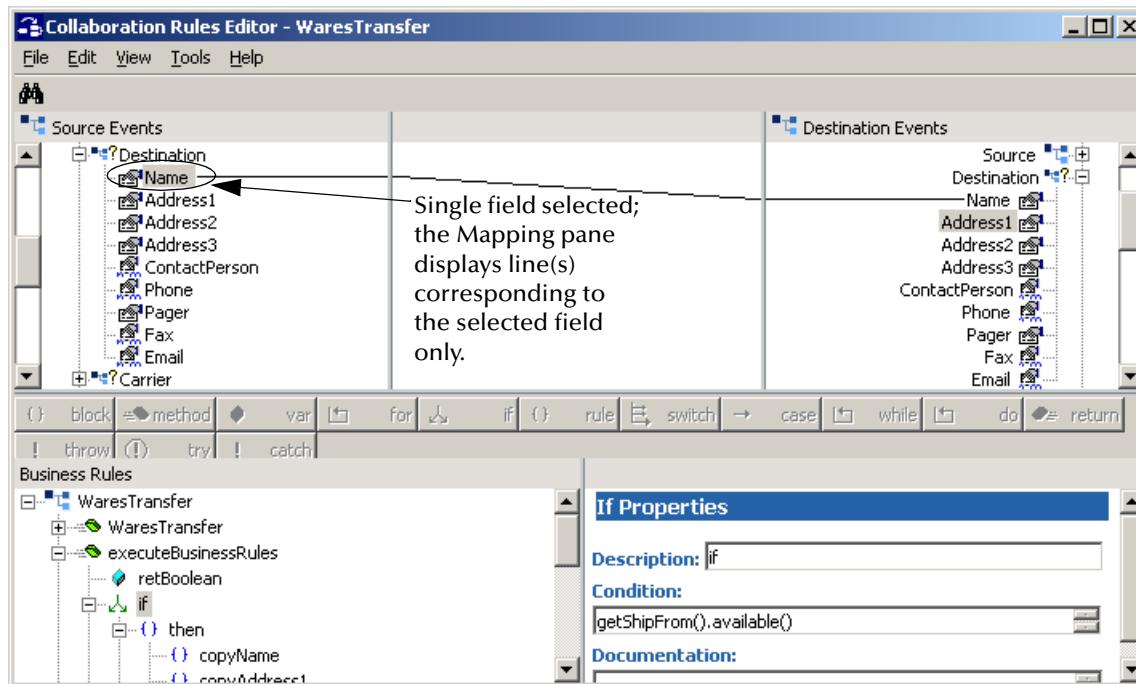
Note: *The **Trigger** check box is unavailable for outbound-only Events, as they are ineligible for the `receive()` method. The **Manual Publish** check box is unavailable for inbound-only Events, as they are ineligible for the `send()` method.*

7.5.3 The Mapping Pane

The Mapping pane provides a visual indication of the relationship between source and destination fields. When you use drag-and-drop or **Find and Map**, the Editor creates a new business rule that invokes the `getSourceField()` method on the selected Source Event field and the `setDestField()` method on the Destination Event field, and draws a line connecting the two fields.

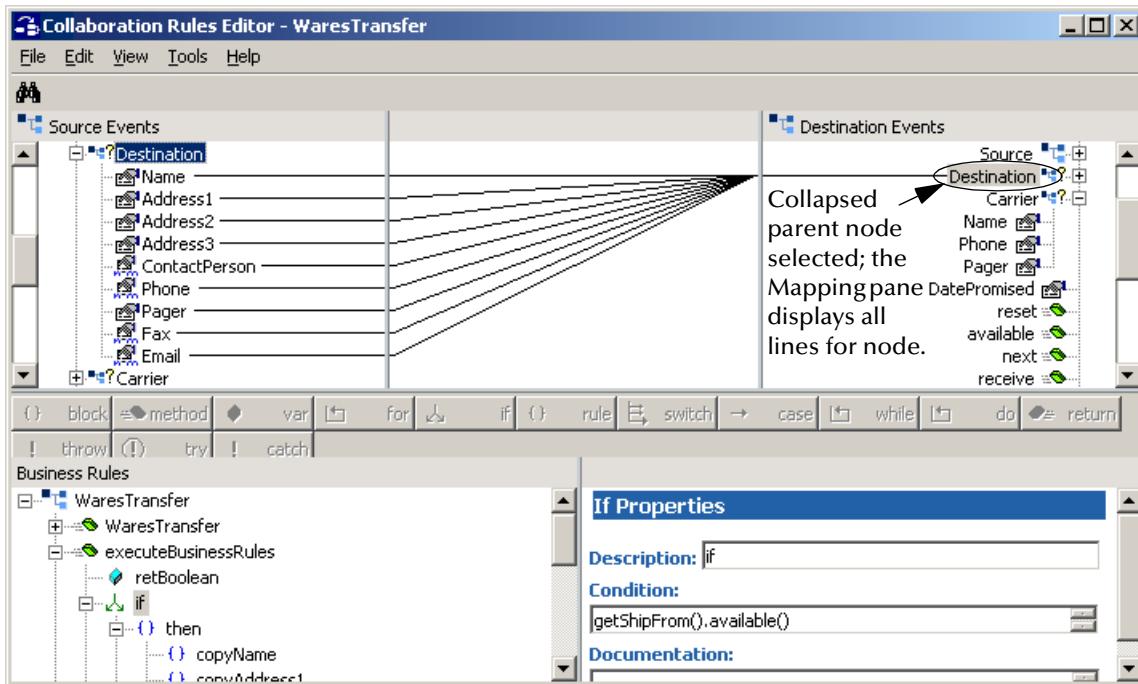
To minimize clutter, this line is displayed in the following circumstances: When either the source or the destination fields is selected, when a parent or grandparent (and so on) node of either field is selected, or when the corresponding business rule is selected. This is illustrated in the figures below. Each of the fields in the Source Events pane—**Name**, **Address1**, and so on—is copied to its corresponding field in the Destination Events pane. When just one field is selected, such as the **Name** field shown in Figure 116, the Mapping pane displays only the lines for that field.

Figure 116 Mapping Pane When One Field Is Selected



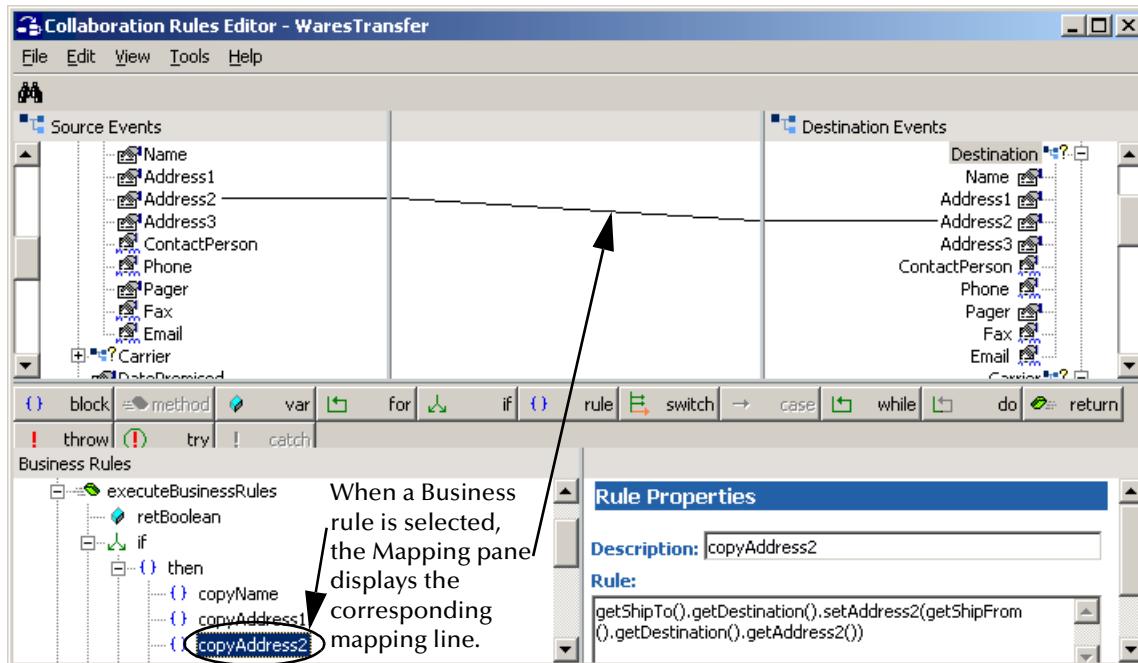
When a parent node is selected, as shown in Figure 117, the Mapping pane shows many connecting lines corresponding to the business rules for all the node's child fields; the mapping lines converge in a point if one of the parent nodes is collapsed.

Figure 117 Mapping Pane for Expanded and Collapsed Parent Nodes



When a business rule is selected, such as the **copyAddress2** rule in Figure 118, the Mapping pane shows connecting lines corresponding to the code in the fragment.

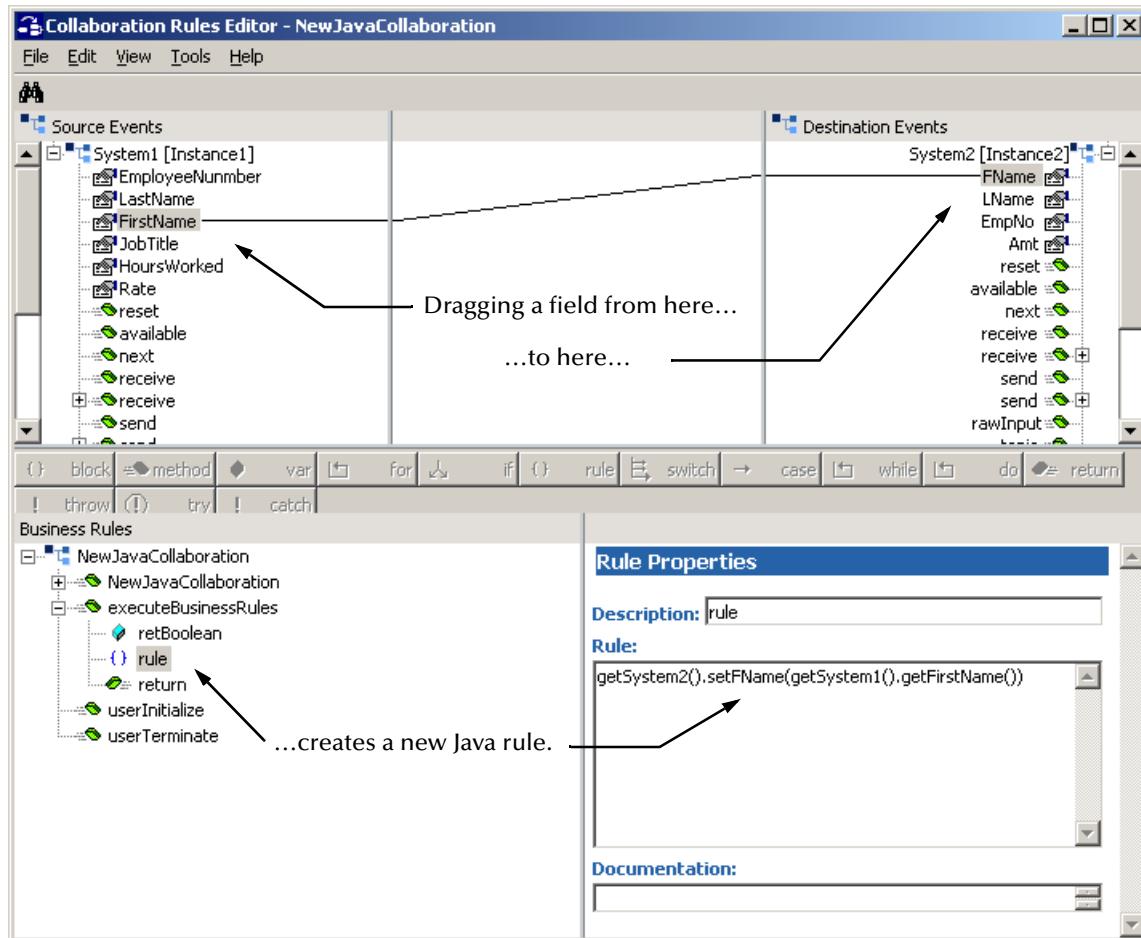
Figure 118 Mapping Pane When a Business Rule Is Selected



Dragging and Dropping Fields

In the Collaboration Rules illustrated in the three figures above, the code for getting and setting field data was generated by dragging the source field and dropping it onto the corresponding destination field, within the context of a specific business rule—in the three Figures above, the context is the **then** block of the first **if** rule component in the **executeBusinessRules()** method, whereas in Figure 119 below, the context is the **executeBusinessRules()** method itself.

Figure 119 Creating Rules by Dragging Fields



You can map a parent node to a parent node, provided that both nodes' tree structures are identical in every way. For example, a node with six children and no grandchildren can map to another node with exactly six children *provided that the data types of the children fields are compatible and all names identical*—thus not to a node with two children or nine, or one with grandchildren, or one whose children are named differently.

Data Type Conversion

If you use drag-and-drop (or the **Find and Map** command—see below) to map between fields of different data types, the Editor issues a warning if appropriate (for example, if you try to map a **String** to a **long**, but not if you map a **long** to a **String**) and invokes one of the **e*Gate STCTypeConverter.toDataType()** methods.

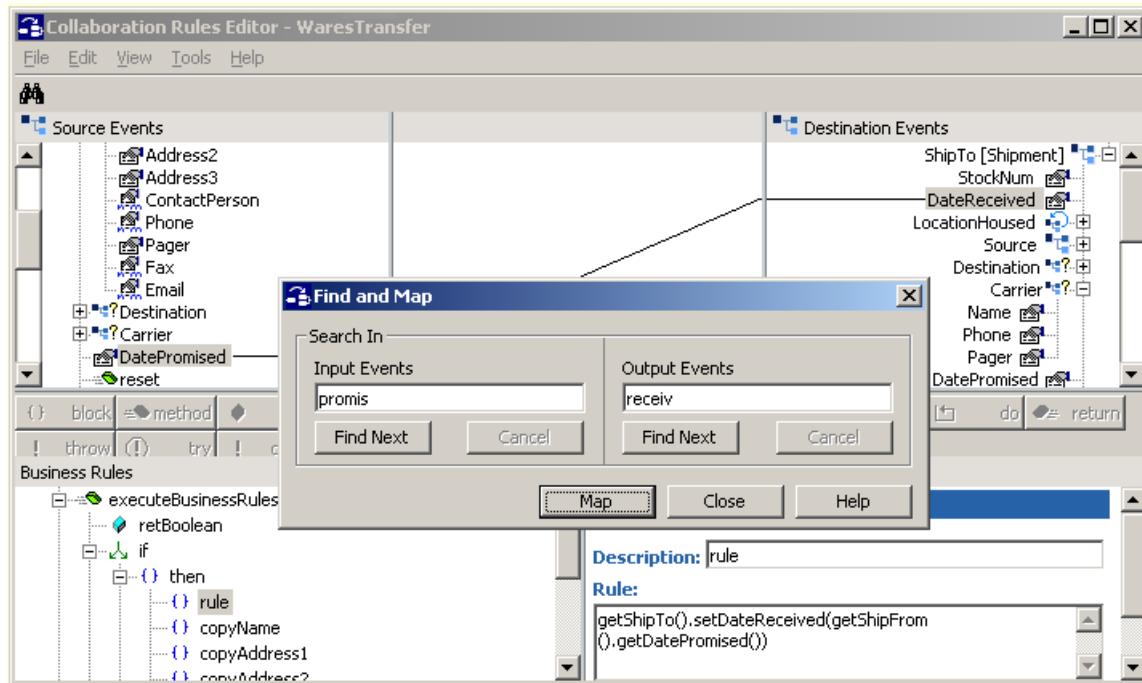
Using Find and Map

For large ETDs, or for complicated mappings, you can use **Find and Map**  to locate fields and nodes in the source and destination Events. To locate and highlight an element in the source Event, enter a text string in **Input Events** box and click **Find**; if necessary, continue to click **Find Next** until the correct element has been located. To locate and highlight an element in the destination Event, enter a text string in **Output Events** box and click **Find** or **Find Next**.

After you have found the correct elements in both the source and the destination ETDs, you can scroll through the Business Rules pane, select or add the rule component that sets the context of the get/set operation, and click **Map**. The Editor acts as if you had dragged the source to the destination, and creates the corresponding code fragment in a new **rule** rule component.

As you can see in Figure 120 below, the text string you enter can be a substring of the field or node name you want to match, and the search is case-insensitive.

Figure 120 Find and Map



7.5.4 The View Commands

The **View** menu has four commands that allow you to change the display of the GUI:

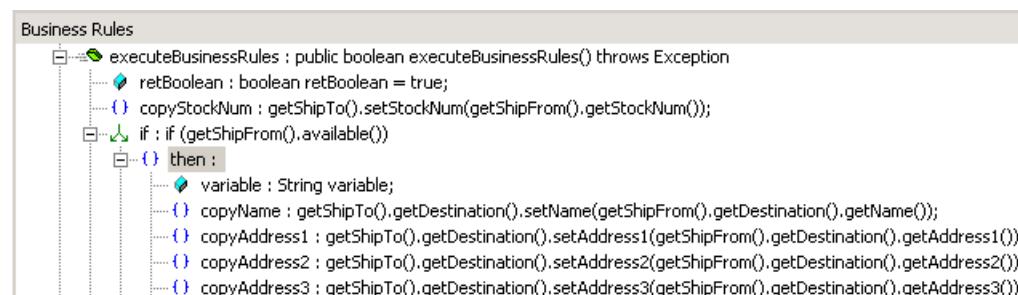
- Keep **Display Tool Labels** turned on while you are learning the interface; when you are familiar with the toolbar icons, you can save some screen space by turning off the labels. All examples in this book have the tool labels turned on. Figure 121 below shows the toolbar with the **Display Tool Labels** turned off.

Figure 121 Business Rules Toolbar With Labels Turned Off



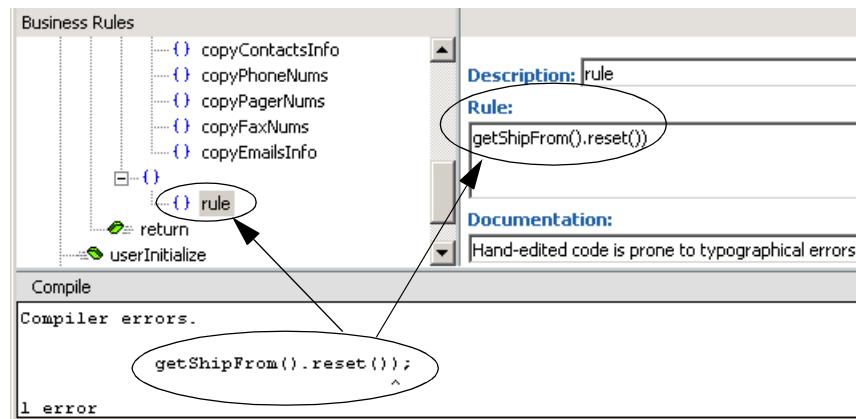
- Turn on **Display Code** when you want to quickly browse through all the code in the Business Rules pane. To provide more space as you browse, temporarily slide the pane divider to the right edge of the screen, and then restore it to the middle when you are done. All examples in this book have the code display turned off. Figure 122 shows the Business Rules pane with **Display Code** turned on.

Figure 122 Business Rules Pane with Code Display Turned On



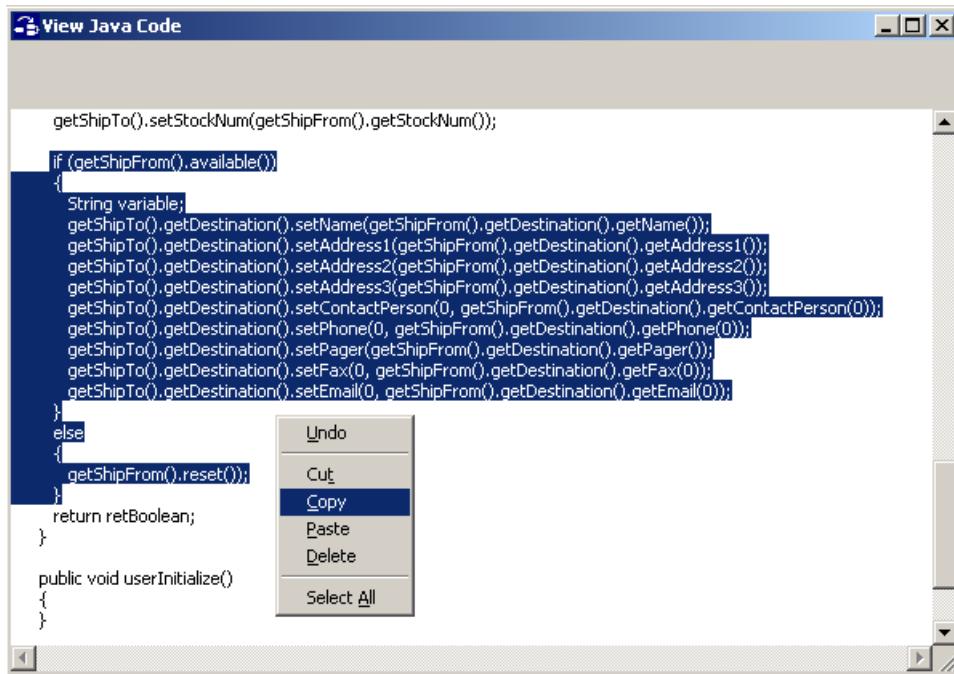
- Turning on **Display Output** allows you to view the Compile pane. This helps you to work with a Collaboration Rule that doesn't compile cleanly—just double-click the compiler message to place focus on the rule that apparently caused the problem. For an example involving unmatched parentheses, see Figure 123 below.

Figure 123 Code Error Exposed by Double-Clicking a Compile Pane Message



- Turning on **View Code** displays the **View Java Code** window, which allows you to see the Java source code generated by the Editor. Although you cannot search the code stream or use keyboard commands like Delete or CTRL+C, you can highlight a selection and right-click to use the shortcut menu commands **Cut**, **Copy**, **Paste**, **Delete**, and **Select All**. However, it is not useful to make any changes within this window, since all changes are erased when the **View Java Code** window is closed. For an example, see Figure 124 below.

Figure 124 The View Java Code Window



7.5.5 Saving, Compiling, and Promoting Collaboration Rules

While working on a Collaboration Rule, you may want to save your work from time to time before the Rule is ready to be compiled. For example, if an ETD changes while you are working on a Collaboration Rule using that ETD, you should save your work, close the Editor, and then re-start the Editor so you can incorporate any changes.

When you have completed work on a Collaboration Rule, you must compile it before you can use it in a Collaboration. When a compilation is unsuccessful, the Compile pane displays messages that can be helpful in locating and correcting errors. Any time there is a change to any ETD referenced by a Collaboration, the Collaboration must be recompiled.

After the Collaboration Rule is tested and approved, you can promote it to run-time.

Procedures for saving, compiling, and promoting Collaborations are given below.

To save the Collaboration Rule without compiling

Do one of the following:

- On the **File** menu, click **Save**.

This regenerates the current **.xpr** and **.xts** files that save the current project settings (GUI layout) and details of the Collaboration Rule.

- On the **File** menu, click **Save As** and then, in the Save dialog box, enter a new file name for the Collaboration Rule.

This generates new **.xpr** and **.xts** files.

To compile the Collaboration Rule

- 1 On the **File** menu, click **Compile**.
- 2 Check the Compile pane for errors and warnings. If the compilation succeeds without problem, the Compile pane will contain only one message:

Compile Completed.

Successful compilation causes the `collabRule.ctl`, `collabRule.class`, and `collabRuleBasic.class` to be committed to the sandbox environment.

Resolving compiler errors

With most common errors, you can get a good clue to the location of the problem by double-clicking the error text in the Compile pane. For example, see [Figure 123 on page 279](#).

To promote the Collaboration Rule from sandbox to run time

- On the **File** menu, click **Promote**.

A dialog box displays the message **Successfully Promoted all Files to Runtime**.

Note: If the compile is unsuccessful, it fails to commit .ctl or .class files to the Sandbox environment. Therefore, if you attempt to promote a new Collaboration Rule that has never been successfully compiled, an error dialog box displays error messages prompting you to save your session before promoting to the run-time environment. However, after a Collaboration Rule has been successfully compiled at least once, the .ctl and .class files from the most recent successful compilation are committed to the Sandbox.

7.5.6 Enabling and Disabling ELS

Event Linking and Sequencing (ELS) consists of a set of independent methods that can be invoked prior to the `executeBusinessRules()` method. These methods allow you to link a group of Events that share a defined unique key (called a *Link Identifier*) and then work with the group as a whole using their Link Identifier, Event Type (also known as *Topic*), and age. Processing of these groups can be sequenced relative to one another because their timers can be set, stopped, reset, or restarted according to need. For a complete discussion of ELS, see [Chapter 12](#).

When ELS is enabled, you can use the **ELS Wizard** command under the **Tools** menu.

To enable ELS

- On the **File** menu, click **Enable ELS** if it is not already checked.

Three new methods are displayed in the Business Rules pane: `retrieveLinkIdentifier()`, `isLinkingComplete()`, and `onExpire()`.

To disable ELS

- 1 On the **File** menu, clear the **Enable ELS** check mark (if it is already checked).
- 2 In the **Disable ELS** dialog box, click **Yes**.

All ELS methods are deleted from the Collaboration Rule, and the **Enable ELS** menu check mark is cleared.

7.5.7 Setting Classpath and Package Options

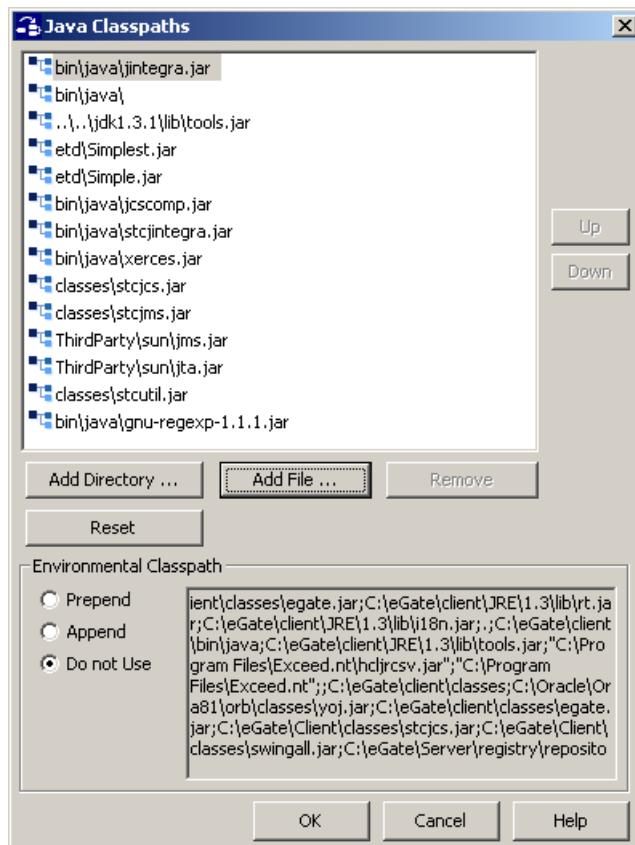
To add classpaths or packages to your Collaboration Rule, use the **Tools** menu:

- The **Options** command opens the **Java Classpaths** dialog box, whose **Classpath** tab displays the default classpath and allows you to add directories and files.
- The **Java Imports** command opens the **Java Imports** dialog box, which allows you to add or remove packages and classes. To add a package or class, you can either enter its name and click **Add**, or you can navigate to it in the **Classes in Classpath** list, click the package name, and then click **>>**. You can move a package higher or lower in the **Selected** list sequence by selecting it and then clicking **Up** or **Down**.

To make external (third-party or custom) .jar or .class files available to your Collaboration

- 1 Use the operating system to copy the **.jar** files into **<eGate-client>\classes** and **<eGate-registry>\repository\<YourSchemaName>\runtime**. Although the Editor allows you to find and use a **runtime\>.jar** file even if it is not in **client\classes**, the compile will fail, because it cannot generate the **.ctl** file.
- 2 In the Collaboration Rules Editor, on the **Tools** menu, click **Options**. See Figure 125.

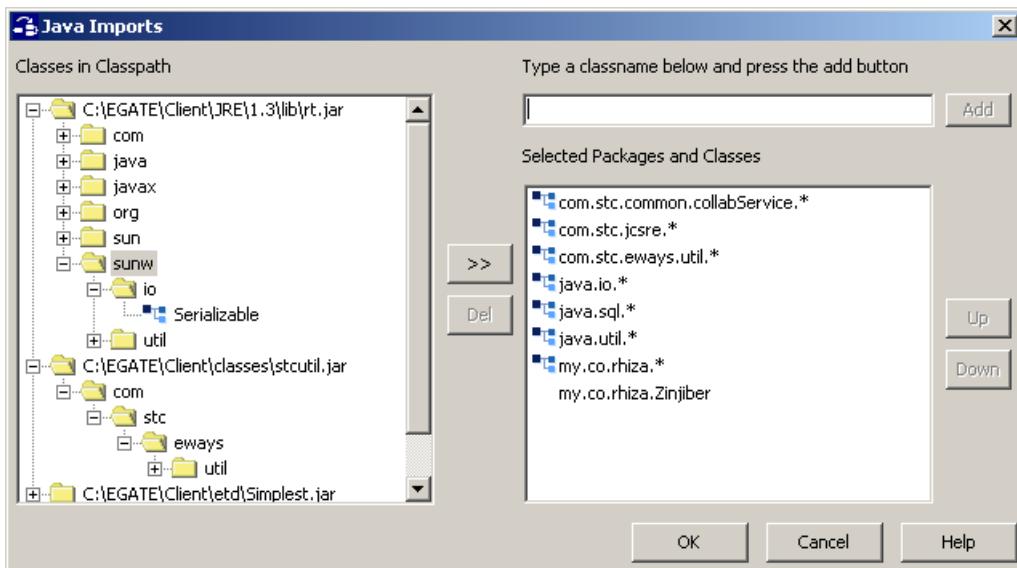
Figure 125 Java Classpaths Dialog Box



- 3 In the **Java Classpaths** dialog box, click **Add Directory** and **Add File** as needed to navigate to and add classpaths and **.jar** files. When you have added all you need, click **OK**.

- 4 As needed, specify how the system should regard the environmental classpath:
 - Click **Prepend** to use the classpath *before* files and directories specified above.
 - Click **Append** to use the classpath *after* files and directories specified above.
 - Click **Do not Use** to cause the environmental classpath to be ignored.
- 5 On the Tools menu, click **Java Imports**. See Figure 126.

Figure 126 Java Imports Dialog Box



Note: Many important classes and interfaces are supplied in the *SeeBeyond* packages *com.stc.jcsre.** and *com.stc.common.collabService.**, such as the *com.stc.jcsre.JCollaboration* and *com.stc.jcsre.Base64* classes and the *com.stc.common.collabService.JCollabController* class.

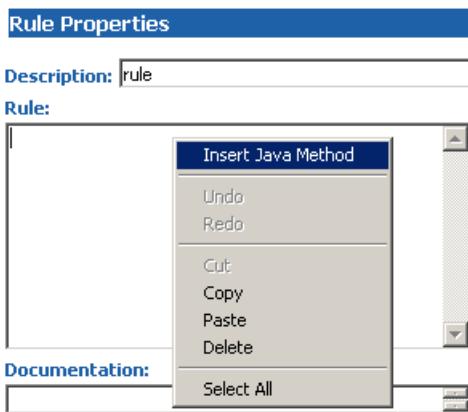
- 6 If you know the name of the class, you can enter the classname in the text box, click **Add**, and repeat as needed.
- 7 If you know the container of the class, navigate to it in the **Classes in Classpath** list and then click **>>** to move its name to the **Selected Packages and Classes** list. Repeat as needed.
- 8 When you have added all packages and classes you need, click **OK**.

To view and use methods from an imported class

Note: Exercise care when using **Insert Java Function**—for example, if you accidentally paste imported code in the middle of another line of code, the result can be extremely time-consuming to sort out. You can use **Undo** (accessed via CTRL+Z or the right-click shortcut menu) to cancel the most recent action.

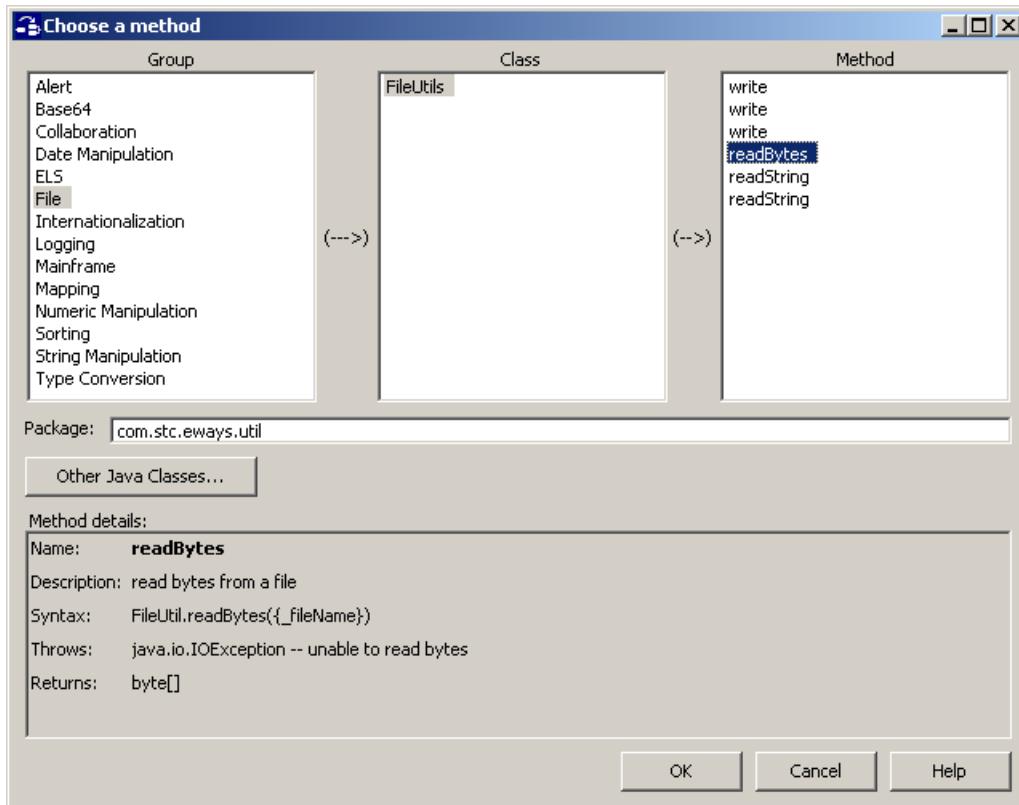
- 1 Right-click in the Rule area of the **Rule Properties** pane (or the Condition area of the **If Properties** pane, or the Initial Value area of the **Variable Properties** pane, and so on—any area except Description or Documentation). Then, on the shortcut menu, click **Insert Java Function**. See Figure 127.

Figure 127 Shortcut Menu in Rule Properties Pane



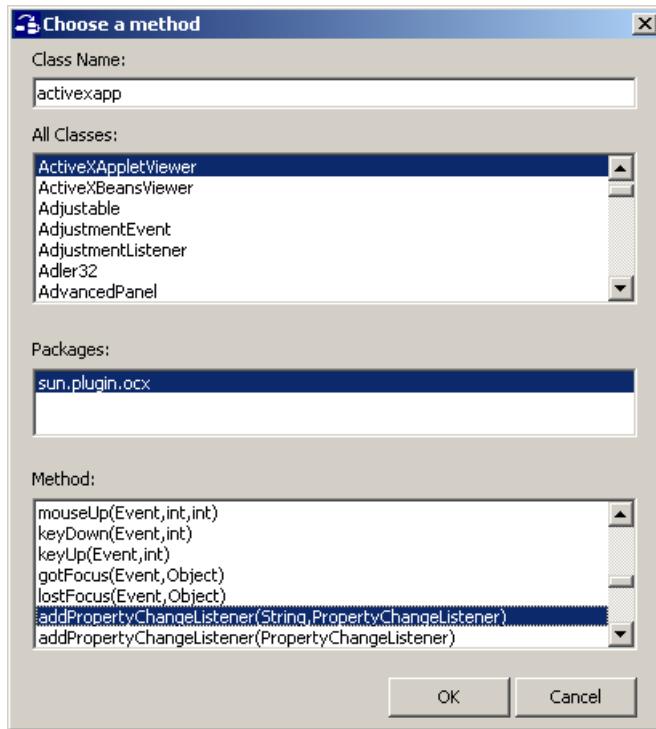
- 2 In the **Choose a method** dialog box, you can browse the categories of available classes and browse the methods signatures within each class. See Figure 128.

Figure 128 Choose a method Dialog Box



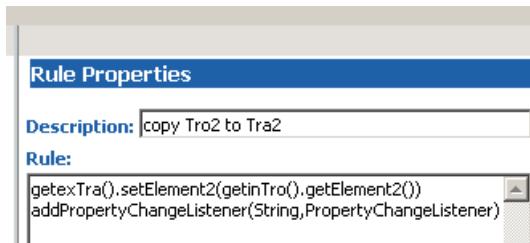
- ♦ If the method you need appears in the **Method** list, click the method name and then click **OK** to paste it into the Java Collaboration Rules Editor at the current insertion point of your cursor. Skip ahead to [Figure 130 on page 286](#).
 - ♦ Or, to browse for methods in other Java classes, click **Other Java Classes** and continue with step 3.
- 3 In the **Choose a method** dialog box, enter or scroll to the class name you want. Its Packages box displays a package name for each class as you scroll. See Figure 122.

Figure 129 Choose a method Dialog Box



- 4 Scroll to the method you want and click **OK** to paste it into the Java Collaboration Rules Editor at the current insertion point of your cursor. See Figure 130.

Figure 130 Results of Inserting a Java Method



- 5 If you make a mistake inserting a Java method, use **Undo**—accessed via CTRL+Z or the right-click shortcut menu—to cancel the most recent action. Otherwise, replace any dummy parameters—such as <parm1> and <parm2>—with appropriate real ones as needed, and continue building your Collaboration.

7.5.8 Searching and Replacing Within a Collaboration

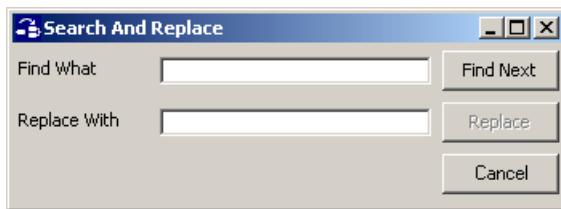
The Editor allows you to search for a particular text string through much of the editable portions of the code in your Collaboration, and also provides you with limited abilities to search-and-replace strings.

To find a text string

- 1 On the **Edit** menu, click **Search and Replace**.

The **Search And Replace** dialog box appears. See Figure 131.

Figure 131 The **Search And Replace** Dialog Box



- 2 In the **Find What** text box, enter the text string you want to find.

The string can include any ASCII character you can enter from the keyboard and any ASCII text string you can copy from the Clipboard.

Note: *The search is case-insensitive.*

- 3 Click **Find Next**.

The system searches all editable text in Description, Name, Throws, and Rule fields. (It does **not** look through text in Value, Parameter Name, or Documentation fields.) The search stops at the next point the text is found; if it is not found below the current point, the search continues at the top of the Collaboration until the original search point has been reached again.

To replace a text string

- 1 Follow steps 1 through 3; if the text string is found, continue with step 2 below.
- 2 In the **Replace With** text box, enter the text string you want to substitute.
- 3 Click **Replace**.
- 4 Continue to find and replace the text string as needed.

Caution: *Watch for case-sensitive substitutions: For example, replacing all instances of erin with Erin would make the userInitialize method unintelligible, as usErinitalizate.*

7.6 Creating Custom Java Methods

You can create your own custom Java methods for use in Collaborations.

To create a custom Java Method

- 1 In Enterprise Manager, on the **File** menu, click **Edit File**.
- 2 In the **Open** file dialog box, open the **collaboration_rules** folder, click **UserFunctions.xml**, and click **Open**.

Figure 132 The **UserFunctions.xml** Template for Custom Java Methods

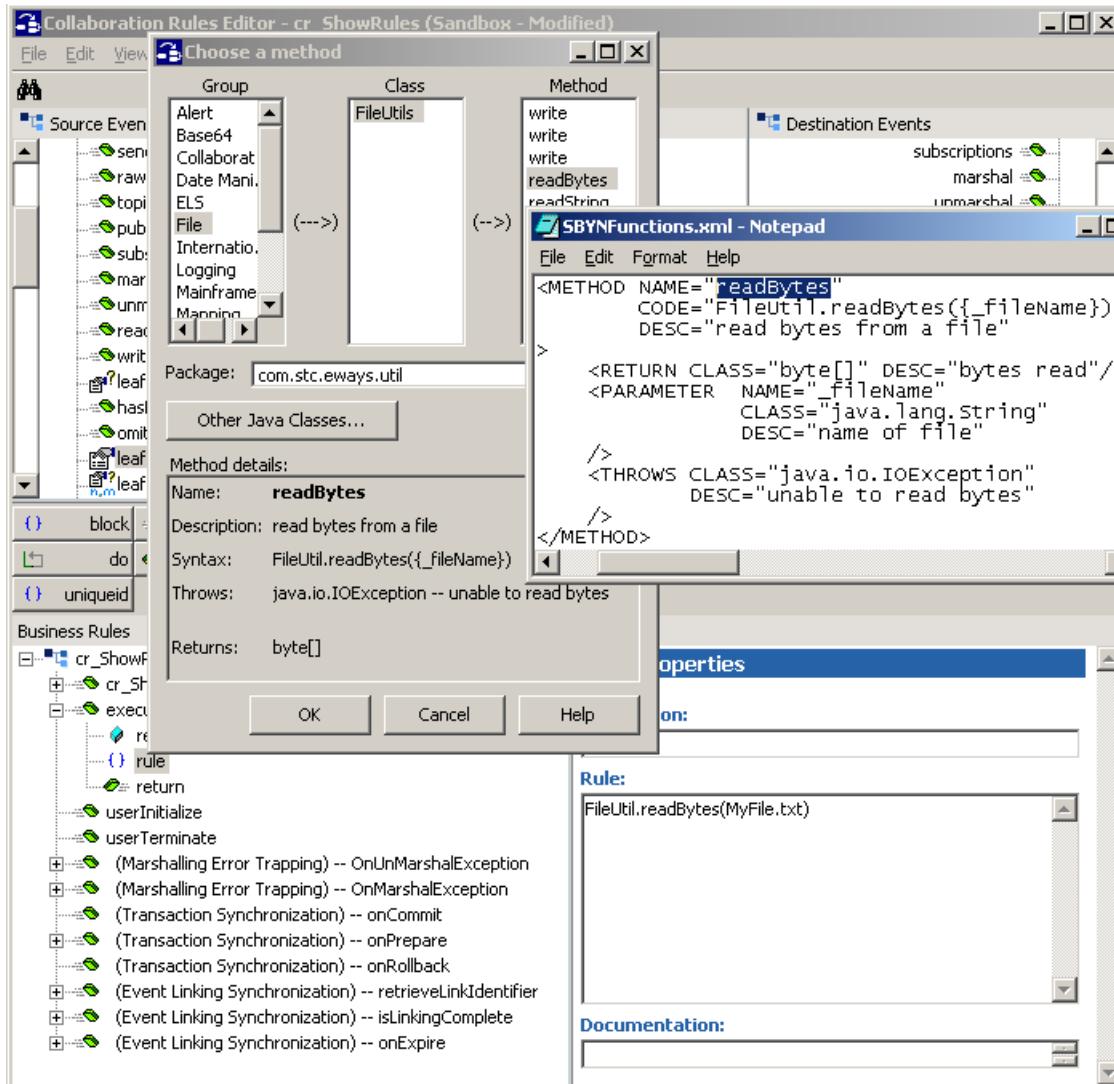
The screenshot shows a Windows Notepad window titled "UserFunctions.xml - Notepad". The content is an XML template for defining custom Java methods. It includes sections for groups, classes, methods, parameters, and throws clauses. A note at the bottom provides instructions for adding entries to the file.

```
<METHOD_BROWSER_DATA_FILE version='2.0'>
<!!-- template
<GROUP NAME="name">
    <DESC>multi-line description</DESC>
    <CLASS PACKAGE="package name"
        NAME="class name"
        DESC="one line description."
    >
        <DESC>multi-line description</DESC>
        <SEE URL="url"/>
        <METHOD NAME="method name"
            CODE="Class.method({param1}, {param2}, {param3})"
            DESC="one line description"
        >
            <DESC>multi-line description</DESC>
            <SEE URL="url"/>
            <RETURN CLASS="class name" DESC="one line description"/>
            <PARAMETER NAME="param1"
                CLASS="class name"
                DESC="one line description"
            />
            <PARAMETER NAME="param2"
                CLASS="class name"
                DESC="one line description"
            />
            <PARAMETER NAME="param3"
                CLASS="class name"
                DESC="one line description"
            />
            <THROWS CLASS="exception class name"
                DESC="one line description"
            />
        </METHOD>
    </CLASS>
</GROUP>
-->
<!!-- directions
The above template can be used to add entries to this file.
Copy and paste portions of the template below this comment
block, and fill in the information.
See SBYNFunctions.xml for examples.
-->
</METHOD_BROWSER_DATA_FILE>
```

- 3 In the text editor window, after all comments (<!-- [...] --> tags) but before the final tag (</METHOD_BROWSER_DATA_FILE [...]>), add XML tags for your methods. Use the example in the template and follow the directions provided in the file. You may also want to consult **SBYNFunctions.xml** for examples.

Figure 133 shows the relationship between **SBYNFunctions.xml** (which contains methods supplied by SeeBeyond) and the Java Collaboration Rules Editor. **UserFunctions.xml** (which contains your user-defined methods) operates in exactly the same way as **SBYNFunctions.xml**.

Figure 133 Relating UserFunctions.xml, SBYNFunctions.xml, and the GUI



- 4 When you have finished making your changes, close the text editor, and then click **Commit** to commit the updated version of **UserFunctions.xml** to your Sandbox. Save **UserFunctions.xml** in the **collaboration_rules** folder.
- 5 From this point onward, if you defined your methods correctly, their class name[s] and package name will appear under the Group you created for them (in the above example, the Group name is "File"); selecting the class in that Group will display all methods in the class.

7.7 Using the Business Rules

Each of the tools on the Business Rules toolbar creates a block of code, a single statement, or a code fragment to be added by the Collaboration Rule.

The tools on the Business Rules toolbar are:

- [block](#) on page 291
- [method](#) on page 310—includes information on the four business rules methods that are presupplied when you start the Editor, such as `executeBusinessRules()`
- [variable](#) on page 324
- [for](#) on page 303
- [if, then, else](#) on page 306
- [rule](#) on page 314
- [switch, case, default](#) on page 315
- [case](#) on page 292
- [while](#) on page 325
- [do, while](#) on page 300
- [return](#) on page 313
- [throw](#) on page 317
- [try, catch, finally](#) on page 321
- [copy](#) on page 294
- [datamap](#) on page 296
- [list lookup](#) on page 307
- [timestamp](#) on page 318
- [uniqueid](#) on page 322

block

Description

Clicking the **block** tool adds a placeholder for a block (a group of Java statements) immediately below the selected rule component. For example, when you click the **if** tool, the Editor automatically adds two blocks—one to hold a group of statements under the **then** rule component, and the other to hold a group statements under the **else** rule component.

The **block** tool is unavailable when the selected rule component is **method** or **return**. A block may contain or immediately follow another block.

Syntax

```
{  
    statement1;  
    statement2;  
}
```

Item	Description
statement<n>	Any valid Java statement.

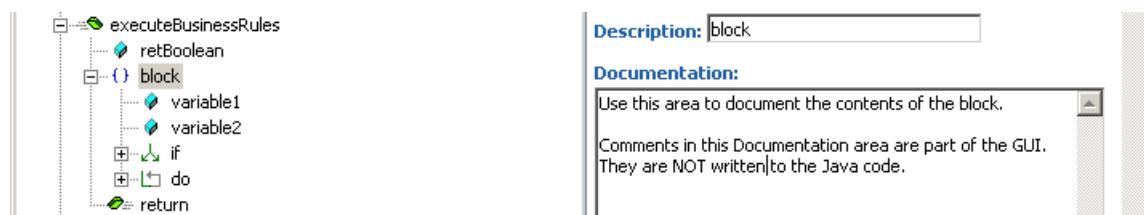
Example

```
{  
    getInput1();  
    getInput2();  
    getInput3()  
}
```

Properties

Property	Description
Description	A descriptive short name you can give to this group of statement. Note that this is not the same as a Java label—the contents of the field are not written to the Java code.
Documentation	Any documentation or comments you want to give to this group of statements. Note that this is not the same as a Java comment—the contents of the field are not written to the Java code.

Figure 134 The **block** Rule and Its Properties



case

Description

Clicking the **case** tool adds a **case** rule fragment under the selected **switch** or **case** rule fragment. For more information, see “[“switch, case, default” on page 315](#)”.

Syntax

```
switch (Expression)
    case ConstantExpression1 : Statement1;
        break;
    case ConstantExpression2 : Statement2;
        break;
    default : StatementLast
```

Properties

Property	Description
Description	Provides an opportunity for you to give a descriptive name to this case statement.
Value	The value this case requires to receive control.
Documentation	Any documentation or comments for this rule fragment.

catch

Description

Clicking the **catch** tool adds a **catch** rule fragment under the selected **try** or **catch** rule fragment. For more information, see “[try, catch, finally](#)” on page 321. When multiple **catch** rule fragments are able to handle the same exception, the exception is handled by the first applicable **catch**.

Syntax

```
try {
    firstblock;
} catch (ExceptionClass1 parm1) {
    block1;
} catch (ExceptionClass2 parm2) {
    block2;
} finally {
    lastblock;
}
```

Item	Description
ExceptionClass	An Exception object you anticipate might be thrown by the <i>firstblock</i> code.
parm	An exception parameter; must be of type Throwable or a subclass of Throwable .

Properties

Property	Description
Description	Provides an opportunity for you to give a descriptive name to this catch statement.
Exception	Specifies the exception class name and parameter name: <code>catch (<i>Exception</i> <i>e1</i>)</code>
Documentation	Any documentation or comments for this rule fragment.

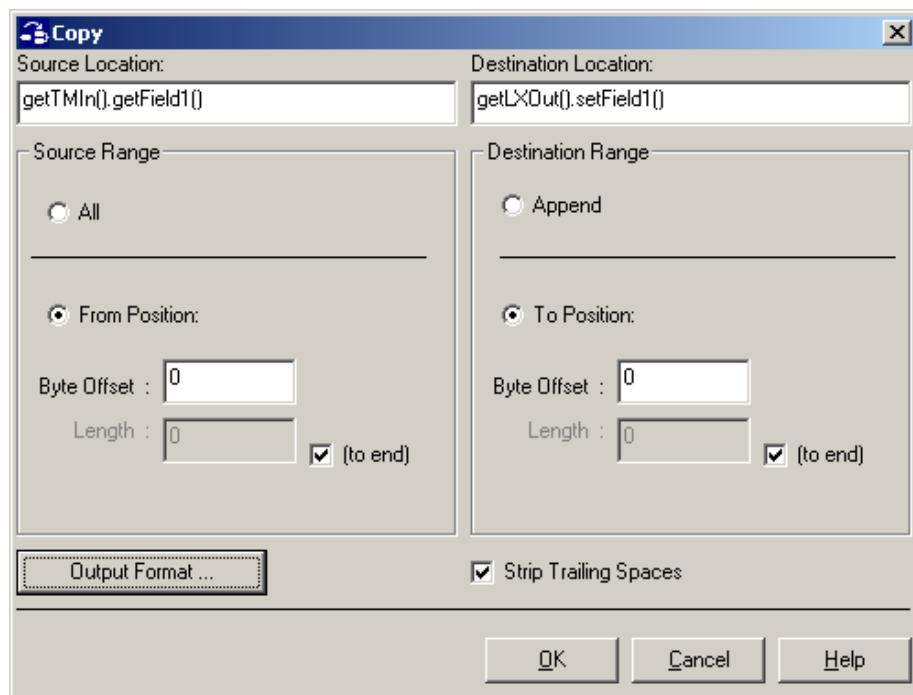
copy

Description

Clicking the **copy** tool adds code for a **copy** rule that copies byte data from a location in a node of the source Event Type instance to a location in a node of the destination Event Type instance. The **copy** button becomes available when nodes are selected in both the Source Events and Destination Events panes, but only if both nodes are of type String. If you click **copy** when one or both nodes are repeating, you will need to specify how to handle the repetitions; see “[Common Dialog Boxes for Business Rules](#)” on page 326.

You can only edit the **copy** rule by using the **Copy** dialog box. See Figure 135.

Figure 135 The **Copy** Dialog Box



Enter information in this dialog box as follows.

- 1 In the **Source** column, select either **All** (to copy all of the data in the node), or **From Position** (to copy only a portion of the data in the node).

If you select **From Position**, then also do the following.

- In the **Byte Offset** box, enter the beginning byte location of the data you are copying to the destination node. Bytes are numbered starting at zero.
- To specify a byte length, clear the **(to end)** check box, and then enter a byte length in the **Length** box. The minimum byte length is 1.

- 2 In the **Destination** pane, select either **Append to End** (to append the copied data to the end of the destination node), or **To Position** to copy the data to a specific position within the node. Copied data *replaces* any original data at that position.

If you select **To Position**, then also do the following.

- ♦ In the **Byte Offset** box, enter the beginning byte location for the copied data. Bytes are numbered starting at zero.
 - ♦ To specify a byte length, clear the **(to end)** check box, and then enter a byte length in the **Length** box. The minimum byte length is 1. For fixed-length nodes, count the number of bytes from 1. For delimited nodes, where field length is variable, leave the **(to end)** check box selected to set the length to the end of the field.
- 3 To retain blank spaces at the end of the copied bytes, clear the **Strip Trailing Spaces** check box; otherwise, keep the box checked (the default) to delete blank spaces.
 - 4 To customize the copied data's output format, click **Set Output Format** and make the appropriate settings. See "[The Output Format Dialog Box](#)" on page 327.

Syntax

```
getet_Dest().setdestNodeName(CollabUtils.copy(  
    (getet_Srce()).getsrceNodeName(),  
    int srceByteOffset, int srceByteLength,  
    int destByteOffset, int destByteLength,  
    java.lang.String formatPattern, java.lang.String emptyString,  
    boolean stripTrailingSpaces)  
) ;
```

Table 38 Parameters for Business Rule **copy**

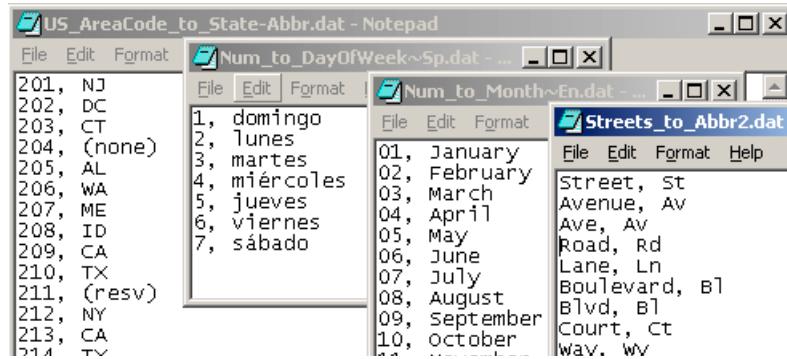
Item	Type	Description
et_Dest		The name of the destination Event Type instance.
destNodeName		The name of the node to copy the data into.
et_Srce		The name of the source Event Type instance.
srceNodeName		The name of the node to copy the data from.
srceByteOffset	int	The number of leading bytes to skip in the source: must be a nonnegative integer.
srceByteLength	int	The total number of bytes to output; a nonnegative integer, or else -1 to signify "keep copying through to the end."
destByteOffset	int	The number of leading bytes to skip in the destination: must be a nonnegative integer.
destByteLength	int	The total number of bytes to output; a nonnegative integer, or -1 to signify "keep copying through to the end."
formatPattern	java.lang.String	A code for formatting string output; see " The Output Format Dialog Box " on page 327.
emptyString	java.lang.String	The zero-length string ("").
stripTrailingSpaces	boolean	When true, specifies that blank characters are removed from the end of the string.

datamap

Description

Clicking the **datamap** tool adds code for a **dataMap** rule that attempts to match the string in the source node to a string in a specified ASCII text file and, if a match is found, output the matchstring to the destination node. Figure 136 shows some uses.

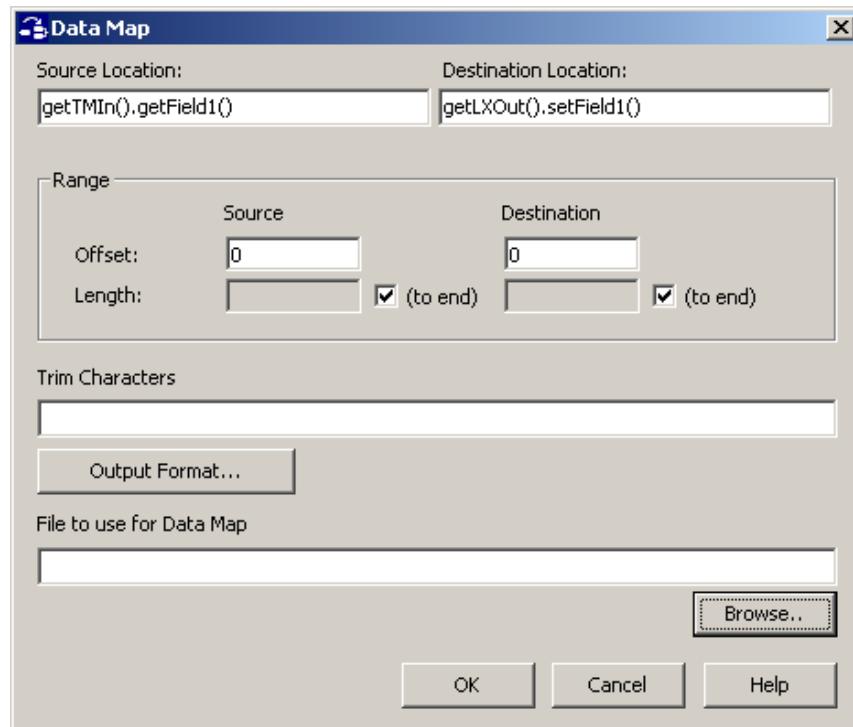
Figure 136 Sample Text Files for the **dataMap** Rule



The **datamap** button becomes available when nodes are selected in both the Source Events and Destination Events panes, but only if both nodes are of type String. If you click **datamap** when one or both nodes are repeating, you will need to specify how to handle the repetitions; see “[Common Dialog Boxes for Business Rules](#)” on page 326.

You can only edit this rule using the **Data Map** dialog box. See Figure 137.

Figure 137 The **Data Map** Dialog Box



Enter information in this dialog box as follows.

- 1 The **Source Location** and **Destination Location** boxes tell you the hierarchy and names of the two nodes you selected. There is normally no reason to edit these.
- 2 Use the **Byte Range** pane to specify whether all or part of the source node's data will be mapped to all or part of the destination node:
 - In the **Byte Offset** box, you can specify the beginning byte location of the data you are mapping from or to (or both). Bytes are numbered starting at zero.
 - To specify a byte length for source or destination, clear the **(to end)** check box, and then enter a byte length in the **Length** box. The minimum byte length is 1. For fixed-length nodes, count the number of bytes from 1. For delimited nodes, where field length is variable, leave the **(to end)** check box selected to set the length to the end of the field.
- 3 To strip one or more specified characters from the beginning and end of the source string before checking it for a match, enter the characters in the **Trim Characters** box. For example, when checking for a match on an abbreviation, you might want to trim the “.” character.
- 4 To customize the copied data's output format, click **Set Output Format** and make the appropriate settings. See [“The Output Format Dialog Box” on page 327](#).
- 5 In the **File to use for Data Map** text box, specify a map file; you can either type the path and file name or by click **Browse** and locate and select the map file you want to use.

Syntax

```
getet_Dest().setdestNodeName(CollabUtils.dataMap(  
    int srceByteOffset, int srceByteLength,  
    int destByteOffset, int destByteLength,  
    this.jCollabController.getEgateBaseDirectory(),  
    java.lang.String fileName,  
    getet_Srce().getsrceNodeName(),  
    java.lang.String formatPattern, java.lang.String trimChars)  
) ;
```

Table 39 Parameters for Business Rule **dataMap**

Item	Type	Description
et_Dest		The name of the destination Event Type instance.
destNodeName		The name of the node to copy the data into.
srceByteOffset	int	The number of leading bytes to skip in the source: must be a nonnegative integer.
srceByteLength	int	The total number of bytes to output; must be a nonnegative integer, or else -1 to signify “keep copying through to the end.”
destByteOffset	int	The number of leading bytes to skip in the destination: must be a nonnegative integer.

Table 39 Parameters for Business Rule **dataMap** (Continued)

Item	Type	Description
destByteLength	int	The total number of bytes to output; must be a nonnegative integer, or -1 to signify “keep copying through to the end.”
fileName	java.lang.String	The path and filename of the mapping file to be used for matching the string.
et_Srce		The name of the source Event Type instance.
srceNodeName		The name of the node to copy the data from.
formatPattern	java.lang.String	A code for formatting string output; see “The Output Format Dialog Box” on page 327 .
trimChars	java.lang.String	Zero or more characters to be removed from the input string before searching for a match.

Property	Description
Description	Provides an opportunity for you to give a descriptive name to this datamap rule.
Generated Code	Allows you to see the Java code that was automatically created for this rule. To modify the code, click the ... button
Documentation	Allows you to enter full documentation or comments for this rule.

default

Description

A placeholder for a **default** block is added automatically when you add a **switch** rule; for more information, see [switch, case, default](#) on page 315.

Properties

Property	Description
Description	Provides an opportunity for you to give a descriptive name to the default block of statements.
Documentation	Any documentation or comments for this statement or block.

do, while

Description

Clicking the **do** tool adds a **do, while** loop. The **do** statement declares a loop that causes a statement or block of statements to be continuously executed one or more times until a specified Boolean condition (the **while** expression) becomes false.

The only difference between **do, while** and **while** is that statements under a **do** statement are guaranteed to be executed at least once.

Syntax

```
do {
    block
} while (condition);
```

Item	Description
block	A statement or block of statements executed until <i>condition</i> is evaluated and found to be false.
condition	A Boolean expression controlling the do loop. The <i>condition</i> expression is not evaluated until after <i>block</i> has been executed once.

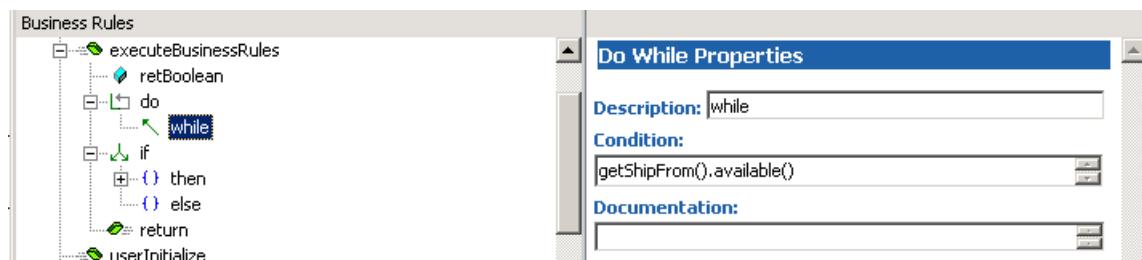
Example

```
do {
    block
} while (i<10);
```

Properties of do, while

Property	Description
Description	Provides an opportunity for you to give a descriptive name to the do, while block.
Condition	The Boolean expression that, when true, triggers the execution of the block.
Documentation	Any documentation or comments for this statement.

Figure 138 The **do, while** Rule and Its Properties



else

Description

A placeholder for an **else** block is added automatically when you add an **if** rule; for more information, see **if, then, else** on page 306.

Properties

Property	Description
Description	Provides an opportunity for you to give a descriptive name to the else block of statements.
Documentation	Any documentation or comments for this statement or block.

finally

Description

A placeholder for a **finally** rule component is added automatically when you add a **try** rule; for more information, see [try, catch, finally](#) on page 321.

Properties

Property	Description
Description	Provides an opportunity for you to give a descriptive name to the finally statement.
Documentation	Any documentation or comments for this statement or block.

for**Description**

Clicking the **for** tool adds a **for** loop that causes a block of statements to be iteratively executed zero or more times until a specified condition becomes false.

If you select a repeating node before clicking the **for** tool, the Editor asks you whether you want the **for** loop generated for the Source node or Destination node and then automatically creates a counter variable named <FieldName>Counter, and generates the code for initialization, condition, and update; see [Figure 139 on page 304](#).

Syntax

```
for (initialization; condition; update) {  
    block  
}
```

Item	Description
initialization	An assignment statement that specifies the counter's initial value.
condition	Boolean test, usually against the counter's current value; while true, the loop continues; when false, the loop ends. Important: In repeating nodes, always use a "less than" (<) test. Never use a "less-than-or-equals" (<=) test.
update	An assignment that causes the counter's value to change.
block	A statement or block of statements executed every time the condition is evaluated and found to be true.

Example 1: Inbound node is repeating, outbound node is nonrepeating

```
for (int QtyCounter=0; QtyCounter<10; QtyCounter++)  
{  
    getShipTo().setQty(getShipFm().getQty(QtyCounter))  
}
```

Example 2: Inbound node and outbound node are both repeating

```
for (int QtyCounter=0; QtyCounter<10; QtyCounter++)  
{  
    getShipTo().setQty(QtyCounter, getShipFm().getQty(QtyCounter))  
}
```

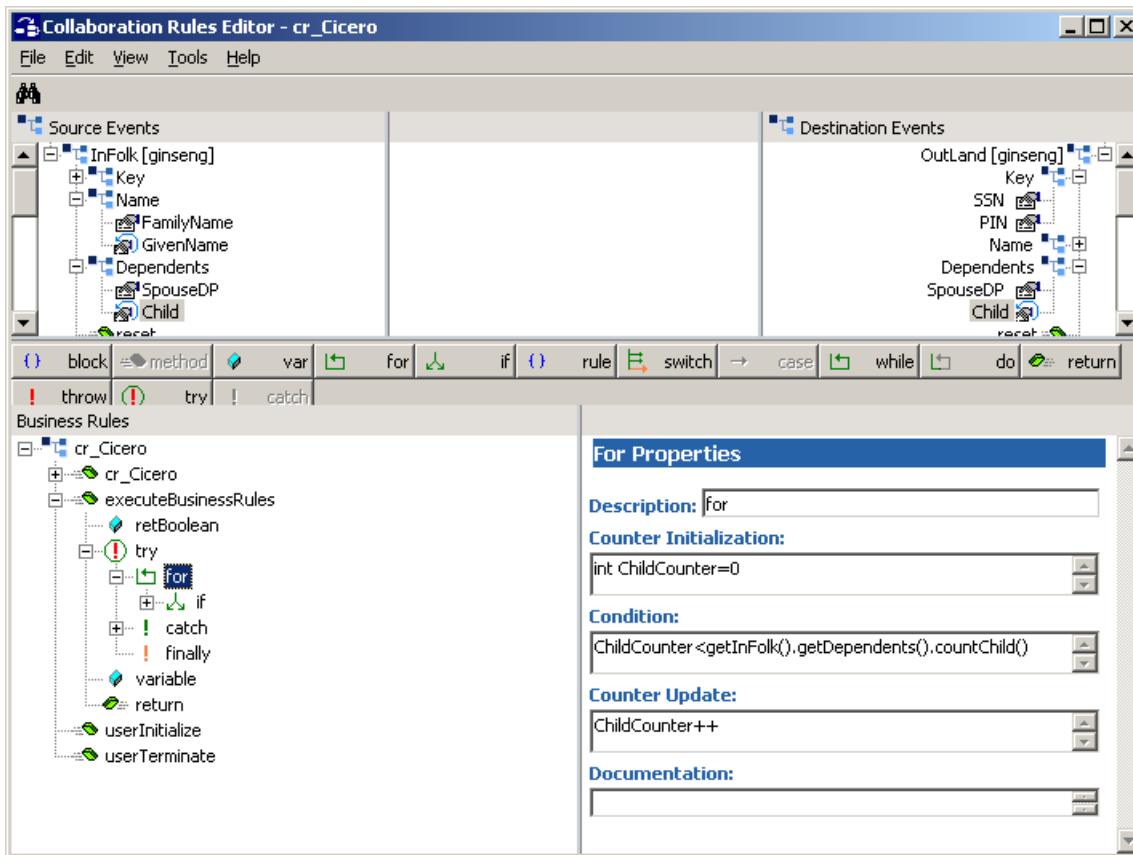
Also see [Figure 139 on page 304](#).

Properties

Property	Description
Description	Provides an opportunity for you to give a descriptive name to the for statement.

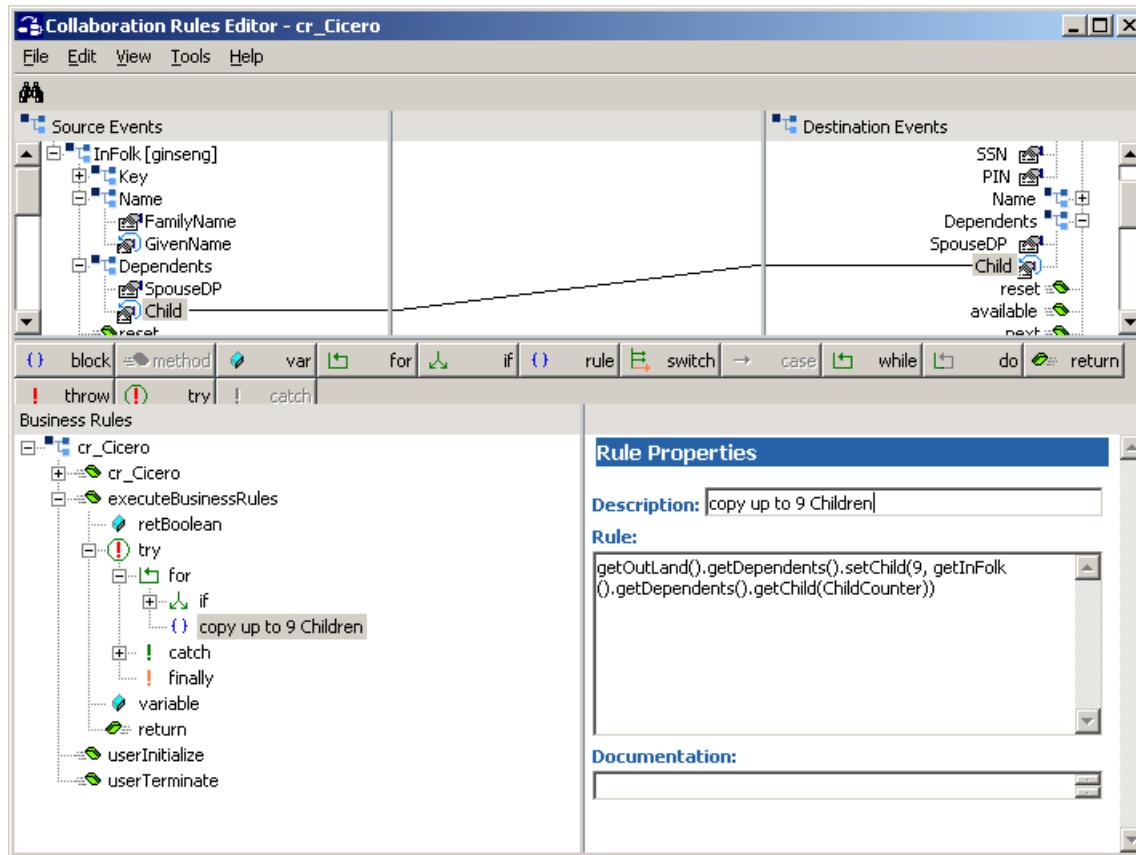
Property	Description (Continued)
Counter Initialization	Specifies an expression setting the value of the counter at the start of the loop.
Condition	Specifies an expression used to test whether the loop should continue.
Counter Update	Specifies an expression to change the value of the counter.
Documentation	Any documentation or comments for this loop.

Figure 139 Code Generated When “for” Is Used on a Repeating Node



If a drag-and-drop within a **for** loop involves a repeating node, e*Gate allows you to use the counter in the mapping by means of the **Select Repetition Instance** dialog box; see [Figure 148 on page 326](#). The results are shown in [Figure 140 on page 305](#).

Figure 140 Result of Mapping a Repeating Node in a “for” Loop



This type of automatic loop control is particularly helpful when you have nested **for** loops and therefore must track nested sets of counters.

if, then, else

Description

Clicking the **if** tool adds an **if, then, else** block. This causes conditional execution of a statement or block of statements (the **then** block) only when a specified condition is evaluated to be true; you can optionally specify another statement or block (the **else** block) to be executed only when the specified condition is false. When you add an **if** rule fragment, the Editor automatically creates empty **then** and **else** blocks beneath it.

Syntax

```
if (condition) then {
    block1;
} else {
    block2;
}
```

Item	Description
condition	A boolean test; when true, <i>block1</i> (the then block of statements) is executed; when false, <i>block2</i> (the else block of statements) is executed.

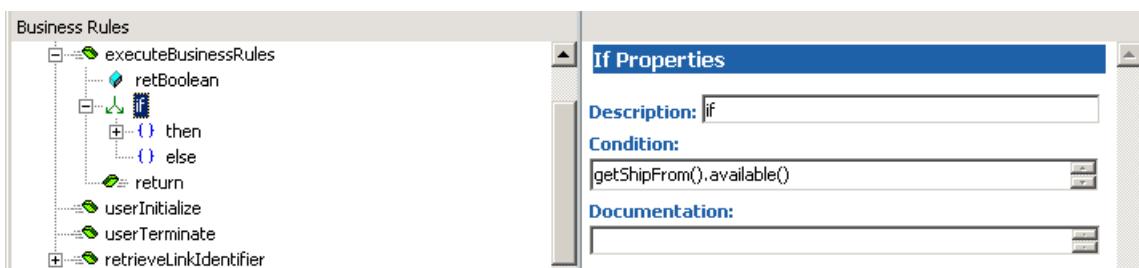
Example

```
if (i<10) then {
    block1;
} else {
    block2;
```

Properties

Property	Description
Description	Provides an opportunity for you to give a descriptive name to this if statement.
Condition	The Boolean expression that, when true, triggers the execution of the then block.
Documentation	Any documentation or comments for this block.

Figure 141 The **if, then, else** Rule and Its Properties



list lookup

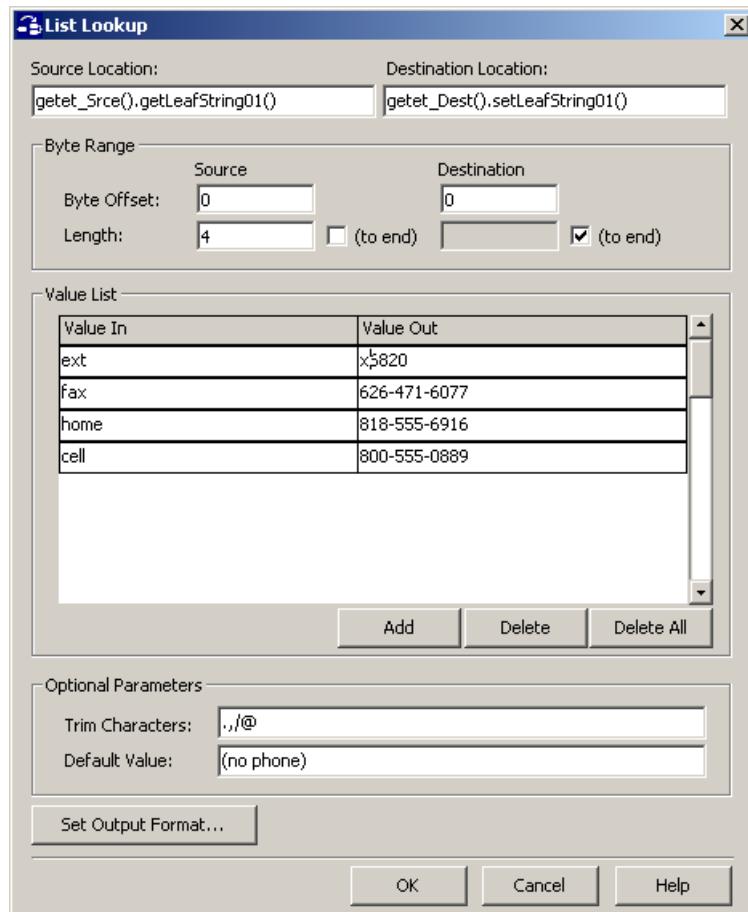
Description

Clicking the **list lookup** tool adds code for a **lookup** rule that allows you to replace a string in a source node with a different but corresponding string in the destination node; for example, you could substitute a department code with a full department name, or you could use it to hold a few lines of commonly accessed information.

The **list lookup** button becomes available when nodes are selected in both the Source Events and Destination Events panes, but only if both nodes are of type String. If you click **list lookup** when one or both nodes are repeating, you will need to specify how to handle the repetitions; see “[Common Dialog Boxes for Business Rules](#)” on page 326.

You can only edit the **lookup** rule using the **List Lookup** dialog box. See Figure 137.

Figure 142 The **List Lookup** Dialog Box



The **lookup** rule searches its list for the string contained in the input Event Type instance. If it finds a match, then it inserts the corresponding string in the output node. In the sample list shown above, if the input node contains the string “cell”, then the string “818-555-0889” is placed in the output node.

The default value—in this case, “(no phone)”—is written when the **lookup** rule finds no match for the input string. If no default value is specified, then the output node will receive an empty string *if it is delimited*; if it is fixed-length, it will receive no data.

Use the **lookup** rule when you only need to match an input Event string to a few mappings (no more than six is recommended); otherwise, use the **dataMap** rule. See [“datamap” on page 296](#).

Syntax

```
getet_Dest().setdestNodeName(CollabUtils.lookup(  
    int srceByteOffset, int srceByteLength,  
    int destByteOffset, int destByteLength,  
    java.lang.String concatStringPairs,  
    java.lang.String defaultOutString,  
    getet_Srce().getsrceNodeName(),  
    java.lang.String formatPattern, java.lang.String trimChars)  
);
```

Table 40 Parameters for Business Rule **lookup**

Item	Type	Description
et_Dest		The name of the destination Event Type instance.
destNodeName		The name of the node to copy the data into.
srceByteOffset	int	The number of leading bytes to skip in the source: must be a nonnegative integer.
srceByteLength	int	The total number of bytes to output; must be a nonnegative integer, or else -1 to signify “keep copying through to the end.”
destByteOffset	int	The number of leading bytes to skip in the destination: must be a nonnegative integer.
destByteLength	int	The total number of bytes to output; must be a nonnegative integer, or -1 to signify “keep copying through to the end.”
concatStringPairs	java.lang.String	A single string containing the values in the lookup table, using the following format: “inString1,outString1 inString2,outString2 inString3,outString3 ... inStringN,outStringN ”
defaultOutString	java.lang.String	The string to be output if there is no match. If not specified, then the empty string (“”) is written to a delimited output node, and no data is written to a fixed-length node.
et_Srce		The name of the source Event Type instance.
srceNodeName		The name of the node to copy the data from.
formatPattern	java.lang.String	A code for formatting string output; see “The Output Format Dialog Box” on page 327 .
trimChars	java.lang.String	Zero or more characters to be removed from the input string before searching for a match.

Property	Description
Description	Provides an opportunity for you to give a descriptive name to this lookup rule.
Generated Code	Allows you to see the Java code that was automatically created for this rule. To modify the code, click the ... button
Documentation	Allows you to enter full documentation or comments for this rule.

method

Description

Clicking the **method** tool allows you to add a user-defined method to the four methods supplied by the Editor at the outset (see “[Methods Presupplied When You Start the Editor” on page 312\).](#)

Syntax

```
access return methodName(parm1 parm2[]) throws Exceptions
```

Property	Description
access	Specifies the visibility of the method, such as public , protected , or private .
return	Specifies the return type for the method, such as void , boolean (or other primitive datatype), String (or other system-defined object type), or a user-defined object type. “[]” indicates that the value returned is an array.
methodName	Specifies the name of the method.
parm1 parm2	Parameters the method expects to receive. “[]” indicates that the parameter is an array.
Exceptions	Exceptions thrown by the method.

Example

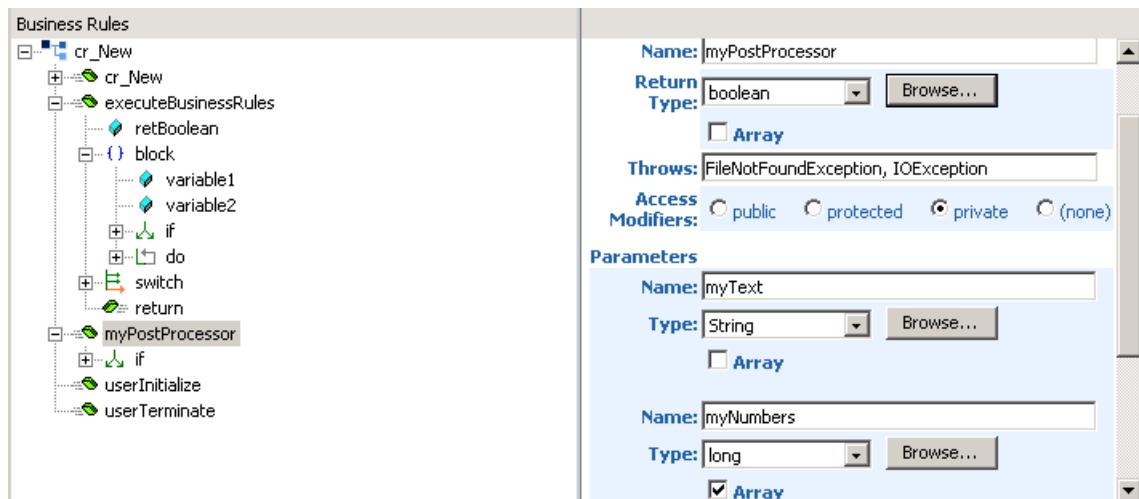
```
public byte[] translate(byte[] inputEvent)
    throws CollabConnException, CollabDataException
```

Properties

Property	Description
Description	Provides an opportunity for you to give a descriptive local name to this method.
Name	Specifies the name of the method: <code>public byte[] name(byte[] inputEvent) throws CollabConnException, CollabDataException</code>
Return Type	Specifies the return type of the method: <code>public byte[] name(byte[] inputEvent) throws CollabConnException, CollabDataException</code>

Property	Description (Continued)
Array	When checked, indicates the return type is an array: <pre>public byte[] name(byte[] inputEvent) throws CollabConnException, CollabDataException</pre>
Throws	Specifies the list of exceptions thrown: <pre>public byte[] name(byte[] inputEvent) throws CollabConnException, CollabDataException</pre>
Access Modifiers	Specifies the visibility of the method: <pre>public byte[] name(byte[] inputEvent) throws CollabConnException, CollabDataException</pre>
Parameter Name	Parameters for the method: <pre>public byte[] name(byte[] inputEvent) throws CollabConnException, CollabDataException</pre>
Parameter Type	Type of parameter used by the method: <pre>public byte[] name(byte[] inputEvent) throws CollabConnException, CollabDataException</pre>
Documentation	Any documentation or comments for this method.

Figure 143 A method Rule and Its Properties



Methods Presupplied When You Start the Editor

When you start the Editor, the following four methods are already supplied:

- a method of the same name as the Collaboration Rule itself
- **executeBusinessRules()**—A placeholder for all user-written code that will be executed after initialization. If your Collaboration uses Event Linking and Sequencing (ELS), this method is invoked for each group of Events passed to it by ELS.
- **userInitialize()**—A placeholder for code to be executed when the component starts.
- **userTerminate()**—A placeholder for code to be executed when the component shuts down.

Although these four methods have the same syntax and properties as a user-defined method, the following restrictions apply:

- Do not change the **Name** property of any of these four methods.
- For the method of the same name as the Collaboration Rule:
 - ♦ Do not delete the rule that invokes the **super()** method.
- For the **executeBusinessRules()** method:
 - ♦ Keep the **Return Type** property set to **boolean**.
 - ♦ Do not alter the **retBoolean** variable or the **return**.
 - ♦ An internal **try...catch** already exists in **executeBusinessRules()**, so that exceptions do not generate compilation errors; however, in **userInitialize()** and **userTerminate()**, you must add your own **try...catch** blocks.

return

Description

Clicking the **return** tool adds a **return** statement to the code; when this is executed, control is passed back to the invoker of the method or constructor.

Syntax

`return Expression`

Note: For methods whose return was declared as **void**, the **return** statement takes no Expression.

Item	Description
Expression	The value to be returned to the invoker.

Properties

Property	Description
Description	Provides an opportunity for you to give a descriptive name to this return statement.
Result	The value returned.
Documentation	Any documentation or comments for this statement or block.

rule

Description

Clicking the **rule** tool adds an unspecified expression statement to your code. This allows you to use the methods associated with the ETD of the Event instance by dragging the method from the Source Events pane into the Rule Properties pane.

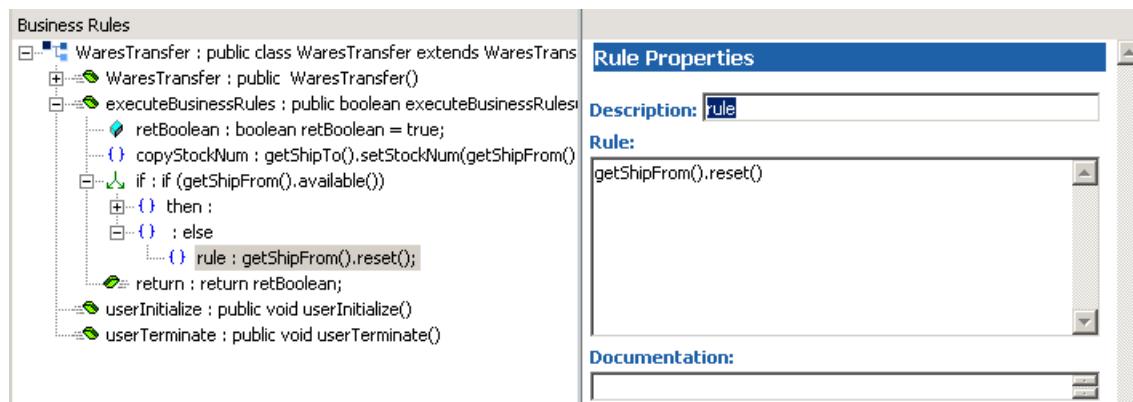
Syntax

Open. The text you enter in the **Rules** area of the Rule Properties pane is simply Java code that you add on your own, without any validation from the GUI. For example, if you want to add code comments or Javadoc comments, you must use the **rule** tool.

Properties

Property	Description
Description	Provides an opportunity for you to give a descriptive local name to this rule.
Rule	Any combination of the following: <ul style="list-style-type: none"> ▪ The code generated automatically by the Editor when you use drag-and-drop (or Find and Map) to map one field or node to another. ▪ The method or methods you drag from the Source Events pane. ▪ Code or comments that you type or paste.
Documentation	Any documentation or comments for this code that you want saved with the GUI instead of being written into the code.

Figure 144 The **rule** Rule and Its Properties



switch, case, default

Description

Clicking the **switch** tool adds a **switch, case, default** block. This sets up a conditional transfer of control to exactly one statement depending on the value of an expression. When you add a **switch** statement, the Editor automatically creates empty **case** and **default** blocks beneath it.

Syntax

```
switch (Expression)
    case ConstantExpression1 : Statement1;
    break;
    case ConstantExpression2 : Statement2;
    break;
    default : StatementLast
```

Item	Description
<i>Expression</i>	A variable or expression of type char, byte, short, or int.
<i>ConstantExpression<n></i>	A variable or expression of the same type as <i>Expression</i> .
<i>Statement<n></i>	The statement to which control will pass if the <i>ConstantExpression</i> of this case statement matches the <i>Expression</i> value.
<i>StatementLast</i>	The statement to which control will pass if no <i>ConstantExpression</i> of any case statement matches the <i>Expression</i> value.

Example

```
switch (k) {
    case 1: System.out.println("this prints when k=1");
    break;
    case 2: System.out.println("this prints when k=2");
    break; // each break statement is added by the Editor
    case 3: System.out.println("this prints when k=3");
    break;
    default: System.out.println("this prints otherwise");}
```

Properties

Property	Description
Description	Provides an opportunity for you to give a descriptive name to the switch statement.
Expression to be evaluated	The variable or expression that might match one of the case statements.
Documentation	Any documentation or comments for this statement or block.

then

Description

A **then** block and an **else** block are both added automatically when you add an **if** rule; for more information, see [if, then, else](#) on page 306.

Properties

Property	Description
Description	Provides an opportunity for you to give a descriptive name to the then block of statements.
Documentation	Any documentation or comments for this statement or block.

throw

Description

Clicking the **throw** tool adds a **throw** statement to the code. When a **throw** statement is executed, an exception is generated and transfer passes to a **try** statement (if possible). The type of exception thrown depends on the evaluation of its expression.

Syntax

```
throw Expression;
```

Item	Description
Expression	A variable or value; must be of type Throwable or a subclass of Throwable .

Properties

Property	Description
Description	Provides an opportunity for you to give a descriptive name to this throw statement.
Exception	The expression thrown by the statement.
Documentation	Any documentation or comments for this statement.

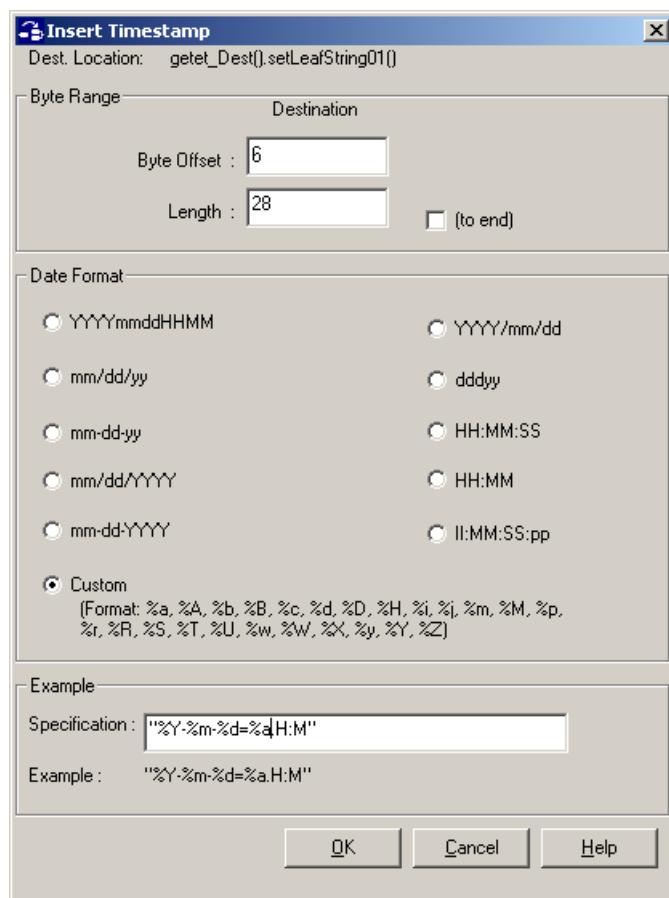
timestamp

Description

Clicking the **timestamp** tool adds code for a **timeStamp** rule that places the current date and time (of the Participating Host) into a location in a node of the destination Event Type instance. The **timestamp** button is available only when a node of type String is selected in the Destination Event panes. If you click **timestamp** when the node is a repeating node, you will need to specify how to handle the repetitions; see “[Common Dialog Boxes for Business Rules](#)” on page 326.

You can only edit the **timeStamp** rule by using the **Insert Timestamp** dialog box. See Figure 145.

Figure 145 The **Insert Timestamp** Dialog Box



Enter information in this dialog box as follows.

- 1 In the **Byte Offset** box, enter the beginning byte location of the data you are copying to the destination node. Bytes are numbered starting at zero.
- 2 To specify a byte length, clear the **(to end)** check box, and then enter a byte length in the **Length** box. The minimum byte length is 1.
- 3 Click one of the presupplied date formats and look at the **Example** line see how it looks; or, if none of the formats is right, click **Custom** and create your own format.

Syntax

```
getet_Dest().setdestNodeName(CollabUtils.timeStamp(
    int destByteOffset, int destByteLength,
    java.lang.String timestampPattern)
);
```

For complete details on syntax, see “[timeStamp\(\)](#)” on page 560.

Table 41 Parameters for Business Rule **timeStamp**

Item	Type	Description
et_Dest		The name of the destination Event Type instance.
destNodeName		The name of the node to receive the timestamp data.
destByteOffset	int	The number of leading bytes to skip in the destination: must be a nonnegative integer.
destByteLength	int	The total number of bytes to output; a nonnegative integer, or -1 to signify “keep copying through to the end.”
timestampPattern	java.lang.String	A code for formatting date/time output; see Table 42.

Table 42 Date and Time Format Codes for the **timeStamp** Rule

Time Division	Code	Description	Value, Range, or Example
Years	%Y	year, including century; four digits	Examples:
	%y	year in current century; two digits	Range: 00 through 99
Months	%m	month number; two digits	Range: 01 through 12
	%b	month, using site-defined abbreviations	Example: Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov, Dec
	%B	month, using site-defined spellings	Example: Januar Februar, März, April, Mai, Juni, Juli, usw
Weeks	%U	week of year, considering Sunday the first day of the week; two digits	Range: 01 through 52
	%W	week of year, considering Sunday the first day of the week; two digits	Range: 01 through 52
Days	%w	day of week. Sunday is day 0.	Range: 0 through 6
	%a	day of week, using site-defined abbreviations	Example: dom, lun, mar, mie, jue, vie, sab
	%A	day of week, using site-defined spellings	Example: Sunday, Monday, Tuesday, etc.
	%d	day of month; two digits	Range: 01 through 31
	%e	day of month; single digit is left-padded with a space	Range: 1 through 31
	%j	day of year (Julian day); three digits	Range: 001 through 366

Table 42 Date and Time Format Codes for the **timeStamp** Rule (Continued)

Time Division	Code	Description	Value, Range, or Example
Hours	%H	hour in 24-hour clock; two digits	Range: 00 through 23
	%I	hour in 12-hour clock; two digits	Range: 01 through 12
Minutes	%M	minutes; two digits	Range: 00 through 59
Seconds	%S	seconds; two digits	Range: 00 through 59
meridian	%p	morning / afternoon; two characters	Value: Either AM or PM
time zone	%Z	abbreviation for current time zone	Examples: EST, PDT
composites	%D	date, as %m/%d/%y	Examples: 12/31/99; 03/01/02
	%R	time, as %H:%M	Example: 14:15; 09:05
	%T	time, as %H:%M:%S	Example: 14:15:03; 09:04:55
	%r	time, as %I:%M:%S%p	Example: 02:15:03 PM; 09:04:55 AM
	%x	site-defined standard date format	Examples: 03/01/02; 0131Thu1659

Property	Description
Description	Provides an opportunity for you to give a descriptive name to this timeStamp rule.
Generated Code	Allows you to see the Java code that was automatically created for this rule. To modify the code, click the . . . button
Documentation	Allows you to enter full documentation or comments for this rule.

try, catch, finally

Description

Clicking the **try** tool adds a **try**, **finally** block. This allows you to protect code by anticipating and handling possible exceptions it might throw. Between the **try** rule fragment and the **finally** statement, you can add one or more **catch** rule fragments to deal gracefully with the exceptions you anticipate.

Syntax

```
try {  
    firstblock;  
} catch (ExceptionClass1 parm1) {  
    block1;  
} catch (ExceptionClass2 parm2) {  
    block2;  
} finally {  
    lastblock;  
}
```

Item	Description
firstblock	The statement or block that you want to protect against throwing exceptions.
ExceptionClass<n>	An Exception object you anticipate might be thrown by the <i>firstblock</i> code.
parm<n>	An exception parameter; must be of type Throwable or a subclass of Throwable .
block<n>	Statements or blocks that are executed only if a particular exception is thrown
lastblock	A statement or block of statements that is guaranteed to execute, whether or not an exception is thrown.

Example

```
try {  
    //code that might throw exceptions;  
} catch (FileNotFoundException e1) {  
    //code for handling missing-file exceptions;  
} catch (InterruptedException e2) {  
    //code for handling Interruption exceptions;  
} finally {  
    next();  
}
```

Properties

Property	Description
Description	Provides an opportunity for you to give a descriptive name to this try statement.
Documentation	Any documentation or comments for this block.

uniqueid

Description

Clicking the **uniqueid** tool adds code for a **uniqueId** rule that places a unique set of digits into a location in a node of the destination Event Type instance. The **uniqueid** button is available only when a node of type String is selected in the Destination Event panes. If you click **uniqueid** when the node is a repeating node, you will need to specify how to handle the repetitions; see “[Common Dialog Boxes for Business Rules](#)” **on page 326**.

You can only edit the **uniqueid** rule by using the **Insert Unique ID** dialog box. See Figure 146.

Figure 146 The **Insert Unique ID** Dialog Box



Enter information in this dialog box as follows.

- 1 In the **Byte Offset** box, enter the beginning byte location of the data you are copying to the destination node. Bytes are numbered starting at zero.
- 2 To specify a byte length, clear the **(to end)** check box, and then enter a byte length in the **Length** box. The minimum byte length is 1.

Syntax

```
getet_Dest().setdestNodeName(CollabUtils.uniqueId(  
    int destByteOffset, int destByteLength)  
)
```

Table 43 Parameters for Business Rule **uniqueId**

Item	Type	Description
et_Dest		The name of the destination Event Type instance.
destNodeName		The name of the node to receive the unique ID data.
destByteOffset	int	The number of leading bytes to skip in the destination: must be a nonnegative integer.
destByteLength	int	The total number of bytes to output; a nonnegative integer, or -1 to signify “keep copying through to the end.”

Property	Description
Description	Provides an opportunity for you to give a descriptive name to this uniqueId rule.
Generated Code	Allows you to see the Java code that was automatically created for this rule. To modify the code, click the ... button
Documentation	Allows you to enter full documentation or comments for this rule.

variable

Description

Adds a user-defined variable.

Syntax

access-modifier type variablename = initvalue

Item	Description
access-modifier	The scope of the variable: public, private, protected, or none.
type	The datatype of the variable.
type[]	The datatype of an array variable.
variablename	A name you enter for the variable.
initvalue	The initial value for the variable.

Example

```
public String[] wkDayEng = { "Mon", "Tue", "Wed", "Thu", "Fri" }
```

Properties

Property	Description
Description	Provides an opportunity for you to give a descriptive label for this variable. Default: variable
Name	Allows you to rename the variable. Default: variable
Type	Allows you to change the datatype of the variable. Default: String
Array	If checked, declares the variable as an array. Default: (not checked)
Initial Value	Allows you to set the value of the variable before its first use. Default: (not set)
Access modifiers	Allows you to declare the variable as public, private, protected, or (none). Default: (not set)
Documentation	Any documentation or comments for this variable.

while

Description

A **while** statement checks the Boolean value of a specified condition and, whenever the condition is found to be true, executes a statement or block of statements. The only difference between **while** loop and **do, while** loop (see [do, while](#) on page 300) is that statements under a **while** loop might never be executed, but statements under a **do** loop are guaranteed to be executed at least once.

Syntax

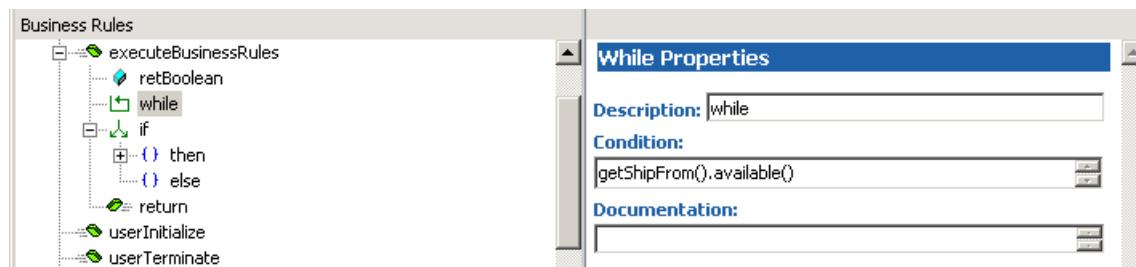
```
while (condition) {  
    block;  
}
```

Item	Description
condition	A Boolean expression controlling the while loop. The <i>condition</i> expression is evaluated before <i>block</i> is executed; thus nothing is executed if <i>condition</i> is false.
block	A statement or block of statements executed each time <i>condition</i> is evaluated and found to be true.

Properties

Property	Description
Description	Provides an opportunity for you to give a descriptive name to this while statement.
Condition	The Boolean expression that, when true, triggers the execution of the block.
Documentation	Any documentation or comments for this statement.

Figure 147 The **while** Rule and Its Properties



7.7.1 Common Dialog Boxes for Business Rules

Several of the business rules require or permit special processing in certain cases. For example:

- When a repeating node is involved in a drag-and-drop or **Find and Map** operation, or in any child rule in a **do**, **for**, or **while** loop, or in a **copy**, **datamap**, **list lookup**, **timestamp**, or **uniqueid** rule, the system requires you to supply the necessary information before continuing with the operation.

See “[Dealing with Repeating Nodes](#)” on page 326.

- e*Gate offers a wide variety of text formatting options when you are outputting a free-form data to a node using the **copy**, **datamap**, or **list lookup** rules. You can specify whether the data should be output as string text, as numeric digits, or in floating-point form; signed or unsigned; right-aligned or left-aligned; and so forth.

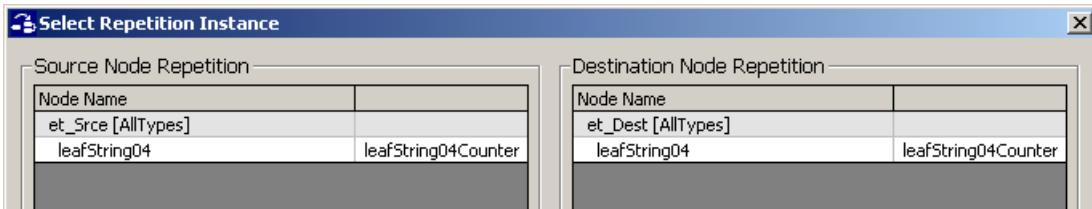
See “[Formatting Output](#)” on page 327.

Dealing with Repeating Nodes

The **Select Repetition Instance** dialog box prompts you to specify how the system should deal with a repeating node.

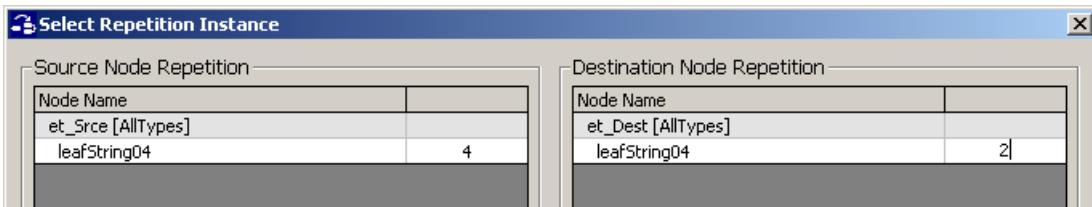
Typically, when you are copying data from one repeating node to another, the operation is performed under the control of a **do**, **for**, or **while** loop, and you will have already defined a variable you are using as a counter. In this case, you can simply drag the variable into the appropriate text box. Figure 148 shows a simple example; more complicated cases can occur when you have loops nested within loops.

Figure 148 The **Select Repetition Instance** Dialog Box With a Counter



When you want to copy from or to a particular instance of a repeating node, simply type the appropriate index value or values in the text box. See Figure 149.

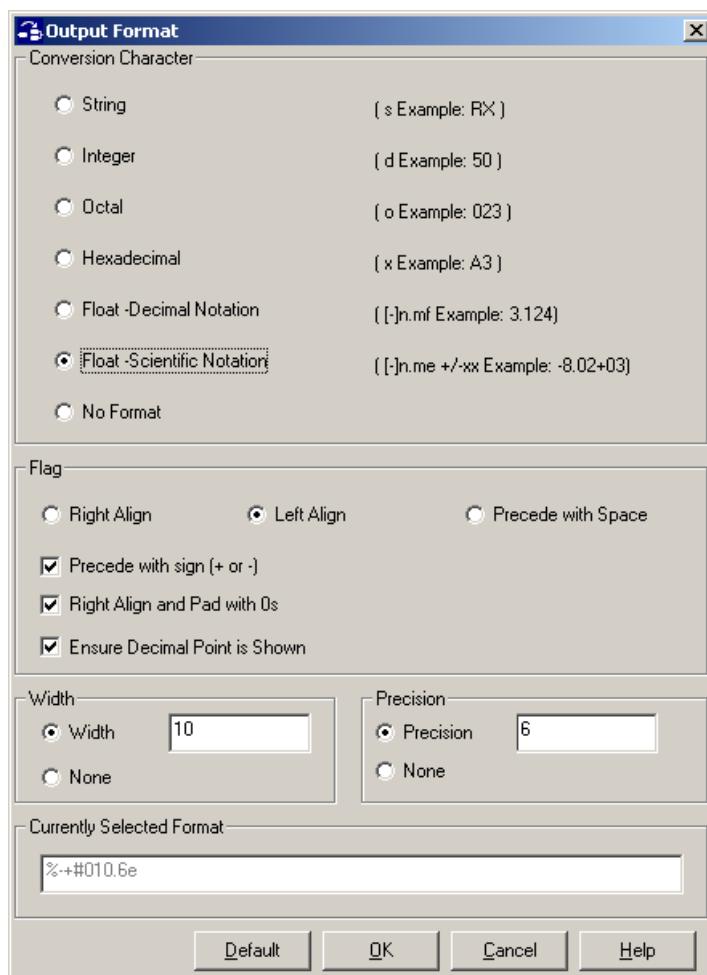
Figure 149 The **Select Repetition Instance** Dialog Box Without a Counter



Formatting Output

You can use the **Output Format** dialog box to control many aspects of the output from such rules as **copy**, **datamap**, or **list lookup**. The option buttons and check boxes allow you to specify and customize output options, such as: string, integer, or floating-point representation; alignment; sign; padding; width (total number of output characters); and precision. The codes corresponding to the output options are echoed in the Current Selected Format box. For a complete explanation of these codes and their effects, see ["Formatting of Output Text" on page 662](#).

Figure 150 The **Output Format** Dialog Box



7.7.2 Methods for Elements and Fields of ETDs

For an element or field named **MyNode**, the following methods are supplied:

- [“get_MyNode_0” on page 330](#)—this method is supplied for all elements and fields.
- [“set_MyNode_0” on page 332](#)—this method is supplied for all elements and fields.
- [“has_MyNode_0” on page 331](#)—this method is supplied for any element or field whose **minOccurs** property has the value **0**.
- [“count_MyNode_0” on page 329](#)—this method is supplied for any element or field whose **maxOccurs** property is **unbounded** or has a value greater than its **minOccurs** value.

count_MyNode_()

Syntax

```
int countMyNode()
```

Description

Counts the number of repetitions of the *MyNode* repeating node.

Parameters

None.

Return Type

int

Comments

This method is defined only for elements or fields whose **maxOccurs** property is unbounded, or has a value greater than its **minOccurs** value.

get_MyNode_()

Syntax

```
MyNode javaType getMyNode( )
MyNode getMyNode(int i)
MyNode[ ] getMyNode( )
```

Description

Retrieves current data in the *MyNode* node.

Parameters

For nonrepeating nodes: None.

For repeating nodes: Either none, or a counter variable.

Return Type

For nonrepeating nodes: Same as the **javaType** value of *MyNode*—often **java.lang.String**.

For repeating nodes: A single *MyNode* object, or an array of *MyNode* objects.

Comments

This method is defined for all elements and fields. For repeating nodes, the method can be overloaded with several different signatures.

has_MyNode_()

Syntax

boolean hasMyNode()

Description

Inquires whether *MyNode* exists.

Parameters

None.

Return Type

boolean

Comments

This method is defined only for elements and fields whose **minOccurs** property has the value 0.

set_MyNode_()

Syntax

```
void setMyNode(MyNode.javaType data)
void setMyNode(MyNode[] data)
void setMyNode(int i, MyNode.javaType data)
```

Description

Writes data to the *MyNode* node.

Parameters

For nonrepeating nodes: A single value, of the same type as the **javaType** property of *MyNode*—often **java.lang.String**.

For repeating nodes, two signatures are supplied:

- An array of values of the same type as the **javaType** property of *MyNode*.
- A pair of arguments, consisting of a counter variable and a single value of the same type as the **javaType** property of *MyNode*.

Return Type

void

Comments

This method is defined for all elements and fields. For repeating nodes, the method can be overloaded with several different signatures, as noted above.

7.7.3 Methods for Standard Java-enabled ETDs

Standard ETD methods supplied with the Java Collaboration Rules Editor

The methods supplied with all standard Java-enabled ETDs are as follows:

- [available\(\)](#) on page 334
- [marshal\(\)](#) on page 335
- [next\(\)](#) on page 336
- [publications\(\)](#) on page 337
- [rawInput\(\)](#) on page 338
- [readProperty\(\)](#) on page 339
- [receive\(\)](#) on page 341
- [reset\(\)](#) on page 342
- [send\(\)](#) on page 343
- [subscriptions\(\)](#) on page 344
- [topic\(\)](#) on page 345
- [unmarshal\(\)](#) on page 346
- [writeProperty\(\)](#) on page 347

For the methods `executeBusinessRules()`, `userInitialize()`, and `userTerminate()`, see “[Methods Presupplied When You Start the Editor](#)” on page 312.

available()

Syntax

```
boolean available()
```

Description

Checks to see whether this Event Type instance contains any data. Use this method when there are multiple triggering subscriptions and the Collaboration needs to know which Event Type triggered it.

Parameters

None.

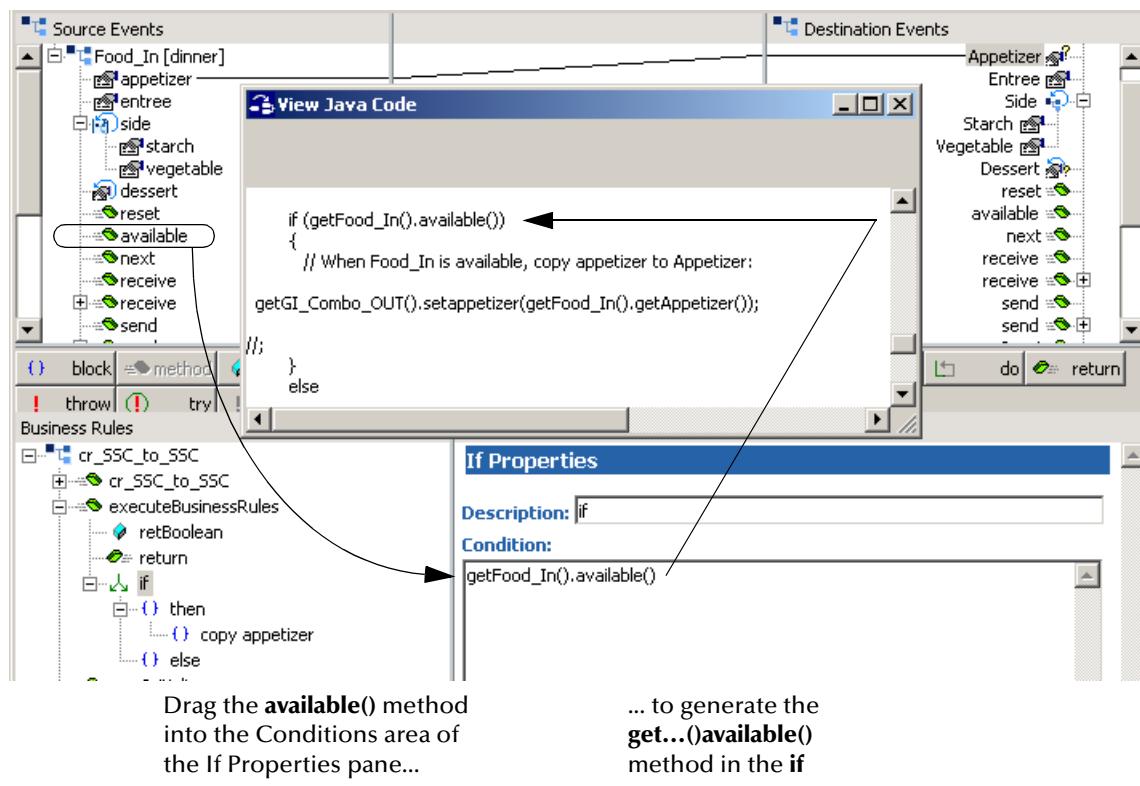
Return Type

boolean

Other Properties

Property	Description
comment	Code comments associated with this method.
signature	available()Z

Example



marshal()

Syntax

```
byte[] marshal()
```

Description

Organizes the data of this Event Type instance into a serialized byte stream (a BLOB). Since this flattens the instance—that is, converts it from a hierarchical structure to a linear one—**marshal()** allows a node with subnodes to be copied to a single destination node.

Parameters

None.

Return Type

byte[]—in other words, an array of bytes.

Throws

com.stc.jcsre.MarshalException—thrown when unable to marshal the ETD for this Event Type instance.

Other Properties

Property	Description
comment	Code comments associated with this method.
signature	marshal()B

next()

Syntax

```
boolean next()
```

Description

Checks to see whether there is more data waiting to be processed. If so, the data is unmarshalled (parsed) into this Event Type instance. For more information on the **unmarshal()** method, see “[“unmarshal\(\)” on page 346](#)”.

Use **next()** in conjunction with Event Linking and Sequencing (ELS). After ELS has handed off a enumeration of Events to the **executeBusinessRules()** placeholder, you can invoke **next()** to advance over the enumeration.

Parameters

None.

Return Type

boolean

Throws

`com.stc.common.collabService.CollabDataException`

Other Properties

Property	Description
comment	Code comments associated with this method.
signature	next()Z

publications()

Syntax

```
java.lang.Vector publications()
```

Description

Retrieves the names of all Event Types published by this Event Type instance.

Parameters

None.

Return Type

java.util.Vector—in other words, a Vector of Event Type object names.

Throws

CollabDataException

Other Properties

Property	Description
comment	Code comments associated with this method.
signature	publications()Ljava/util/Vector

rawInput()

Syntax

```
byte[] rawInput()
```

Description

This represents the original data contents of this Event Type instance before it was unmarshalled (parsed).

Parameters

None.

Return Type

byte[]—in other words, an array of bytes.

Throws

com.stc.common.collabService.CollabDataException

Other Properties

Property	Description
comment	Code comments associated with this method.
signature	rawInput()B

readProperty()

Syntax

```
java.lang.String readProperty(propName)
```

Description

Retrieves the value for a particular property that was assigned to this Event Type instance. Used in conjunction with [writeProperty\(\)](#) on page 347. Together, these two methods allow you to store data about an Event instance, outside of the Event itself. For example, [writeProperty\(\)](#) and [readProperty\(\)](#) can be used to store the name of the file containing the Event instance, the date and time it was published, the name of the publishing application, or even a checksum or signature. They can also be used to calculate and store key fields used by Event Linking and Sequencing (ELS).

For information on `this.jCollabController` methods to list and copy ETD properties, see “[getPropertyNames\(\)](#)” on page 577 and “[copyProperties\(\)](#)” on page 575, respectively.

Note: Properties that are set by `writeProperty()` are valid for the rest of the instances of the current Collaboration. Such properties retain their values between Collaborations only if published across a JMS e*Way Connection.

Example 1: To retrieve the UID of the publisher of an Event Type named YourEvent

```
String pubUUID = getYourEvent().readProperty("PUBLISHER_UUID")
```

Example 2: To retrieve the logical name of the publisher of an Event Type named MyEvent

```
String pubName = getMyEvent().readProperty("PUBLISHER_NAME")
```

Parameters

Name	Type	Description
propName	java.lang.String	Name of the property to read, or null if no such property exists.
A property with a specified name exists only in two cases: <ul style="list-style-type: none">▪ A property of that name was previously written to the message header by a prior call to <code>writeProperty()</code>; or▪ A property of that name is predefined by the Collaboration Rule itself. Two such predefined properties – namely, “PUBLISHER_UUID” and “PUBLISHER_NAME” – are shown in the examples above.		

Return Type

`java.lang.String`

Other Properties

Property	Description
comment	Code comments associated with this method.

Property	Description
signature	readProperty(Ljava/lang/String;)Ljava/lang/String;

receive()

Syntax

```
boolean receive()  
  
boolean receive(topicName)
```

Description

Applies to inbound Event Type instances only. Pulls any data waiting in the IQ for the current Event Type instance and unmarshals (parses) it into this Event Type instance. For more information on the **unmarshal()** method, see [“unmarshal\(\)” on page 346](#).

If a *topicName* is specified, **receive** pulls only data for a particular Event Type.

Parameters for receive(*topicName*)

Name	Type	Description
topicName	java.lang.String	The name of the topic (Event Type) residing in the IQ.

Return Type

boolean—for **receive()**, returns **true** only if a pertinent **JMsgObject** is available; for **receive(*topicName*)**, returns **true** only if a pertinent **JMsgObject** is available for the specified Event Type.

Throws

com.stc.common.collabService.CollabDataException—thrown when **JMsgObject** data cannot be unmarshalled.

Other Properties

Property	Description
comment	Code comments associated with this method.
signature	receive()Z receive(Ljava/lang/String;)Z

reset()

Syntax

```
boolean reset()
```

Description

Clears the contents of this Event Type instance. This method sets the values of all fields to null.

Parameters

None.

Return Type

boolean

Other Properties

Property	Description
comment	Code comments associated with this method.
signature	reset()Z

send()

Syntax

```
void send  
void send(topicName)
```

Description

Applies to outbound Event Type instances only. Sends the entire data content of this Event Type instance as output in its marshalled (serialized BLOB) form.

If *topicName* is specified, **send** only sends data for the specified Event Type; otherwise, it sends to all topics associated with the instance.

Parameters for send(*topicName*)

Name	Type	Description
topicName	java.lang.String	The name of the topic (Event Type) residing in the IQ.

Return Type

void

Other Properties

Property	Description
comment	Code comments associated with this method.
signature	send()V send(Ljava/lang/String;)V

subscriptions()

Syntax

```
subscriptions()
```

Description

Retrieves the names of all Event Types this Event Type instance subscribes to.

Parameters

None.

Return Type

java.util.Vector—in other words, a Vector of Event Type object names.

Throws

CollabDataException

Other Properties

Property	Description
comment	Code comments associated with this method.
signature	subscriptions()Ljava/lang/Vector;

topic()

Syntax

```
java.lang.String topic()
```

Description

Retrieves the name of the Event Type for this Event Type instance.

Note: If `topic()` is called when the Event Type instance has no Events available, it returns null. Therefore, it is good practice—especially for Event Type instances that are defined as In/Out—to call `available()` first and then call `topic()` only after learning that there is data waiting to be processed.

Parameters

None.

Return Type

`java.lang.String`

Throws

`com.stc.common.collabService.CollabDataException`

Other Properties

Property	Description
comment	Code comments associated with this method.
signature	<code>topic()Ljava/lang/String;</code>

unmarshal()

Syntax

```
void unmarshal(blob)
```

Description

De-serializes a particular byte stream and parses it into an appropriate hierarchical form for this Event Type instance.

Parameters

Name	Type	Description
blob	paramtype = byte[] javatype = blob	Byte arrays of the values to be unmarshalled.

Return Type

None.

Throws

com.stc.jcsre.UnmarshalException—thrown when unable to unmarshal the BLOB into the ETD for this Event Type instance.

Other Properties

Property	Description
comment	Code comments associated with this method.
signature	unmarshal((B)V

writeProperty()

Syntax

```
void writeProperty(java.lang.String propName,
                  java.lang.String propValue)
```

Description

Creates a user-defined property for Events in this Event Type instance. Used in conjunction with “[readProperty\(\)](#)” on page 339.

Together, these two methods allow you to store data about an Event Type instance, outside of the Event itself. For example, `writeProperty()` and `readProperty()` can be used to store the name of the file containing the Event instance, the date and time it was published, the name of the publishing application, or even a checksum or signature. They can also be used to calculate and store key fields used by Event Linking and Sequencing (ELS).

For information on `this.jCollabController` methods to list and copy ETD properties, see “[getPropertyNames\(\)](#)” on page 577 and “[copyProperties\(\)](#)” on page 575, respectively.

Note: Properties that are set by `writeProperty()` are valid for the rest of the instances of the current Collaboration. Such properties retain their values between Collaborations only if published across a JMS e*Way Connection.

Parameters

Name	Type	Description
propName	java.lang.String	User-defined name of a property to store in the message header. Applies only to Events being sent to a JMS e*Way Connection.
propValue	java.lang.String	The value to store for this property. Applies only to Events being sent to a JMS e*Way Connection.

Return Type

void

Other Properties

Property	Description
comment	Code comments associated with this method.
signature	<code>readProperty(Ljava/lang/String;Ljava/lang/String;)V</code>

7.8 Subcollaboration Rules

7.8.1 Terminology

Depending on how it is used, every Collaboration Rule is either:

- Used as a Root Collaboration Rule—in other words, a Collaboration Rule invoked by e*Gate itself; or
- Used as a Subcollaboration Rule, invoked by a parent Collaboration Rule.

Prior to release 4.5.2, Collaboration Rules could only be used as Root Collaboration Rules.

7.8.2 Purpose, Concepts, and Caveats

A Collaboration Rule, when used as a Root Collaboration Rule, is like a main program; when used as a Subcollaboration Rule, it is like a subroutine. For example:

- A Subcollaboration Rule allows you to reuse a valuable piece of work in another context without having to reinvent it or reconstruct it from scratch.
- Typically, a Subcollaboration Rule takes care of details or special-purpose parsings and transformations, allowing the parent Collaboration Rule to be simpler and more general.
- Sub[sub[...]]collaboration Rules can nest to indefinite depth, limited only by system resources (memory, stack, and so forth).
- A Subcollaboration Rule is invoked programmatically, whereas a Root Collaboration Rule, like a main program, is called by the e*Way itself.

Every Collaboration Rule runs in a *mapping environment* defined by its container:

- A Root Collaboration Rule's mapping environment is defined via the e*Gate GUI—namely, the **Collaboration Properties** dialog box.
- A Subcollaboration Rule's mapping environment is defined programmatically, via its parent's call to **setInstanceMap()**.

The following caveats apply to Collaboration Rules invoked as Subcollaboration Rules:

- Transactionality — in other words, Prepare/Commit/Rollback — can only be handled at the Root Collaboration Rule level, never by a Subcollaboration Rule.
- A Collaboration Rule that uses ELS can be invoked as a Subcollaboration Rule, but its **executeBusinessRules()** code runs immediately, bypassing its ELS-specific code.
- Collaboration Rules that use an API ETD—other than database ETDs—are ineligible for being used as Subcollaboration Rules.
- When an outbound Event Type instance is set to **Manual Publish**, its data is handled by its ETD's **send()** method (or not at all), and cannot be intercepted by its container. In other words, for a manually published Event Type instance, its data goes wherever **send()** sends it—typically an IQ or a JMS e*Way Connection.

7.8.3 Working with Subcollaboration Rules

Before you begin: You must have one or more Collaboration Rules that are eligible for use as Subcollaboration Rules. For example, they must not use ELS and must not use any API ETDs other than database ETDs.

For each Subcollaboration Rule you call from your Collaboration Rule, do the following.

To call a Subcollaboration Rule

- 1 Start the Java Collaboration Rules Editor and open the parent Collaboration Rule.
- 2 Immediately under the name of the Collaboration Rule, add a variable of type JSubCollabMapInfo.

For example:

```
mapForSub1
```

This variable keeps track of the Subcollaboration Rule's mapping information.

- 3 Within the **userInitialize()** method, add a rule to create the mapping information for the Subcollaboration Rule and assign it to the variable you added in step 2.

Generically:

```
<variable> = ...createSubCollabMapInfo("name", this)
```

where *name* is the Collaboration Rule named defined in Enterprise Manager.

For example:

```
mapForSub1 =
this.jCollabController.createSubCollabMapInfo("crCollabTMtoLX", this)
```

- 4 In the body of the Collaboration Rule, add a method. Within this method, do the following.

- A Add one variable of type **boolean**, to check on the Subcollaboration Rule's success.

For example:

```
isSub1_OK
```

- B Add a variable of type **byte[]** for each output you want to harvest from the Subcollaboration Rule.

For example:

```
outSub1OutET02
```

- C For each of the Subcollaboration Rule's inbound Event Type instances, add a rule that calls **setInstanceMap()** to populate the instance.

Generically:

```
<variable>.setInstanceMap("name", getParentNode(), data[], topic)
```

or, if no queue operation is being done:

```
<variable>.setInstanceMap("name", null, data[], topic)
```

For example:

```
mapForSub1.setInstanceMap("TMIn", getgenOut(),
getFromFTP().getPayload(), null)
```

- D For each of the Subcollaboration Rule's harvestable outbound Event Type instances, add a rule that calls `setInstanceMap()` again to harvest the instance.

Generically:

```
<variable>.setInstanceMap("name", get...(), null, null)
```

For example:

```
mapForSub1.setInstanceMap("LXOut", getgenOut(), null, null)
```

- E Assign a value to the variable you defined in step 3A. For example:

```
isSub1_OK = this.jCollabController.invoke(mapForSub1)
```

- F Add any additional logic and processing that you want to occur. The following Subcollaboration Rule-specific methods are helpful:

- `getCallingCollaboration()`
- `getInputData()`
- `getOutputData()`
- `getRuleName()`
- `isManualPublish()`
- `isPublisher()`
- `isTrigger()`

To call any of these methods, prepend the name of the variable you set in step 2. For example:

```
outData1_LX = mapForSub1.getOutputData("LXOut")
```

For complete details on these methods and others, see [“JSubCollabMapInfo Class \(com.stc.common.collabService\)” on page 603](#).

Note: If your Root Collaboration Rule is transactional and the Subcollaboration Rule fails, you can simply pass the boolean `false` upwards to `executeBusinessRules()` so that the entire transaction will roll back.

- 5 Within the `executeBusinessRules()` method, add a rule to call the method you defined in step 4.

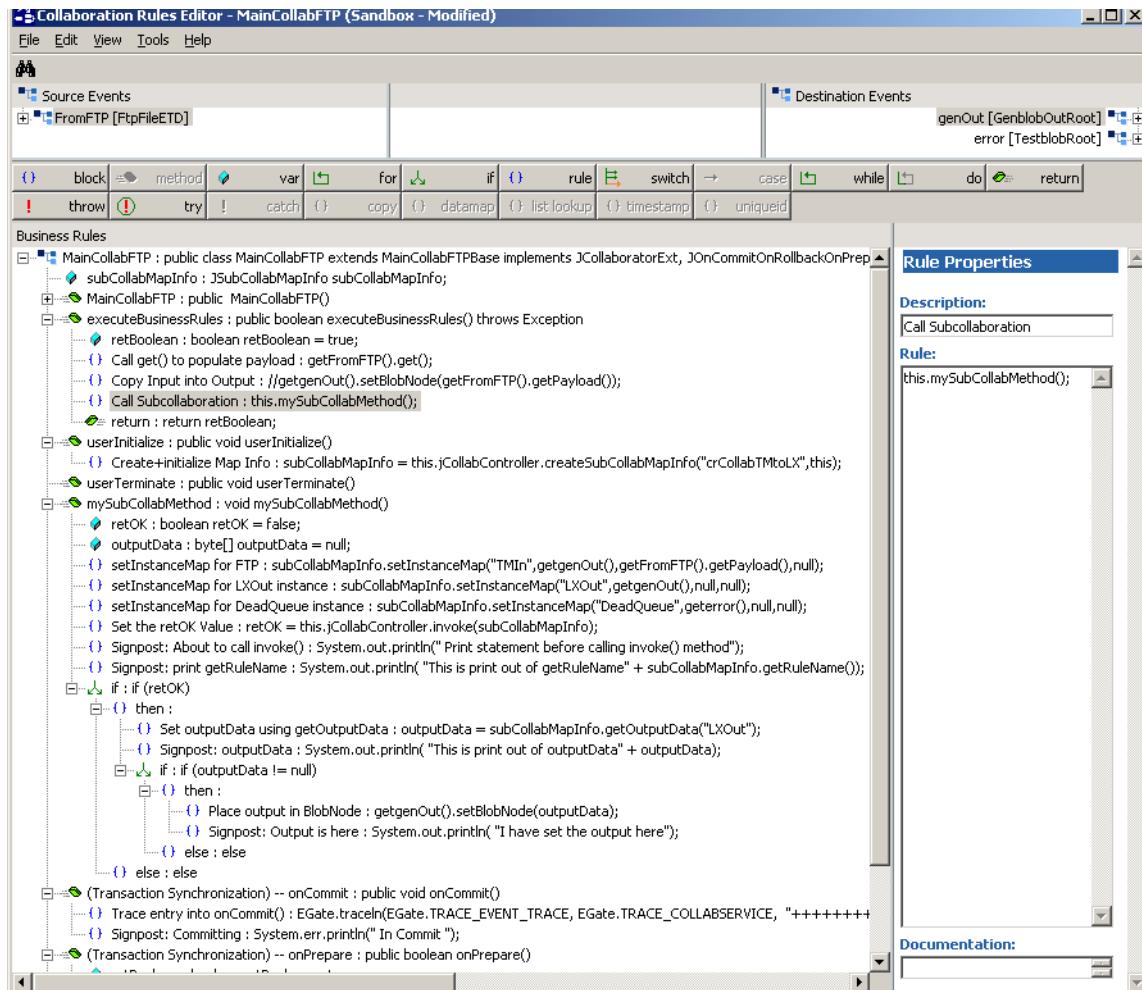
For example:

```
this.mySubCollabMethod();
```

Also add any additional logic and processing that might be needed before and after the method is called.

An example of a Root Collaboration Rule calling a Subcollaboration Rule is shown in [Figure 151 on page 351](#).

Figure 151 Root Collaboration Rule Calling a Subcollaboration Rule



Monk Collaboration Rules Editor

This chapter explains how to use the Monk Collaboration Rules Editor to create, add, and define Monk Collaboration Rules scripts in the e*Gate system.

8.1 Overview: Monk Collaboration Rules

The Monk Collaboration Rules Editor is the Monk language-based Collaboration Editor for e*Gate. These Collaboration Rules are Monk scripts that allow you to define how the e*Gate system validates, converts, and transforms Events (data packages) within a Collaboration.

The Monk Collaboration Rules Editor allows you to define these scripts that instruct the e*Gate system how to build specified output Events Types (classes of Events) from input Event Types.

Note: For more information on Event Types and Event Type Definitions (ETDs), see [Chapter 6](#) and [Chapter 5](#).

You can create Collaboration Rules, using the Collaboration Rules Editor window graphical user interface (GUI). This chapter explains the features of each element contained within or accessed from this feature.

8.1.1 Collaboration Rules Scripts and Types

There are several types of Monk scripts available for use as Collaboration Rules. For details on these scripts and types, see [Table 10 on page 107](#). Also, see [“Creating Collaboration Rules and Scripts” on page 106](#) for an explanation of Collaboration Rules/scripts and how they fit into the e*Gate setup operation.

8.1.2 Before You Begin

Before you define your Monk Collaboration Rules, ensure you have completed the following:

- Added all Event Types.
- Created and built your Monk ETDs: at least one source (inbound) ETD and at least one destination (outbound) ETD.

- Added the Collaboration Rules component. To do this, choose any Monk Collaboration Service—do not Pass Through—and set your subscription/publication relationships for the appropriate inbound/outbound Event Types.

Important: *This chapter discusses many operations related to using the Monk programming language. Advanced knowledge of Monk is not necessary to use the Monk Collaboration Rules Editor, but some knowledge can be helpful. For more information, see the **Monk Developer's Reference**.*

8.1.3 Task List

The primary tasks for defining Monk Collaboration Rules are:

- Choose the source and destination ETDs you want to use to define the Collaboration Rules. The source ETD represents the input Event format, while the destination ETD represents the output Event format.
- Define the Collaboration Rules you want the e*Gate system to follow in creating an output Event from an input Event. Add as many rules as needed to transform an input Event, according to your specifications.

8.2 Collaboration Rules Editor Window

This section describes the features contained within or accessed from the SeeBeyond Collaboration Rules Editor window.

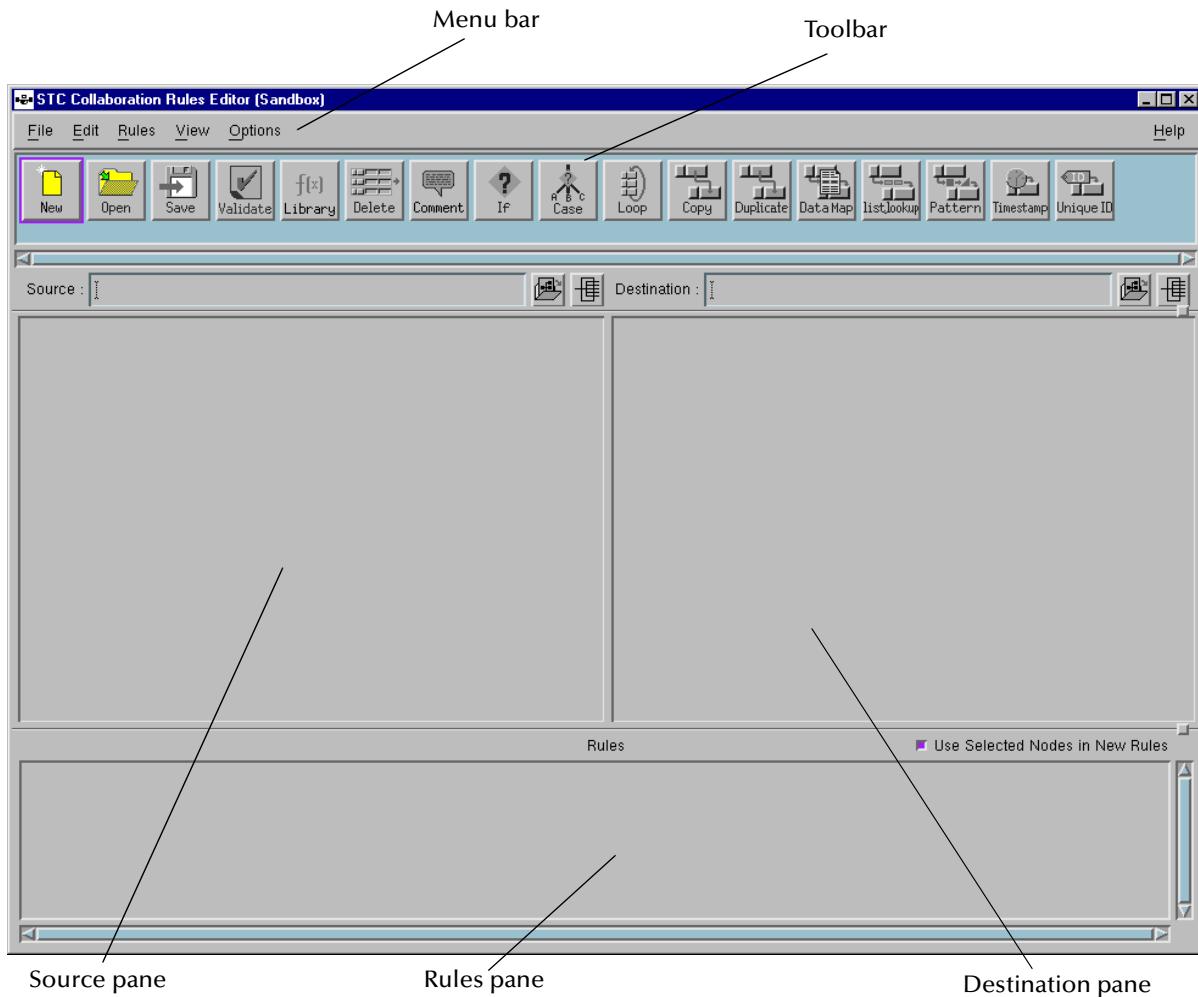
To access the Monk Collaboration Rules Editor window

From Enterprise Manager, do one of the following:

- On the toolbar, click .
- On the Tools menu, click **Collaboration Editor**.
- In the **Collaboration Rules Properties** dialog box, click **New**.

The Monk Collaboration Rules Editor window opens with no Collaboration Rules files displayed (see [Figure 152 on page 354](#)).

Figure 152 Monk Collaboration Rules Editor GUI Map



The Collaboration Rules Editor window (see Figure 152 above) has the following main sections:

- **Menu bar** — Contains the Collaboration Rules Editor menus.
- **Toolbar** — Contains buttons that allow you easy access to often-used features.
- **Source pane** — Shows the ETD Tree node diagram of the current input ETD file.
- **Destination pane** — Shows the ETD Tree node diagram of the current output ETD file.
- **Rules pane** — Displays the current Monk-scripted Collaboration Rules you are using.

8.2.1 Toolbar Buttons

Table 44 shows the Monk Collaboration Rules Editor buttons in the toolbar, including their functions. Operational details for each tool can be found in “[Basic Collaboration Rules Operations](#)” on page 379.

Table 44 Toolbar Buttons

Button	Function
	Creates a new Collaboration Rules file. If you currently have a Collaboration Rules displayed, clicking this button also clears the Collaboration Rules Editor window.
	Opens an existing Collaboration Rules file. The Open SeeBeyond Collaboration Rules dialog box opens. Select the desired Collaboration Rules file to open.
	Saves the current Collaboration Rules file.
	Verifies that all rules have been completely filled out and contain correct values.
	Displays the Function Library so that you can insert a function in your Collaboration Rules.
	Deletes the selected rule from the Rules list.
	Inserts a Comment rule in the Rules list.
	Inserts an If rule in the Rules list.
	Inserts a Case rule in the Rules list.
	Inserts a Loop rule in the Rules list.
	Inserts a Copy rule in the Rules list.
	Inserts a Duplicate rule in the Rules list.
	Inserts a Data Map rule in the Rules list.
	Inserts a List Lookup rule in the Rules list.

Table 44 Toolbar Buttons (Continued)

Button	Function
 Pattern	Inserts a Pattern rule in the Rules list.
 Timestamp	Inserts a Timestamp rule in the Rules list.
 Unique ID	Inserts a Unique ID rule in the Rules list.

Other Window Controls

The controls listed in Table 45 are found next to the **Source** and **Destination** text boxes.

Table 45 Window Controls

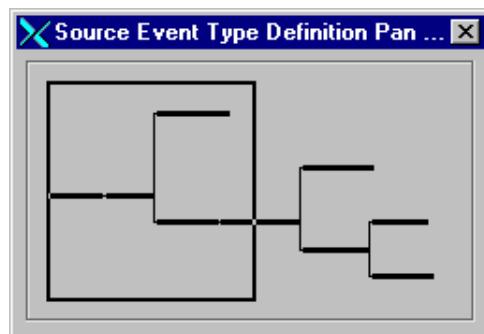
Control	Description
	Click to select a source or destination ETD.
	Opens the Source and Destination ETD Panning Windows.

ETD Panning Windows

Use the Source ETD Panning window and Destination ETD Panning window to view different parts of the input and output ETDs, respectively.

Using these windows has the same effect as using the scroll bars, that is, they enable you to see different portions of a larger structure; see Figure 153. Also, the windows also give you a condensed view of the ETDs as a whole.

Figure 153 ETD Panning Window



Note: You do not need to close the Panning Window to resume work in the Collaboration Rules Editor window. You can leave the windows open as long as you like.

Rules Pane Controls

The controls listed in Table 46 manage the Rules List display.

Table 46 Rules Pane Controls

Control	Description
	Displays additional details for the selected rule.
	Hides additional details for the selected rule.
	Use the rectangular control to resize the pane in which the Rules list is displayed (located at the right edge of the application window). Drag the button to resize the pane.

8.2.2 Menu Bar

The Monk Collaboration Rules Editor provides the following menus:

- “File Menu” on page 358
- “Edit Menu” on page 359
- “Rules Menu” on page 359
- “View Menu” on page 360
- “Options Menu” on page 361
- “Help Menu” on page 361

This section explains the options available under each of these menus. See “[Basic Collaboration Rules Operations](#)” on page 379 for an explanation of general Collaboration Rules operations.

Some menu selections open dialog boxes. For an explanation of how to use a specific dialog box, see the section later on in this chapter that explains the associated rule or operation.

File Menu

The Collaboration Rules Editor **File** menu commands are explained in Table 47.

Table 47 File Menu Commands

Command	Function
New	Displays the New SeeBeyond Collaboration Rules dialog box, where you specify basic parameters for the new Collaboration Rules, including the file name and input/output ETDs.
Open	Displays the Open SeeBeyond Collaboration Rules dialog box, allowing you to choose a Collaboration Rules file to display. The Collaboration Rules Editor window displays the selected Collaboration Rules file. You can only have one Collaboration Rules file open at a time.
Save and Edit Using External Editor	Saves the current Collaboration Rules file and opens it using the designated external Editor. Note: This command is only available when the external Editor is designated in the Enterprise Manager.
Reload From Local Machine	Opens a local copy of the currently displayed Collaboration Rules file rather than the copy in the e*Gate registry. Use this option only when directed to do so by SeeBeyond support staff.
Use As	Allows you to select the type of Collaboration Rules file for the current Collaboration Rules.
Open a Source Event Type Definition	Displays the Open Source ETD dialog box, where you choose the input ETD for the current Collaboration Rules. The ETD displays in the Source pane.
Open a Destination Structure	Displays the Open Destination ETD dialog box, where you choose the output ETD for the current Collaboration Rules. The ETD displays in the Destination pane.
Save	Saves the current Collaboration Rules file. Click OK to complete the save operation.
Save As	Opens the Save SeeBeyond Collaboration Rules As dialog box. If you are saving a new Collaboration Rules file, for which a file name has not been assigned, the system opens the Save As file selection dialog box. Use the dialog box to select a directory and assign a new file name to the current file.
Validate	Checks your current Collaboration Rules file for syntax errors.
Promote to Run Time	Promotes the current Collaboration Rules file out of your Sandbox folder and into the run-time (production) system.
Remove	Deletes the current Collaboration Rules file from the Sandbox folder.
Monk Function Header	Allows you to enter parameters for the Monk function. Note: This command is only available when you are working with a Monk function.
Main Comment	Provides access to a dialog box where you can enter comments that apply to the entire Collaboration Rules. There are no syntax restrictions.
Try/Catch Statement	Allows you to enter code for Monk exception handling.
Close	Closes the Collaboration Rules Editor window.

Edit Menu

The Collaboration Rules Editor **Edit** menu commands are explained in Table 48.

Table 48 Edit Menu Commands

Command	Function
Cut	Deletes the selected rule from the Rules list, temporarily placing it to the Windows Clipboard. You can then use the Paste option to place the rule back in the Rules list.
Copy	Temporarily copies a selected rule to the Clipboard. You can then use the Paste option to place the rule back in the Rules list.
Paste	Pastes the rule in the Rules list that you last placed on the Clipboard with the Cut or Copy option.
Delete Rule	Deletes the selected rule from the Rules list.
Find First Rule	Associated with Source ETD — Select a node in the Source ETD pane then choose this option to find the first rule associated with the selected node, in the Rules list. Associated with Destination ETD — Select a node in the Destination ETD pane then choose this option to find the first rule associated with the selected node, in the Rules list.
Find Next Rule	Associated with Source ETD — Finds the next occurrence of a rule, in the Rules list, associated with the selected Source ETD node. Associated with Destination ETD — Finds the next occurrence of a rule, in the Rules list, associated with the selected Destination ETD node. NOTE: Once you reach the end of the Rules list, you are asked if you want to continue the search at the beginning of the list.
Find Nodes Associated with Rule	Highlights the nodes in the Source and Destination panes, which are referenced in the currently selected rule in the Rules list.
Find Node	In Source ETD — Allows you to search through your Source ETD for a node. You search by node name. In Destination ETD — Allows you to search through your Destination ETD for a node.

Rules Menu

The Collaboration Rules Editor **Rules** menu commands are explained in Table 49.

Table 49 Rules Menu Commands

Command	Function
Add If	Adds an If rule.
Add Loop	Adds a Loop rule.
Add Case	Adds a Case rule.
Add Comment	Adds a Comment rule.
Add Copy	Adds a Copy rule.

Table 49 Rules Menu Commands (Continued)

Command	Function
Add Display	Adds a Display rule.
Add Duplicate	Adds a Duplicate rule.
Add Data Map	Adds a Data Map rule.
Add List Lookup	Adds a List Lookup rule.
Add Change Pattern	Adds a Change Pattern rule.
Add Timestamp	Adds a Timestamp rule.
Add Unique ID	Adds a Unique ID rule.
Add Let	Adds a Let rule.
Add Set!	Adds a Set! rule.
Add Function	Adds a Function rule.
Add User Function	Adds a User Function rule.
Add Set Regex	Adds a Set Regex rule.

View Menu

The Collaboration Rules Editor **View** menu commands are explained in Table 50.

Table 50 View Menu Commands

Command	Function
Function Library	Displays a library of functions you can use in your Collaboration Rules, to perform special operations.
Float Toolbar	Opens a floating version of the toolbar that you can drag to a convenient location in the Collaboration Rules Editor window.
Float Source ETD Panning Window	Places the Source ETD Panning Window on a floating palette you can place anywhere in the Collaboration Rules Editor window. Use the pane to scroll across and down the open source ETD.
Float Destination ETD Panning Window	Places the Destination ETD Panning Window on a floating palette you can place anywhere in the Collaboration Rules Editor window. Use the pane to scroll across and down the open destination ETD.
Expand	Source ETD — Fully expands your view of the source ETD Tree. Destination ETD — Fully expands your view of the destination ETD Tree. Both ETDs — Fully expands your view of both the source and destination ETD Trees.

Options Menu

The Collaboration Rules Editor **Options** menu commands are explained in Table 51.

Table 51 Options Menu Commands

Command	Function
Select Drag Drop Rule	Selects a custom Monk function to insert when a node is dragged from the Source ETD to the Destination ETD (only applicable to functions that you write, not predefined Monk functions). If no function is specified, a Copy rule is inserted.
Use Selected Nodes in new Rule	Automatically inserts the currently selected source and destination ETD nodes in a newly added rule to the Rules list.
Add Rule AFTER Selected Rule	Causes new rules to be inserted in the Rules list after the currently selected rule. This option is the default selection.
Add Rule BEFORE Selected Rule	Causes new rules to be inserted in the Rules pane before the currently selected rule.

Help Menu

The Collaboration Rules Editor **Help** menu commands are explained in Table 52.

Table 52 Help Menu Commands

Command	Function
e*Gate Help Topics	Activates the online Help system for the Collaboration Rules Editor GUI.
Monk Developer's Reference	Opens the Monk Developer's Reference.
About e*Gate	Displays basic information about the current e*Gate software version.

8.3 Creating Monk Collaboration Rules Scripts

This section explains the basic procedures of how to create new Collaboration Rules scripts, under the following topics:

- [Creating New Monk Collaboration Rules](#) on page 362
- [Adding and Arranging Rules](#) on page 366

If you want to write your Collaboration Rules in the Monk programming language mode using a text editor, see the *Monk Developer's Reference* for details. This chapter explains enough of basic Monk operations to allow you to effectively use the Collaboration Rules Editor feature.

8.3.1 Getting Started

Make sure you have completed the following Event-related tasks *before* creating Collaboration Rules:

- Added all Event Types.
- Created and built source and destination Event Type Definitions (ETDs).
- Added Collaboration Rules components.

For details on these operations, see [Chapter 4](#).

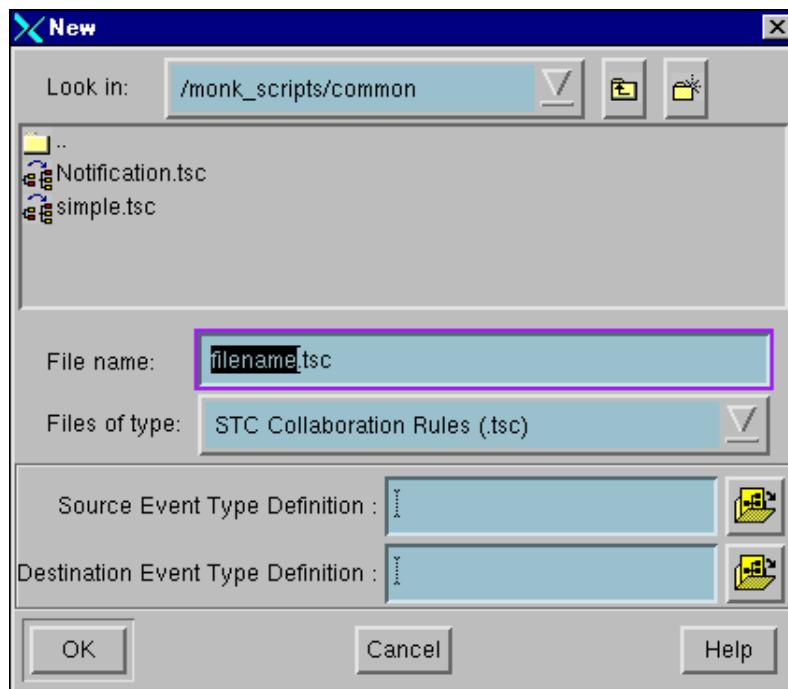
8.3.2 Creating New Monk Collaboration Rules

Creating new Monk Collaboration Rules requires that you know your source and destination ETDs.

To create new Monk Collaboration Rules scripts

- 1 From the Collaboration Rules Editor, on the **File** menu, click **New** to open the **New** dialog box. See Figure 154.

Figure 154 New Dialog Box

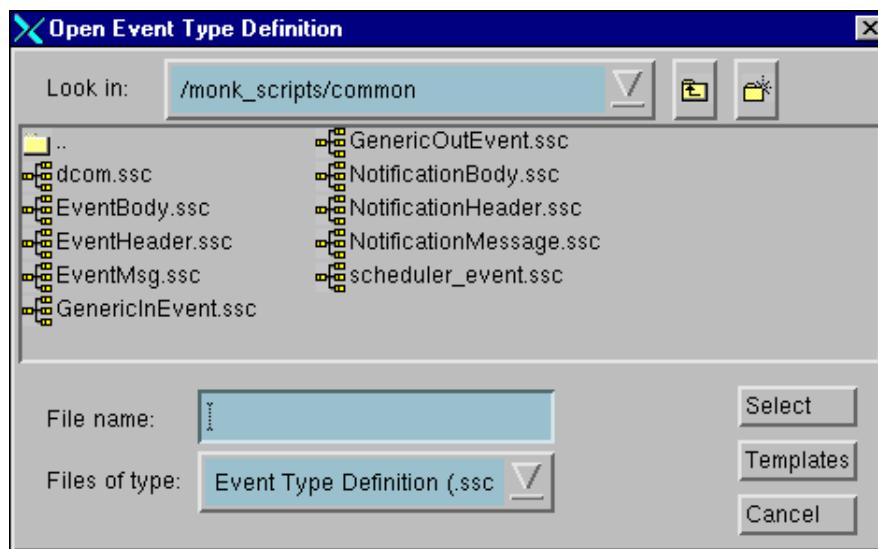


- 2 In the **New** dialog box, enter the following information:

- ♦ In the **File name** box, enter the name of the file you want to create. Use the directory selection window above to select the directory in which to store the file. The file extension, **.tsc**, is the default file extension for Collaboration Rules. It is automatically appended to your file name, so you do not need to specify a file extension.

- ♦ From the **Files of type** list, select the type of file you want to create.
 - ♦ Continue with the rest of the steps in this procedure, based on the type of file you want to create.
- 3 Click  to the right of the **Source Event Type Definition** text box to open the **Open Event Type Definition** dialog box. See Figure 155.

Figure 155 Open Event Type Definition Dialog Box



- 4 In the **Open Event Type Definition** dialog box, select the ETD file representing the input data.

Click **Templates** to display and choose from a list of standard ETDs delivered with the Collaboration Rules Editor: HL7, X12, and EDIFACT. By default, standard structures are stored in the following directories:

```
$EGATE_ENV/library/etds/HL7
$EGATE_ENV/library/etds/X12
$EGATE/library/etds/EDIFACT
```

- 5 Click **OK** to close the **Open Event Type Definition** dialog box.
- 6 Click  to the right of the **Destination Event Type Definition** text box. The **Open Event Type Definition** dialog box opens again.
- 7 Select the ETD file representing the output Event resulting from the current Collaboration. Click **Templates** to display and choose from the list of standard ETDs: HL7, X12, and EDIFACT.
- 8 Click **OK** to close the **Open Event Type Definition** dialog box.
- 9 Click **OK** to close the **New** dialog box.

You have now completed creating a new Collaboration Rules component. Before you begin adding the rules that describe the how the Collaboration processes an incoming Event, see “[How e*Gate Processes Event Data](#)” on page 364.

For new Collaboration Rules files, database poll scripts, or Screen Scripter functions

- If you are creating a database poll script, skip this step. Otherwise, in the **Source Event Type Definition** box, enter the name of the Event Type Definition file that represents the input Event. Click  for assistance in selecting a file.
- In the **Destination Event Type Definition** box, enter the name of the Event Type Definition file that represents the output Event. Click  for assistance in selecting a file.

Note: To use a predefined ETD for either the source or destination ETD, click , then click **Templates** for a list of available template files.

For new Monk functions

- In the **Usage** text box, enter text that illustrates the syntax of the new function. Click **Add Source Path** or **Add Destination Path** to insert reminder text for <source-path> or <destination-path> within the **Usage** text box.
- To enable this function to be used as a drag-and-drop rule, check **Can be used as a default Drag Drop rule**.

8.3.3 How e*Gate Processes Event Data

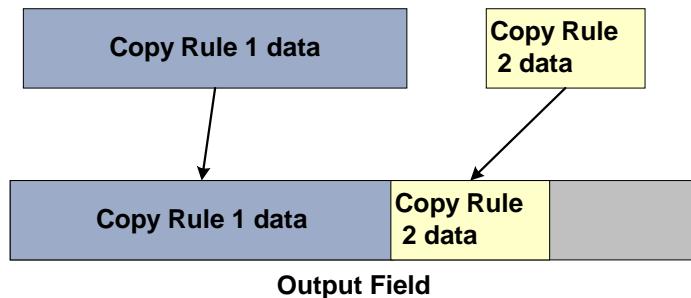
This section explains the guidelines e*Gate follows when creating output Events from input Events. Understanding these concepts can help you in creating Collaboration Rules scripts that bring about these processes.

Appending Data

You can define multiple rules that write data to the same output Event field, without any data's being overwritten. e*Gate appends each set of data to the end of the previous set of data within the Event field.

For example, if you set up two **Copy** rules for the same output field, both sets of copied data are placed in the output field, one after the other. Data is appended in the order in which the Collaboration Rules appear in the Collaboration Rules file. See Figure 156.

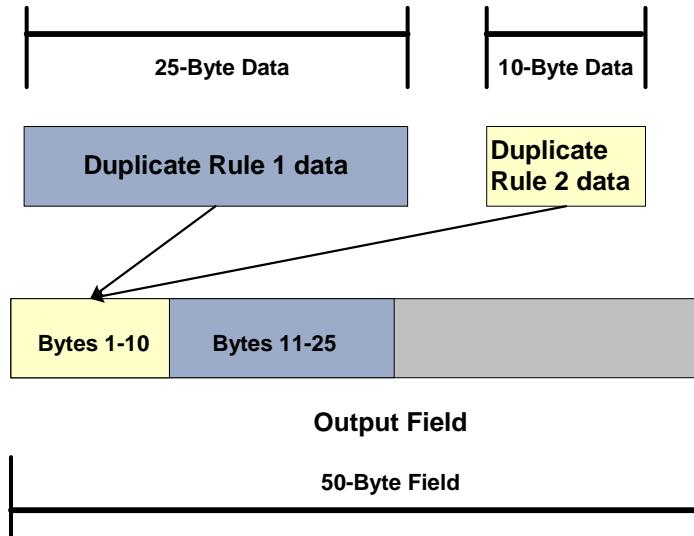
Figure 156 Appending Data with the **Copy** Rule



Duplicate Rule

Data written to the output Event field using the **Duplicate** rule is replaced if you write additional data to that output Event field with another **Duplicate** rule. See Figure 157.

Figure 157 Appending Data with the Duplicate Rule

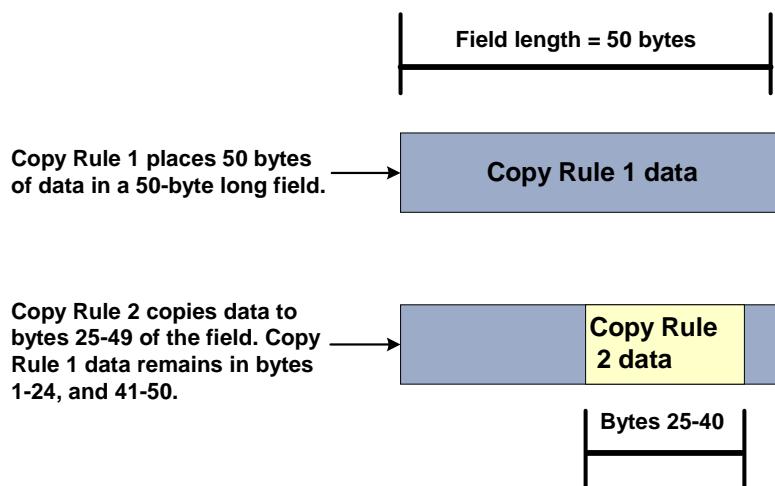


If you write additional data using the **Copy** rule, however, the data is appended as shown in Figure 157 above.

Assigning Offset Values

A set of data is not appended to the end of a previous set of data when you specify an offset that places data in the middle of other data. Figure 158 shows an example.

Figure 158 Offset Data



If data is written to a delimited output Event destination, and that data is offset, then leading spaces are inserted up to the beginning of the offset data.

Trailing Spaces

When you write data from an input Event to an output Event, any trailing spaces in the input Event data are also written to the output Event. To strip trailing spaces from input Event data, use the **Function** rule.

Fixed Data Lengths

When fixed-length data from an input Event is written to a smaller fixed-length destination in an output Event, the input Event data is truncated.

If fixed-length data from an input Event is smaller than its fixed-length output Event destination, the extra bytes in the output Event are filled with leading and trailing spaces.

8.3.4 Adding and Arranging Rules

After you create a new Collaboration Rule script, you define it by adding rules using the Collaboration Rules Editor window. The added rules appear in the Rules pane. You can further arrange (or rearrange) those rules once they have been added.

Adding Rules and Elements

Depending on how you want to create your list of rules, choose either **Add Rule After Selected Rule** or **Add Rule Before Selected Rule** in the **Options** menu.

To add a rule

Use one of the following methods:

- Select the desired rule from the **Rules** menu.
- To insert a rule using the toolbar, click the button representing the desired rule.

Note: *The most frequently used rules are available on the toolbar.*

To add to a list of rules

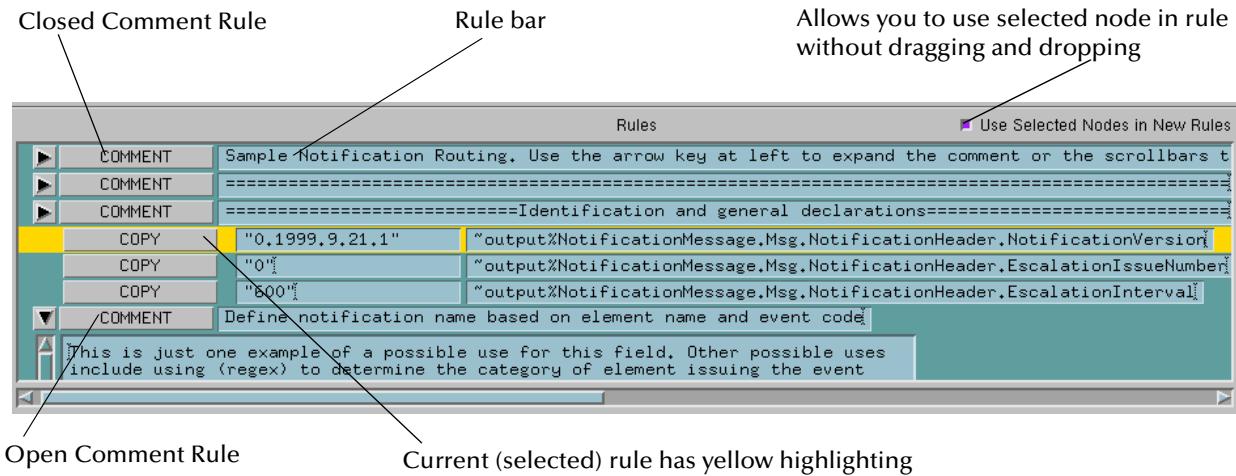
- 1 Place the mouse pointer over the desired icon (for example, a node).
- 2 Click and hold the middle mouse button.
- 3 Drag the icon to the desired position in the Rules pane.
- 4 Release the mouse button. The rule is inserted for you.

Note: *If your mouse does not have a middle button, hold down both mouse buttons to have the same effect as pressing the middle mouse button.*

When you add a rule, it is automatically selected (highlighted) and becomes the current rule. This feature allows you to add several rules in a row, if you have the **Add Rule**

After Selected Rule option selected. The rules are added one after another in the Rules pane. See Figure 159.

Figure 159 Rules Pane with Added Rules



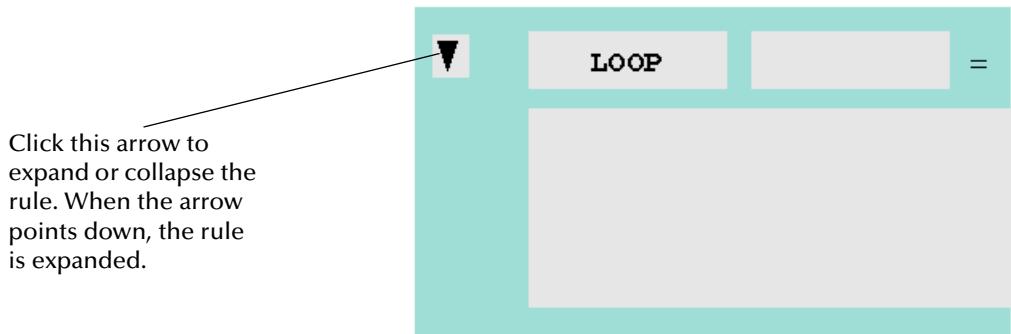
Arranging Rules

To drag a rule to a new position in the Rules pane, position your mouse over the rule, click and hold the middle mouse button, and drag the rule to the desired position. Release the mouse button. The rule is moved to the new position.

Additional Information

When a rule is selected, it is highlighted a bright yellow. Very long rules, such as the **Let** and **Loop** rules, can be collapsed to take up less space and give you a better view of other rules. See Figure 160.

Figure 160 Collapsing and Expanding Rules



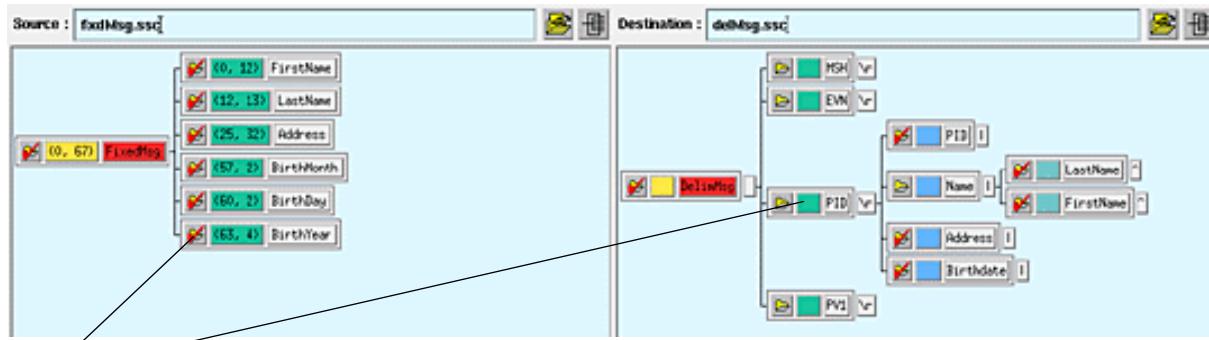
Selecting Event Elements

Defining Collaborations requires you to specify the Event elements in both the source and destination ETDs on which Collaboration operations are performed. Select Event elements when defining Collaboration Rules, using any of the following ways:

- Point to and click on Event elements with the left mouse button before inserting the corresponding rule in the Collaboration Rules Editor window.
- Insert a rule in the Collaboration Rules Editor window; then, drag and drop Event elements with the middle mouse button onto the rule.
- Fill in Event element addresses directly in the Rule bar.

The point-and-click method is more useful when you are first defining your Collaboration Rules. The drag-and-drop method is more useful when you are modifying Collaboration instructions, and need to quickly change the Event elements on which a particular operation are performed. See Figure 161.

Figure 161 Specifying Event Elements



Click on a node in your source or destination ETD to select it to include in a Collaboration Rules component. Or, use the drag-and-drop method.

To fill Event element locations directly in a Rule bar, you must use Monk syntax; however, you only need to do this when defining an **If**, **Loop**, **Let**, **Set!**, or **Function** rule. See “[Defining ETD Paths in a Loop Rule](#)” on page 399 for details.

To use the point-and-click method

- 1 In the Options menu, make sure **Use Selected Nodes in New Rules** option is selected.
- 2 Position the mouse arrow on an Event element in the source or destination ETD.
- 3 Press and release the left mouse button.

If you select either an Event element that repeats, or belongs to a higher-level node that repeats, the **Select Repetition Instance** dialog box appears. Specify the instance of the repeating element you want to perform the Collaboration Rules on. For details on how to use this dialog box, see “[Defining Instances of Repeating Event Elements](#)” on page 371.

Once you have selected Event elements this way, you can insert a rule and the selected elements are automatically included in the rule definition.

To use the drag-and-drop method

- 1 Add a Collaboration Rules component.
- 2 Position the mouse arrow on an Event element in the source or destination ETD.
- 3 Press and hold down the middle mouse button as you move the mouse to the desired position in the rule. Then release the button.

If you select either an Event element that repeats, or belongs to a higher-level node that repeats, the **Select Repetition Instance** dialog box appears. Specify the instance of the repeating element you want to perform the Collaboration Rules on. For details on how to use this dialog box, see “[Defining Instances of Repeating Event Elements](#)” on page 371.

The Event element is inserted into the rule.

Note: Before dragging and dropping nodes on a Rule bar, it's a good idea to delete the sample text that appears in the Rule bar. Just click and hold your mouse button over the beginning of the text, drag the cursor to the end of the text, release the mouse button, and press the **Delete** key.

Defining ETD Paths

When you define a rule in a Collaboration Rules component, you need to specify both input and output Event locations.

These Event locations are called *path expressions*. Path expressions point to a specific location in a ETD, such as a segment or field. Path expressions tell the rule where to perform its operation.

You specify path expressions by dragging elements from the ETDs and dropping them onto the Rule bar. Sometimes your ETD cannot contain all the levels you need to include in a Collaboration Rules component. In this case, you can add information to the path expression to access lower-level Event elements missing from your ETD.

Using Path Expressions

When you reference an Event element in a rule, you can see that the element has the following syntax in the Rule bar:

`~Event-name%pathelement1.pathelement2.pathelementN`

The variables in this expression are:

Event-name

The name of the Event buffer, either *input* or *output*, preceded by a tilde (~).

pathelement

A list of Event locations separated by periods (.); each element of the list can be:

- A name assigned to the selected ETD node

- An integer that represents the Event element's location. The first Event element at a given level (that is, the first segment, the first field, and so on) is counted as 0.

For example:

~input%MSG.ST

This path expression represents the ST segment of the MSG group in an input ETD.

Referencing an Event Element Missing from Your ETD

To reference an Event element missing from your ETD, use integers that represent the Event element's location. The first Event element at a given level (that is, the first segment, the first field, and so on) is counted as 0.

Table 53 shows some examples.

Table 53 Sample Events with Path Locations

This Path	Locates This Event Element
~input%MSHGRP.MSH.4	Field 4 (the fifth field) of the MSH segment
~input%MSHGRP.MSH.4.3	Sub-field 3 (fourth sub-field) of field 4 (fifth field) in the MSH segment

Referencing Byte Location

Use the following syntax in a Rule bar to reference the byte location of an Event element:

final_path element:byte_offset,length

where:

byte_offset

The beginning byte position, counted from the first byte of the Event location.

Note: *The first byte is counted as 0.*

length

The length of the bytes referenced, counted from one.

For example:

N1:2,10

This path expression references the N1 segment, starting from byte 3, and extending for a length of 10 bytes.

Referencing Instances of a Group

If you are referencing an instance of a repeating Event element that is not included in your ETD, use the following syntax:

Path element [Index]

where **Index** is an integer that represents the instance of the repeating Event element desired. Count Event element instances from 0, the beginning of the Event. For example, the third instance of an element has an index of 2.

Here is a sample path expression that references an instance of a group:

~input%MSG.DTM[2].4

This path expression references field 4 (the fifth field) of the third instance of the DTM segment of the MSG group.

Note: You only need to specify instances in a path expression when you are referencing an Event element that is missing from your ETD. When selecting repeating elements from a ETD to specify paths, the Collaboration Rules Editor presents you with a dialog box where you can define instances of repeating elements. See “[Defining Instances of Repeating Event Elements](#)” on page 371 for details.

Defining Instances of Repeating Event Elements

The **Select Repetition Instance** dialog box automatically appears when you select one of the following types of Event elements to include in a Collaboration Rules component:

- An element that repeats, such as a telephone number field that repeats twice, where the first instance contains a home number, and the second contains a work number.
- A nonrepeating element that belongs to higher-level repeating elements (which you can think of as the element’s parents), such as a Comment field that belongs to a segment, NTE, that repeats.
- A repeating element that also has repeating parents.

Note: The terms *parent*, *child*, and *sibling* elements are used here in the context of node elements. See “[ETD Creation and Nodes](#)” on page 205 for a complete explanation of nodes in Events.

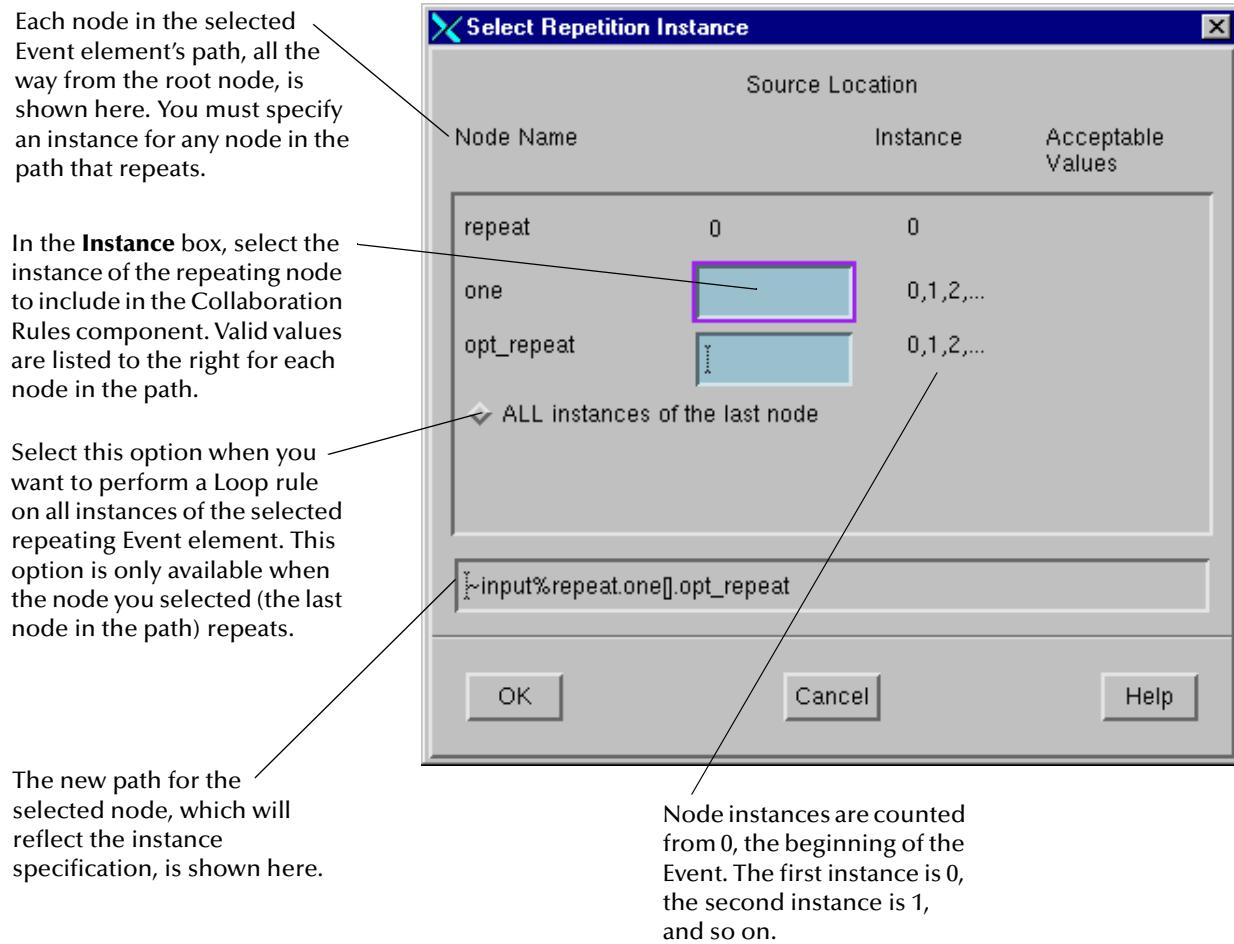
When you select a repeating Event element, or an element that has repeating parents, the **Select Repetition Instance** allows you to specify the repetition, or **instance**, of the repeating node you want to include in the Collaboration Rules component.

To include the element you selected for a Collaboration Rules component, you have to tell the system which instance of the element, its repeating parents, or both, to which you want to apply the Collaboration Rules. By asking you to specify an instance, the system is asking, “Out of this group of elements that repeats, which one do you want me to work on?”

Select Repetition Instance Dialog Box

This feature allows you to specify the desired repeating node in the rule. This dialog box only displays when performing a **Change Pattern**, **Copy**, **Duplicate**, or **List Lookup** rule on a repeating node (see [Figure 162 on page 372](#)).

Figure 162 Select Repetition Instance



To use the Select Repetition Instance dialog box

- For each repeating node in the path to the selected node, type the desired instance to which you want to apply the selected rule. The full path is displayed at the bottom of the dialog box.
- If the selected node is a repeating node and you want to apply the rule to all instances of that node, select **All instances of the last node** (this option is only available if the selected node is a repeating node).

Selecting Repetition Instances

When you select a repeating Event element or an element that has repeating parents, the system wants to know whether:

- You want it to perform Collaboration Rules on a particular member of a repeating group of elements.
- You want it to perform Collaboration Rules on all members.
- You want it to perform Collaboration Rules on a range of members.

If you do not provide this information, the system does not know where to start. Here are the conditions under which you define instances:

- When you want to perform Collaboration Rules on a particular instance of a repeating group, that is, “Do this Collaboration Rules only on this member of the repeating group of elements.”
- When you want to perform Collaboration Rules on all instances of a repeating group, that is, “Do this Collaboration Rules on the entire group of repeating elements.”

Important: You can only select all instances when setting up a **Loop** rule.

- When you want to link a repeating group to a variable; this is only applicable to Collaboration Rules included in a **Loop** rule (see “[Defining ETD Paths in a Loop Rule](#)” on page 399 for details about loop variables). Use this method to perform Collaboration Rules on a range of members in a repeating group.

Identifying Repeating Event Elements

A repeating Event element can be identified in a ETD when one of the symbols listed in Table 54 appears inside it.

Table 54 Repeating Event Elements

Node with Repetition Type	Symbol Contained
Range of repetitions	<n-m> (where n and m are integers)
Occurs zero or more times (optional-repeating)	*
Occurs one or more times	+

To select an Event element’s node in either an input or output ETD

- Press and hold down the middle mouse button on the node, and begin dragging the mouse cursor to the desired position in the rule.
- Or, if you are defining a **Copy** rule, you can drag a node from the input ETD to a node in the output ETD.

Selecting multiple levels of instances

Sometimes when you choose an element to include in a Collaboration Rule component, the **Select Repetition Instance** dialog box asks you to define several levels of instances. This happens when the element you picked repeats on multiple levels in the ETD hierarchy.

For example, if you choose an Event element that is a repeating field, and the segment it belongs to also repeats, the system wants you to tell it which instance of both the segment and the field to work on.

Each blank Instance field must be filled in before you are able to include the Event element in a Collaboration Rules component.

Specifying a Variable as an Instance

The only time you specify a variable as an instance is when you're setting up Collaboration Rules inside a **Loop** rule. Since a **Loop** rule works on multiple instances of an Event element, it uses a variable to keep track of how many times it has performed a Collaboration Rules on an Event element. Once the variable reaches a specified value, the **Loop** rule knows it has reached its goal, and stops.

For more details about the **Loop** rule and **Loop** rule variables, see [“Using the Loop Rule” on page 396](#).

Note: If you are defining a **Loop** rule that loops on multiple levels of repeating nodes (this is known as a nested loop), you also specify variables for the instances of a repeating node's repeating parents. See the **Monk Developer's Reference** for a nested loop example.

Using the Select Repetition Instance Dialog Box

Table 55 provides guidelines on using out the **Select Repetition Instance** dialog box.

Table 55 Select Repetition Instance Dialog Box Entries

Desired Action	Does Node Have Repeating Parents?	Procedure
Select a particular instance of a node's repeating group.	Yes	<ol style="list-style-type: none"> 1 Type an integer or variable in each Instance field preceding the node's Instance field. 2 Type an integer in the selected node's Instance field. 3 Click OK.
	No	<ol style="list-style-type: none"> 1 Type an integer in the node's Instance field. 2 Click OK.
Select a node's entire repeating group to include in a Loop rule	Yes	<ol style="list-style-type: none"> 1 Type an integer or variable in each Instance field preceding the node's Instance field. 2 Click the All instances button next to the node's Instance field. 3 Click OK.
	No	<ol style="list-style-type: none"> 1 Click the All instances button next to the selected node's Instance field. 2 Click OK.

Table 55 Select Repetition Instance Dialog Box Entries (Continued)

Desired Action	Does Node Have Repeating Parents?	Procedure
Link a node to a variable	Yes	<ol style="list-style-type: none"> 1 Type an integer or variable in each Instance field preceding the node's Instance field. 2 Type a variable in the node's Instance field. 3 Click OK.
	No	<ol style="list-style-type: none"> 1 Type a variable in the node's Instance field. 2 Click OK.

See the *Monk Developer's Reference* for a nested loop example.

Note: Remember that valid integer values are displayed in the **Instance** column, where **0** represents the first instance of a repeating element, **1** the second instance, and so on.

Filling in Rule Details

When you create Collaboration Rules, there are details that must be specified in order for the rule to be executed according to your needs as follows:

- You can use dialog boxes available for some rules which provide forms for filling out parameters. To access these dialog boxes, click the rule's name button in its Rule bar. See Figure 163.

Figure 163 Rule's Name Button



- You can fill in rule details directly in the text boxes in a Rule bar. The only Collaboration Rules in which you need to do this are the **If**, **Loop**, **Let**, **Set!**, and **Function** rules. See “[Using Collaboration Rules](#)” on page 386 for a summary of all the Collaboration Rules, including the **If**, **Loop**, **Let**, **Set!**, and **Function** rules.

Where applicable, the procedures in this chapter are targeted for those who want to use dialog boxes to specify Collaboration Rules details.

Specifying byte locations

If you want to target a specific byte location within a selected Event element, there are two ways to do it:

- By specifying a byte offset and/or length in a rule's dialog box

- By specifying a byte location directly in the Event element's path expression; this is only applicable to rules that have no dialog boxes, including the **If**, **Loop**, **Let**, **Set!**, and **Function** rules. See “[Defining ETD Paths in a Loop Rule](#)” on page 399 for details about how to specify a byte location in an Event element's path expression.

Specifying a byte location is optional; by default, a rule is applied to the entire contents of any Event element you select from a ETD. However, if you need to target a subset of the data contained in an Event element, then you need to specify a byte location. For details, see Table 56.

Table 56 Showing Byte Locations in Rule Dialog Boxes

Text Box	Definition
Byte Offset	<p>Defaults to entire Event element selected.</p> <p>Source – For the source Event, fill in the byte location from which to start the operation.</p> <p>Destination – For the destination Event, fill in the beginning data placement position.</p> <p>To count bytes – For both fixed and delimited Events, count the offset from byte 0, the beginning of the selected Event element.</p>
Length	<p>Defaults to entire Event element selected.</p> <p>Source – For the source Event, fill in the length, in bytes, of the input Event data on which to operate.</p> <p>Destination – For the destination Event, fill in the length, in bytes, of the output data.</p> <p>To count bytes – For fixed Events, count the number of bytes from 1. For delimited Events, where field length is variable, leave the (to end) button selected to set the length to the end of the Event element.</p>

Rules for specifying byte locations in Event elements

These rules are:

- Count the byte offset (the beginning byte position, counted from the first byte of the Event element) starting from **0**; therefore, the second byte in an Event element has a byte offset of **1**.
- If you want to limit the byte location to a certain number of bytes, fill in a byte length. Count bytes starting from **1**. To calculate the targeted length, subtract the beginning byte from the ending byte. For example, to target bytes 2 through 10, subtract 2 from 10 to get a length of 8.

Figure 164 shows how you would target bytes 2 through 10 of an Event element in any of the rule dialog boxes.

Figure 164 Specifying a Byte Location



Using the Function Library to Define Rules

The Collaboration Rules Editor provides a library of functions you can use in your Collaboration Rules to perform special operations. Each function is a prewritten formula that takes a user-specified value or values, performs an operation, and returns a value or values.

To use the function library in a Collaboration Rules component

- 1 From the Collaboration Rules Editor, on **Options** menu, clear the **Use Selected Nodes in New Rule** option.

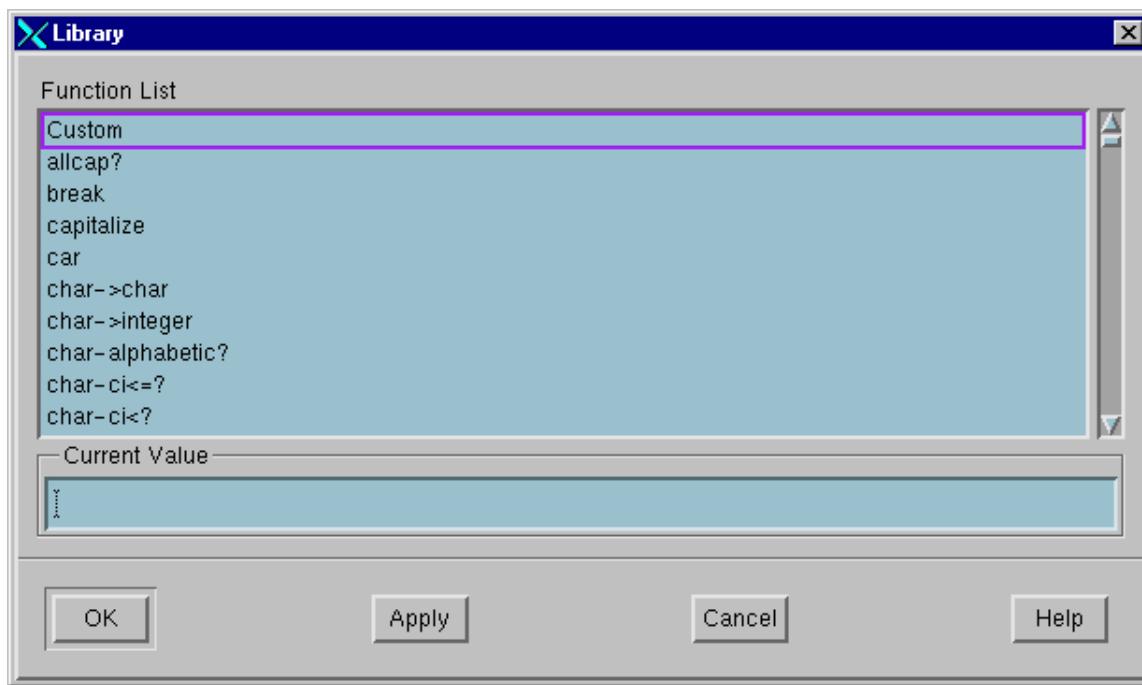
This action prevents nodes in either your **Source** or **Destination** structure from being referenced in the Collaboration Rules you add next. You can select node(s) to include in the rule later in this procedure.

- 2 Add the Collaboration Rules in which you want to use a function to the Rules pane. Click on the location in the Rules List where you want to insert the function.

- 3 On the toolbar or **View** menu, click  **Function Library**.

The **Library** dialog box appears. This feature allows you to select a function to insert into the current rule at the cursor position. See [Figure 165 on page 378](#).

Figure 165 Library Dialog Box

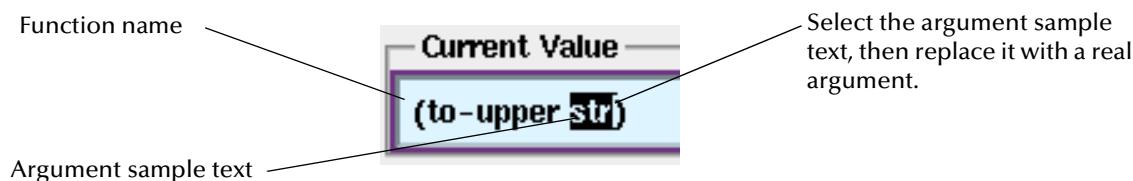


Note: You can drag and drop ETD nodes into arguments in a **Function** rule.

- 4 In the **Function List**, select the function you want to use.

The function and its arguments are shown in the list of functions in the **Current Value** text box; see Figure 166. You must define your value(s) for the function here.

Figure 166 Current Value Text Box



- 5 Select the argument sample text.
 - 6 Type or drag and drop your argument(s). When specifying a ETD node as an argument, click and hold the middle mouse button on the node, drag the mouse cursor to the selected argument text, and release the button.
- The argument text is replaced with a reference to the selected node. Be sure the argument sample text is completely replaced by a real argument. If there is more than one argument, you must separate each argument with a space.
- 7 Click **Apply** to add the function to the Rules List location you selected in step 2.
 - 8 Click **Cancel** to close the **Functions** dialog box.

Note: To include selected nodes in other rules: On **Options** menu, turn on **Use Selected Nodes in New Rule**.

Example

You need to copy a field from the input Event to your output Event, and in the process you want to convert the input data to uppercase letters. Here is how you would do it, using the Function library:

- 1 Add the **Copy** rule, making sure that the **Use Selected Nodes in New Rule** option is turned off.
- 2 In the Copy rulebar, click inside the box containing the text <Source Location>.
- 3 On the toolbar, click  **Library**.
- 4 In the **Function** dialog box, select the function called **to-upper**.
- 5 In the **Current Value** text box, select the argument text, **str**.
- 6 Click and hold the middle mouse button on the field node in the **Source** ETD; drag the mouse cursor to the selected **str** text in the **Current Value** text box. Release the mouse button over the **str** text.

The **str** text is replaced with a reference to the input Event field node.

- 7 Click **OK** to place the function inside the <Source Location> text box and dismiss the **Function** dialog box.
- 8 Drag and drop the destination node to the Copy rule's second text box, which currently contains the text, <Destination Location>.

For more information on Monk functions, see the *Monk Developer's Reference*.

8.4 Basic Collaboration Rules Operations

This section explains basic operations, for example opening, saving, and validating, that you do with Collaboration Rules files.

This section explains:

- “Opening a Collaboration Rules Component” on page 380
- “Saving a Collaboration Rules Component to a New Name” on page 380
- “Entering Comments About Collaboration Rules” on page 381
- “Changing Collaboration Rules Scripts” on page 381
- “Validating Collaboration Rules” on page 382
- “Finding Nodes” on page 383
- “Finding Nodes” on page 383

8.4.1 Opening a Collaboration Rules Component

Remember that you can only have one Collaboration Rules component open at a time.

Caution: If you try to open a file with more than 6500 lines in the Collaboration Rules Editor, you get an error message, and the file does not open. If you need to open a larger file, use a text editor or word processor capable of handling large files.

To open an existing Collaboration Rules file

- 1 On the toolbar or **File** menu, click  **Open**.

If you currently have a Collaboration open and have not saved that file, the system warns you, and asks whether you want to save your current file, proceed without saving, or cancel. Unless you cancel, the **Open Collaboration Rules** dialog box appears.

By default, the **Filter** is set to look for files in your current schema's Collaboration Rules subdirectory:

`<schema-name>\monk_scripts\common\ *.tsc`

The Filter lists all files with the file extension **.tsc** because the default naming convention for Collaboration Rules is a file name followed by the extension **.tsc**.

- 2 Select a file to open and click **OK**.

The Collaboration Rules file appears in the Collaboration Rules Editor window.

8.4.2 Saving a Collaboration Rules Component to a New Name

Use this procedure to save your Collaboration Rules to a different file name than the file name last saved to.

To save a Collaboration Rules component to a new name

- 1 Before you begin: On the **File** menu, click **Validate** to make sure that there are no errors in your Collaboration Rules. For details, see “[Validating Collaboration Rules](#)” on page 382.
- 2 On the **File** menu, click **Save As** to open the **Save Collaboration As** dialog box. The **Filter** is set to `<schema-name>\monk_scripts\common\ *.tsc`, by default. This setting means that your file is saved to your current schema. Do not change the Filter unless you need to save a Collaboration outside your schema.
- 3 Notice that in the **Selection** text box, your cursor is placed before the default file extension, **.tsc**. Type a file name without changing the position of the cursor.

Caution: Although you can type over the default **.tsc** file extension, you are not able to include the Collaboration in an Event route if you use a different extension.

- 4 Click **OK** to save the file.

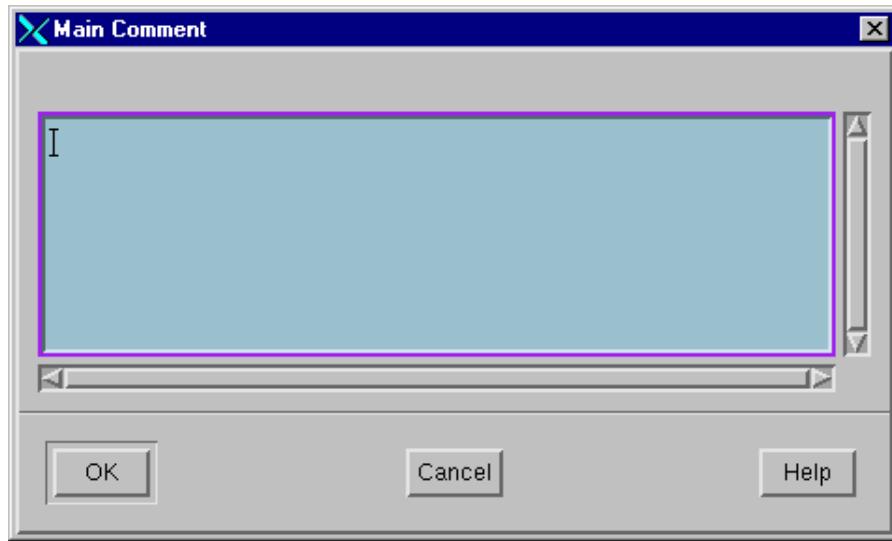
If you changed the default file extension and clicked **OK**, the system warns you that the Collaboration cannot be included in an Event route. At this point:

- ♦ You can append your file extension to the default file extension, **.tsc**. This allows the Collaboration to be included in an Event route.
- ♦ Or, you can keep your file extension. This means that you are not able to include the Collaboration in an Event route.

8.4.3 Entering Comments About Collaboration Rules

This feature allows you to enter comments in your Collaboration Rules. See Figure 167.

Figure 167 Main Comment Dialog Box



To use the Main Comment dialog box

- Enter a general comment describing the contents of this file. The comment is only for your reference, and you can use any characters you like.
- Click **OK**.

8.4.4 Changing Collaboration Rules Scripts

You can use the following methods to make changes to Collaboration Rules scripts:

- Deleting rules
- Changing rule parameters
- Changing source and destination ETDs

Deleting Rules

- 1 In the Rules pane, select the rule you want to delete.
- 2 On the toolbar or **Edit** menu, click **Delete Rule**.

The current rule is removed from the Collaboration Rules component.

Changing Rule Parameters

The ways to change rule parameters are shown in Table 57.

Table 57 Changing Rule Parameters

This method...	... applies to these rules:
Clicking the rule's name button in the Rule bar to access its parameter dialog box and filling in changes.	Insert, Copy, Data Map, Map, Change Pattern, Timestamp, and Unique ID
Filling in changes in the Rule bar's text boxes using Monk syntax.	All rules; for more information about using Monk syntax, see the <i>Monk Developer's Reference</i>
Dragging and dropping new Source or Destination ETD elements on a rule.	All rules

When you are finished, on the toolbar or **File** menu, click  **Save**.

Note: If a rule has parameter dialog boxes, click on the rule name button to display them. You can also fill in rule parameters in the Rule bar text boxes, using Monk syntax.

Changing Source/Destination ETDs

After you have created and defined a Collaboration, you can change the source and destination ETDs at any time.

To change the Source or Destination ETD in a Collaboration Rules component

- 1 To the right of either the **Source** or **Destination** ETD boxes, click .
- A file selection dialog box appears, listing all available input or output ETDs.
- 2 Select an ETD from the list and click **OK**.
- The new ETD appears in the Collaboration Rules Editor window.

Note: When you change the ETD used in your Collaboration Rules component, you see a warning asking you to confirm your action before the new structure appears in the window. Keep in mind that if you change the ETD in a Collaboration, you must also make changes to rule parameters so that they are applied to the correct Event elements.

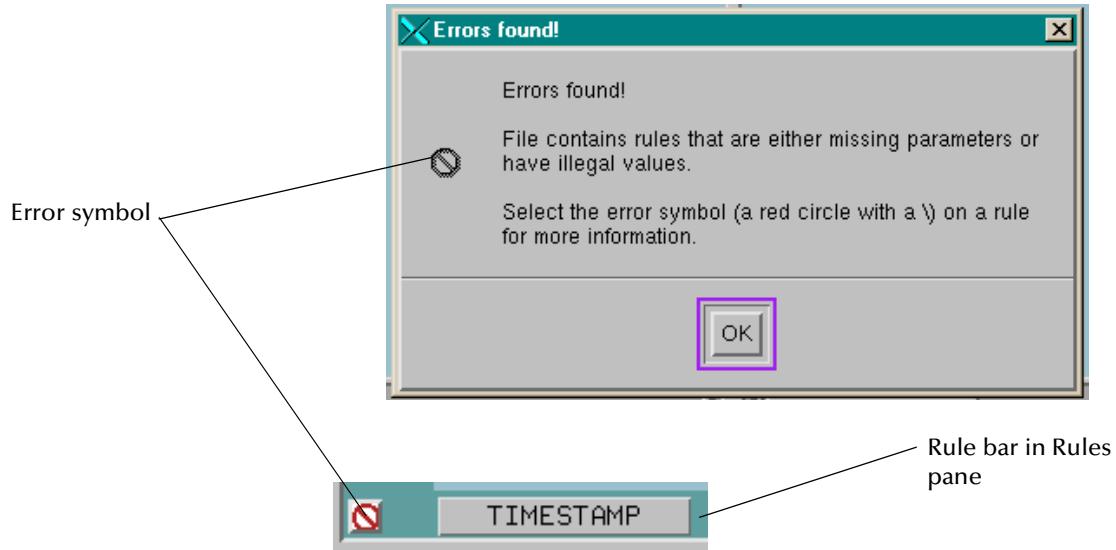
8.4.5 Validating Collaboration Rules

When you are done with a Collaboration Rules component, on the toolbar or **File** menu, click  **Validate** to verify that all Collaboration Rules have been completely and correctly entered.

If you have any Collaboration Rules rule that are not completely filled out or contain incorrect values, the system displays an error message to warn you. Check the Rules

pane and look for the rules that contain an error symbol in their Rule bars. Click the error symbol to display a list of the errors in that rule. For an example, see Figure 168.

Figure 168 Error Message with Symbol



Once you have corrected all errors, on the toolbar or **File** menu, click **Save**.

Note: Be sure to validate every Collaboration Rules file before adding it to the system's run-time schema, to prevent processing errors.

8.4.6 Finding Nodes

Use the **Find Node** feature to search through either your source or destination ETD and find a particular node.

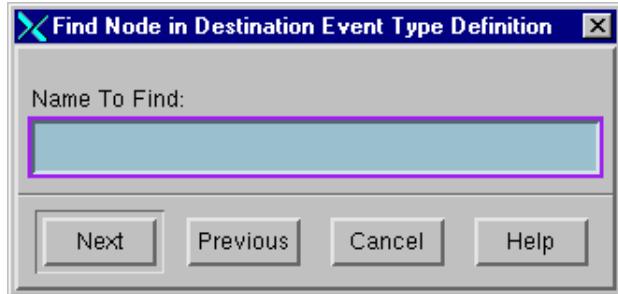
To find the desired node

- 1 To start at the beginning of the ETD, select the root node.
- 2 On the **Edit** menu, click **Find Node** and then take one of the following steps:
 - ♦ To search for a node in your **Source** ETD, choose **In Source ETD**.
 - ♦ To search for a node in your **Destination** ETD, choose **In Destination ETD**.
- 3 Enter the node name in the appropriate dialog box. You must enter the complete node name; the Collaboration Rules Editor cannot find a node if you only enter a partial name.

Find Node in Destination Event Type Definition Dialog Box

This feature allows you to find the named node in the destination ETD. See Figure 169.

Figure 169 Find Node in Destination Event Type Definition Dialog Box



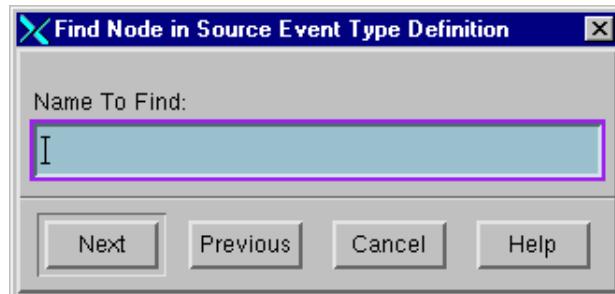
To use the Find Node in Destination Event Type Definition dialog box

- In the **Name to Find** box, enter the name of a node in the destination ETD.
- Click **Next** to search for the next occurrence of the node later in the ETD, or **Previous** to search for the next node earlier in the ETD.

To use the Find Node in Source Event Type Definition dialog box

This feature allows you to find the named node in the source ETD. See Figure 170.

Figure 170 Find Node in Source Event Type Definition Dialog Box



To use the Find Node in Source Event Type Definition dialog box

- In the **Name to Find** box, enter the name of a node in the source ETD.
- Click **Next** to search for the next occurrence of the node later in the ETD, or **Previous** to search for the next node earlier in the ETD.

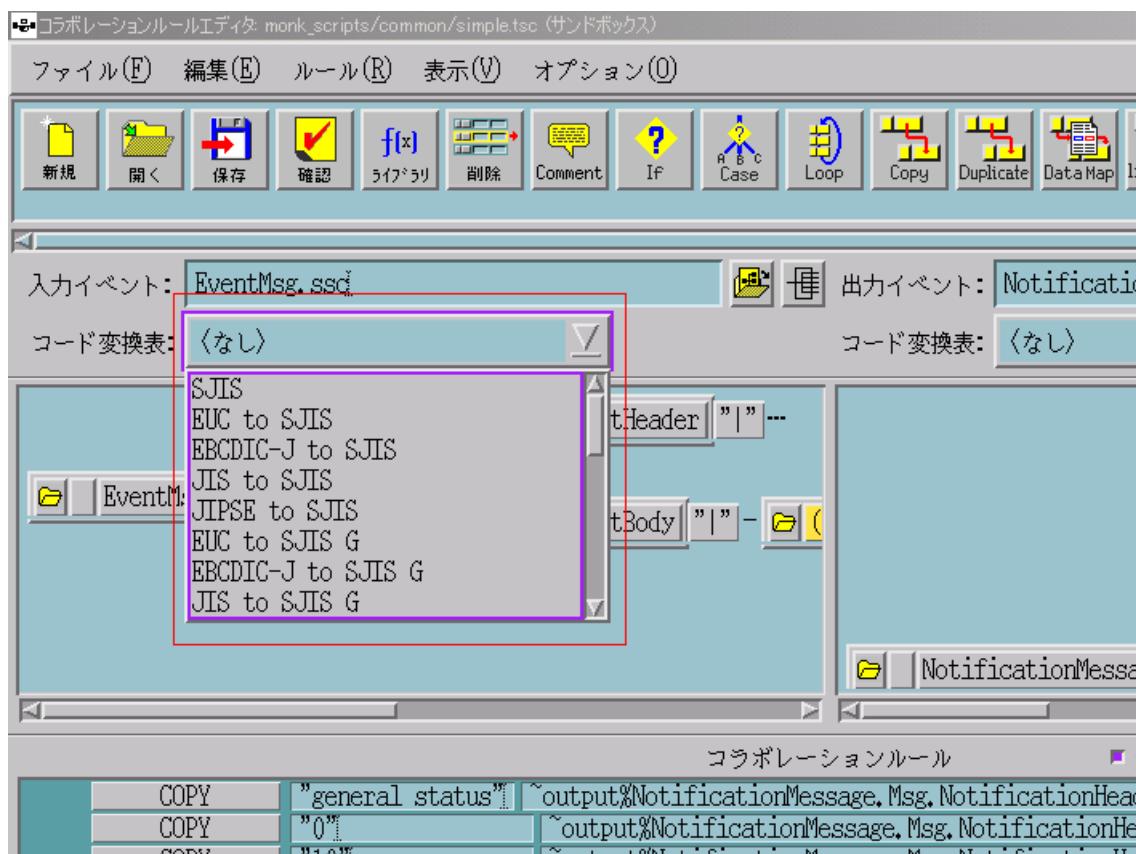
8.4.7 Converting to and from Double-Byte Character Encodings

Special considerations apply when Monk Collaborations run in Japanese or Korean operating systems, whose native character encoding is not single-byte. For e*Gate to parse Events and apply string functions correctly, the encoding method for inbound and outbound Events must match the native code. To support encoding methods other than the native code, use a code converter to convert the inbound or outbound Event to the appropriate native code. See Table 58 and [Figure 171 on page 385](#).

Table 58 Monk Functions for Code Conversion

To do this operation, use these functions:	For more information, see:
Query, set, or match a character encoding method	char-type char-type! char-type?	Monk Developer's Reference, Chapter 5 ("Character Expressions")
Query, set, or match a string encoding method	string-type string-type! string-type?	Monk Developer's Reference, Chapter 6 ("String Expressions")
Set the encoding method for an entire file	set-file-encoding-method	Monk Developer's Reference, Chapter 12 ("File I/O Expressions")
Convert EUC, JIS, or shift-JIS to or from another encoding method	ebcdic2sjis, ebcidic2sjis_g, jef2sjis*, jipse2sjis*, jis2*, sjis2*, utf82sjis*	Monk Developer's Reference, Chapter 21 ("International Conversion Functions")
Convert to or from gaiji	init-gaiji, init-utf8gaiji, set-gaiji-table, set-utf8gaiji-table,	
Convert UHC to or from another encoding method	ebcdic2uhc, ebcidic2uhc_g, uhc2*, utf82uhc	

Figure 171 Accessing the Code Conversion Functions



8.5 Using Collaboration Rules

When you create Collaboration Rules, the e*Gate system uses the rules you specify to create output Events. All the rules are accessible from the **Rules** menu (see “[Rules Menu” on page 359\). You can also quickly access the most frequently used rules with your mouse using the toolbar \(see “\[Toolbar Buttons” on page 355\\).\]\(#\)](#)

In addition to a quick-reference table, this section provides detailed explanations of all the Monk Collaboration Rules and how to use them as the building blocks for your Collaboration Rules scripts.

8.5.1 Collaboration Rules Reference Table

Table 59 provides a quick reference for brief explanations of all the Collaboration Rules. The column on the left tells you where in this section you can go to find a detailed explanation of the given rule.

Table 59 Monk Collaboration Rules

Rule name	Description	For more information, see:
If	If a defined condition is true, performs a block of rules on a particular Event element. If the defined condition is false, can perform an alternate block of rules.	“Using the If Rule” on page 387
Loop	Initiates a set of rules for a particular Event element, which are performed repeatedly, either a fixed number of times or until some condition is true or false.	“Using the Loop Rule” on page 396
Case	Allows you to select a list of rules for execution based on the value of a test expression.	“Using the Case Rule” on page 406
Comment	Inserts a comment into the Collaboration Rules component; used for the Collaboration developer’s reference only.	“Using the Comment Rule” on page 410
Copy	Copies all or part of the input Event to the output Event.	“Using the Copy Rule” on page 411
Display	Allows you to print whatever information you include in the rule to a destination of your choice.	“Using the Display Rule” on page 416
Duplicate	Copies all or part of the input Event to the output Event; replaces input Event delimiters with output Event delimiters.	“Using the Duplicate Rule” on page 417
Data Map	Matches a string in the input Event with a string stored in an ASCII text file. The mapping data associated with the matching string is inserted in the output Event.	“Using the Data Map Rule” on page 420
List Lookup	Inserts an output Event associated with a matching input Event string.	“Using the List Lookup Rule” on page 424

Table 59 Monk Collaboration Rules (Continued)

Rule name	Description	For more information, see:
Change Pattern	Specifies an output string to replace an input string or regular expression.	"Using the Change Pattern Rule" on page 429
Timestamp	Inserts the current date and time (of the system's host) into the output Event.	"Using the Timestamp Rule" on page 432
Unique ID	Inserts the current date and time (of the system's host) into the output Event. Differs from Timestamp in that it outputs milliseconds; timestamp does not. Unique ID is output in only one format.	"Using the Unique ID Rule" on page 435
Let	Allows you to define a variable and associated condition that can be used to control Collaboration Rules.	"Using the Let Rule" on page 437
Set!	Allows you to change the value of a variable that has been defined in a Let rule.	"Using the Set! Rule" on page 442
Function	Allows you to choose a preformatted Monk function to use in a Collaboration, or to develop your own function.	"Using the Function Rule" on page 443
User Function	Allows you to define your own Monk functions. In effect, you create a named subroutine composed of one or more calls to existing functions.	"Using the User Function Rule" on page 446
Set Regex	Allows you to define a variable whose value is a regular expression.	"Using the Set Regex Rule" on page 446

The rest of this section explains in detail each of these Collaboration Rules listed in the previous table.

Note: For more information on any rule construction, see the *Monk Developer's Reference*.

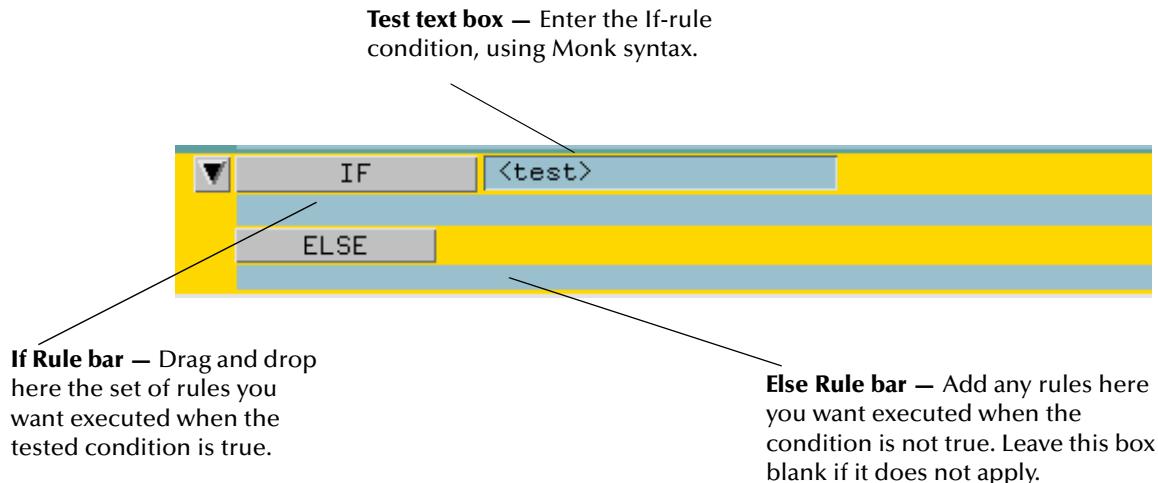
8.5.2 Using the If Rule

The **If** rule allows you to define a set of operations that can be executed only if a defined condition is tested and proved true.

Like the **Loop**, **Let**, **Set!**, and **Function** rules, the **If** rule requires you to set up its special syntax directly in the Rule bar text boxes. See [Figure 172 on page 388](#) for an example.

Note: Before typing, make sure you delete the sample text already provided in the pane.

Figure 172 If-rule Setup in the Rules Pane



To define an If rule

- 1 On the toolbar or **Rules** menu, click **Add If**.
The **If** rule appears in the Rules pane.
- 2 Click inside the **Test** text box, and then select and delete the **<test>** text.
- 3 Define the test condition in the **Test** text box, using the following Monk syntax:
`(test)`
where **test** is any legal Monk expression.
See [Table 60 on page 389](#) for a list of ways you can set up the **If**-rule tests. This table directs you to detailed instructions later in this section on how to set up the desired **If**-rule test.
- 4 Click **If** in the Rules pane to activate the **If Rule bar**.
- 5 In the **If Rule bar**, add rules to be executed if the test condition is true. For an explanation of how to add rules, see "[Adding and Arranging Rules](#)" on page 366.
- 6 Click **Else** to activate the **Else Rule bar**.
When activated, this Rule bar turns dark blue. If the previous test condition is false, the current **If** rule executes any **Else** rules added here.
- 7 Add the rule(s) to be executed in the **Else Rule bar** (see "[Adding and Arranging Rules](#)" on page 366).
- 8 When you are finished, on the toolbar or **File** menu, click **Save**.

If-rule Test Setup Table

[Table 60 on page 389](#) explains basic setup procedures for **If**-rule tests.

Table 60 If-rule Test Setups

Test Application	Condition of If-rule Operation	Reference for More Information
Comparing an Event element to a regular expression	The contents of an input Event element match a specified regular expression.	See “ Comparing an Event Element to a Regular Expression ” on page 390.
Testing for a false condition	An input Event element tests false for a conditional expression.	See “ Testing for a False Condition ” on page 394.
Performing multiple tests with an If rule	Multiple tests on an input Event element are all proven true.	See “ Performing Multiple Tests with an If Rule ” on page 394.
Performing alternate tests with an If rule	Alternate tests on an input Event element are performed and at least one proven true.	See “ Performing Alternate Tests with an If Rule ” on page 395.
Naming If conditions	A defined condition is proven true.	See “ Naming a Condition Using the Let Rule ” on page 439.
Comparing an Event element to a number using an operator:		
<	The contents of an input Event element are less than a user-specified number.	See “ Comparing an Event Element to a Number Using < ” on page 390.
<=	The contents of an input Event element are less than or equal to a user-specified number.	See “ Comparing an Event Element to a Number Using <= ” on page 391.
>	The contents of an input Event element are greater than a user-specified number.	See “ Comparing an Event Element to a Number Using > ” on page 392.
>=	The contents of an input Event element are greater than or equal to a user-specified number.	See “ Comparing an Event Element to a Number Using >= ” on page 392.
=	The contents of an input Event element are equal to a user-specified number.	See “ Comparing an Event Element to a Number Using = ” on page 393.
not =	The contents of an input Event element are not equal to a user-specified number.	See “ Comparing an Event Element to a Number Using not = ” on page 393.

Path Location Syntax

When you drag and drop a desired Event element into the **Test** text box, the Collaboration Rules Editor automatically fills in the correct path location for the input Event element you placed in the text box. In syntax expressions, the term **source_path** represents this variable.

Caution: Make sure there is always an equal number of open and closed parentheses in your If-rule test expression; otherwise, the expression creates an error.

Repeating Events — If you select a repeating Event element for *source_path*, the Select Repetition Instance dialog box appears. See “[Defining Instances of Repeating Event Elements](#)” on page 371 for details about how to use this dialog box.

Accessing Lower-level Event Elements — If you want to add information to any path expression, which allows the current Collaboration Rules to access lower-level Event elements missing from the current ETD, see “[Defining ETD Paths](#)” on page 369.

The rest of this section explains how to construct the If-rule tests listed in [Table 60 on page 389](#).

Comparing an Event Element to a Regular Expression

You can define an If rule so that if the input Event matches a specified regular expression, the If-rule operations are executed.

Syntax

Use the following syntax in the Test text box:

```
(regex reg_exp source_path)
```

The variables in this expression you need to define are:

reg_exp

The regular expression you want matched to the selected input Event element. See the *Monk Developer’s Reference* for details on how to write a regular expression. Be sure to enclose the regular expression in double-quotation marks.

source_path

The path location of the Event element you want tested.

To construct this If-rule test

- 1 In the Rules pane, type **(regex** at the beginning of the **Test** text box.
- 2 Next to your first entry, type the appropriate regular expression enclosed by double-quotation marks.
- 3 Next to the regular expression, drag and drop the input Event element that matches that expression.
- 4 Type a parenthesis **()** at the end of the rule (after the Event/path name).

See [Figure 172 on page 388](#) for an example.

Comparing an Event Element to a Number Using <

You can define an If rule so that when the contents of a selected Event element are less than a number you specify, the If-rule operations are executed.

Syntax

Use the following syntax in the **Test** text box:

```
(< (string->number source_path) n)
```

The variables in this expression you need to define are:

source_path

The path location of the Event element you want tested.

n

The number you want compared to the selected input Event element. When the contents of a selected Event element are less than this number, the If-rule test is true.

To construct this If-rule test

- 1 In the Rules pane, type **(< (string->number** at the beginning of the **Test** text box.
- 2 Next to your first entry, drag and drop the input Event element that is compared to the specified number **n**.
- 3 Type **) n)** at the end of the rule (after the Event/path name), where **n** is your specified number.

See [Figure 172 on page 388](#) for an example.

Comparing an Event Element to a Number Using **<=**

You can define an If rule so that when the contents of a selected Event element are less than or equal to a number you specify, the If-rule operations are executed.

Syntax

Use the following syntax in the **Test** text box:

```
(<= (string->number source_path) n)
```

The variables in this expression that you need to define are:

source_path

The path location of the Event element you want tested.

n

The number you want compared to the selected input Event element. When the selected Event element contents are less than or equal to this number, the If-rule test is true.

To construct this If-rule test

- 1 In the Rules pane, type **(<= (string->number** at the beginning of the **Test** text box.
- 2 Next to your first entry, drag and drop the input Event element that is compared to the specified number **n**.
- 3 Type **) n)** at the end of the rule (after the Event/path name), where **n** is your specified number.

See [Figure 172 on page 388](#) for an example.

Comparing an Event Element to a Number Using >

You can define an **If** rule so that when the contents of a selected Event element are greater than a number you specify, the If-rule operations are executed.

Syntax

Use the following syntax in the **Test** text box:

```
(> (string->number source_path) n)
```

The variables in this expression that you need to define are:

source_path

The path location of the Event element you want tested.

n

The number you want compared to the selected input Event element. When the selected Event element content is greater than this number, the **If**-rule test is true.

To construct this If-rule test

- 1 In the Rules pane, type **(> (string->number** at the beginning of the **Test** text box.
- 2 Next to your first entry, drag and drop the input Event element that is compared to the specified number **n**.
- 3 Type **) n)** at the end of the rule (after the Event/path name), where **n** is your specified number.

See [Figure 172 on page 388](#) for a diagram.

Comparing an Event Element to a Number Using >=

You can define an **If** rule so that when the contents of a selected Event element are greater than or equal to a number you specify, the If-rule operations are executed.

Syntax

Use the following syntax in the **Test** text box:

```
(>= (string->number source_path) n)
```

The variables in this expression that you need to define are:

source_path

The path location of the Event element you want tested.

n

The number you want compared to the selected input Event element. When the contents of the selected Event element are greater than or equal to this number, the **If**-rule test is true.

To construct this If-rule test

- 1 In the Rules pane, type **(>= (string->number** at the beginning of the **Test** text box.

- 2 Next to your first entry, drag and drop the input Event element that is compared to the specified number **n**.
- 3 Type **) n**) at the end of the rule (after the Event/path name), where **n** is your specified number.

See [Figure 172 on page 388](#) for a diagram.

Comparing an Event Element to a Number Using =

You can define an If rule so that when the contents of a selected Event element are equal to a number you specify, the If-rule operations are executed.

Syntax

Use the following syntax in the **Test** text box:

```
(= (string->number source_path) n)
```

The variables in this expression that you need to define are:

source_path

The path location of the Event element you want tested.

n

Fill in the number you want compared to the selected input Event element. When the contents of the selected Event element are equal to this number, the If rule test is true.

To construct this If-rule test

- 1 In the Rules pane, type **(= (string->number** at the beginning of the **Test** text box.
- 2 Next to your first entry, drag and drop the input Event element that is compared to the specified number **n**.
- 3 Type **) n**) at the end of the rule (after the Event/path name), where **n** is your specified number.

See [Figure 172 on page 388](#) for a diagram.

Comparing an Event Element to a Number Using not =

You can define an If rule so that when the contents of a selected Event element are not equal to a number you specify, the If-rule operations are executed.

Syntax

Use the following syntax in the **Test** text box:

```
(not (= (string->number source_path) n))
```

The variables in this expression that you need to define are:

source_path

The path location of the Event element you want tested.

n

Fill in the number you want compared to the selected input Event element. When the contents of a selected Event element are not equal to this number, the If rule test is true.

To construct this If-rule test

- 1 In the Rules pane, type **(not (= (string->number** at the beginning of the **Test** text box.
- 2 Next to your first entry, drag and drop the input Event element that is compared to the specified number **n**.
- 3 Type **) n))** at the end of the rule (after the Event/path name), where **n** is your specified number.

See [Figure 172 on page 388](#) for an example.

Testing for a False Condition

You can define an If rule so that when the contents of a selected input Event element test false for a conditional expression, the If-rule operations are executed.

Syntax

Use the following syntax in the **Test** text box:

(not test)

The variable in this expression that you need to define is:

test

Any legal Monk expression that you want to test the input Event element against. For example, you might want to test for a particular string or regular expression. See the *Monk Developer's Reference* for details on how to write a regular expression. Be sure to enclose the expression in double-quotation marks.

To construct this If-rule test

- 1 In the Rules pane, type **(not** at the beginning of the **Test** text box.
- 2 Next to your first entry, type the Monk expression that defines the condition you want to test. Be sure to enclose the expression in double-quotation marks.
- 3 Type a parenthesis **()** at the end of the rule (after the expression).

See [Figure 172 on page 388](#) for an example.

Performing Multiple Tests with an If Rule

You can define an If rule so that multiple tests are performed on the selected input Event element and proved true before the If-rule operations are executed.

Syntax

Use the following syntax in the **Test** text box:

(and test1 test2 testN)

The variables in this expression that you need to define are:

test1 test2

Fill in multiple legal Monk expressions that you want to test the input Event element for. For example, you might want to test for particular strings or regular expressions. See *Monk Developer's Reference* for details on how to write a regular expression. Be sure to enclose each expression in double-quotation marks.

To construct this If-rule test

- 1 In the Rules pane, type **(and** at the beginning of the **Test** text box.
- 2 Next to your first entry, type one or more Monk expressions that define the condition you want to test. Be sure to enclose each expression in double-quotation marks.
- 3 Type a parenthesis **()** at the end of the rule (after the last expression).

See [Figure 172 on page 388](#) for an example.

Performing Alternate Tests with an If Rule

You can define an **If** rule so that alternate tests are performed on the selected input Event element and all proved true before the **If**-rule operations are executed.

Syntax

Use the following syntax in the **Test** text box:

(or test1 test2 testN)

The variables in this expression that you need to define are:

test1 test2

Fill in multiple legal Monk expressions that you want to test the input Event element for. For example, you might want to test for particular strings or regular expressions. See *Monk Developer's Reference* for details on how to write a regular expression. Be sure to enclose each expression in double-quotation marks.

To construct this If-rule test

- 1 In the Rules pane, type **(or** at the beginning of the **Test** text box.
- 2 Next to your first entry, type one or more Monk expressions that define the condition you want to test. Be sure to enclose each expression in double-quotation marks.
- 3 Type a parenthesis **()** at the end of the rule (after the last expression).

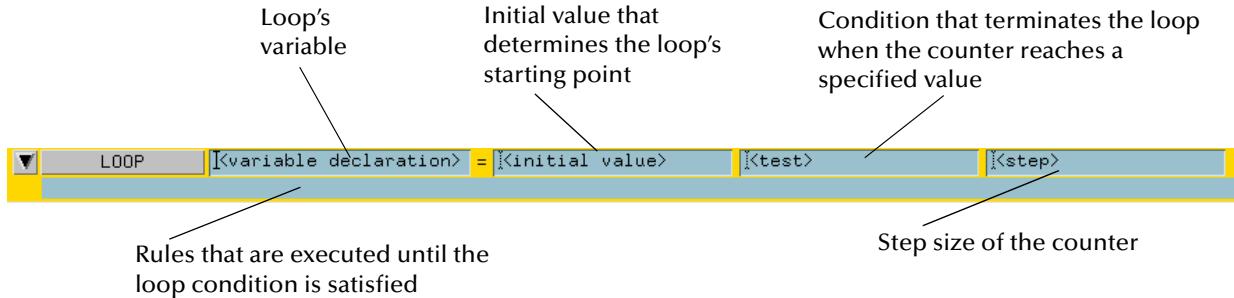
See [Figure 172 on page 388](#) for an example.

Type everything else as shown in the expression above. See the following graphic for an example of how to construct this expression.

8.5.3 Using the Loop Rule

The **Loop** rule allows you to perform Collaboration Rules repeatedly on selected Event elements, such as a set of repeating segments or fields. Figure 173 shows the **Loop** rule's elements.

Figure 173 Loop Rule Structure



This section explains:

- [“Loop Rule Overview” on page 396](#)
- [“Creating a Loop Rule” on page 397](#)
- [“Executing a Loop Rule” on page 399](#)
- [“Defining ETD Paths in a Loop Rule” on page 399](#)
- [“Looping on a Computed Range of Event Elements” on page 401](#)
- [“Looping on a Fixed Range of Event Elements” on page 404](#)

Loop Rule Overview

When you define a loop, you are saying to the Collaboration Rules Editor, “Do this rule to this Event element. Now, loop back and do it again to the next Event element.” Using the Loop rule, you can perform Collaboration operations on all instances of a repeating element.

Table 61 shows the main elements that make up a **Loop** rule.

Table 61 Loop Rule Elements

Loop Rule Parameter	Description	Default Value
<variable declaration>	Alphanumeric symbol to represent the current count of iterations performed on the specified set of Event elements. The system compares the current value of this variable to the test condition to determine when to stop looping.	i
<initial value>	Initial value of the variable. This is your starting count for the Loop rule iterations. The default, 0, tells the system to start looping on the first instance of the repeating Event element.	0
<test>	Termination condition for the Loop rule, which, when true, causes the Loop rule to stop. When the system processes the default test, it keeps track of how many repeating Event elements it has found, and compares that count at the beginning of each loop iteration to the value of i. When i is greater than or equal to the Event element count, the loop terminates.	($\geq i$ (<i>count path</i>)) where <i>path</i> is the path expression for the node you selected in step 1. See “ Defining ETD Paths in a Loop Rule ” on page 399 for more information about path expressions.
<step>	The value by which the Loop rule variable is updated on each iteration. The default step value, (+ i 1), ensures that each instance of the repeating Event element is looped on. If you specify (+ i 2), for example, every other instance is looped on.	(+ i 1) where i is the variable name, and 1 is the step value
Collaboration Rules	Collaboration operations that the loop performs on each instance of a repeating Event element, until the loop’s termination condition is met.	N/A

The **Loop** rule requires a special syntax to set it up. Like the **If**, **Let**, **Set!**, and **Function** rules, the **Loop** rule requires you to set up its special syntax directly in the Rule bar text boxes.

Creating a Loop Rule

Follow this procedure to create a **Loop** rule that loops on all instances of a repeating Event element.

To add and define a Loop rule

- In the **Source** ETD, select the node representing the repeating Event element you want to loop on.

Note: To loop on a group of elements that repeats (rather than a single element that repeats), select the group's set node before adding the **Loop** rule. A set node appears one level above grouped Event elements in the ETD.

- 2 On the toolbar or **Rules** menu, click  **Add Loop**.

A **Loop** rule is added to the Rules pane (see [Figure 173 on page 396](#)). Text boxes for the **Loop** rule parameters are already set up for you as follows:

- ♦ <variable declaration>
- ♦ <initial value>
- ♦ <test>
- ♦ <step>

- 3 Enter the appropriate information for each parameter. For details see [Table 61 on page 397](#).

- 4 Click **Loop** inside the Rule bar to activate the **Loop** rule.

When activated, the box at the bottom of the **Loop** rule turns dark blue. This is where you add the rules to be executed repeatedly.

- 5 Add rules to the **Loop** rule. Set up one or more rules in the blue bar, in the way as you would set up the rules in any other context.

Collaboration Rules that you add to the **Loop** rule appear inside and at the bottom of the **Loop** rule.

- 6 When the **Select Repetition Instance** dialog box appears, type the **Loop** rule variable **i** in the selected nodes' **Instance** fields (see ["Defining Instances of Repeating Event Elements" on page 371](#) for details on this dialog box).

- 7 Be sure that the path expression in each rule within the **Loop** includes the loop variable, for example [**<i>**], as shown in the following example:

~input% MSG.DTM [<i>]

To complete the **Loop** rule, you must include the loop variable name in the path expressions of the Event elements on which the Collaboration Rules are executed.

- 8 When you are finished, on the toolbar or **File** menu, click  **Save**.

For additional applications of the **Loop** rule, see the following sections:

- ["Using the Loop Rule" on page 396](#)
- ["Defining ETD Paths in a Loop Rule" on page 399](#)
- ["Looping on a Computed Range of Event Elements" on page 401](#)
- ["Looping on a Fixed Range of Event Elements" on page 404](#)

For more information about additional applications of the **Loop** rule, see the *Monk Developer's Reference*.

Executing a Loop Rule

The system executes a **Loop** rule as follows:

- 1 The loop variable is assigned its initial value.
- 2 The loop iterations begin:
 - A The loop test is evaluated; as long as the test condition is false, the loop continues.
 - B The Collaboration Rules within the loop are executed, in sequential order.
 - C The loop variable is incremented by the amount of the step size (the loop now goes back to step A).

Defining ETD Paths in a Loop Rule

When you define a **Loop** rule in a Collaboration Rule, you need to specify both input and output Event locations, that is, their path expressions. Path expressions tell the **Loop** rule where to perform its operations. For the **Loop** rule, you must define path expressions for repeating elements, such as an individual field that repeats.

Note: To see the basic syntax of a path expression and a description of its components, see [“Defining ETD Paths” on page 369](#).

You specify path expressions by dragging elements from the ETDs and dropping them onto the **Loop** Rule bar. When defining ETD paths in a **Loop** rule, you need to know how to:

- Select a repeating Event element (see [“Selecting a Repeating Event Element” on page 399](#)).
- Specify the instance(s) of the repeating Event element included in the loop (see [“Specifying the Instance\(s\) of a Repeating Event Element” on page 400](#)).
- Reference a particular byte location (optional; see [“Referencing Byte Location” on page 370](#)).

This section explains how to do each of these tasks.

Selecting a Repeating Event Element

You can use a **Loop** rule to perform a sequence of Collaboration Rules on a repeating Event element. To identify a repeating Event element in a ETD, look in its for one of the symbols shown in Table 62

Table 62 Node Repetition Symbols

Node Repetition Type	Node Symbol
Range of repetitions	<n-m> (where n and m are integers)
Occurs zero or more times (optional-repeating)	*
Occurs one or more times	+

To select a repeating Event element in either an input or output ETD, press and hold down the middle mouse button on the node, and begin dragging the mouse cursor to the desired position in the rule.

The **Select Repetition Instance** dialog box appears. This is where you specify the instance of the repeating node you want to loop on. Go to “[Specifying the Instance\(s\) of a Repeating Event Element](#)” on page 400 for information on what to do next.

Specifying the Instance(s) of a Repeating Event Element

Whenever you select a repeating Event element to include in a rule, the **Select Repetition Instance** dialog box appears. In a **Loop** rule, you need to specify an instance of a repeating Event element:

- When defining the loop condition, otherwise known as the loop’s goal. For the applications of the **Loop** rule discussed in this chapter, you will select an entire repeating group.
- When defining the loop operations. These are the Collaboration Rules that the loop performs over and over on selected Event elements until it reaches its goal. In this case, you will specify the instance as a variable. For details about how the variable works, see “[Including Variables in a Loop’s Operations](#)” on page 400.

To specify the instance of a repeating Event element in a **Loop** rule, fill in the information in the **Select Repetition** dialog box as provided in Table 63.

Table 63 Repetition Information for Loop Rule

Repeating Node To Include In	Select Repetition Instance Dialog Box Entries
The Loop condition	Click the All instances of the last node button below the node’s Instance field. Click OK to add the node’s path to the Loop rule.
Loop rule operations	Enter the Loop rule variable in the selected node’s Instance field (just type the variable; the Collaboration Rules Editor automatically fills in the additional syntax required). Click OK to add the node’s path to the Loop rule.

Note: If you selected a repeating Event element that belongs to one or more levels of repeating elements, you need to specify the instance(s) of the higher-level node(s) on which to perform the **Loop** rule. For example, if you are defining a loop on a repeating field that belongs to a repeating segment, you must specify the instance of the repeating segment before following the guidelines listed in the previous table for the repeating field.

For more details about the **Select Repetition Instance** dialog box, see “[Defining Instances of Repeating Event Elements](#)” on page 371.

Including Variables in a Loop’s Operations

As noted in the previous section, when you set up a loop’s Collaboration Rules, you need to include the loop variable in each Collaboration Rules’ path expression. This establishes the relationship between the current value of the variable (which is the loop’s counter) and the instance of the Event element that is being looped on.

To include a variable in a path expression, type the **Loop** rule variable in the **Select Repetition Instance** dialog box, which appears when you add a repeating Event element inside the **Loop** rule.

Example

In the **Select Repetition Instance** dialog box, MSG is ADM's nonrepeating parent node. You need to include a **Loop** rule variable in a repeating Event element's path expression. Type the variable in the element's **Instance** field, that is, give ADM an instance of **i**. Then give MSG an instance of **0**.

Note that the element's entire path, back to its root node, is shown in this dialog box. The variable then appears in the path expression in the Loop Rule bar:

```
~input%MSG.ADM[<i>]
```

Referencing Byte Count

For most of the Collaboration Rules, you can specify byte count inside the dialog boxes available from their Rule bars. However, the **Loop** rule requires you to specify a particular byte location directly in a path expression. You must specify a byte location after the final element in the expression. Use the following syntax in the **Loop** Rule bar:

```
finalpath:element:byte_offset, length
```

This syntax contains the following variables:

byte_offset

The beginning byte position, counted from the first byte of the Event location

Note: *The first byte is counted as 0.*

length

The number of bytes referenced, counted from one.

For example:

```
N1:2,10
```

This path expression references the N1 segment, starting from byte 3, and extending for a length of 10 bytes.

Looping on a Computed Range of Event Elements

You can define a **Loop** rule that performs a sequence of Collaboration Rules on a computed range of Event elements.

In this type of loop, the system computes for you the total number of elements in a set, which is the range's maximum value. You then tell the loop where to start in the Event element set (minimum).

To define this loop, you need to complete the following steps:

- 1 Define a **Let** rule.
- 2 Define a **Loop** rule.
- 3 Add the Collaboration Rules to be executed as part of the loop.

For more information on how to define the **Let** rule, see “[Using the Let Rule](#)” on [page 437](#).

Syntax

Let Rule — Use the following syntax in **Let** rule text boxes:

max

This is a variable that represents the total number of Event elements in the selected set. The **Let** rule first determines the value of this variable; the **Loop** rule then uses this value to determine when it has looped on the last Event element in the set.

value

Fill in the Monk **count** function here which will add up the number of Event elements in the set to be looped on.

Loop Rule — Use the following syntax in **Loop** rule text boxes:

variable name

Fill in the alphanumeric symbol that represents the current count of iterations the loop expression has performed on the specified set of Event elements. The system compares this variable to the test condition to determine when to stop the **Loop** rule.

min

Fill in an integer for the initial value of the variable. This is your starting count for the Loop rule iterations. For example, if you fill in **2**, then the loop starts with the third Event element in the set.

test

Fill in the termination condition for the Loop rule, which, when true, causes the **Loop** rule to stop. In this case, you will set up a test that allows the loop to process while **min** is less than or equal to **max** (the number of Event elements in the set).

step

Fill in the value by which the **Loop** rule variable is updated on each iteration. Use the following syntax: **(+i n)**, where **i** is the variable name and **n** is the step value.

To construct the Let rule

- 1 On the Rules menu, click **Add Let**.

The **Let** Rule bars appear in the Rules pane. To the right of the **Let** name button, the bar contains the words **Add Declaration** and **Remove Declaration** (see [Figure 191 on page 437](#)).

Below this first **Let** Rule bar is another Rule bar with two text boxes.

- 2 In the first text box type the variable:

max

- 3 In the second text box in the same Rule bar, type:

(count

- 4 Drag and drop into the same text box, the node representing the set of repeating segments or fields.
- 5 In the **Select Repetition Instance** dialog box, click **All Instances** for the node.
- 6 After the path expression, type:
 `)`

Now you need to nest the **Loop** rule in the **Let** rule. This action links the variable you set up with **Let** to the **Loop** rule operations.

Note: If you are setting up a **Loop** rule for a group of elements that repeats (rather than a single element that repeats), then you must drag and drop the group's set node into the **Let** rule's **Initial Value** text box.

To construct the Loop rule

- 1 Click **Let** inside the **Let Rule bar**.
- 2 On the **Rules** menu, click **Add Loop** to add a **Loop** rule inside the **Let** rule.
The **Let Rule bar** appears in the **Rules pane**. This bar has four text boxes to the right of the **Let** name button (see [Figure 173 on page 396](#)).
- 3 In the first text box **<variable declaration>**, type the following variable:
`i`
- 4 In the second text box **<initial value>**, type the initial value for the variable, that is, the appropriate integer.
- 5 In the third text box **<test>**, type:
`(<= max i)`
- 6 In the fourth text box **<step>**, type:
`(+i 1)`

Example

You want the following operation:

The variable, **i**, must be initialized to **2** and incremented by **1** for each iteration. The test condition must compare the count of **max** against the current value of **i**. While **i** is less than or equal to **max**, the Collaboration Rules in the **Loop** rule must be executed. The value for **max** was defined in the **Let** rule.

The **Loop Rule bar** text box entries for this example must be:

- **<variable declaration>** — **i**
- **<initial value>** — **2**
- **<test>** — `(<= max i)`
- **<step>** — `(+i 1)`

To define the Collaboration Rules

- 1 Select the **Loop** rule and add the Collaboration Rules to be executed.

To complete the **Loop** rule, you need to include the loop variable name in the path expressions of the Event elements on which the Collaboration Rules are executed.

- 2 Drag and drop the nodes, representing the input and output Event elements, into the appropriate text boxes in the Rule bar.
- 3 When the **Select Repetition Instance** dialog box appears, type the **Loop** rule variable **i** in the selected nodes' **Instance** text boxes.
- 4 When you are finished, on the toolbar or **File** menu, click  **Save**.

For details about the **Select Repetition Instance** dialog box, see “[Defining Instances of Repeating Event Elements](#)” on page 371.

See “[Defining ETD Paths in a Loop Rule](#)” on page 399 for more information about defining instances of repeating Event elements, including using variables in path expressions.

Looping on a Fixed Range of Event Elements

You can define a **Loop** rule that performs a sequence of Collaboration Rules on a fixed range of Event elements in a set.

In this type of **Loop** rule, the loop starts at a user-specified point in the Event element set (minimum value), and ends at a user-specified point in the Event element set (maximum value).

To define this loop, you need to complete the following steps:

- 1 Define a **Let** rule.
- 2 Define a **Loop** rule.
- 3 Add the Collaboration Rules to be executed as part of the loop.

For more information on how to define the **Let** rule, see “[Using the Let Rule](#)” on page 437.

Syntax

Let Rule — Use the following syntax in the **Let** rule text boxes:

min and max

These variables represent the loop's starting and ending points in the Event element set. The Loop rule uses **min** and **max** to determine on which Event element to start the loop, and on which element to stop the loop.

value

Fill in the value of **min** and **max**, as integers.

Note: You must set up two declarations (variables) in the **Let** rule.

Loop Rule — Use the following syntax in the **Loop** rule text boxes:

variable name

Fill in the alphanumeric symbol that represents the current count of iterations the loop expression has performed on the specified set of Event elements. The system compares this variable to the test condition to determine when to stop the **Loop** rule.

min

Fill in the variable representing the loop's starting point in the Event element set, as defined in the **Let** rule: **min**.

test

Fill in the termination condition for the **Loop** rule, which, when true, causes the **Loop** rule to stop. In this case, you set up a test that allows the loop to process while **min** is less than or equal to **max**.

step

Fill in the value by which the **Loop** rule variable is updated on each iteration. Use the following syntax: **(+i n)**, where **i** is the variable name, and **n** is the step value.

To construct the **Let** rule

- 1 On the **Rules** menu, click **Add Let**.

The **Let** Rule bars appear in the Rules pane. To the right of the **Let** name button, the bar contains the words **Add Declaration** and **Remove Declaration** (see [Figure 191 on page 437](#)).

Below this first **Let** Rule bar is another Rule bar with two text boxes.

- 2 In the first text box in the first Rule bar, type the variable:

min

- 3 In the second text box, type the value for **min** as an integer.

- 4 Click **Add Declaration** to add another declaration (variable).

An additional Rule bar with two text boxes appears.

- 5 In the first text box in the second Rule bar, type the variable:

max

- 6 In the second text box, type the value for **max** as an integer.

Once you construct the **Let** rule, you can place the **Loop** rule inside the **Let** rule. This action links the variables you set up in **Let** to the **Loop** rule operations.

To construct the **Loop** rule

- 1 Click **Let** inside the **Let** Rule bar.

- 2 On the **Rules** menu, click **Add Loop** to add a **Loop** rule inside the **Let** rule.

The **Let** Rule bar appears in the Rules pane. This bar has four text boxes to the right of the **Let** name button. See [Figure 173 on page 396](#).

- 3 In the first text box <variable declaration>, type the following variable:

i

4 In the second text box <initial value>, type the following variable:

min

5 In the third text box <test>, type:

(<= max i)

6 In the fourth text box <step>, type:

(+i 1)

Example

You want the following operation:

The variable, **i**, must be initialized to the value of **min** and incremented by **1** for each iteration. The test condition must compare the count of **max** against the current value of **i**. While **i** is less than or equal to **max**, the Collaboration Rules in the **Loop** rule must be executed. The values for **min** and **max** were defined in the **Let** rule.

The **Loop** Rule bar text box entries for this example must be:

- <variable declaration> — **i**
- <initial value> — **min**
- <test> — **(<= max i)**
- <step> — **(+i 1)**

To define the Collaboration Rules

1 Select the **Loop** rule and add the Collaboration Rules to be executed.

To complete the **Loop** rule, you need to include the loop variable name in the path expressions of the Event elements on which the Collaboration Rules are executed.

2 Drag and drop the nodes, representing the input and output Event elements, into the appropriate text boxes in the Rule bar.

3 When the **Select Repetition Instance** dialog box appears, type the **Loop** rule variable **i** in the selected nodes' **Instance** text boxes.

4 When you are finished, on the toolbar or **File** menu, click  **Save**.

For details about the **Select Repetition Instance** dialog box, see “[Defining Instances of Repeating Event Elements](#)” on page 371.

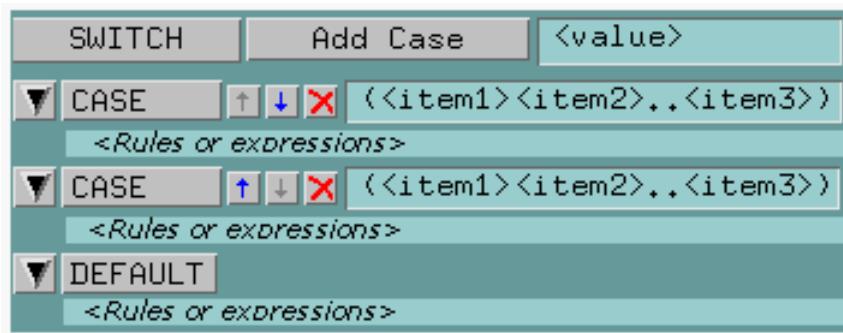
See “[Defining ETD Paths in a Loop Rule](#)” on page 399 for more information about defining instances of repeating Event elements, including using variables in path expressions.

8.5.4 Using the Case Rule

The **Case** rule allows you to select a list of rules for execution based on the value of a test expression. The **Case** rule is most often used in place of a sequence of **If** rules.

The example in Figure 174 shows the **Case** rule setup in the Rules pane.

Figure 174 Case Rule Setup



Syntax

The **Case** rule uses the following syntax:

<value>

A variable or Monk expression to be evaluated and can be a number, character, or symbol.

(<item1> ... <item3>)

Value(s) that, when matched, trigger the execution of a given list of rules. A value can be a number, character, or symbol. See the special instructions in “[Case Rules and Strings](#)” on page 408 and “[Case Rules and Integers](#)” on page 409 for comparing strings and integers.

<Rules or expressions>

The rules/functions to be executed. The **DEFAULT** case is executed if none of the other Case statements are selected.

Controls

Table 64 explains the **Case** rule’s GUI control features in the Rules pane.

Table 64 Case Rule Control Features

Add Case	Adds another CASE entry under the same SWITCH .
CASE	Selects the field to enter rules or functions to be executed.
DEFAULT	Selects the field to enter rules to be executed if <i>none</i> of the preceding CASE entries are selected.
	Moves the selected field up or down in the sequence.

Table 64 Case Rule Control Features (Continued)

	Deletes the selected entry.

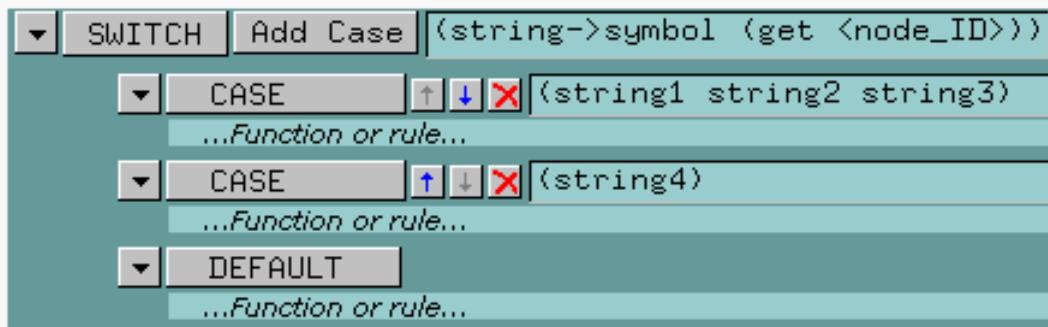
Creating Case Rules

When creating **Case** rules, you must observe important guidelines when entering strings and integers as explained in this section.

Case Rules and Strings

When using strings in **Case** rules, you must use the Monk **string->symbol** function and enter the strings as symbols within parentheses, as shown in Figure 175.

Figure 175 Case Rule Strings



In Figure 175 above, **<node_ID>** is the ETD node against which the strings within the **Case** rule are compared.

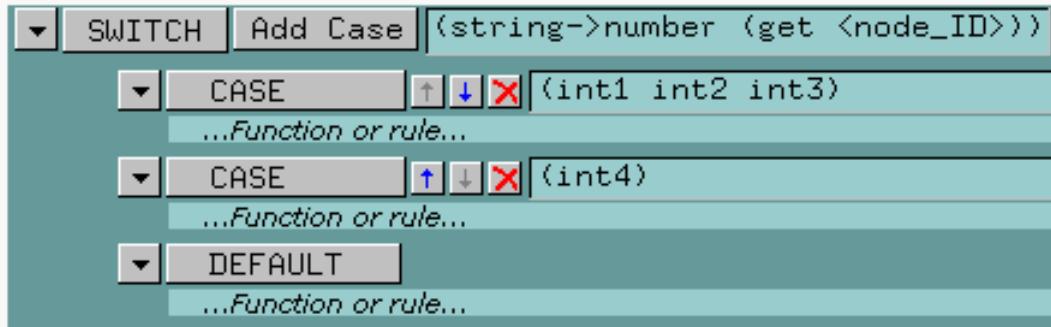
When entering strings in **Case** rules, observe the following guidelines:

- Be sure there are no spaces in the **string->symbol** function name.
- The strings to be compared must be enclosed in parentheses. Do not use double quotation marks.
- Lists of strings to be compared are delimited by spaces. In the example above, if the input node contains any of the strings **string1**, **string2**, or **string3**, the first function/rule block executes.

Case Rules and Integers

When using integers in **Case** rules, you must use the Monk **string->number** function and enter the integers as symbols within parentheses, as shown in Figure 176.

Figure 176 Case Rule Integers



In Figure 175 above, <node_ID> is the ETD node against which the integers within the **Case** rule are compared.

When entering integers in **Case** rules, observe the following guidelines:

- Be sure there are no spaces in the **string->number** function name.
- The integers to be compared must be enclosed in parentheses.
- Lists of integers to be compared are delimited by spaces. In the example in Figure 175, if the input node contains any of the integers **int1**, **int2**, or **int3**, the first function/rule block executes.

To create a Case rule

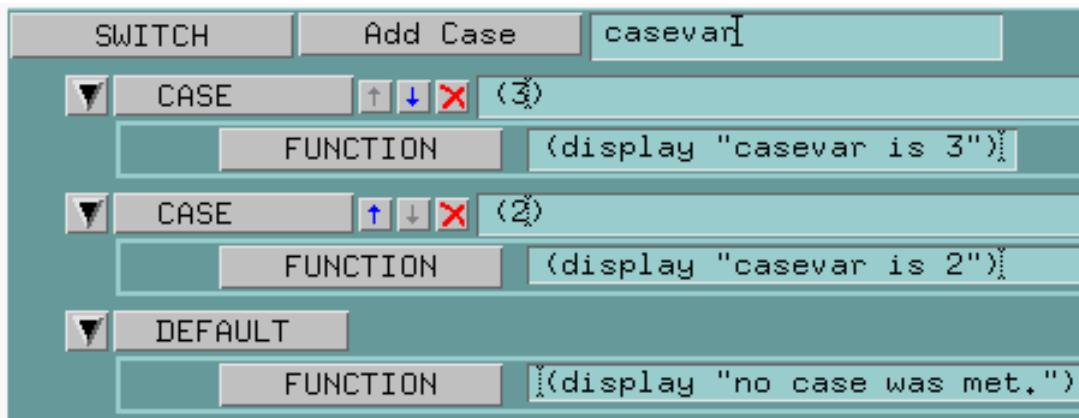
- 1 In the Rules pane, select the rule below which you wish to insert the new rule. If there are no rules in the list, continue with step 2.
- 2 On the toolbar, click .
- 3 In the <value> text box, enter the test expression. If the test involves ETD nodes, you can drag and drop the desired nodes into the expression.
- 4 In the <item> list, enter the result(s) of the test expression which trigger the execution of the first set of rules. Items in the result list must be separated by spaces and enclosed by parentheses.
- 5 Click **CASE**.
- 6 Enter the rule(s) or function(s) to be executed. Be sure to follow the rules for comparing strings and integers.
- 7 To add additional cases, click **Add Case**, then repeat steps 4 through 6.

- 8 To add a case to be executed if all other cases evaluate false, click **DEFAULT** and repeat steps 4 through 6.
- 9 To change the order of cases, click or on the desired **Case** statement to move the statement in the desired direction.

Example

The example in Figure 177 uses the **Case** rule to test whether a variable (**casevar**) has a value of 2 or 3.

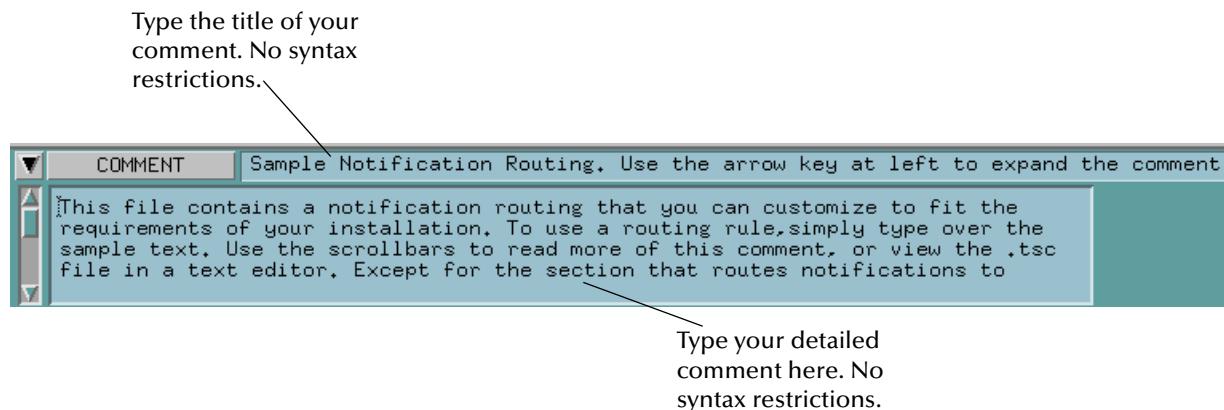
Figure 177 Case Rule Example



8.5.5 Using the Comment Rule

The **Comment** rule allows you to insert a comment into the Collaboration Rules component. It is for your reference only, and is not used by the e*Gate system for Collaboration purposes. [Figure 178 on page 410](#) shows an example of the Rules pane with a **Comment** rule entered.

Figure 178 Comment Rule Example



To enter a comment

- 1 From the **Rules** menu, choose **Add Comment**.
- 2 Select the sample text provided in the rule, and delete it (**<title>** and **<multi_line_comment>**).
- 3 Fill in the title and detailed comment in the **Comment** Rule bar text boxes. The multi-line comment text box can hold an extensive amount of text; however, if you run out of room, add another comment rule.

Note: If you only have a short comment to fill in, you can just type it in the **<title>** text box, and leave the **<multi_line_comment>** text box empty.

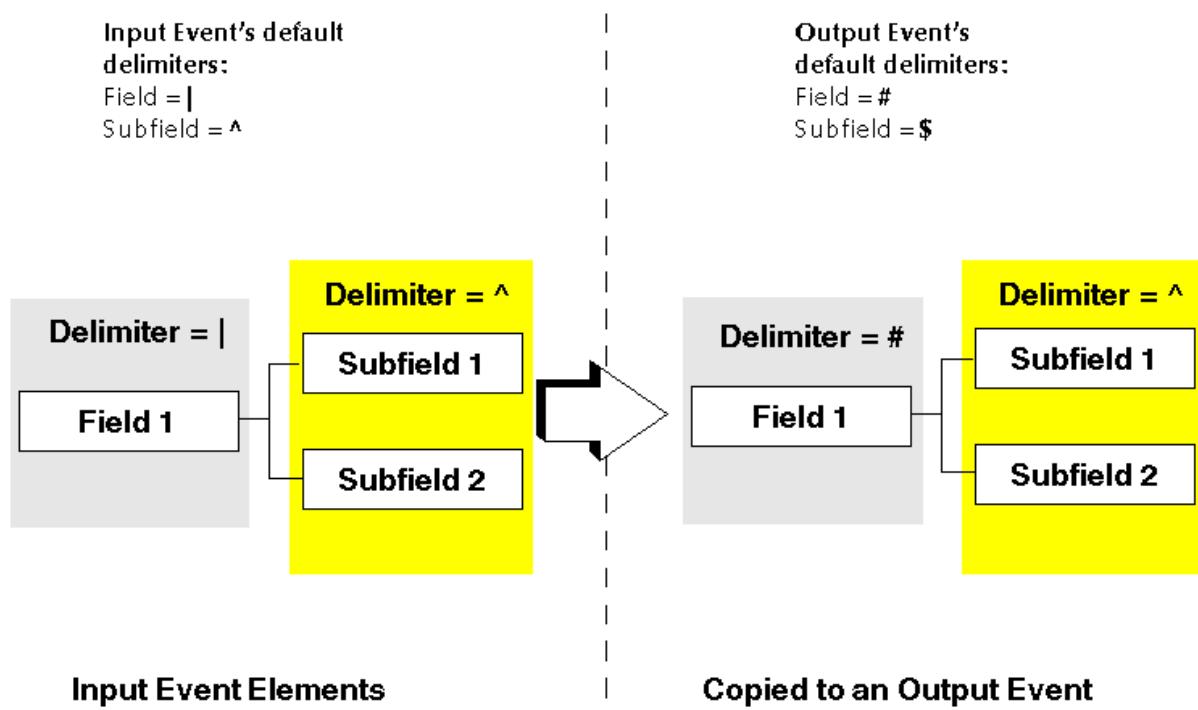
8.5.6 Using the Copy Rule

The **Copy** rule copies data from a location in a source Event to a location in a destination Event.

It is important to keep in mind how the **Copy** rule works when the input Event's delimiters are different from the output Event's delimiters. When input and output Event delimiters are different, a copied Event element uses the output Event's delimiters.

However, if the copied element contains lower-level elements, then the lower-level elements retain the delimiters assigned to them in the input Event. [Figure 179 on page 411](#) shows a diagram of how the **Copy** rule handles delimiters in output Events.

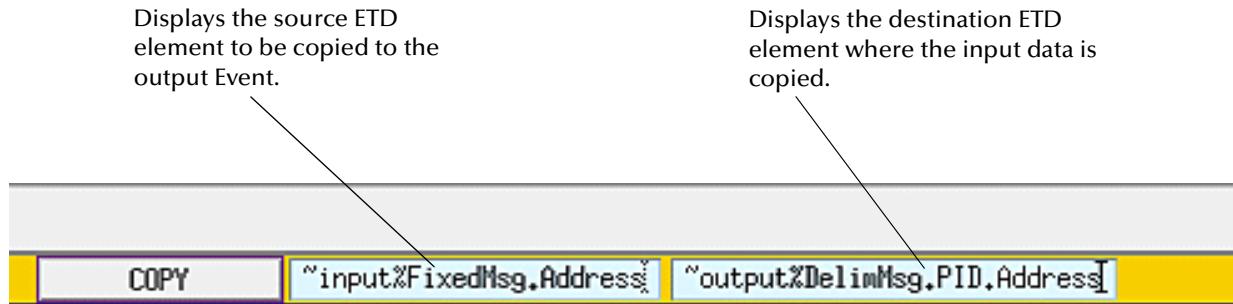
Figure 179 How the **Copy** Rule Handles Input-to-output Event Delimiters



Note: If you want to completely replace an Event element's input Event delimiters with the output Event's delimiters, then use the **Duplicate** rule. For details on the **Duplicate** rule, see “[Using the Duplicate Rule](#)” on page 417.

Figure 178 shows an example of the Rules pane with a **Copy** rule entered.

Figure 180 Copy Rule Example



To enter a Copy rule

- 1 Select the element to be copied using one of the ways shown in Table 65.

Table 65 Copy Rule Use Methods

Point and Click	Drag and Drop
<p>Make sure that Use Selected Nodes in New Rules option is selected.</p> <p>In the source ETD, select the element that you want to copy to the output Event.</p> <p>In the destination ETD, select the location where the element will be copied.</p> <p>Go to step 2.</p>	<p>In the source ETD, click and hold the middle mouse button on the element of the Event to be copied.</p> <p>Drag the pointer to the location in the destination ETD where the element will be copied, and then release the middle mouse button.</p> <p>If you selected an element that repeats, or belongs to higher-level nodes that repeat, the Select Repetition Instance dialog box appears. See “Defining Instances of Repeating Event Elements” on page 371 for details on how to use this dialog box.</p> <p>Go to step 3.</p>

- 2 On the toolbar or Rules menu, click **Add Copy**.

If you selected an element that repeats or belongs to higher-level nodes that repeat, the **Select Repetition Instance** dialog box appears. See “[Defining Instances of Repeating Event Elements](#)” on page 371 for details on how to use this dialog box.

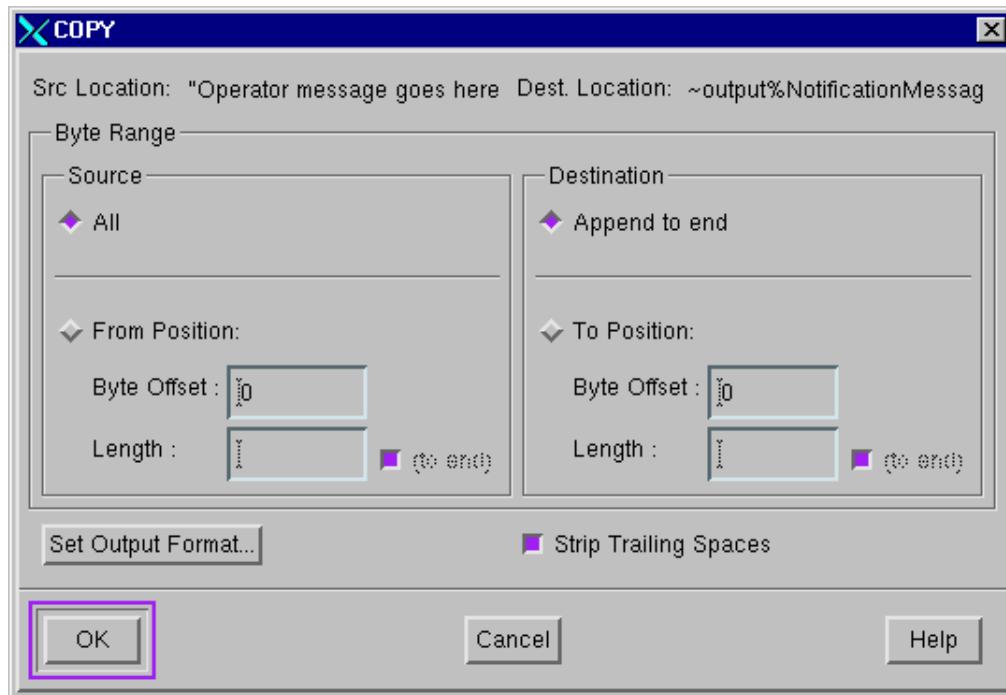
A **Copy** rule is added to the Rule bar in the Rules pane.

Note: For information on how to access lower-level Event elements missing from your ETD, see “[Defining ETD Paths](#)” on page 369.

- 3 To specify additional **Copy** rule properties, click **COPY** in the Rule bar (this step is optional).

The **Copy** dialog box appears. This dialog box allows you to copy data “as is” from one node to another. The original delimiters of child nodes are retained (see [Figure 181 on page 413](#)).

Figure 181 Copy Dialog Box



Enter information in this dialog box as follows:

- In the **Source** column, select **All** to copy all of the data in the node, or **From Position** to copy only a portion of the data in the node.
- If you select **From Position**:
 - In the **Byte Offset** box, enter the beginning byte location of the data you are copying to the output Event. Bytes are numbered starting at zero.
 - To specify a byte length, clear the **(to end)** check box, then enter a byte length in the **Length** box. The minimum byte length is 1.
- In the **Destination** column, select **Append to end** to append the copied data to the end of the output Event. Select **To Position** to copy the data to a specific position within the node. Copied data replaces any original data at that position.
- If you select **To Position**:
 - In the **Byte Offset** box, enter the beginning byte location for the copied data. Bytes are numbered starting at zero.

- ♦ To specify a byte length, clear the **(to end)** check box, then enter a byte length in the **Length** box. The minimum byte length is 1.
- ♦ Check **Strip Trailing Spaces** to delete trailing spaces from the input Event data. By default this box is checked.
- ♦ To customize the copied data's output format, click **Set Output Format**.

Table 66 gives you additional information on how to use the **Copy** dialog box.

Table 66 Using the Copy/Duplicate Dialog Box

Text Box	Definition
Byte Offset	<p>Defaults to the entire Event element selected (optional).</p> <p>Source — For the source Event element, fill in the beginning byte location of the data you are copying to the output Event.</p> <p>Destination — For the destination Event element, specify a beginning byte location for the copied data.</p> <p>To Count Bytes — For both fixed and delimited Events, count the offset from byte 0, the beginning of the selected Event element.</p>
Length	<p>Defaults to the entire length of the selected Event element (optional).</p> <p>Source — For the source Event element, fill in the length, in bytes, of the input Event data you are copying.</p> <p>Destination — For the destination Event element, specify the length, in bytes, of the output data.</p> <p>To Count Bytes — For fixed Events, count the number of bytes from 1. For delimited Events, where field length is variable, leave the (to end) button selected to set the length to the end of the Event element.</p>

Note: For details on the **Duplicate** rule, see “[Using the Duplicate Rule](#)” on page 417.

For more details about specifying byte offset and length, see “[Specifying byte locations](#)” on page 375.

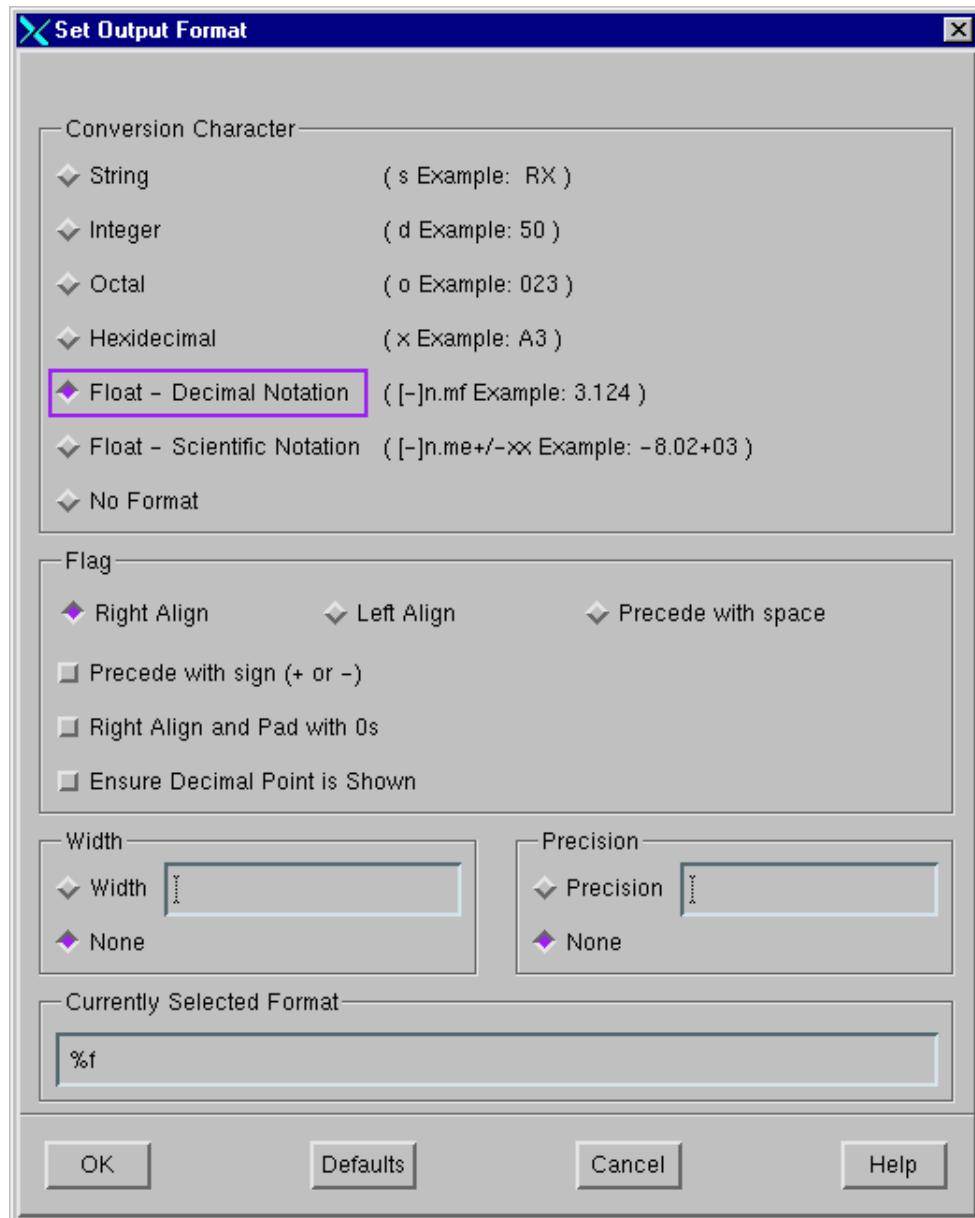
- 4 By default, the **Strip Trailing Spaces** button is automatically selected. Deselect this button if you do not want trailing spaces deleted from the input Event data.

If the input Event data being copied is followed by any empty spaces, the **Strip Trailing Spaces** option ensures that those empty spaces are deleted. This means that your output Event has no empty spaces.

- 5 If you want to customize the copied data's output format, click **Set Output Format**.

The **Set Output Format** dialog box appears. This dialog box allows you to format the display of the output data. See [Figure 182 on page 415](#).

Figure 182 Set Output Format Dialog Box



Enter information in this dialog box as follows:

- ◆ Under **Conversion character**, select the data format to which output data is converted. The default is to perform no conversion (**No Format**).
- ◆ For integer or float formats only: Under **Flag**, select an alignment for the output string. The default for all formats is **Right Align**.
- ◆ Under **Width**, specify the minimum width in bytes of the output data field. If the input data width is less than the output field width, the data is right-aligned in the output field. If no minimum value is specified or the input data width is greater than this value, the output field width is expanded to contain the results of the conversion. The default, **None**, imposes no width limit.

- Under **Precision**, enter the maximum number of characters (in bytes) to be printed in the output field. For integers only, this field determines the minimum number of digits to be printed. The default, **None**, imposes no limit.
- To reset all values to their defaults, click **Defaults**.

Note: The **Currently Selected Format** text box shows the Monk codes used to impose the format you have selected. The contents of this text box are determined by the selections you make in the dialog box controls; you cannot edit this text box's contents directly.

Table 67 gives you additional information on how to use this dialog box.

Table 67 Using the Set Output Format Dialog Box

Option	Description
Conversion Character	Specifies the method of displaying numbers and letters in the inserted string. Select a method from the list.
Flag	Specifies the alignment of contents of the inserted string. Select the preferred alignment from the list.
Width	The minimum width, in bytes, of the output data field. If the input data width is less than the output field width, the data is right-aligned in the output field. If no minimum value is specified or the input data width is greater than this value, the output field width is expanded to contain the results of the conversion.
Precision	The maximum number, in bytes, of characters to be printed in the output field. For integers, it is the minimum number of digits to be printed.

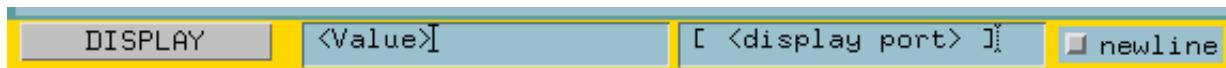
As you select formatting options, the **Currently Selected Format** text box displays your choices. See the “[Example](#)” on page 431 for output format examples.

- Click **OK**.
- Click **OK** to exit the **Copy** dialog box.
- When you are finished, on the toolbar or **File** menu, click  **Save**.

8.5.7 Using the Display Rule

Use the **Display** rule to print whatever information you include in the rule to a destination of your choice. Figure 183 shows an example of the Rules pane setup for this rule.

Figure 183 Display Rule Setup



To enter a Display rule

- On the **Rules** menu, click **Add Display**.

A Rule bar for the **Display** rule appears in the Rules pane.

2 Enter information for this rule as follows:

- <Value> — Enter the information you wish to include.
- [<display port>] — Enter the destination for your information, for example, a text file. The default is to the log file associated with the current component.
- **newline** — Includes the information on its own separate line in the destination, for example, a text file.

3 When you are finished, on the toolbar or **File** menu, click  **Save**.

8.5.8 Using the Duplicate Rule

Like the **Copy** rule, the **Duplicate** rule copies data from a location in a source Event to a location in a destination Event. However, the **Copy** and **Duplicate** rules differ in how they handle delimiters.

Where the **Copy** rule in some cases allows the input Event delimiters to be retained in the output Event, the **Duplicate** rule always replaces the input Event's delimiters with the output Event's delimiters. Figure 184 illustrates how the **Duplicate** rule handles delimiters in output Events.

Figure 184 How the **Duplicate** Rule Handles Input-to-Output Event Delimiters

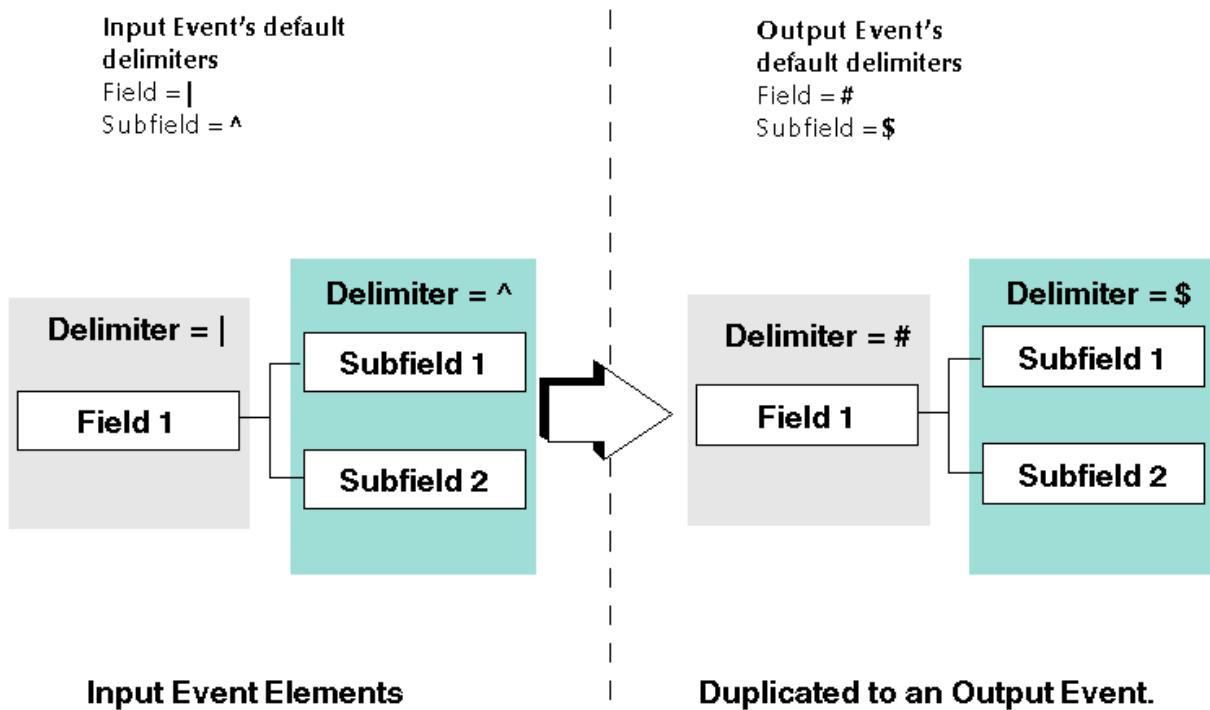
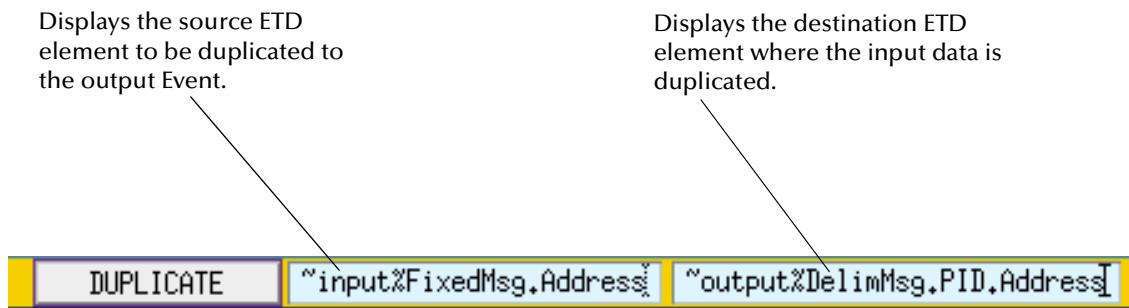


Figure 185 shows an example of the Rules pane with a **Duplicate** rule entered.

Figure 185 Duplicate Rule Example



To enter a Duplicate rule

- 1 Make sure that the **Use Selected Nodes in New Rule** option is selected.
- 2 In the source ETD, select the element that you want to duplicate to the output Event.
- 3 In the destination ETD, select the location where the element will be duplicated.
- 4 On the toolbar or **Rules** menu, click  **Add Duplicate**.

Note: *An easy way to duplicate an entire Event is to select the root nodes of both source and destination ETDs, then add the Duplicate rule.*

If you selected an element that repeats, or belongs to higher-level nodes that repeat, the **Select Repetition Instance** dialog box appears. See "["Defining Instances of Repeating Event Elements" on page 371](#)" for details on how to use this dialog box.

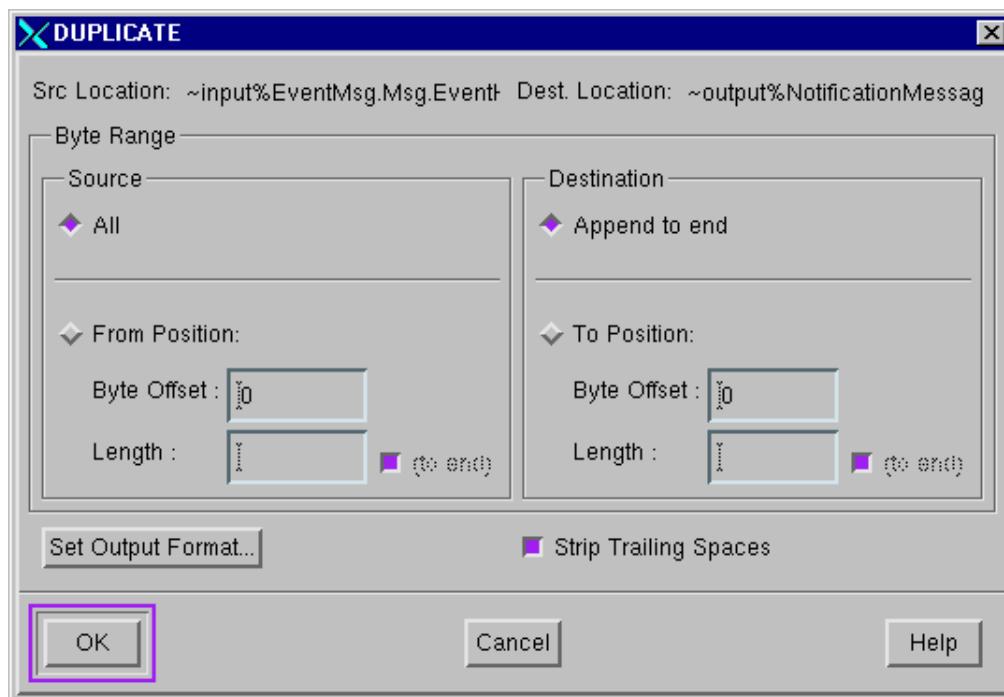
A **Duplicate** rule is added to the Rule bar in the Rules pane.

Note: *For information on how to access lower-level Event elements missing from your ETD, see "["Defining ETD Paths" on page 369](#)".*

- 5 To specify additional **Duplicate** rule properties, click **DUPLICATE** in the Rule bar (this step is optional).

The **Duplicate** dialog box appears. This dialog box allows you to duplicate data from one node to another. The original delimiters of child nodes are replaced (see [Figure 186 on page 419](#)).

Figure 186 Duplicate Dialog Box



Enter information in this dialog box as follows:

- In the **Source** column, select **All** to duplicate all of the data in the node, or **From Position** to duplicate only a portion of the data in the node.
- If you select **From Position**:
 - In the **Byte Offset** box, enter the beginning byte location of the data you are duplicating to the output Event. Bytes are numbered starting at zero.
 - To specify a byte length, clear the **(to end)** check box, then enter a byte length in the **Length** box. The minimum byte length is 1.
- In the **Destination** column, select **Append to end** to append the duplicated data to the end of the output Event. Select **To Position** to duplicate the data in a specific position within the node. Duplicated data replaces any original data at that position.
- If you select **To Position**:
 - In the **Byte Offset** box, enter the beginning byte location for the duplicated data. Bytes are numbered starting at zero.
 - To specify a byte length, clear the **(to end)** check box, then enter a byte length in the **Length** box. The minimum byte length is 1.
- Check **Strip Trailing Spaces** to delete trailing spaces from the input Event data. By default this box is checked.
- To customize the copied data's output format, click **Set Output Format**.

For additional information on how to use the **Duplicate** dialog box, see [Table 66 on page 414](#).

Note: For more details about specifying byte offset and length, see “[Specifying byte locations](#)” on page 375.

- 6 By default, the **Strip Trailing Spaces** button is automatically selected. Deselect this button if you do not want trailing spaces deleted from the input Event data.

If the input Event data being duplicated is followed by any empty spaces, the **Strip Trailing Spaces** option ensures that those empty spaces are deleted. This means that your output Event has no empty spaces.

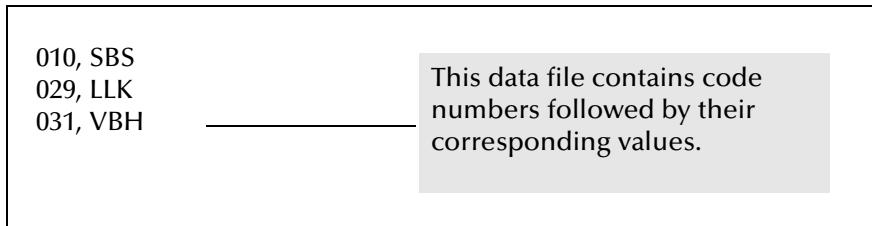
- 7 If you want to customize the copied data’s output format, click the **Set Output Format** button. See the [procedure on page 415](#) for more information on how to use this dialog box.
- 8 Click the **OK** button.
- 9 Click the **OK** button to exit the **Duplicate** dialog box.

- 10 When you are finished, on the toolbar or **File** menu, click  **Save**.

8.5.9 Using the Data Map Rule

The **Data Map** rule allows you to match a string in the inbound Event with a string stored in an ASCII text file. The mapping data associated with the matching string is inserted into the output Event. For example, if you want to substitute a code in the input Event with its corresponding value in the output Event, you could use a data map text file with contents like the following figure:

Figure 187 Sample Text File for **Data Map** Rule



The **Data Map** rule will look at the code in the input Event element, match it to its value in the text file, and insert the code’s value in the output Event. “010” in the input Event would be translated to “SBS” in the output Event.

Note: If the **Data Map** rule finds no match, an empty Event element is written to a delimited output Event, and no data will be written to a fixed output Event. An exception is returned and the data-map function fails; for details, see the **Monk Developer’s Reference**. However, you can enter a default value in the data file which the system will use when no match is found. Read on for details.

ASCII File Formatting

When you develop an ASCII text file to use with the **Data Map** rule, format the entries to be matched as follows:

string to be matched, mapping data

where the string to be matched is what's found in the input Event, and the mapping data is what's written to the output Event.

Default Value in Data File

You can specify a default value in the data file that is written to the output Event when no matching value is found for the input Event element string. Use the following syntax for the default value:

%default%, mapping data

where mapping data is the default value written to the output Event.

You can also enter the default value with the following syntax:

, mapping data

where you precede the mapping data with a comma.

Special Character Handling in the Data File

Because a comma is used as the delimiter in the data file, a comma must be preceded by a backslash (\) if it appears in either the string to be matched, or the mapping data.

To represent a backslash in the data, enter two backslashes (\\).

A backslash before a NewLine character at the end of a data file line is interpreted as a literal and the NewLine character is written to the output Event.

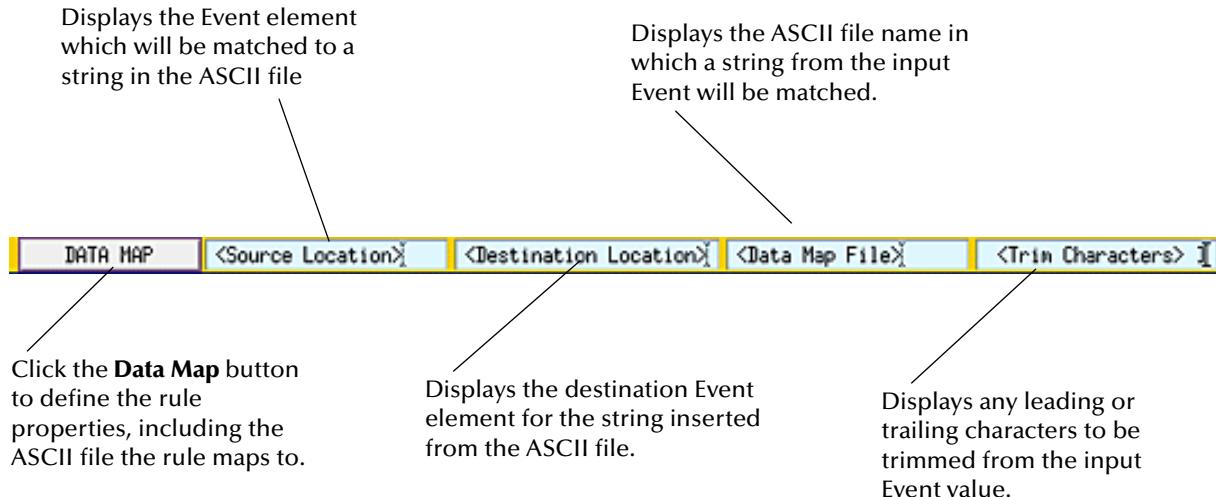
The e*Gate system supports the ANSI-C interpretation of escape sequences for Data Map data files. See *Monk Developer's Reference* for details.

Note: *If changes are made to the Data Map data file after system startup, you need to reload the data file.*

Syntax

The following figure shows the **Data Map** rule syntax.

Figure 188 Data Map Rule Syntax



Procedure

- 1 Make sure that **Use Selected Nodes in New Rules** in the **Options** menu is selected.
- 2 In the **Source** ETD, select the element containing a string that will be matched to a string in an ASCII text file.
- 3 In the **Destination** ETD, select the location where the matched string will be inserted.

Note: For information on how to access lower-level Event elements missing from your ETD, see “[Defining ETD Paths](#)” on page 369.

- 4 From the **Rules** menu, select **Add Data Map**.

The **Data Map** rule appears in the **Collaboration**.

If you selected a ETD element that repeats, or belongs to higher-level nodes that repeat, the **Select Repetition Instance** dialog box appears. See “[Defining Instances of Repeating Event Elements](#)” on page 371 for details on how to use this dialog box.

- 5 In the **Collaboration**, click the **Data Map** button.

The **Data Map** dialog box appears.

- 6 Enter the following information in the **Data Map** dialog box (see the following table).

Table 68 Data Map Dialog Box Entries

Text Box	Description
Byte Offset	<p>(Optional) Defaults to entire Event element selected.</p> <p>Source: For the source Event, fill in the beginning byte location of the data you are matching to an ASCII file.</p> <p>Destination: For the destination Event, fill in a beginning byte location where the matched string will be inserted.</p> <p>To count bytes: For both fixed and delimited Events, count the offset from byte 0, the beginning of the selected Event element</p>
Length	<p>(Optional) Defaults to entire length of selected Event element.</p> <p>Source: For the source Event, fill in the length, in bytes, of the input Event data you are matching to an ASCII file.</p> <p>Destination: For the destination Event, specify the length, in bytes, of the output data.</p> <p>To count bytes: For fixed Events, count the number of bytes from 1. For delimited Events, where field length is variable, leave the (to end) button selected to set the length to the end of the Event element.</p>
Trim Characters	<p>This text box is, by default, active when you open the dialog box. If necessary, you can re-activate it by clicking inside it.</p> <p>Type any leading or trailing characters to be trimmed from the input Event values before matching against the specified set of match strings. Be sure to type the string between the two double-quotation marks (" ") provided for you. All characters in the set are interpreted as literals. Do not separate characters, even with spaces; spaces are also interpreted literally.</p> <p>You can specify a variable in this box; just delete the double-quotation marks before typing in your variable.</p> <p>For details about how to define a variable, see "Using the Let Rule" on page 437.</p>
Directories/Files	Select the directory and file name of the ASCII text file you want to match the input string to.

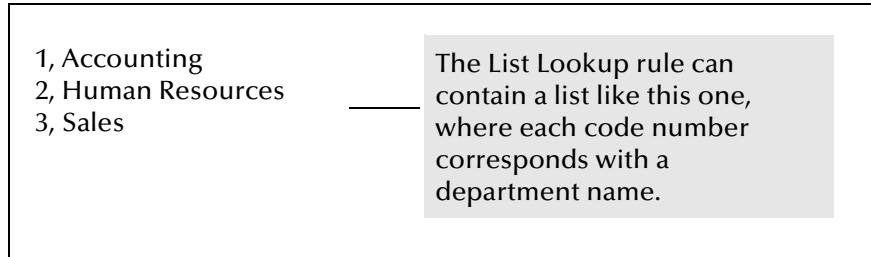
For more details about specifying byte offset and length, see ["Specifying byte locations" on page 375](#).

For more details about how e*Gate places data in output Events, see ["How e*Gate Processes Event Data" on page 364](#).

- 7 Click **OK** to save your work and exit the **Data Map** dialog box.

8.5.10 Using the List Lookup Rule

The List Lookup rule allows you to substitute a value in an input Event with a different, although corresponding, value in the output Event. For example, you could substitute a department code from the input Event with the full department name in the output Event.

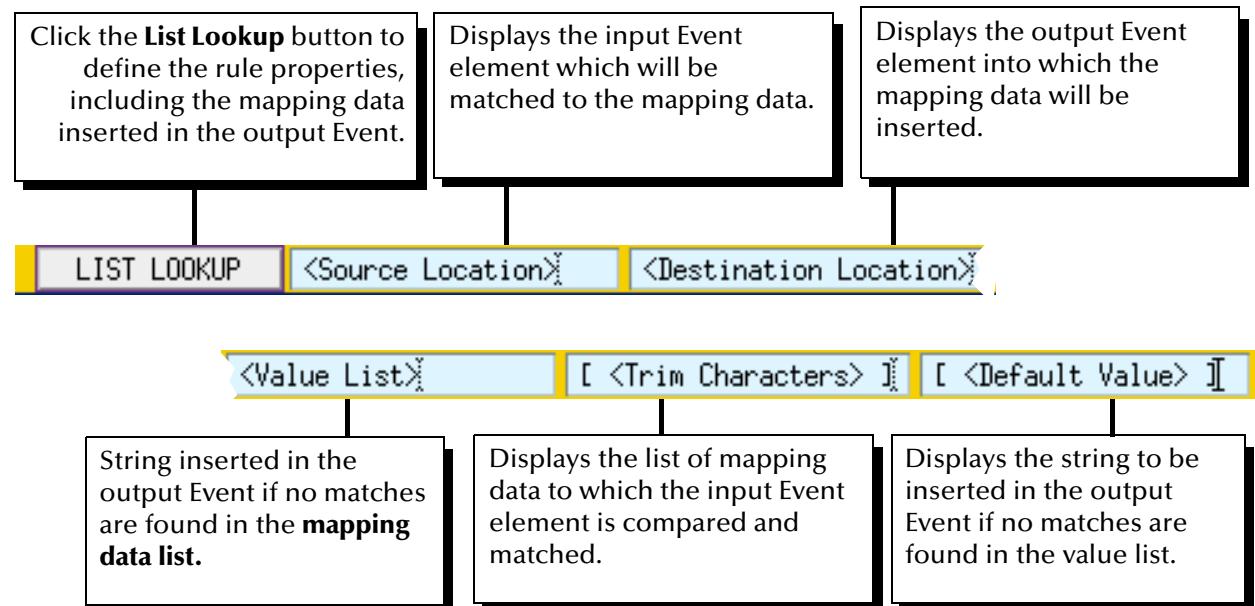


The List Lookup rule will search the input Event for a department code in its list; if it finds a match, then it inserts the corresponding full department name in the output Event. In the sample list shown above, if the input Event contains code “1”, then “Accounting” is placed in the output Event.

Note: *If the List Lookup rule finds no match in the input Event, an empty Event element is written to a delimited output Event, and no data is written to a fixed output Event. However, you can enter a default value in the rule's dialog box which the system will use when no match is found. Read on for details.*

Use the List Lookup rule when you only need to match an input Event string to a few mappings (no more than six is recommended); otherwise, use the Data Map rule ([“Using the Data Map Rule” on page 420](#)).

Syntax



Procedure

- 1 Make sure that **Use Selected Nodes in New Rules** in the **Options** menu is selected.
- 2 In the **Source** ETD, select the input Event element containing a string that will be matched to the rule's list.
- 3 In the **Destination** ETD, select the location where you want the mapping data to be inserted.

Note: For information on how to access lower-level Event elements missing from your ETD, see “[Defining ETD Paths](#)” on page 369.

- 4 From the **Rules** menu, select **Add List Lookup**.

The List Lookup rule appears in the **Collaboration**.

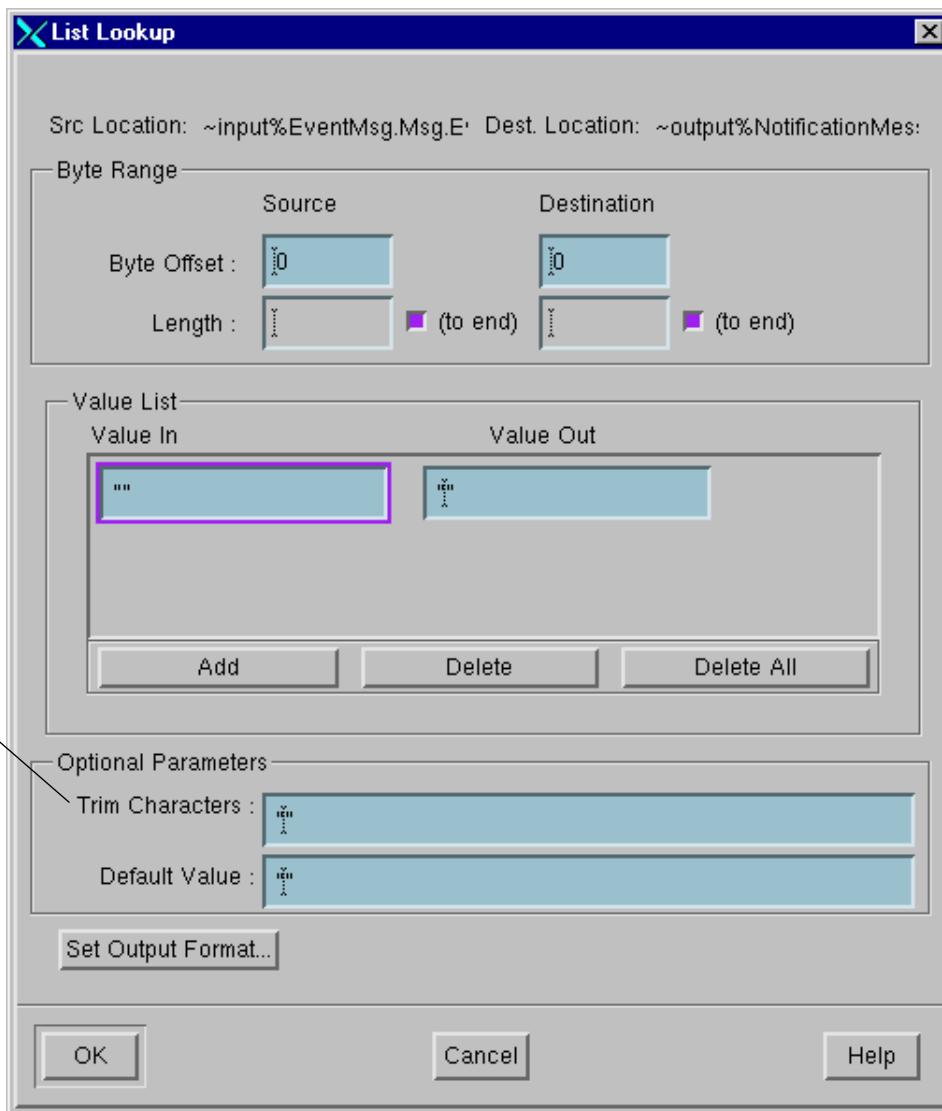
If you selected a ETD element that repeats, or belongs to higher-level nodes that repeat, the **Select Repetition Instance** dialog box appears. See “[Defining Instances of Repeating Event Elements](#)” on page 371 for details on how to use this dialog box.

- 5 In the **Collaboration**, click the **List Lookup** button.

The **List Lookup** dialog box appears.

This feature allows you to replace the input string with the output string (see [Figure 189 on page 426](#)).

Figure 189 List Lookup Dialog Box



- Under **Byte Range**, in the **Source** column, **Byte Offset** box, enter the beginning byte location of the data you are copying to the output Event. In the **Destination** column, enter the beginning byte location for the copied data. Bytes are numbered starting at zero.
- To specify a byte length for either source or destination Events, clear the appropriate **(to end)** check box, then enter a **Source** or **Destination** byte length in the appropriate **Length** box. The minimum byte length is 1.
- To specify a **Value In** and **Value Out** pair of values, click **Add**, then enter the values.
- To delete a **Value In** and **Value Out** pair, select either element of the pair and click **Delete**. To delete all pairs in the list, click **Delete All**.
- In the **Trim Characters** box, enter a quoted string or (unquoted) variable name that contains the character(s) to be trimmed from the input Event before matching them against the values. All characters are taken as literals, including spaces.

- In the **Default Value** box, enter the quoted string or (unquoted) variable name that contains the default value to be inserted in the output Event if no values in the **Value In** list are matched.
- To customize the copied data's output format, click Set Output Format.

6 Fill in the following:

Text Box	Description
Byte Offset	<p>(Optional) Defaults to entire Event element selected.</p> <p>Source: For the source Event, fill in the beginning byte location of the data string you want to match to the rule's list.</p> <p>Destination: For the destination Event, fill in a beginning byte location where the mapping data will be inserted.</p> <p>To count bytes: For both fixed and delimited Events, count the offset from byte 0, the beginning of the selected Event element.</p>
Length	<p>(Optional) Defaults to entire Event element selected.</p> <p>Source: For the source Event, fill in the length, in bytes, of the input Event data on which to operate.</p> <p>Destination: For the destination Event, specify the length, in bytes, of the output data.</p> <p>To count bytes: For fixed Events, count the number of bytes from 1. For delimited Events, where field length is variable, leave the (to end) button selected to set the length to the end of the Event element.</p>

Text Box	Description
Value List	<p>Click the Add button for each Value In/Value Out pair you want to add to the list. Select a pair and click Delete to remove it from the list. Click Delete All to remove the entire list.</p> <p>Value In: Enter the list of values to be matched against the input Event value.</p> <p>Value Out: Enter the list of values to be substituted for the input Event value. Each entry corresponds with an entry in the Value In list.</p> <p>For both the Value In and Value Out lists: If you are specifying a string literal, be sure to type the value between the double-quotation marks (" ") provided. If you are specifying a variable, delete both double-quotation marks before typing the value.</p> <p>For details about how to define a variable, see “Using the Let Rule” on page 437.</p> <p>To include a double-quote or backslash in an output Event string, you must escape the interpreted character in the Value List by preceding it with the backslash (\) character.</p>
Optional Parameters	<p>Trim Characters: This text box is, by default, active when you open the dialog box. If necessary, you can re-activate it by clicking inside it. Specify any leading or trailing characters to be trimmed from the input Event values before matching against the specified set of match strings. Be sure to type the trim characters between the two double-quotation marks (" ") provided for you if you want them to be interpreted as literals. Do not separate characters, even with spaces; spaces are also interpreted literally.</p> <p>You can specify a variable in this box; just delete the double-quotation marks before typing in your variable.</p> <p>Default Value: Specify the string to be inserted in the output Event if no matches are found in the Value In list. Be sure to type the default value between the two double-quotation marks (" ") provided for you if you want it to be interpreted as a literal string. If you want to specify a variable in this box, delete the double-quotation marks before typing in your variable.</p> <p>For details about how to define a variable, see “Using the Let Rule” on page 437.</p>

For more details about specifying byte offset and length, see [“Specifying byte locations” on page 375](#).

For more details about how e*Gate places data in output Events, see “[How e*Gate Processes Event Data](#)” on page 364.

Note: Each **Value In** string is matched exactly against the input Event. If the string found in the input Event element is shorter or longer than the **Value In** string, then the comparison fails and no match is found. Usually, this is a concern only when reading fixed Events. If the string in the input Event can be padded with leading or trailing spaces, use the **Trim Characters** option to ensure that a match is found. An alternate method is to include the spaces as part of the **Value In** value.

- 7 Click the **Set Output Format** button.

The **Set Output Format** dialog box appears.

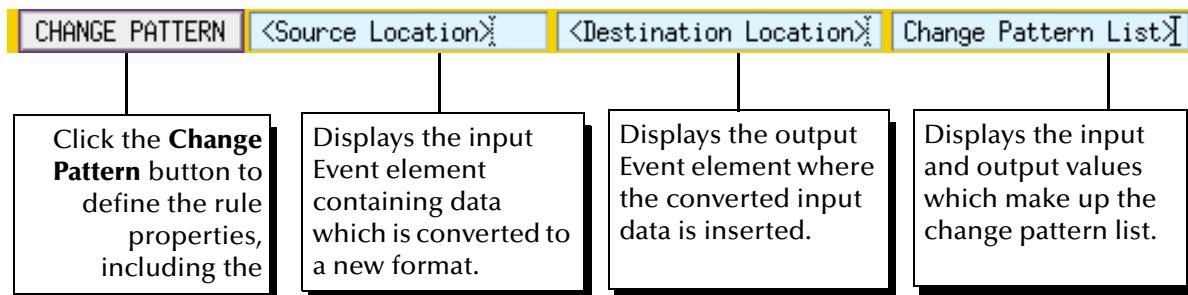
- 8 Fill in your Event output requirements. For more details on how to use this dialog box, see the [procedure on page 415](#) where the output format options are detailed.
- 9 Click **OK** to exit the **Set Output Format** dialog box. Click **OK** to exit the **List Lookup** dialog box.

8.5.11 Using the Change Pattern Rule

The Change Pattern rule allows you to specify an output string to replace an input string or regular expression; its primary feature is that it converts an input Event string to the desired format in the output Event.

As you set up the Change Pattern rule, notice that you need to define a list of input values that correspond with a converted output value. When you develop the Change Pattern list, you are telling the system, “If you find this value in the input Event, convert it to this format in the output Event.”

Syntax



Procedure

- 1 Make sure that **Use Selected Nodes in New Rules** in the **Options** menu is selected.
- 2 In the **Source** ETD, select the element containing a string that will be converted to a new format.

- 3 In the **Destination** ETD, select the location where the converted string will be inserted.

Note: For information on how to access lower-level Event elements missing from your ETD, see “[Defining ETD Paths](#)” on page 369.

- 4 From the **Rules** menu, select **Change Pattern**.

The Change Pattern rule appears in the **Collaboration**.

If you selected a ETD element that repeats, or belongs to higher-level nodes that repeat, the **Select Repetition Instance** dialog box appears. See “[Defining Instances of Repeating Event Elements](#)” on page 371 for details on how to use this dialog box.

- 5 In the **Collaboration**, click the **Change Pattern** button.

The **Change Pattern** dialog box appears.

- 6 Fill in the following:

Text Box	Description
Byte Offset	<p>(Optional) Defaults to entire Event element selected.</p> <p>Source: For the source Event, fill in the beginning byte location of the input Event data to be converted.</p> <p>Destination: For the destination Event, fill in a beginning byte location where the converted string will be placed.</p> <p>To count bytes: For both fixed and delimited Events, count the offset from byte 0, the beginning of the selected Event element.</p>
Length	<p>(Optional) Defaults to entire Event element selected.</p> <p>Source: For the source Event, fill in the length, in bytes, of the input Event data to be converted.</p> <p>Destination: For the destination Event, specify the length, in bytes, of the output data.</p> <p>To count bytes: For fixed Events, count the number of bytes from 1. For delimited Events, where field length is variable, leave the (to end) button selected to set the length to the end of the Event element.</p>

Text Box	Description (Continued)
Change Pattern List	<p>Click Add for each In Pattern/Out Pattern pair you want to add to the list. Click Delete to remove a selected pair from the list. Click Delete All to remove the entire list.</p> <p>In Pattern: Fill in a string or other regular expression to be matched to the input Event element. For details on how to write a regular expression, see the <i>Monk Developer's Reference</i>.</p> <p>Out Pattern: Fill in a string to be output, replacing the corresponding In Pattern string. Each Out Pattern must have a corresponding In Pattern.</p> <p>For both the In Pattern and Out Pattern: If you are specifying a string literal, type the value between the double-quotation marks (" ") provided. If you are specifying a variable, delete both double-quotation marks before typing the value. For details about how to define a variable, see "Using the Let Rule" on page 437.</p>

For more details about specifying byte offset and length, see "[Specifying byte locations](#)" on page 375.

For more details about how e*Gate places data in output Events, see "[How e*Gate Processes Event Data](#)" on page 364.

Caution: If the data in the input Event element and the **In Pattern** don't match, no Collaboration takes place and an empty field or field element is written to a delimited output Event. No data is written to a fixed Event. To avoid losing data, use the If rule to write a conditional expression that tests for the correct string or pattern.

- 7 Click the **Set Output Format** button.
- The **Set Output Format** dialog box appears.
- 8 Fill in your Event output requirements. For more details on how to use this dialog box, see [procedure on page 415](#) where the output format options are detailed.
 - 9 Click **OK** to exit the **Set Output Format** dialog box. Click **OK** again to exit the **Change Pattern** dialog box.

Example

Delimited input and output Events. Replace the input Event string, "LPC", with another string in the output Event: "Laboratory Patient Center."

Event examples:

```

MSH | ^& | ADT | AMC | LAB | LPC | |
MSH | *& | TMS | AMC | LAB | LPC | |
MSH | %# | Patinfo | AMC | Path | PPC | |
MSH | ^& | TMS | AMC | LAB | LPC | |

```

Select the input Event element representing field 5 of the MSH segment for the Change Pattern rule. Then, select the output Event element representing field 5 of the MSH segment.

In the **Change Pattern** text box, fill in the following values.

For this rule parameter	Use this value	Result
Change Pattern List: In Pattern	LPC	System looks for this value in the selected input Event element.
Change Pattern List: Out Pattern	Laboratory Patient Center	If the selected input Event element contains the value, "LPC", it is replaced by the value, "Laboratory Patient Center", in the output Event element.

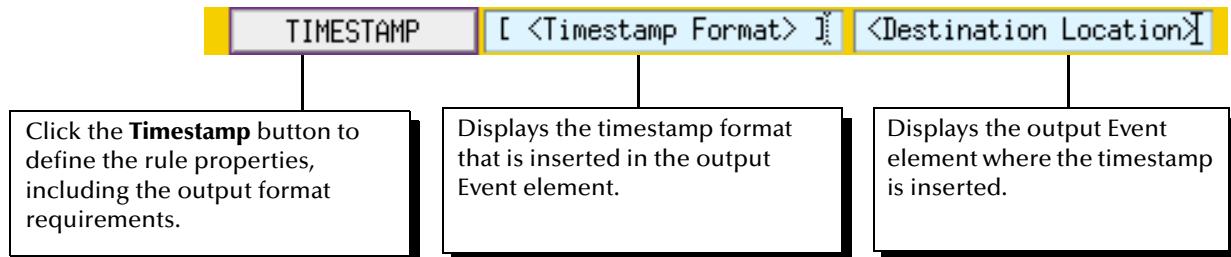
Example output:

```
MSH|^&|ADT|AMC|LAB|Laboratory Patient Center|||
MSH|^&|TMS|AMC|LAB|Laboratory Patient Center|||
MSH|^#|Patinfo|AMC|Path|||||
MSH|^&|TMS|AMC|LAB|Laboratory Patient Center|||
```

8.5.12 Using the Timestamp Rule

The Timestamp rule allows you to insert the current date and time (of the system's host) into the output Event. You can select the format from a list of options.

Syntax



Procedure

- 1 Make sure that **Use Selected Nodes in New Rules** in the **Options** menu is selected.
- 2 In the **Destination** ETD, select the location where the timestamp will be inserted.

Note: For information on how to access lower-level Event elements missing from your ETD, see "["Defining ETD Paths" on page 369](#)".

- 3 From the **Rules** menu, select **Add Timestamp**.

The Timestamp rule appears in the **Collaboration**, with the Destination ETD location displayed in it.

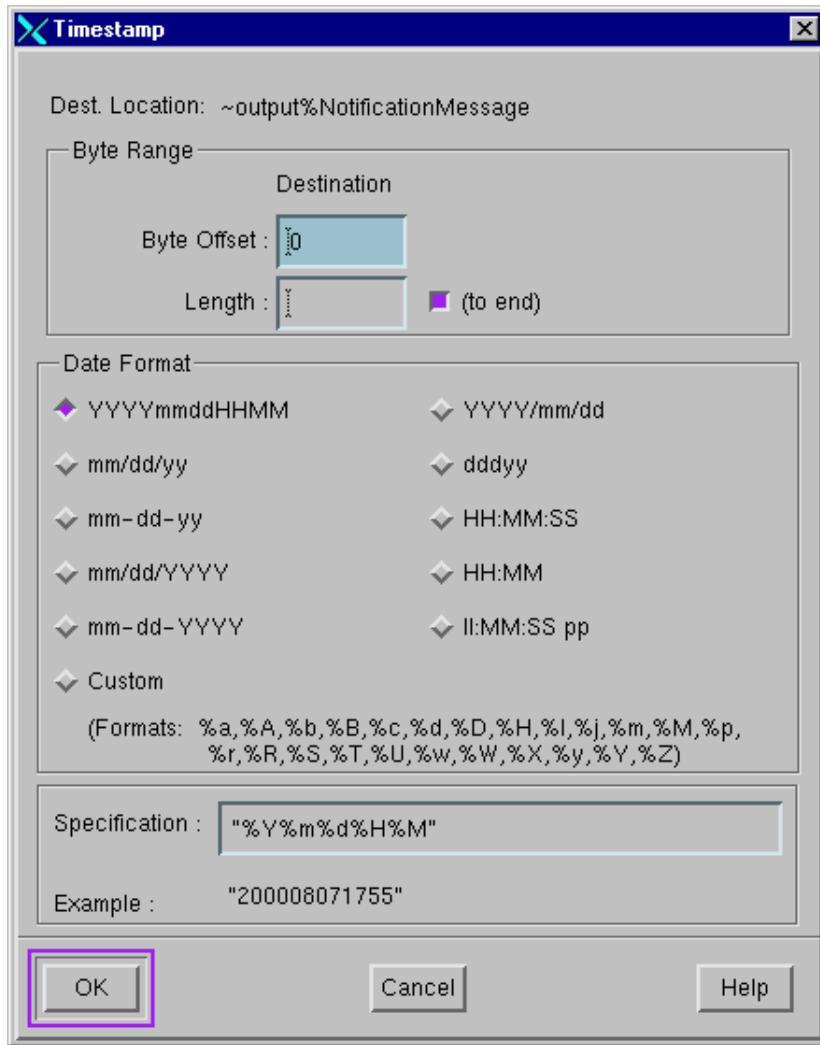
- 4 In the **Collaboration**, click the **Timestamp** button.

If you selected a ETD element that repeats, or belongs to higher-level nodes that repeat, the **Select Repetition Instance** dialog box appears. See "["Defining Instances](#)"

[of Repeating Event Elements” on page 371](#) for details on how to use this dialog box.

The **Timestamp** dialog box appears (see [Figure 190 on page 433](#)).

Figure 190 Timestamp Dialog Box



To use the Timestamp dialog box

- 1 In the **Byte Offset** box, enter the beginning byte location for the copied data. Bytes are numbered starting at zero.
- 2 To specify a byte length, clear the **(to end)** check box, then enter a byte length. The minimum byte length is 1.
- 3 Under **Date Format**, select the timestamp format to be inserted into the output Event. To create a custom format, select **Custom** and type the new format in the **Specification** box, below.
- 4 Sample output formatted using the selection appears in the **Example** box.

Note: The contents of the **Specification** and **Example** boxes are determined by the **Date Format** selection; for predetermined formats you cannot edit them directly.

- 5 Fill in:

Text Box	Description
Byte Offset	<p>(Optional) Defaults to entire Event element selected.</p> <p>Destination: For the destination Event, fill in a beginning byte location where the timestamp will be inserted.</p> <p>To count bytes: For both fixed and delimited Events, count the offset from byte 0, the beginning of the selected Event element.</p>
Length	<p>(Optional) Defaults to entire Event element selected.</p> <p>Destination: For the destination Event, specify the length, in bytes, of the output data.</p> <p>To count bytes: For fixed Events, count the number of bytes from 1. For delimited Events, where field length is variable, leave the (to end) button selected to set the length to the end of the Event element.</p>
Date Format	<p>Lists the available timestamp formats. If you select the last option, Custom, you need to fill out the Specification text box (go to step 6).</p>

For more details about specifying byte offset and length, see “[Specifying byte locations](#)” on page 375.

For more details about how e*Gate places data in output Events, see “[How e*Gate Processes Event Data](#)” on page 364.

- 6 (Optional) If you selected **Custom** in the list of timestamp formats, fill in the **Specification** text box according to the following table.

Time Division	Format Option	Description	Value Range or Sample Output
Days	%w	day of week (Sunday is day 0)	0-6
	%a	day of week, using site-defined abbreviations	for ex., Sun, Mon, Tue, etc.
	%A	day of week, using site-defined spellings	for ex., Sunday, Monday, etc.
	%d	day of month	01-31
	%e	day of month (single digits are preceded by a space)	1-31
	%j	day of year	001-366

Time Division	Format Option	Description (Continued)	Value Range or Sample Output
Weeks	%U	week of year (Sunday is the first day of the week)	01-52
	%W	week of year (Monday is the first day of the week)	01-52
Months	%m	month number	01-12
	%b	month, using site-defined abbreviations	for ex., Jan, Feb, Mar, Apr, etc.
Months	%B	month, using site-defined spellings	for ex., January, February, etc.
Years	%y	year within century	00-99
	%Y	year, including century	ex: 1988
Hours	%H	hour	00-23
	%I	hour	01-12
Minutes	%M	minute	00-59
Seconds	%S	seconds	00-59
Morning or Afternoon	%P	AM or PM	AM or PM
Time Zone Composites	%Z	time zone abbreviation	ex: PDT
	%D	date as %m/%d/%y	ex: 02/05/04
	%R	time as %H:%M	ex:14:15
	%T	time as %H:%M:%S	ex:14:15:03
	%r	time as %I:%M:%S%p	ex: 02:15:03 PM
	%x	site-defined standard date format	ex: 09/12/93

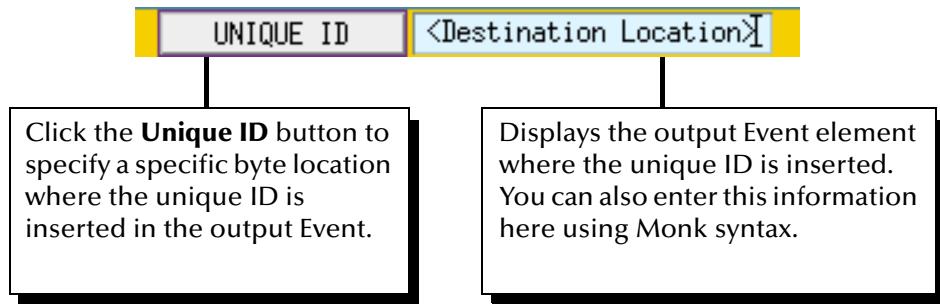
Once you have filled in a format, press the **Tab** key to see an example of that format automatically filled in for you in the **Example** text box.

- 7 Click **OK** to save your work and exit the **Timestamp** dialog box.

8.5.13 Using the Unique ID Rule

The **Unique ID** rule allows you to insert the current date and time (of the system's host) into the output Event. Use this rule to create a unique Event identifier. This rule differs from the **Timestamp** rule in that the **Unique ID** rule outputs milliseconds; **Timestamp** does not. Also, there is only one format for the **Unique ID** rule.

Syntax



Procedure

- 1 Make sure that **Use Selected Nodes in New Rules** in the **Options** menu is selected.
- 2 In the **Destination** ETD, select the element where the current date and time will be inserted.
- 3 On the toolbar or **Rules** menu, click **Add Unique ID**.
The **Unique ID** rule appears in the **Collaboration**.

If you selected a ETD element that repeats, or belongs to higher-level nodes that repeat, the **Select Repetition Instance** dialog box appears. See "["Defining Instances of Repeating Event Elements" on page 371](#)" for details on how to use this dialog box.

- 4 (Optional) To specify a specific byte location for the unique ID in the output Event, click the **Unique ID** button in the **Collaboration**.
The **Insert Unique ID** dialog box appears.
- 5 In the **Insert Unique ID** dialog box, fill in the following:

Text Box	Description
Byte Offset	Defaults to entire output Event element selected. Specifies the beginning data placement position within the output Event element you selected in step 2. For both fixed and delimited Events , is the offset from byte 0, the beginning of the selected Event element.

Text Box	Description (Continued)
Length	Defaults to length of entire output Event element selected. Fill in the length, in bytes, of the output Event element where the current date and time will be inserted. For fixed Events, count the number of bytes from 1. For delimited Events, where field length is variable, leave the (to end) button selected to set the length to the end of the Event element.

For more details about specifying byte offset and length, see “[Specifying byte locations](#)” on page 375.

For more details about how e*Gate places data in output Events, see “[How e*Gate Processes Event Data](#)” on page 364.

- 6 Click **OK** to exit from the **Insert Unique ID** dialog box.

8.5.14 Using the Let Rule

The Let rule allows you to define variables that are used in other rules, such as the If, Loop, and Insert rules.

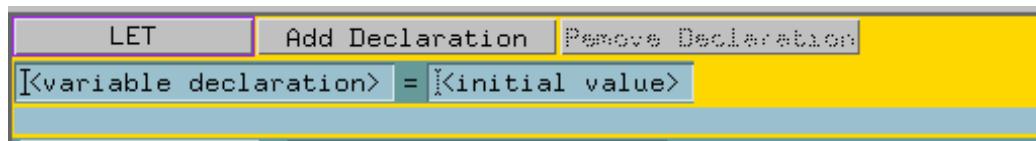
If you want to set up a variable in a rule using Let, first add the Let rule and define the variable(s). Then inside the Let rule, add the rule where you want to use the variable(s). This establishes the link between the variable definition in Let, and the rule where you use the variable(s).

When you define a variable in Let and apply that variable to other rules, the value of the variable is computed before it is bound to the variable and used in the rules.

Note that a Let rule variable only has meaning within the Let rule where it is defined. Let's say that in Let Rule A, you set up a variable where **i** = 2. Now when you set up Let Rule B, you can use the variable **i** again, and assign it a value of 5, without impacting or being impacted by the variable in Let Rule A. The system considers the variables in Let Rules A and B to be completely separate.

Figure 191 shows an example of the Let Rule bar.

Figure 191 Let Rule Bar Structure



For details on how to use the Let rule in combination with the Loop rule, see:

- [“Looping on a Computed Range of Event Elements” on page 401](#)
- [“Looping on a Fixed Range of Event Elements” on page 404](#)

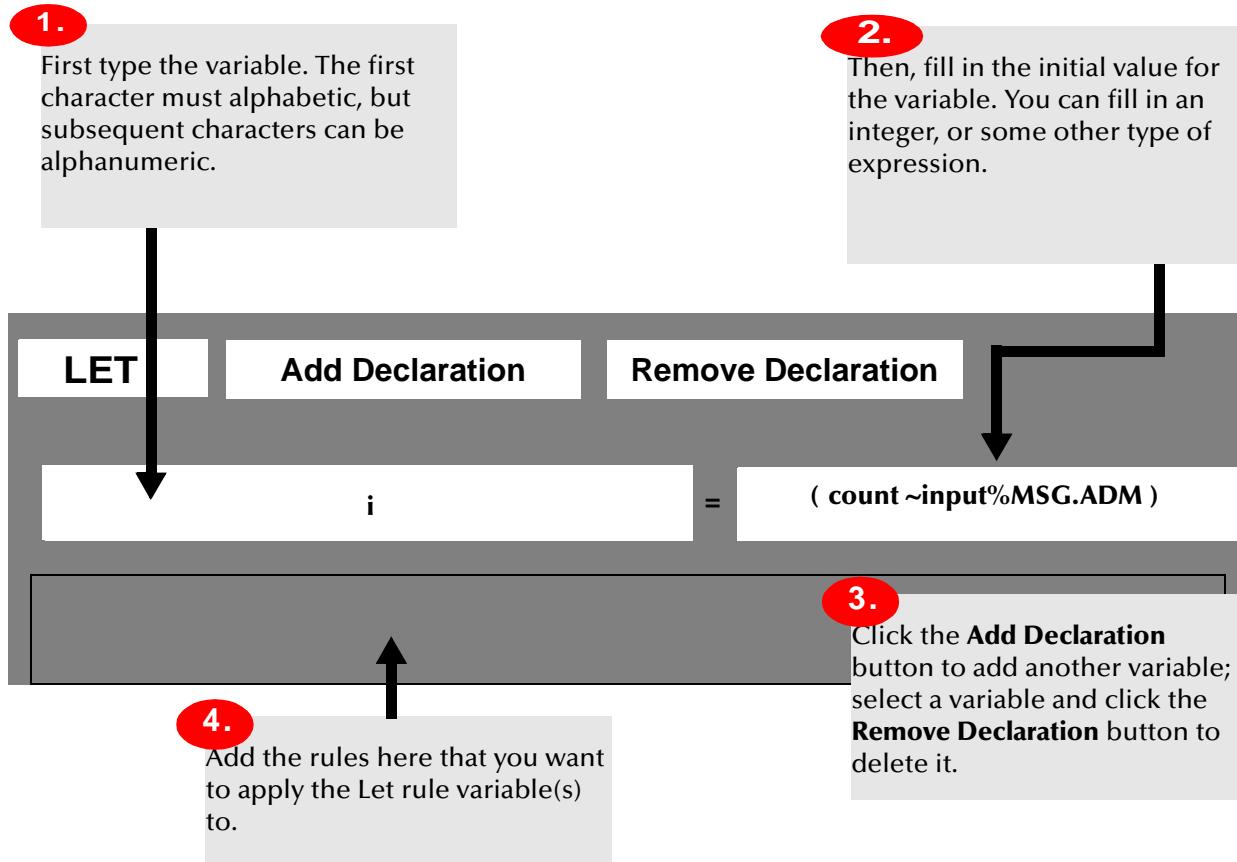
To see an example of how to name a condition using the Let rule, see “[Naming a Condition Using the Let Rule](#)” on page 439.

For details on how to use the Let rule to insert a variable in an Insert rule, see “[Using the Let Rule to Specify a Variable in an Insert Rule](#)” on page 441.

For the rules on naming variables, see “[Naming Variables in the Let Rule](#)” on page 438.

Constructing the Let Rule

Use this graphic to help you construct a Let rule. Before filling out the Let rule text boxes, be sure to delete the sample text provided by the Collaboration Rules Editor.



Naming Variables in the Let Rule

When naming variables in the Let rule, keep the following list of rules in mind:

- 1 These characters are accepted in variable names:

Description	Character(s)
all letters	A-Z, a-z
all digits	0-9

Description	Character(s)
plus-sign	+
dash	-
period	.
asterisk	*
slash	/
left angle bracket	<
equal-sign	=
right angle bracket	>
exclamation mark	!
question mark	?
colon	:
dollar sign	\$
percent sign	%
underscore	_
ampersand	&
tilde	~
caret	^

2 The first character of a variable cannot be:

- a digit (0-9)
- a tilde (~)
- a plus (+)
- a dash (-)
- a period (.)

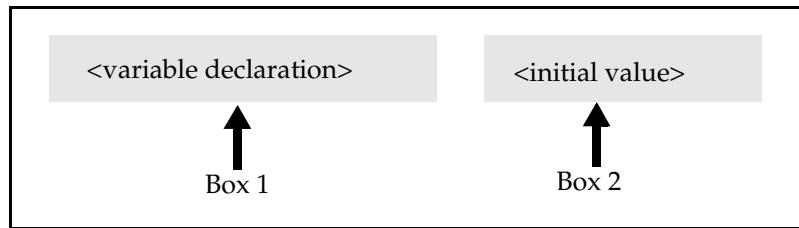
3 Variable name interpretation is case sensitive.

Naming a Condition Using the Let Rule

You can use the Let rule in combination with the If rule to define a condition and then test that condition before executing a Collaboration Rules component.

Syntax

The following syntax is used in the Let rule text boxes:



where:

variable declaration

Variable that represents the condition to be tested before Collaboration Rules are executed. See “[Naming Variables in the Let Rule](#)” on page 438 for a list of rules to keep in mind when naming variables in the Let rule.

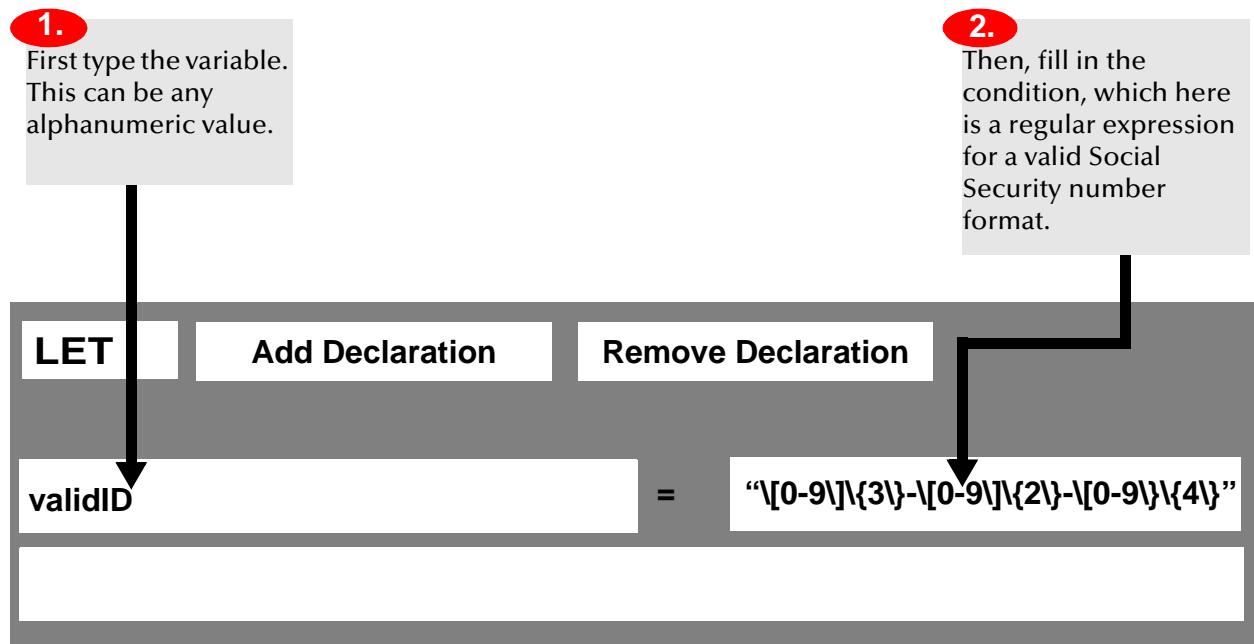
initial value

Condition to be tested. This can be any Monk expression.

See “[Constructing the Let Rule](#)” on page 440 for an example of how to construct the Let rule to name a condition.

Constructing the Let Rule

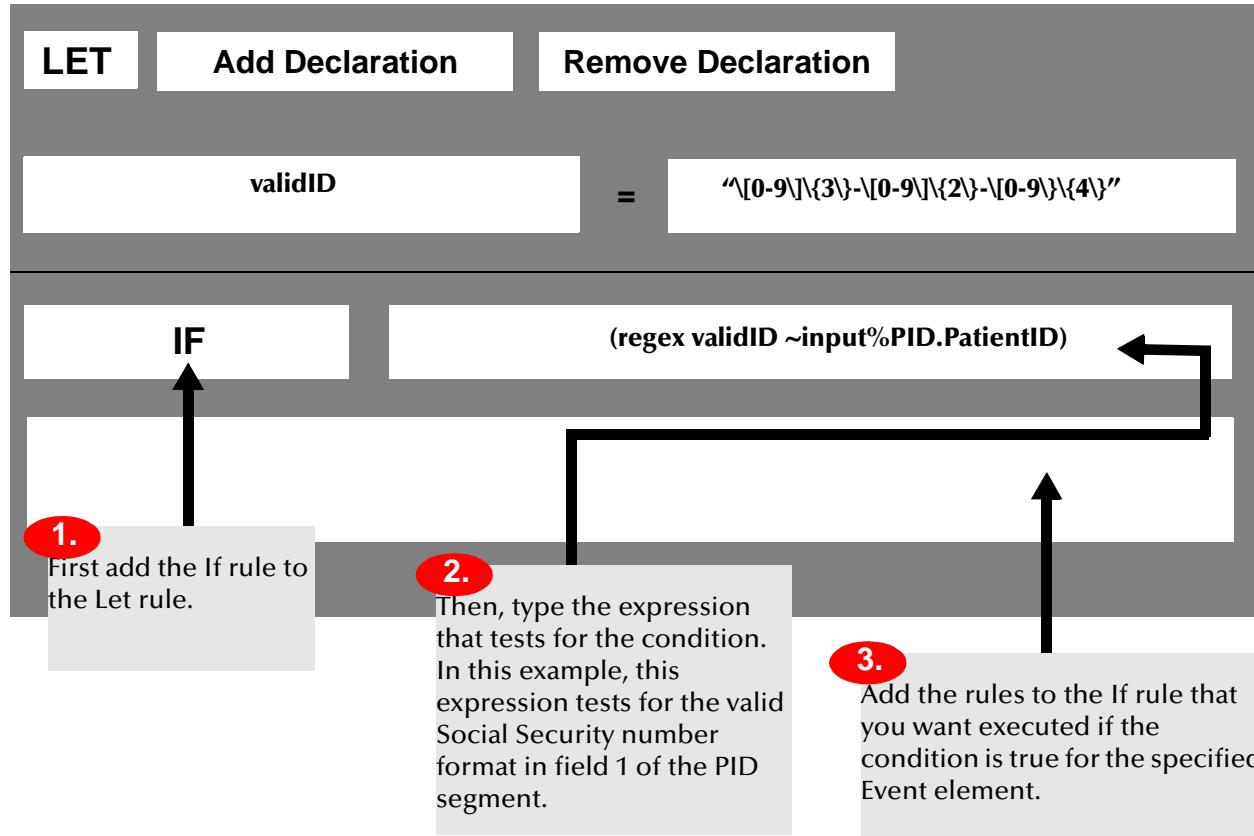
Use this graphic to help you define a condition using the Let rule. In this example, the condition is a valid Social Security number, which will be set up as a regular expression in the **Initial Value** field. Before filling out the Let rule text boxes, be sure to delete the sample text provided by the Collaboration Rules Editor.



Once you define the condition with the Let rule, you can set up the If rule to test for that condition. Continue to “[Constructing the If Rule](#)” on page 441.

Constructing the If Rule

Use this graphic to help you construct an If rule to test for a condition defined in a Let rule. Before filling out the If-rule text box, be sure to delete the sample text provided by the Collaboration Rules Editor.



Add rules to the **Else** portion of the If rule that will be executed when the condition is not met.

For more details about how to construct an If rule, see “[Using the If Rule](#)” on page 387.

Using the Let Rule to Specify a Variable in an Insert Rule

You can define a variable in a Let rule that represents a special Monk function, and then use that variable in an Insert rule to:

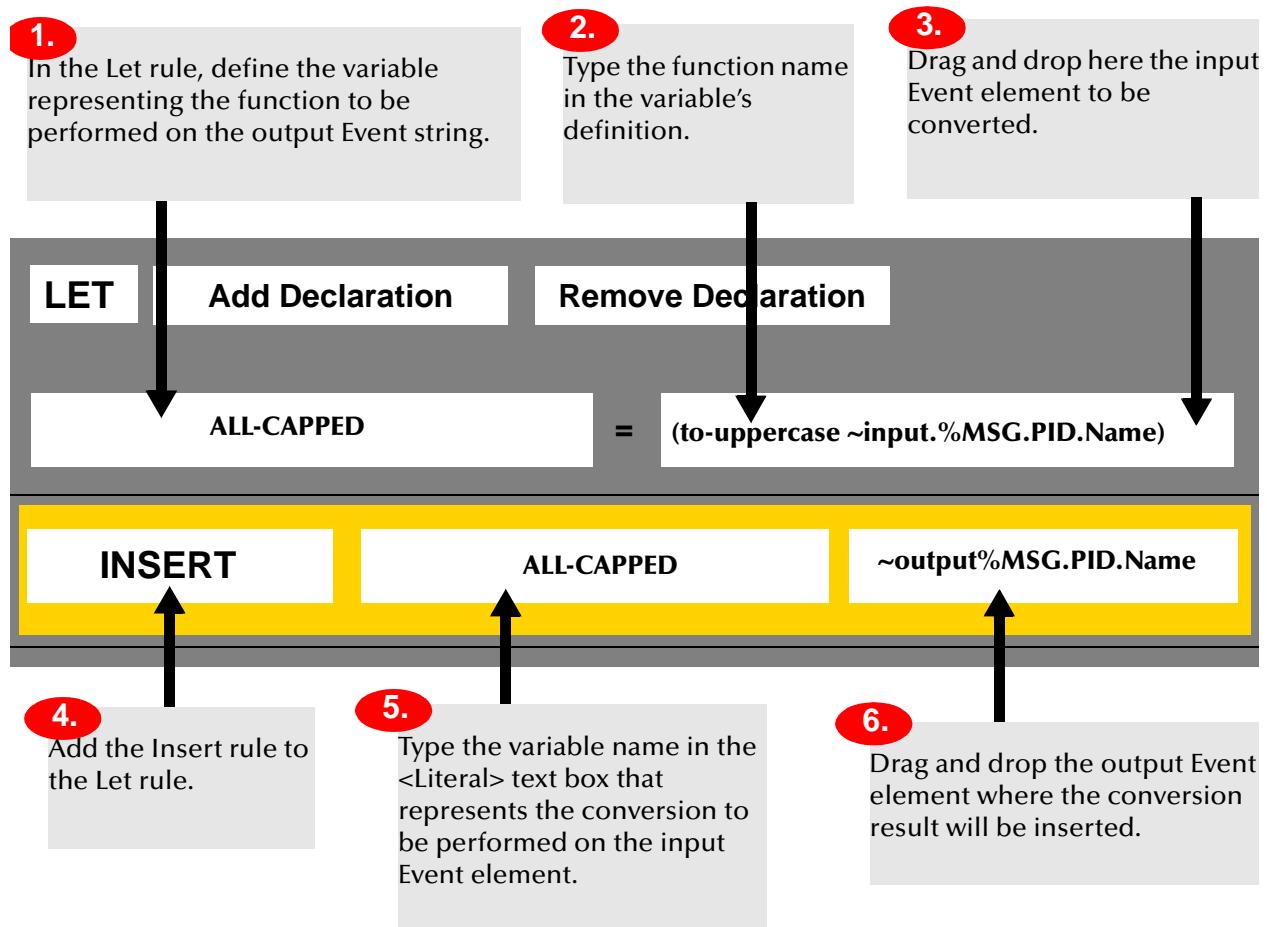
- Perform the function on input Event data
- Insert the result in the output Event.

One such Monk function converts input Event data to all uppercase characters in the output Event. This function is called **to-uppercase**, and is used in this section as an example.

Procedure

You first set up a Let rule, which contains the variable definition and input Event element to be converted. Then you set up an Insert rule, which contains the output Event location for the conversion result.

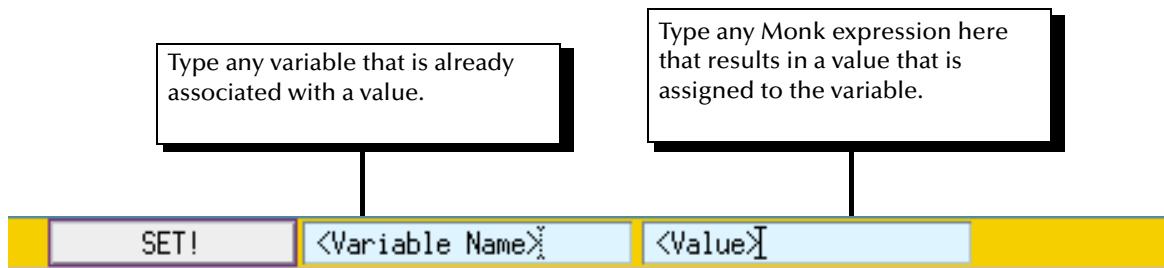
- 1 From the **Rules** menu, choose **Add Let**.
The Let rule appears in the **Collaboration**.
- 2 Select the Let rule by clicking on its Rule bar.
- 3 Now add the Verify rule inside the Let rule by choosing **Add Verify** from the **Rules** menu.
- 4 Set up the Let and Verify rules as shown in the graphic that follows.



8.5.15 Using the Set! Rule

The Set! rule allows you to change the value of a variable that has already been associated with a value. The variable whose value the Set! rule is changing can either have been set locally in the Let rule, or globally.

The Set! rule requires a special syntax to set it up. Like the If, Let, Loop, and Function rules, the Set! rule requires you to set up its special syntax directly in the Rule bar text boxes.



Adding a Set rule

- 1 From the **Rules** menu, select **Add Set!**.
- 2 Fill in the parameters in the Set! rule text boxes:

Variable Name

Name of the variable whose value you are changing

Value

Any Monk expression that results in a new value that is assigned to the variable. This expression changes the value of the variable that has been previously assigned.

If you are using the Set! rule to redefine a variable that was set up in a Let rule, you must place the Set! rule inside the Let rule.

If you are using the Set! rule to redefine a global variable, you can place the Set! rule by itself in the **Collaboration** in the **Collaboration Rules Editor**.

Caution: *Keep in mind that once you use the Set! rule to change the value of a variable, the previous value of the variable is lost.*

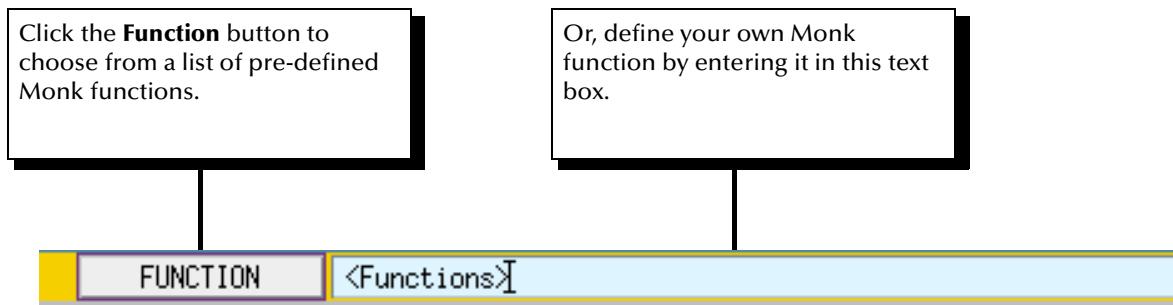
8.5.16 Using the Function Rule

The Function rule allows you to include in your Event ID any Monk function. Some prewritten functions are provided for you. In addition, you can use the Function rule to write your own functions.

This section shows you how to:

- Select a prewritten Monk function ([“Selecting a Prewritten Function” on page 444](#))
- Create your own function ([“Defining Your Own Function” on page 445](#)).

For detailed information about Monk functions, see the *Monk Developer’s Reference*.



Selecting a Prewritten Function

To add a Function rule and select a prewritten Monk function:

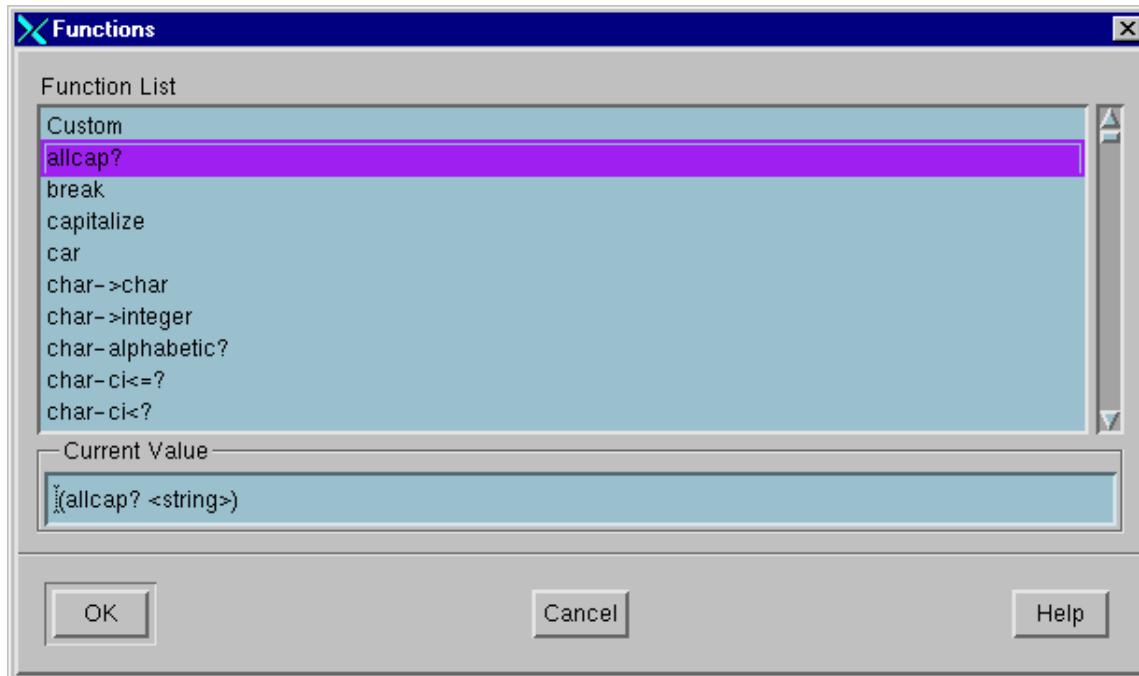
- 1 From the **Rules** menu, choose **Add Function**.

The Function rule is added to the **Collaboration**.

- 2 Click the **Function** button inside the Rule bar.

The **Functions** dialog box appears. This feature allows you to select a function to add to the Rules List. See Figure 192.

Figure 192 Functions Dialog Box



To use a predefined function

- Select the desired function from the **Function List**.

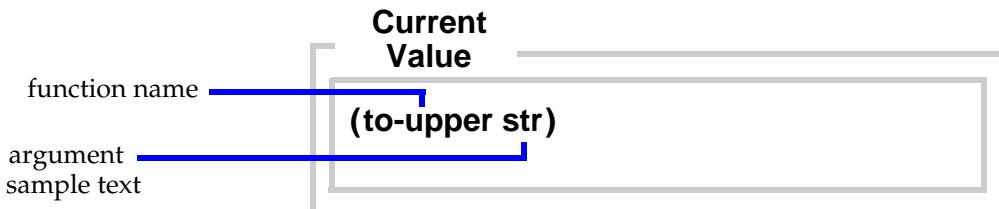
- In the **Current Value** text box, enter any arguments the function requires.
- Click **OK**.

To use a custom function

- From the **Function List**, select **Custom**.
- In the **Current Value** text box, enter the name of the function plus any arguments the function requires.
- Click **OK**.

3 In the **Function List** box, select a function.

The function and its arguments are shown below the list of functions in the **Current Value** text box. This is where you need to define your value(s) for the function.



4 Select the argument sample text. Type or drag and drop your argument(s). When specifying a ETD node as an argument, click and hold the middle mouse button on the node, drag the mouse cursor to the selected argument text, and release the button. The argument text is replaced with a reference to the selected node.

Make sure the argument sample text is completely replaced by a real argument. If there is more than one argument, you must separate each argument with a space.



5 Click the **OK** button to insert the function in the Function Rule bar and dismiss the dialog box.

Defining Your Own Function

To add a Function rule and define your own Monk function

1 From the **Rules** menu, select **Add Function**.

The Function rule is added to the **Collaboration**.

2 Select the <Functions> text in the Rule bar and delete it. Type the name of the Monk function in the text box.

For detailed information about Monk functions and syntax, see the *Monk Developer's Reference*.

8.5.17 Using the User Function Rule

The User Function rule allows you to define your own Monk functions. In effect, you create a named subroutine composed of one or more calls to existing functions.

To create a user function rule

- 1 On the Rules menu, click **Add User Function**.

Figure 193 User Function Rule



- 2 Supply the name and argument list for the function you are creating.
- 3 Add function calls, logic, and comments as needed.

For more information on using Monk functions, see the *Monk Developer's Reference*.

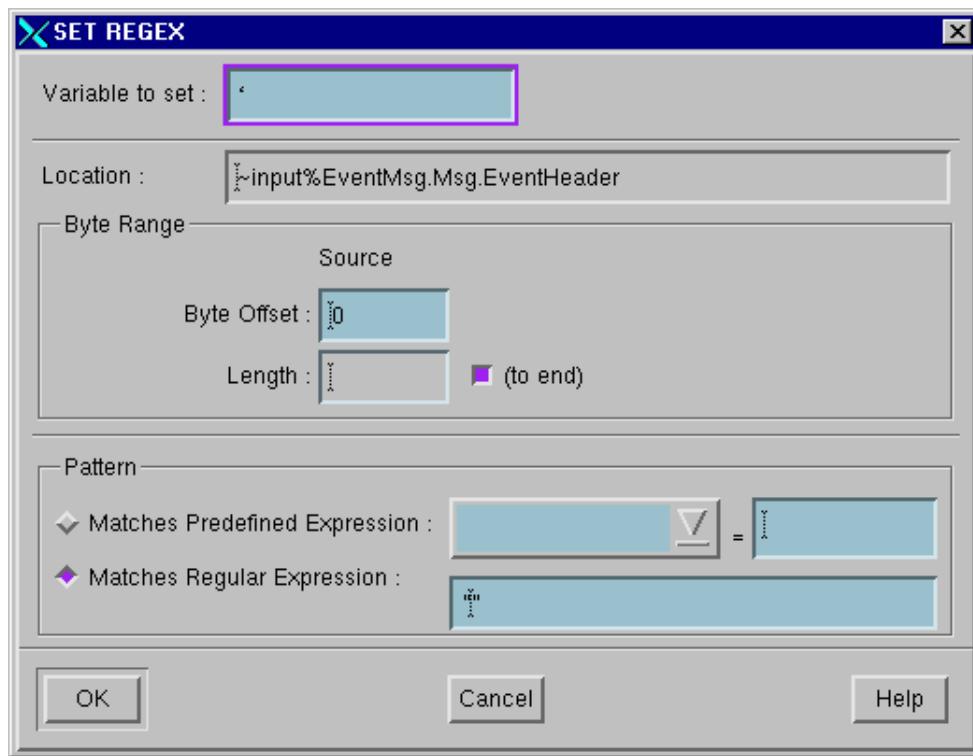
8.5.18 Using the Set Regex Rule

The Set Regex rule allows you to define a variable whose value is a regular expression. For example, you might create a variable named USAPhoneNumber composed of 0-2 access digits, followed by a three-digit area code, followed by a three-digit exchange, followed by a four-digit number, followed by 0-5 digits constituting an extension. If you used this example without deviation, you could use a predefined expression named "Phone Number" (giving it any variable name that suits your purpose). A few of the other predefined variable names include: Name; Social Security Number; ZIP code; Time HH:MM:SS.

To create a Set Regex Rule

- 1 On the Rules menu, click **Add Set Regex Rule**.
- 2 Click the **Set Regex** button and use the Set Regex dialog box to define a name, location, byte range, and pattern for your variable. See Figure 194.

Figure 194 Set Regex Dialog Box



- In the **Variable to set** box, enter the name of the variable to be set based on the evaluation of this regular expression. **The variable must be defined in the box with a back quote (').** (When the variable is used in other rules, the back quote should be omitted.)
 - In the **Location** box, enter the Event node against which to test the expression.
 - In the **Byte Offset** box, enter the beginning byte location of the data you are copying to the output Event. Bytes are numbered starting at zero.
 - To specify a byte length for the source Event, clear the **(to end)** check box, then enter a byte length in the **Length** box. The minimum byte length is 1.
 - Under **Pattern**, do one of the following:
 - Choose **Matches Predefined Expression** to use a pre-programmed expression, and then click to select from the list of available expressions.
 - Choose **Matches Regular Expression**, then enter a regular expression.
- 3 When you have finished defining the regular expression, click **OK**.

Working with e*Ways

This chapter explains e*Way Intelligent Adapter general operation, how to use the e*Way Configuration Editor feature, and the basics of how to configure e*Ways in the e*Gate system. It also explains information to be entered when setting up a Business Object Broker (BOB) component.

9.1 Overview of e*Way Operation

e*Ways provide points of contact between the e*Gate system and external applications. e*Ways handle the communication details necessary to send and receive information, including:

- Responding to or generating positive and negative acknowledgments.
- Rules that govern resend and/or reconnect criteria.
- Time-out logic.
- Data envelope parsing and reformatting.
- Buffer size.
- Retrieval/transmission schedules.

In addition to handling communications, e*Ways are also able to apply business logic within Collaboration Rules to perform any of e*Gate's range of data identification/manipulation/transformation operations.

e*Ways are tailored to meet the communication requirements of a specific application or protocol. SeeBeyond also makes available a "generic" e*Way, which you can extend using the Monk programming language to handle custom communications requirements. For more information, see "[e*Way Operation](#)" on page 123.

The intention of this chapter is to help you understand how to implement and configure e*Ways within the e*Gate Enterprise Manager.

Topics discussed include:

- The components that comprise an e*Way.
- How to create and configure e*Ways using the e*Gate Enterprise Manager.
- How to set e*Way parameters using the e*Way Configuration Editor.
- Troubleshooting tips.

9.1.1 Component Parts

Functionally, each individual e*Way contains the following component parts:

- **Executable Component:** An .exe file, this component is the engine of the e*Way; it does the work necessary to send, receive, and process data.
- **Configuration Files:** These .cfg files store the parameters that govern the e*Way's functions. For example, the configuration for a TCP/IP e*Way specifies the port numbers to send and receive data; the configuration for a file-based e*Way specifies the name of the directory to poll for input data.
- **Library Files:** These files (such as .dll files under Windows) support the operations that the executable component and functions require.
- **Function Definitions:** Depending on the e*Way, these functions can be written in C, Java, or Monk.

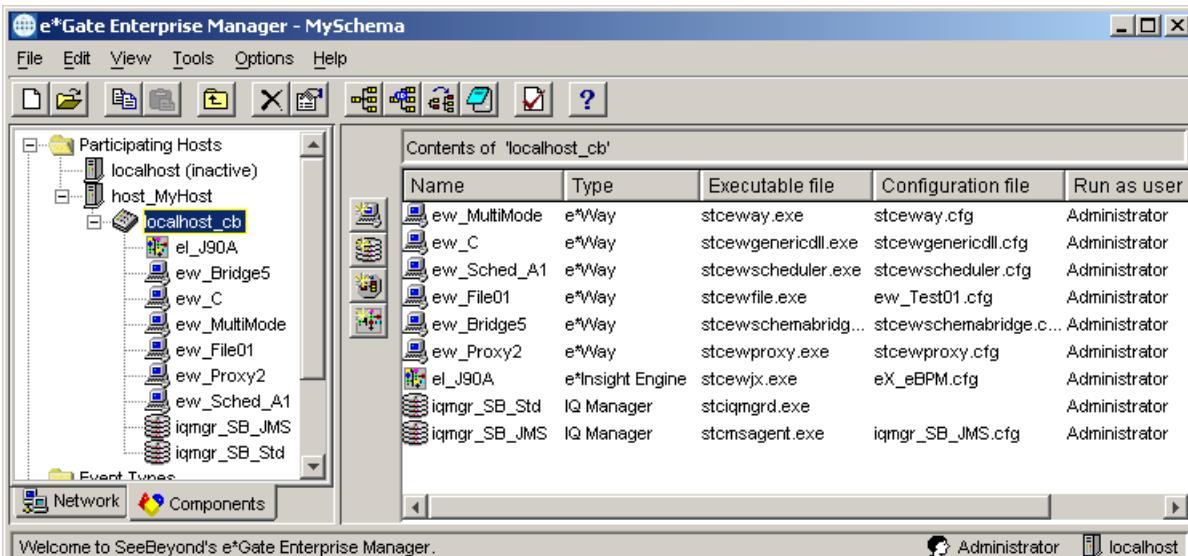
Library files are loaded automatically by the script or executable file that calls them. All the other components listed above are associated with the e*Way using either the Enterprise Manager or the e*Way Configuration Editor.

Note: In e*Gate, BOBs operate like e*Ways, except that their function is entirely internal. Configure BOBs in the same way as e*Ways. For more information on working with BOBs, see “[Adding Business Object Brokers](#)” on page 130.

9.1.2 e*Ways and the Enterprise Manager

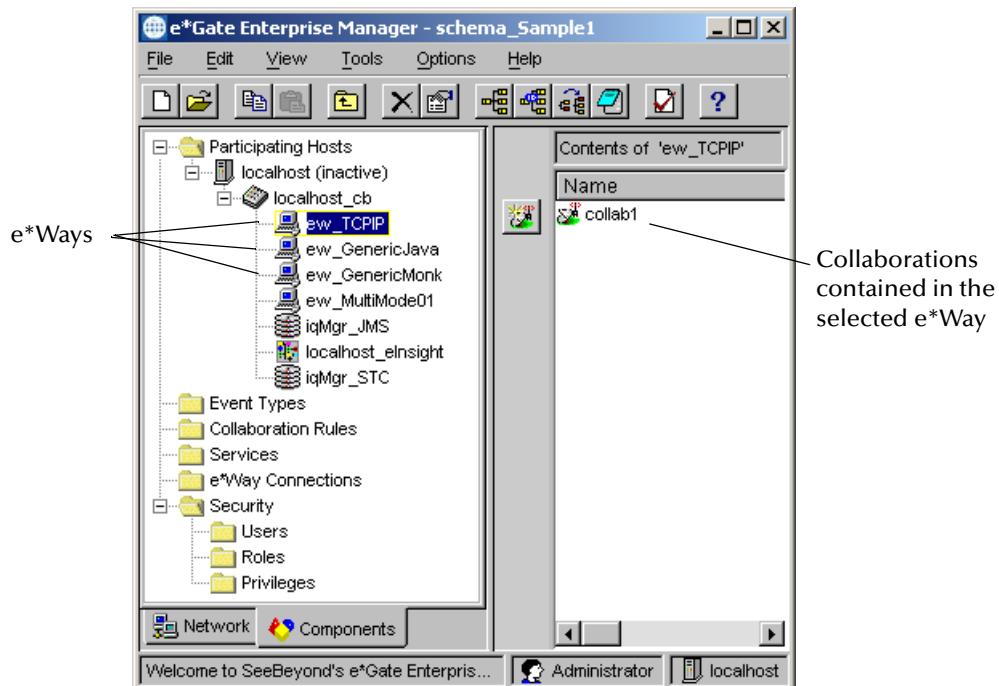
The previous section showed that e*Ways are combinations of many elements operating in concert. However, from the perspective of the Enterprise Manager, each e*Way is a single component, supervised by a Control Broker. An e*Way derives its character from the e*Way executable file (**stcew*.exe**) it uses and the particular configuration file created and tailored for this particular e*Way. See Figure 196.

Figure 195 Enterprise Manager View of e*Way Characteristics



All parts of the e*Way, including the executable file, its configuration file, its Monk scripts and library files (if any), are either properties of the e*Way component or called by elements of those properties. The logic that the e*Way executes to process information is carried out by Collaborations assigned to each e*Way.

Figure 196 Enterprise Manager View of e*Way Contents



The procedures required to create and configure the e*Way component within the Enterprise Manager are discussed in "[Configuring e*Ways with the Enterprise Manager](#)" on page 450. For an explanation of the e*Way Configuration Editor, which is used to configure the operating parameters of the e*Way itself, see "[Configuring e*Ways](#)" on page 457.

9.2 Configuring e*Ways with the Enterprise Manager

This section explains how to define and configure e*Way components within the Enterprise Manager.

9.2.1 Defining e*Way Components

The first step in implementing an e*Way in the Enterprise Manager is to define the e*Way component.

To create an e*Way

- 1 Select the Navigator's **Components** tab.
- 2 Open the host on which you want to create the e*Way.

- 3 Select the Control Broker you want to manage the new e*Way.
- 4 On the Palette, click . The **New e*Way Component** dialog box opens.
- 5 Enter the name of the new e*Way, then click **OK**.

9.2.2 Modifying e*Way Properties

Once an e*Way has been defined, you can modify its properties to do any of the following operations:

- Select an executable file.
- Select or create a configuration file.
- Change command-line parameters.
- Change the e*Gate user name under which the e*Way runs.
- Determine whether the Control Broker starts the e*Way immediately or on a startup/shutdown schedule.
- Activate or modify logging options.
- Activate or modify monitoring thresholds.

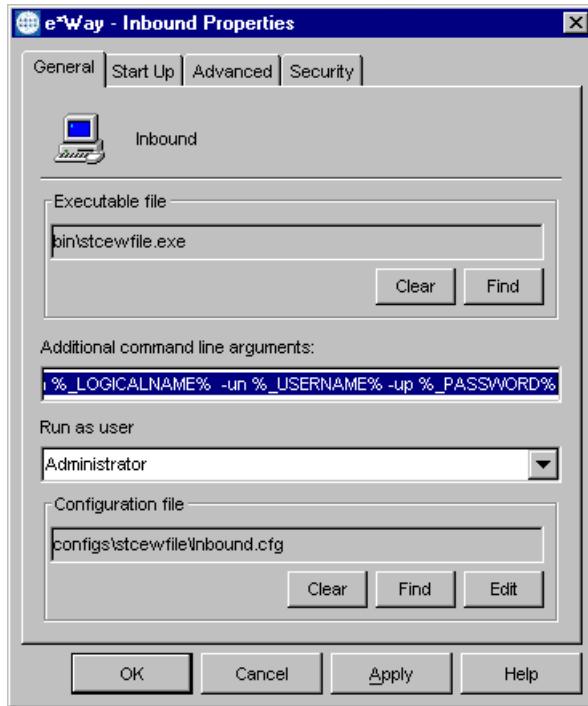
Note: *Each of these operations is described in a separate section later in this chapter.*

To modify an e*Way's properties

- 1 Select the Navigator's **Components** tab.
- 2 Open the Participating Host on which the desired e*Way runs.
- 3 Open the Control Broker that manages the e*Way.
- 4 Select the desired e*Way.
- 5 On the toolbar or **Edit** menu, click  **Properties** to edit the e*Way's properties.

The Properties dialog box for the e*Way opens to the **General** tab. See Figure 197.

Figure 197 e*Way Properties Dialog Box, General Tab



- 6 Make the desired modifications, then click **OK**.

Note: When you shut down an e*Way and open its Properties dialog box in the Enterprise Manager, once you click **OK** or **Apply**, the e*Way immediately restarts. This action only happens if the e*Way is in autostart mode. After you click **OK** or **Apply**, the Registry is automatically updated with any changes, if you made them using the e*Way Configuration Editor (see “[Configuring e*Ways](#)” on page 457).

Important: Selecting the executable file is the first configuration procedure you do once you have created the e*Way.

Selecting an Executable File

Selecting the executable file is the first and *most* important step in configuring the e*Way. This step determines the type of e*Way and what type of external system or communications protocol it supports.

Important: You must know which executable file to select before you perform this procedure.

To select an executable file for an e*Way

- 1 Display the e*Way’s Properties dialog box (see the [procedure on page 451](#)).
- 2 Under the **General** tab, under **Executable file**, click **Find**.
- 3 Use the **File Selection** dialog box to select the executable files. All e*Way executable files have an .exe extension.

Important: You must use the **Find** button to select the executable file. You cannot type its name directly into the **Executable file** text box. The file name depends on the type of e*Way installed. See the user's guide for the desired e*Way for details.

Creating or Selecting a Configuration File

After you have selected an executable file (see the previous procedure), you must select or create a configuration file that contains the operating parameters for the e*Way.

To create a configuration file

- 1 Display the e*Way's Properties dialog box (see the [procedure on page 451](#)).
- 2 Under the **General** tab, under **Configuration file**, click **New**.
The e*Way Configuration Editor window opens (discussed in detail in ["Configuring e*Ways" on page 457](#)).
- 3 Use the e*Way Configuration Editor window to change configuration parameters as required.
- 4 Save the configuration file and exit the e*Way Configuration Editor.

The name of the configuration file you just created appears automatically in the **Configuration file** text box. If for some reason it does not, use the procedure that follows to select it.

To select an existing e*Way configuration file

- 1 Display the e*Way's Properties dialog box (see the [procedure on page 451](#)).
- 2 Under the **General** tab, under **Configuration file**, click **Find**.
- 3 Use the **File Selection** dialog box to select the configuration file. All e*Way configuration files have a **.cfg** extension.

Note: You must use the **Find** button to select the configuration file. You cannot type its name directly into the **Configuration file** text box.

Changing Command-line Parameters

Most e*Ways provided by SeeBeyond require only the default command-line parameters shipped with the Enterprise Manager. Use the procedure in this section only if the e*Way you are configuring requires special command-line options, or if you are directed to do so by SeeBeyond support personnel.

To change an e*Way's command-line options

- 1 Display the e*Way's Properties dialog box (see the [procedure on page 451](#)).
- 2 Under the **General** tab, edit the **Additional command line arguments** text box to include the required arguments. Unless you have a specific need to do so, do not change any of the existing parameters.

To add new parameters to the command-line options

You can also add new parameters to the end of the command line. This feature allows you to automate the process of configuring your e*Ways.

For example, if you have five instances of e*Gate running on the same box, you could add “-rp %_REGPORT%” to the command line, which in turn instructs **stccb** to grab the values listed in the text box with all required parameters when the “-rp” command is encountered. To do this:

- 1 Display the e*Way’s Properties dialog box (see the [procedure on page 451](#)).
- 2 Under the **General** tab, type **-rp %_REGPORT%** in the **Additional command line arguments** text box at the end of the string.

Note: Use this procedure to add new parameters to BOB, Multi-Mode e*Way, and IQ Manager command-line options.

Follow the above procedure to add other arguments to the **Additional command line arguments** text box, such as “-rp %_PORT”

To use “-rp %_REGPORT%” for all components in any schema

To use “-rp %_REGPORT%” for all components created in any schema, do the following to change the setting:

- 1 Edit the **default.txt** file in the **egate/server/registry/repository/import** directory to include “-rp %_REGPORT%” for each desired module under the table **!REG_TBL_MODULE_ID**.
- 2 Import the **default.txt** file, overlaying the changes on the existing default settings.
All new e*Ways, BOBs, Multi-Mode e*Ways, or IQ Managers created after the **default.txt** file was changed will have “-rp %_REGPORT%” in the **Additional command line arguments** text box.

Changing the “Run As” User Name

Like all e*Gate executable components, e*Ways run under an e*Gate user name. By default, all e*Ways run under the “Administrator” user name. You can change this if your site’s security procedures so require.

To change the “run as” user name

- 1 Display the e*Way’s Properties dialog box (see the [procedure on page 451](#)).
- 2 Under the **General** tab, open the **Run as user** list and select the e*Gate user under whose name you want this component to run.

Note: See the *e*Gate Integrator System Administration and Operations Guide* for more information on the e*Gate security system.

Setting Startup Options or Schedules

e*Ways can be started or stopped by any of the following methods:

- The Control Broker can start the e*Way automatically whenever the Control Broker starts.
- The Control Broker can start the e*Way automatically whenever it detects that the e*Way terminated execution abnormally.

- The Control Broker can start or stop the e*Way on a schedule that you specify.
- Users can start or stop the e*Way manually using an interactive monitor.

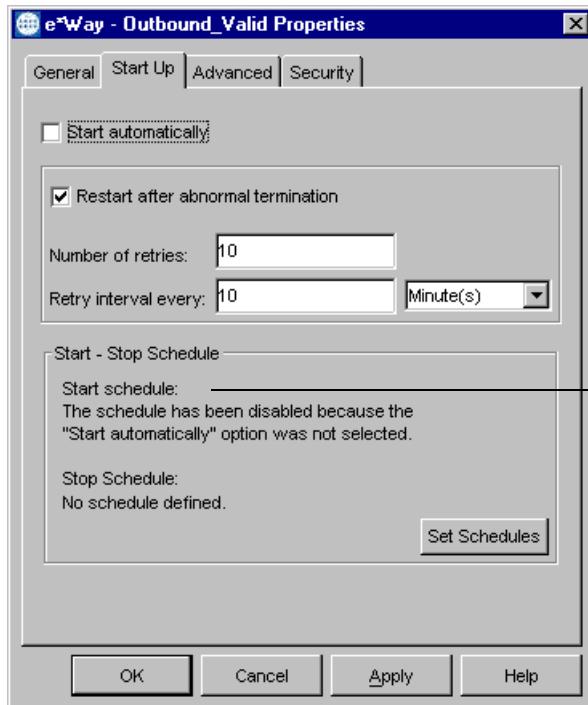
Use the options on the **Start Up** tab on the e*Way's Properties dialog box to determine how the Control Broker starts or shuts down an e*Way. See the *e*Gate Integrator Alert and Log File Reference Guide* for more information about how interactive monitors can start or shut down components.

Caution: *Do not try edit an ETD on a machine while it is running an e*Way or BOB that uses that ETD. Trying to edit an ETD on the same machine that is using the ETD in a running e*Gate module can destabilize both the Editor and the module.*

To set the e*Way to start automatically when the Control Broker starts

- 1 Display the e*Way's Properties dialog box (see the [procedure on page 451](#)).
- 2 Select the **Start Up** tab (shown in Figure 198 below).

Figure 198 e*Way Properties Dialog Box, Start Up Tab



After checking the **Start automatically** check box, the text under "Start schedule" changes to: "This component will start when the Control Broker is started."

- 3 Check **Start automatically** to activate this feature.

Note: Clear the *Start automatically* check box to deactivate this feature.

- 4 Click **OK**.

To set the e*Way to restart automatically

- 1 Display the e*Way's Properties dialog box (see the [procedure on page 451](#)).
- 2 Select the **Start Up** tab (shown in Figure 198).

- 3 Check **Restart after abnormal termination** to activate this feature.

Note: Clear the **Restart after abnormal termination** check box to deactivate this feature.

- 4 If you checked the **Restart after abnormal termination** box, enter the appropriate information in the **Number of retries** and the **Retry interval every** text fields.
- 5 Click **OK**.

Note: The “auto restart” feature does not automatically restart the e*Way if the e*Way is shut down manually by an interactive monitor.

If the e*Way is shut down and you make any configuration changes using the Enterprise Manager, the Control Broker automatically restarts the e*Way when the configuration changes are recorded in the e*Gate Registry. If you do not want the e*Way to restart when configuration changes are made, disable this feature before configuring the e*Way. See the individual e*Way’s user’s guide for details.

Activating or Modifying Logging Options

Logging options enable you to troubleshoot problems with the e*Way, its assigned Collaborations, the Collaboration Rules it executes, the IQs to which the e*Way publishes Events, or its communication with the external application or system. Use these procedures to activate or modify logging options.

Important: Use logging options freely while developing and debugging schemas, but decrease the level of detail before you migrate the schema to a production environment:
Certain logging options and severity settings cause significant slowing of component performance and overall system throughput.

To activate or modify logging options

- 1 Display the Properties dialog box for the e*Way (see the [procedure on page 451](#)).
- 2 Select the **Advanced** tab.
- 3 Click **Log**.
- 4 Select the desired logging options and click **OK**.

See the *e*Gate Integrator Alert and Log File Reference Guide* for more information concerning log files, logging options, logging levels, and debug flags.

Activating or Modifying Monitoring Thresholds

Monitoring thresholds enable you to monitor the throughput of the e*Way. When the monitoring thresholds are exceeded, the e*Way sends a monitoring Event to the Control Broker. The system routes the monitoring Event to the e*Gate Monitor or any of a number of destinations, depending on the nature of the event.

For more information on the e*Gate Monitor, see [Chapter 10](#).

To activate or modify monitoring thresholds

- 1 Display the e*Way's properties dialog box (see the [procedure on page 451](#)).
- 2 Select the **Advanced** tab.
- 3 Click **Thresholds**.
- 4 Select the desired threshold options and click **OK**.

See the *e*Gate Integrator Alert and Log File Reference Guide* and [“Monitoring Resources and Performance” on page 500](#) for more information concerning threshold monitoring or routing specific notifications to specific recipients, or for general information about e*Gate’s monitoring and notification system.

9.2.3 e*Ways and Collaborations

After you have defined and configured the e*Way, you must define a Collaboration that executes the business logic which enables the e*Way to do its intended work. The Collaboration in turn executes a Collaboration script, containing the actual instructions to execute the business logic.

Your e*Way requires at least one Collaboration (more may be required, depending on the tasks you want the e*Way to perform). In turn, each Collaboration requires one or more IQs to which its processed Events are published.

For more information on IQs, see [“Intelligent Queuing Layer” on page 37](#) and [“Adding Intelligent Queues” on page 136](#). For more information on Collaborations, see [“Collaborations” on page 37](#) and [“Adding Collaborations” on page 142](#).

9.3 Configuring e*Ways

The e*Way Configuration Editor enables you to modify all the parameters of an e*Way that control how the e*Way communicates with an external application. This section explains how the Editor stores its configuration information and how to use the Editor’s controls to modify that information.

9.3.1 Concepts

The e*Way’s configuration parameters are stored in a **.def** ASCII text file. The e*Way Configuration Editor provides an easy way to view and change those parameters through a simple graphical interface, creating and maintaining the configuration file (**.cfg**) that you select when you configure the e*Way component in the Enterprise Manager (as discussed in [“Configuring e*Ways with the Enterprise Manager” on page 450](#)).

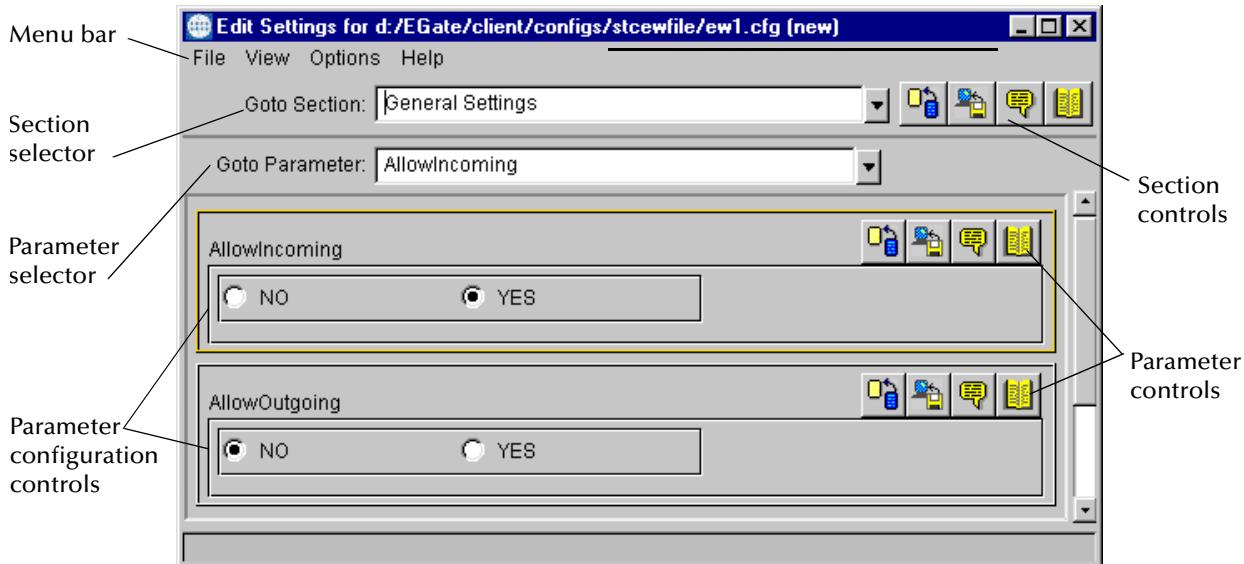
Because every e*Way functions differently to interface with a specific external application or communications protocol, every e*Way’s configuration parameters are different. See the appropriate e*Way user’s guide for instructions to complete the configuration of an individual e*Way.

The e*Way Configuration Editor uses a common set of controls to adjust the various configuration parameters specific to each e*Way. Once you learn how the e*Way Configuration Editor's controls work, you can easily edit any e*Way's configuration.

9.3.2 Controls

Figure 199 below shows a picture of the e*Way Configuration Editor window with its most important features indicated.

Figure 199 e*Way Configuration Editor GUI Map



The e*Way Configuration Editor controls are divided into the following categories:

- **Menu bar:** This feature provides access to basic operations: saving the configuration file, viewing a summary of all parameter settings, starting the Help system, and so on.
- **Section selector:** The e*Way's configuration parameters are grouped into sections. The section selector (labeled **Goto Section**) enables you to select the section whose parameters you want to edit.
- **Section controls:** The section controls enable you to restore the default settings, restore the last saved settings, display tips, or enter comments for the currently selected section.
- **Parameter selector:** Each section contains a list of parameters. Use the scroll bar to scroll through the entire list of parameters, or use the Parameter selector to jump to a specific parameter.
- **Parameter controls:** The parameter controls enable you to restore the default settings, restore the last saved settings, display tips, or enter comments for the currently selected parameter.

Note: Notice the similarity between the parameter controls and the section controls. The available operations are the same, but the scope differs.

- **Parameter Configuration Controls:** The Parameter configuration controls enable you to set the e*Way's various operating parameters.

Section and Parameter Controls

The section and parameter controls are shown in Table 69 below.

Table 69 Parameter and Section Controls

Button	Function
	Restores default values. There is no undo; however, the system issues a warning before issuing the command.
	Restores saved values. There is no undo; however, the system issues a warning before issuing the command.
	Enter user notes.
	Displays tips.

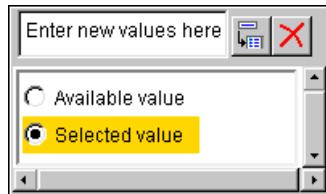
The Section controls for the **Goto Section** selector affect all parameters in the selected section; the controls within the parameter-editing section of the editor affect only the selected parameter.

Parameter Configuration Controls

Parameter configuration controls fall into one of two categories:

- Simple option buttons as shown in [Figure 199 on page 458](#) (setting the **AllowIncoming** and **AllowOutgoing** parameters).
- Selection lists as shown in Figure 200 below.

Figure 200 Sample Selection List Controls



Selection lists have two controls, described in Table 70 below.

Table 70 Selection List Controls

Button	Function
	Adds the value in the text box to the list of available values.
	Displays a Delete Items dialog box, used to delete items from the list.

9.3.3 Using the e*Way Configuration Editor

This section describes basic procedures you can use when working with the e*Way Configuration Editor.

To open the e*Way Configuration Editor

- Under the **General** tab on the e*Way properties dialog box, click **Edit**.
The e*Way Configuration Editor window opens.

Navigating Through the Editor

To display a list of all e*Way parameters and their settings

- On the **View** menu, click **Summary**.
- Use the controls on the Summary list to go to a selected parameter or print a list of parameter settings. When you have finished using the Summary list, click **Cancel**.

To view or edit a section of the e*Way configuration

- In the **Goto Section** list, select a section.

To view or edit a selected parameter

- In the **Goto Section** list, select the section that contains the parameter you want to edit.
- In the **Goto Parameter** list, select a parameter. If all the parameters are not visible, use the scroll bars to scroll to the desired parameter.

To navigate using the summary list

- On the **View** menu, click **Summary**.
- Select a parameter and click **Go To**.

Saving Configuration Settings

To save the current configuration settings

- On the **File** menu, click **Save**.

To promote the current configuration file to run time

- On the **File** menu, click **Promote to Run Time**.

Modifying Configuration Settings

To modify a configuration setting

- 1 Navigate to the parameter whose options you want to change.
- 2 Select the parameter from those that are provided. To add new options to a list of parameters (see [Figure 200 on page 459](#)), enter the option in the parameter's data-entry box and click .
- 3 To remove options from the list, click . A new dialog box displays, in which you do the actual deletions.

Restoring Default Settings

To reload the default current configuration settings for a parameter

- 1 Navigate to the parameter whose defaults you want to restore.
- 2 On the parameter-editing toolbar, click .

To reload the default current configuration settings for a section

- 1 Navigate to the section whose defaults you want to restore.
- 2 On the Section toolbar, click .

Restoring Saved Settings

To reload the saved current configuration settings for a parameter

- 1 Navigate to the parameter whose defaults you want to restore.
- 2 On the Parameter-editing toolbar, click .

To reload the saved current configuration settings for a section

- 1 Navigate to the section whose defaults you want to restore.
- 2 On the Section toolbar, click .

Entering User Notes

User Notes provide you with a means to “comment” your e*Way configuration.

To enter user notes regarding the e*Way's general configuration or operations

- On the **File** menu, click **User Notes**.

To enter user notes regarding a section of the e*Way configuration

- On the Section toolbar, click .

To enter user notes regarding a parameter within the e*Way configuration

On the Parameter-editing toolbar, click .

Creating Business Object Brokers

You can add BOBs to your e*Gate system, using the Enterprise Manager window and procedures similar to those used for e*Ways. BOBs are an optional feature in e*Gate.

For more information on working with BOBs, see “[Adding Business Object Brokers](#)” [on page 130](#).

Using the Online Help System

Use these features when you need help with the e*Way Configuration Editor.

Note: *The online Help provides information on all features discussed in this section.*

To start the e*Way Configuration Editor’s online Help system

- On the **Help** menu, click **Help topics**.

To display tips regarding the general operation of the e*Way

- On the **File** menu, click **Tips**.

To display tips regarding a section of the e*Way configuration

- On the Section toolbar, click .

To display tips regarding a parameter within the e*Way configuration

- On the Parameter-editing toolbar, click .

Note: *Tips are displayed and managed separately from the online Help system associated with the **Help** menu. You cannot search for Tips within the online Help system, can you cannot view online Help topics by requesting Tips.*

For a complete explanation of how to use the e*Gate online Help systems, see “[Online Help Systems](#)” [on page 73](#).

9.4 Troubleshooting e*Ways

This section provides guidelines for troubleshooting e*Way’s operation or performance. However, the range of e*Ways that SeeBeyond provides, and the range of e*Ways you can develop, makes it impossible for an introductory guide to present an exhaustive list of troubleshooting tips.

In the initial stages of developing your e*Gate system, most problems with e*Ways can be traced to configuration. For information on system-wide troubleshooting, see the *e*Gate Integrator Alert and Log File Reference Guide*.

9.4.1 In the Enterprise Manager

- Does the e*Way have the correct Collaborations assigned?
- Do those Collaborations use the correct Collaboration Services?
- Is the logic correct within any Collaboration Rules script employed by this e*Way's Collaborations?
- Do those Collaborations subscribe to and publish Events appropriately?
- Are all the components that "feed" this e*Way properly configured, and are they sending the appropriate Events correctly?
- Are all the components that this e*Way "feeds" properly configured, and are they subscribing to the appropriate Events correctly?

9.4.2 In the e*Way Configuration Editor

- Check that all configuration options are set appropriately.
- Check that all settings you changed are set correctly.
- Check all required changes to ensure they have not been overlooked.
- Check the defaults to ensure they are acceptable for your installation.

9.4.3 On the e*Way's Participating Host

- Check that the Participating Host is operating properly, and that it has sufficient disk space to hold the IQ data that this e*Way's Collaborations publish.

9.4.4 In the e*Way's External Application

- Check that the application is configured correctly, is operating properly, and is sending or receiving the correct data appropriately.
- Check that the connection between the external application and the e*Way is functioning appropriately.
- Once the e*Way is up and running properly, operational problems can be due to:
 - ◆ External influences (network or other connectivity problems).
 - ◆ Problems in the operating environment (low disk space or system errors)
 - ◆ Problems or changes in the data the e*Way is processing.
 - ◆ Corrections required to Collaboration Rules scripts that become evident in the course of normal operations.

One of the most important tools in the troubleshooter's arsenal is the e*Way log file. For more information on log files, see the *e*Gate Integrator Alert and Log File Reference Guide*.

In addition, see the *e*Gate Integrator Alert and Log File Reference Guide* for an extensive explanation of log files, debugging options, and how to use the e*Gate monitoring system to monitor both operations and performance.

9.5 Multi-Mode e*Way

A Multi-Mode e*Way is a multi-threaded component that extends the e*Way concept of routing and transforming data within e*Gate. It uses e*Way Connections to send and receive topics directly to and from multiple external systems and/or SeeBeyond JMS IQ Managers.

The e*Way Connections are gateways to external systems, allowing a single e*Way to adopt several configuration profiles simultaneously to communicate with external systems as well as IQs. e*Way Connection instances have a graphical representation that can be manipulated with the Collaboration Editor to configure a particular kind of interaction with the external system.

Multi-Mode e*Way Characteristics

Multi-Mode e*Ways have the following characteristics:

- **Adapting:** Multi-Mode e*Ways face in two or more directions, as they must interact with and adapt to multiple external systems. They normally communicate with e*Gate as well, but it is possible to configure a Multi-Mode e*Way so that it merely bridges between two or more external systems without bringing data into e*Gate.
- **Transporting:** Acting as “smart” gateways, Multi-Mode e*Ways direct the flow of multiple components of data in and out of e*Gate.
- **Collaborating:** Inbound and outbound e*Gate Collaborations reside in Multi-Mode e*Ways and form the core of their operation. They determine:
 - ◆ The routing (publishing/subscribing) of the Events they handle.
 - ◆ Any transformation of data as it passes through the Multi-Mode e*Way.

In e*Gate, e*Ways interact with Collaborations as follows:

- Every Multi-Mode e*Way requires at least one Collaboration, but it can have more than one.
- Every Multi-Mode e*Way Collaboration that publishes internal e*Gate Events requires at least one IQ.

Caution: While you are running a Multi-Mode e*Way, do not use the same machine to edit any ETD involved in the e*Way—if you do, you will be unable to save the ETD changes and unable to use normal methods to halt either the Editor or the e*Way.

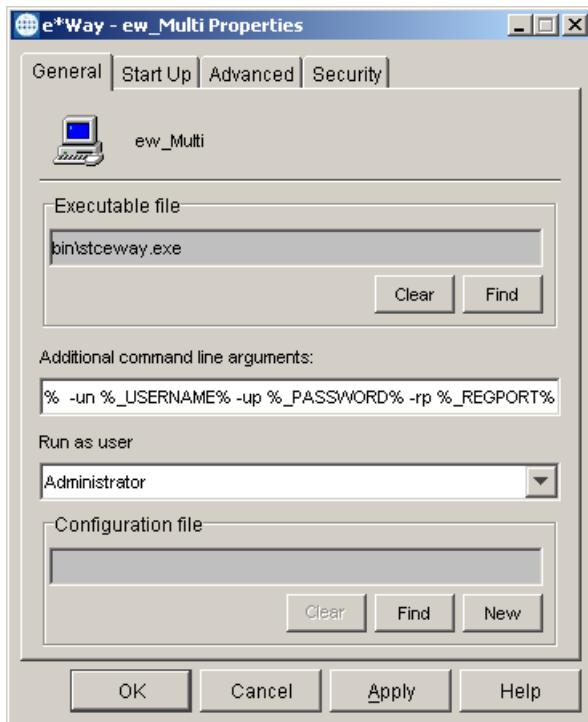
Multi-Mode e*Way properties are set using the Enterprise Manager.

To create and configure a New Multi-Mode e*Way

- 1 Select the Navigator’s Components tab.
- 2 Open the host on which you want to create the e*Way.
- 3 On the Palette, click the icon to create a new e*Way.
- 4 Enter the name of the new e*Way and click **OK**.
- 5 Select the e*Way and edit its properties.

The e*Way Properties dialog box opens to the General tab. See Figure 201.

Figure 201 e*Way Properties Dialog Box – General Tab



- 6 Under **Executable file**, click **Find**. Then, in the **File Selection** dialog box, navigate to **bin** (if necessary) and select **stceway.exe**.

Important: Multi-Mode e*Ways must use the **stceway.exe** executable.

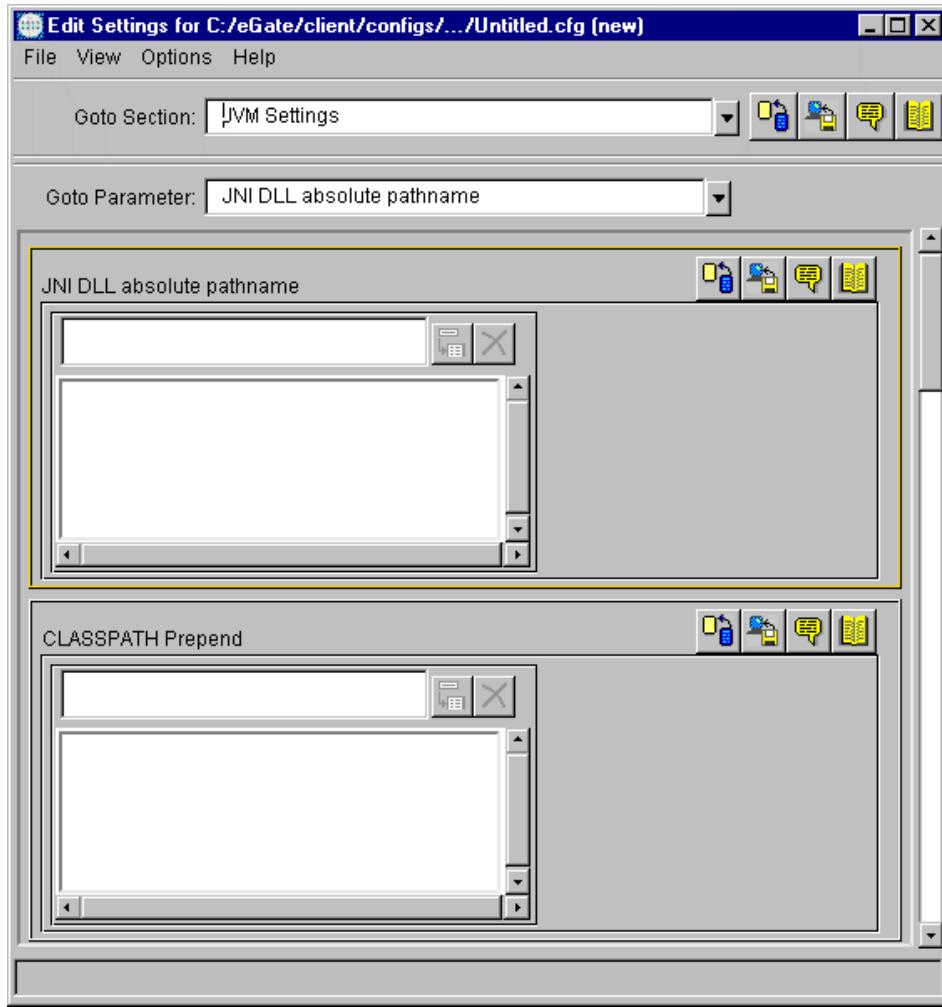
- 7 Click **Select** to register the selection and return to the e*Way Properties dialog box.
- 8 You do not normally need to add or change command-line arguments to be passed to the executable when it is started; see “[Changing Command-line Parameters](#)” on page 472.
- 9 You may want to change the type of User who runs this e*Way; see “[Changing the “Run As” User Name](#)” on page 473.
- 10 You may want to change the **Start Up**, **Security**, and **Advanced** settings for this Multi-Mode e*Way; see “[Setting Startup Options or Schedules](#)” on page 474 and “[Advanced Settings for Multi-Mode e*Ways](#)” on page 476.
- 11 After selecting the desired parameters, save the configuration file. Close the .cfg file and select **OK** to close the e*Way Properties dialog box.

After you have made the e*Way into a Multi-Mode e*Way by selecting executable file **stceway.exe** (see the previous procedure), you must select or create a configuration file to set the JVM Settings for the Multi-Mode e*Way.

9.5.1 JVM Settings

JVM settings control the basic Java Virtual Machine (JVM) settings on Multi-Mode e*Ways. These settings are the configuration parameters for this configuration file, and are organized under the **JVM Settings** (in the **Goto Section** at the top of the **e*Way Configuration Editor** window). See [Figure 209 on page 482](#) and Figure 202 below.

Figure 202 Top Portion of the Multi-Mode e*Way Configuration Editor Window



Important: The information entered in the fields on the e*Way Configuration Editor is e*Way-specific. Make sure you see the appropriate e*Way user's guide for specifics about what information should be entered in each field.

The JVM Settings control basic Java Virtual Machine settings.

Note: Although you can make changes to these configuration options while an e*Way is running, such changes only take effect when the e*Way is next restarted.

JNI DLL Absolute Pathname

Description

Specifies the absolute path name to where the JNI DLL (installed by the *Java 2 SDK*) is located on the Participating Host. This parameter is **mandatory**.

Required Values

A valid pathname.

Additional Information

The JNI DLL name varies on different platforms; see Table 74 below.

Table 71 Java 2 JNI DLL Names

Operating System	Java 2 JNI DLL Name
Windows NT 4.0, Windows 2000	jvm.dll
Solaris 2.6, 7, or 8	libjvm.so
Linux 6.2	libjvm.so
HP-UX 11.0 or 11i	libjvm.sl
AIX 4.3.3	libjvm.a

The value assigned can contain a reference to an environment variable, by enclosing the variable name within a pair of % symbols. For example:

%MY_JNIDLL%

Such variables can be used when multiple Participating Hosts are used on different platforms.

Note: To ensure that the JNI DLL loads successfully, the Dynamic Load Library (DLL) search path environment variable must be set appropriately to include all the directories under the Java 2 SDK (or JDK) installation directory that contain shared libraries (on UNIX) or DLLs (on Windows).

CLASSPATH Prepend

Description

Specifies the paths to be prepended to the CLASSPATH environment variable for the Java Virtual Machine settings.

Required Values

An absolute path or an environment variable. This parameter is optional.

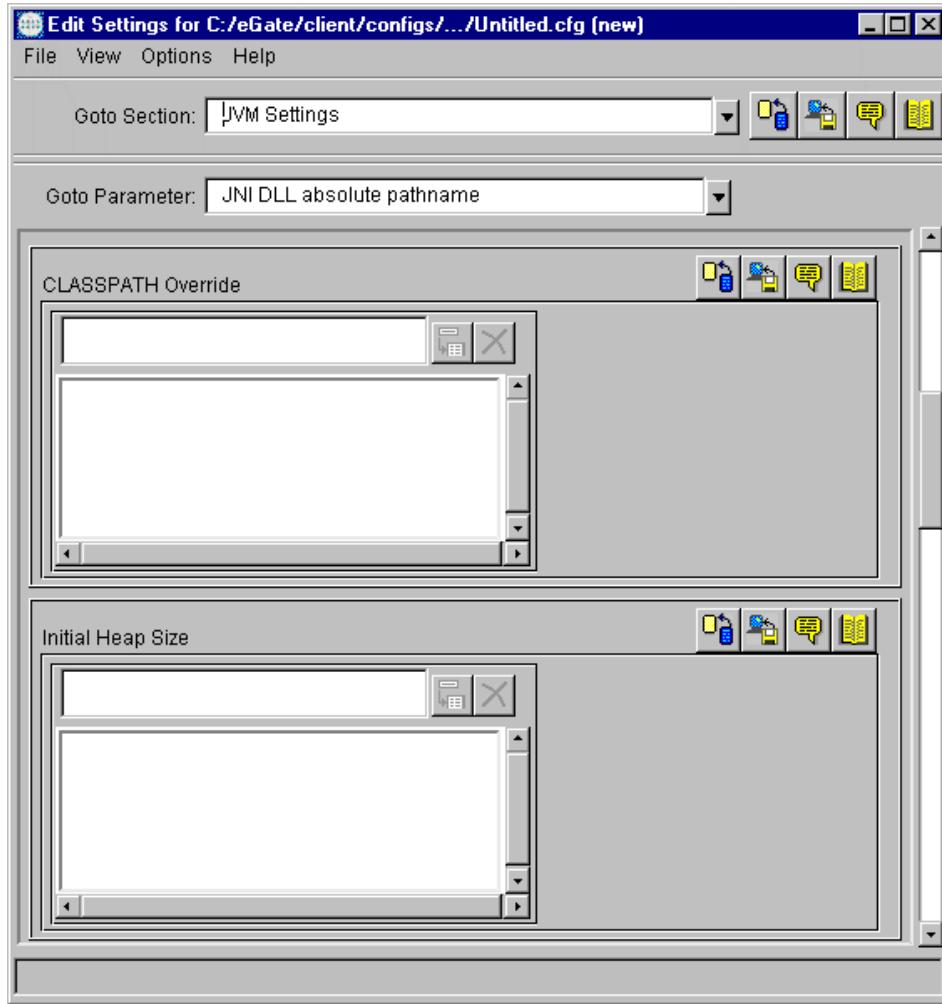
Additional Information

If left unset, no paths will be prepended to the CLASSPATH environment variable.

Existing environment variables can be referenced in this parameter by enclosing the variable name in a pair of % signs. For example:

%MY_PRECLASSPATH%

Figure 203 Second Portion of the Multi-Mode e*Way Configuration Editor Window



CLASSPATH Override

Description

Specifies the complete CLASSPATH variable to be used by the Java Virtual Machine settings, which is either an absolute path or an environment variable. This parameter is optional. If left unset, an appropriate CLASSPATH environment variable (consisting of required e*Gate components concatenated with the system version of CLASSPATH) will be set (see [Figure 210 on page 484](#)).

Note: All necessary .jar and .zip files needed by both e*Gate and the Java VM must be included. It is advised that the **CLASSPATH Prepend** parameter should be used.

Required Values

An absolute path or an environment variable. This parameter is optional.

Additional Information

Existing environment variables can be referenced in this parameter by enclosing the variable name in a pair of % signs. For example:

%MY_CLASSPATH%

Initial Heap Size

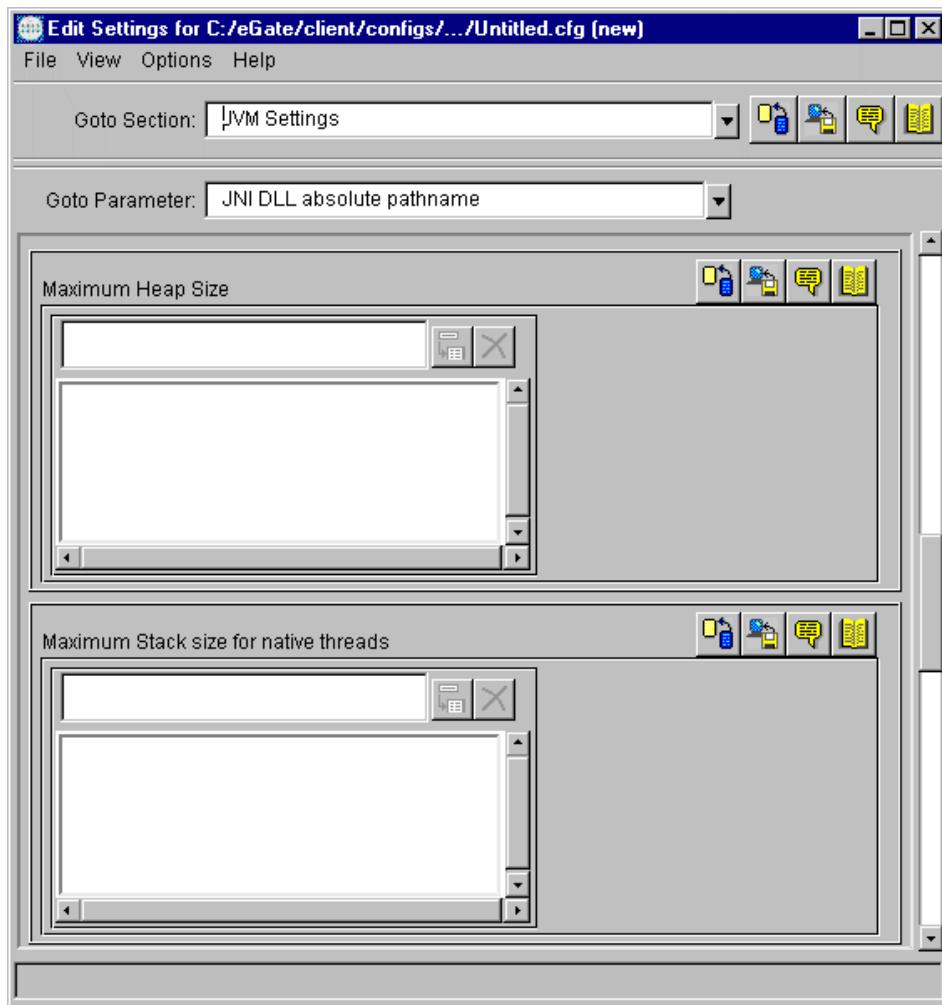
Description

Specifies the value for the initial heap size in bytes. If set to 0 (zero), the preferred value for the initial heap size of the Java VM will be used.

Required Values

An integer between 0 and 2147483647. This parameter is optional. (See [Figure 210 on page 484](#)).

Figure 204 Third Portion of the Multi-Mode e*Way Configuration Editor Window



Maximum Heap Size

Description

Specifies the value of the maximum heap size in bytes. If set to 0 (zero), the preferred value for the maximum heap size of the Java VM will be used.

Required Values

An integer between 0 and 2147483647. This parameter is optional (see [Figure 211 on page 485](#)).

Maximum Stack Size for Native Threads

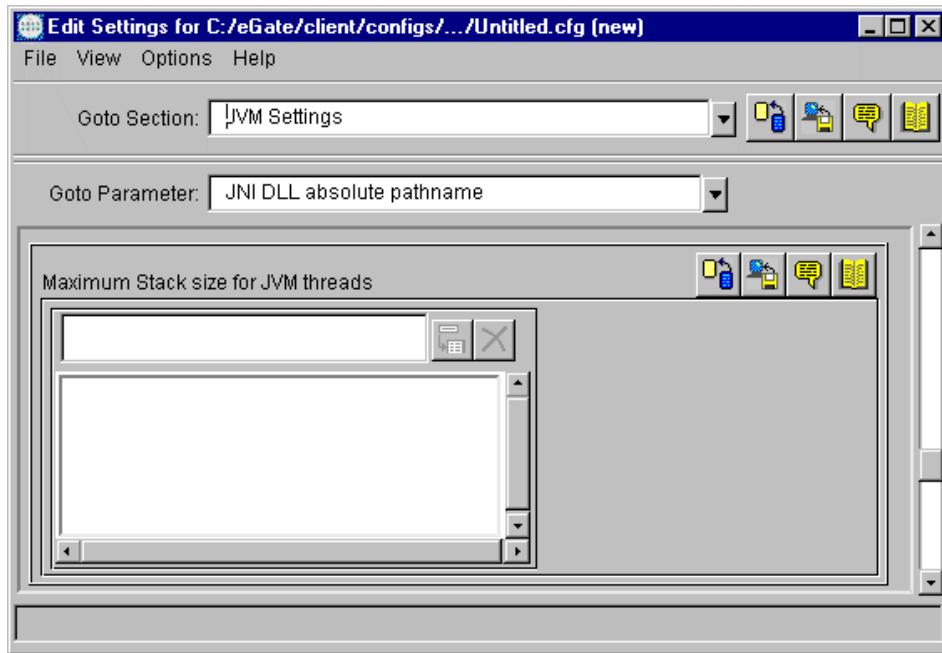
Description

Specifies the value of the maximum stack size in bytes for native threads. If set to 0 (zero), the default value will be used.

Required Values

An integer between 0 and 2147483647. This parameter is optional.

Figure 205 Fourth Portion of the Multi-Mode e*Way Configuration Editor Window



Maximum Stack Size for JVM Threads

Description

Specifies the value of the maximum stack size in bytes for JVM threads. If set to 0 (zero), the preferred value for the maximum stack size of the Java Virtual Machine will be used.

Required Values

An integer from 0 through 2147483647. This parameter is optional (see [Figure 211 on page 485](#)).

Note: The rest of the values are either "Yes" or "No."

Class Garbage Collection

Description

Specifies whether the Class Garbage Collection will be done automatically by the Java VM. The selection affects performance issues.

Required Values

YES or NO.

Garbage Collection Activity Reporting

Description

Specifies whether garbage collection activity will be reported for debugging purposes.

Required Values

YES or NO.

Asynchronous Garbage Collection

Description

Specifies whether asynchronous garbage collection activity will be reported for debugging purposes.

Required Values

YES or NO.

Report JVM Info and all Class Loads

Description

Specifies whether the JVM information and all class loads will be reported for debugging purposes.

Required Values

YES or NO.

Disable JIT

Description

Specifies whether the Just-In-Time (JIT) compiler will be disabled.

Required Values

YES or NO.

Note: This parameter is not supported for Java Release 1.

Remote debugging port number

Description

Specifies the port number by which the e*Gate Java Debugger can connect with the JVM to allow remote debugging. See “[Overview of e*Gate Java Debugger Operation](#)” on page 505.

Required Values

If specified, must be an unused port number in the range 2000 through 65535. If not specified, the e*Gate Java Debugger will be unable to connect to this e*Way.

Suspend option for debugging

Description

Allows you to specify that the e*Way should do no processing until an e*Gate Java Debugger has successfully connected to it. See “[Overview of e*Gate Java Debugger Operation](#)” on page 505.

Required Values

YES (causes the e*Way to suspend operation until a Debugger connects to it), or
NO (the default—the e*Way begins processing immediately upon startup).

Note: Although you can make changes to these configuration options while an e*Way is running, such changes only take effect when the e*Way is next restarted.

9.5.2 Changing Command-line Parameters

Most Multi-Mode e*Ways require only the default command-line parameters shipped with the Enterprise Manager. Use the procedure in this section only if the Multi-Mode e*Way you are configuring requires special command-line options, or if you are directed to do so by SeeBeyond support personnel.

To change a Multi-Mode e*Way’s command-line options

- 1 Display the e*Way’s Properties dialog box (see the [procedure on page 464](#)).
- 2 Under the **General** tab, edit the **Additional command line arguments** text box to include the required arguments. Unless you have a specific need to do so, do not change any of the existing parameters.

To add new parameters to the command-line options

You can also add new parameters to the end of the command line. This feature allows you to automate the process of configuring your Multi-Mode e*Way.

For example, if you have five instances of e*Gate running on the same box, you could add “-rp %_REGPORT%” to the command line, which in turn instructs **stccb** to grab the values listed in the text box with all required parameters when the “-rp” command is encountered. To do this:

- 1 Display the Multi-Mode e*Way Properties dialog box (see the [procedure on page 464](#)).
- 2 Under the **General** tab, append **-rp %_REGPORT%** in the **Additional command line arguments** text box at the end of the string.

Note: You can use this procedure to add new parameters to BOB, e*Way, Multi-Mode e*Way, and IQ Manager command-line options.

Follow the above procedure to add other arguments to the **Additional command line arguments** text box, such as “-rp %_PORT”

To use “-rp %_REGPORT%” for all components in any schema

To use “-rp %_REGPORT%” for all components created in any schema, do the following to change the setting:

- 1 Edit the **default.txt** file in the **<eGate>\Server\registry\repository\import** directory to include “-rp %_REGPORT%” for each desired module under the table **!REG_TBL_MODULE_ID**.
- 2 Import the **default.txt** file, overlaying the changes on the existing default settings.

All new e*Ways, BOBs, or IQ Managers created after the **default.txt** file was changed will have “-rp %_REGPORT%” in the **Additional command line arguments** text box.

9.5.3 Changing the “Run As” User Name

Like all e*Gate executable components, Multi-Mode e*Way run under an e*Gate user name. By default, all Multi-Mode e*Way run under the **Administrator** user name.

You can change this if your site’s security procedures so require.

To change the “run as” user name

- 1 Display the Multi-Mode e*Way Properties dialog box (see the [procedure on page 464](#)).
- 2 Under the **General** tab, open the **Run as user** list and select the e*Gate user under whose name you want this component to run.

Note: See the *e*Gate Integrator System Administration and Operations Guide* for more information on the e*Gate security system.

9.5.4 Setting Startup Options or Schedules

Multi-Mode e*Ways can be started or stopped by any of the following methods:

- The Control Broker can start the Multi-Mode e*Way automatically whenever the Control Broker starts.
- The Control Broker can start the Multi-Mode e*Way automatically whenever it detects that the Multi-Mode e*Way terminated execution abnormally.
- The Control Broker can start or stop the Multi-Mode e*Way on a schedule that you specify.
- Users can start or stop the Multi-Mode e*Way manually using an interactive monitor.

Use the options on the **Start Up** tab on the Multi-Mode e*Way's Properties dialog box to determine how the Control Broker starts or shuts down a Multi-Mode e*Way. See the *e*Gate Integrator Alert and Log File Reference Guide* for more information about how interactive monitors can start or shut down components.

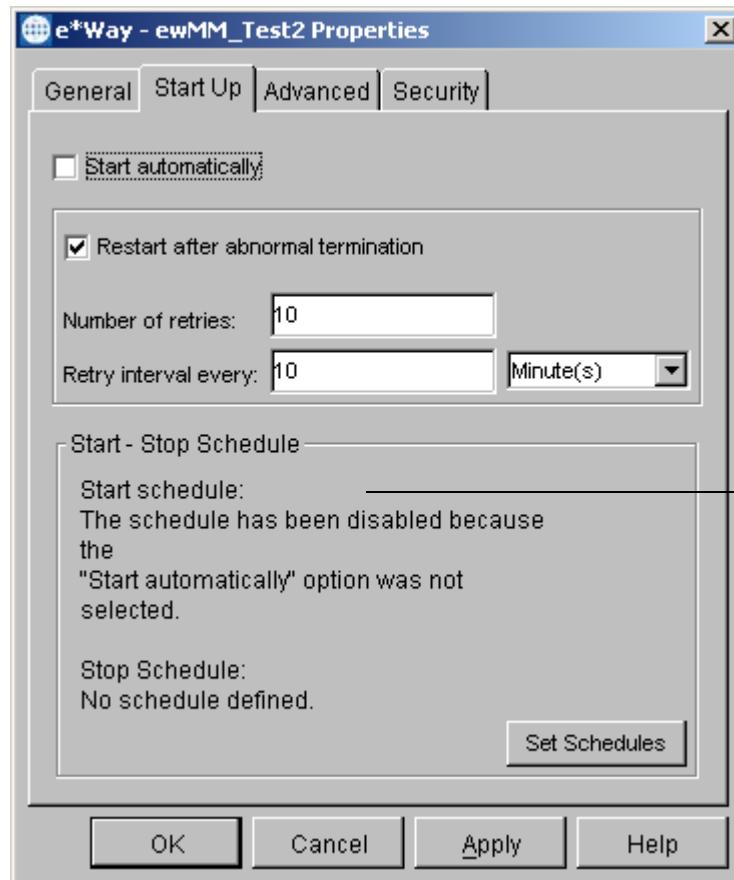
Caution: *Do not try edit an ETD on a machine while it is running a Multi-Mode e*Way that uses that ETD. Trying to edit an ETD on the same machine that is using the ETD in a running e*Gate module can destabilize both the Editor and the module.*

To set the Multi-Mode e*Way to start automatically when the Control Broker starts

- 1 Display the Multi-Mode e*Way's Properties dialog box (see the [procedure on page 464](#)).

- 2 Select the **Start Up** tab (see the following figure).

Figure 206 Multi-Mode e*Way Properties Dialog Box, Start Up Tab



- 3 Select the **Start automatically** check box to activate this feature.

Note: To deactivate this feature, clear the **Start automatically** check box.

- 4 Click **OK**.

To set the Multi-Mode e*Way to restart automatically

- 1 Display the Multi-Mode e*Way's Properties dialog box (see the [procedure on page 464](#)).
- 2 Select the **Start Up** tab (shown in [Figure 206 on page 475](#)).
- 3 Select the **Restart after abnormal termination** check box to activate this feature.

Note: To deactivate this feature, clear the **Restart after abnormal termination** check box.

- 4 If you checked the **Restart after abnormal termination** box, enter the appropriate information in the **Number of retries** and the **Retry interval every** text fields.
- 5 Click **OK**.

Note: The “auto restart” feature does not automatically restart the Multi-Mode e*Way if it is shut down manually by an interactive monitor.

If the Multi-Mode e*Way is shut down and you make any configuration changes using the Enterprise Manager, the Control Broker automatically restarts the Multi-Mode e*Way when the configuration changes are recorded in the e*Gate Registry. If you do not want the Multi-Mode e*Way to restart when configuration changes are made, disable this feature before configuring the Multi-Mode e*Way. See the individual e*Way’s user’s guide for details.

9.5.5 Advanced Settings for Multi-Mode e*Ways

This section explains additional settings available for the Multi-Mode e*Way component.

Activating or Modifying Logging Options

Logging options enable you to troubleshoot problems with the Multi-Mode e*Way, its assigned Collaborations, the Business Rules it executes, the IQs to which the Multi-Mode e*Way publishes Events, or the Multi-Mode e*Way’s communication with the external application or system. Use these procedures to activate or modify logging options.

Important: Use logging options freely while developing and debugging schemas, but decrease the level of detail before you migrate the schema to a production environment.

To activate or modify logging options

- 1 Display the Multi-Mode e*Way’s Properties dialog box (see the [procedure on page 464](#)).
- 2 Select the **Advanced** tab.
- 3 Click **Log**.
- 4 Select the desired logging options and click **OK**.

See the *e*Gate Integrator Alert and Log File Reference Guide* for more information on log files, logging options, logging levels, and debug flags.

Activating or Modifying Monitoring Thresholds

Monitoring thresholds enable you to monitor the throughput of the Multi-Mode e*Way. When the monitoring thresholds are exceeded, the Multi-Mode e*Way sends a monitoring Event to the Control Broker. The system routes the monitoring Event to the e*Gate Monitor or any of a number of destinations, depending on the nature of the Event.

For more information on the e*Gate Monitor, see [Chapter 10](#).

To activate or modify monitoring thresholds:

- 1 Display the Multi-Mode e*Way's Properties dialog box (see the [procedure on page 451](#)).
- 2 Select the **Advanced** tab.
- 3 Click **Thresholds**.
- 4 Select the desired threshold options and click **OK**.

See the *e*Gate Integrator Alert and Log File Reference Guide* and [“Monitoring Resources and Performance” on page 500](#) for more information concerning threshold monitoring or routing specific notifications to specific recipients, or for general information about e*Gate’s monitoring and notification system.

9.5.6 Configuring Multi-Mode e*Ways with e*Way Connections

The e*Way Connection GUI enables you to modify all the parameters of a Multi-Mode e*Way that control how the Multi-Mode e*Way communicates with an external application. This section explains how the Editor stores its configuration information and how to use the Editor’s controls to modify that information.

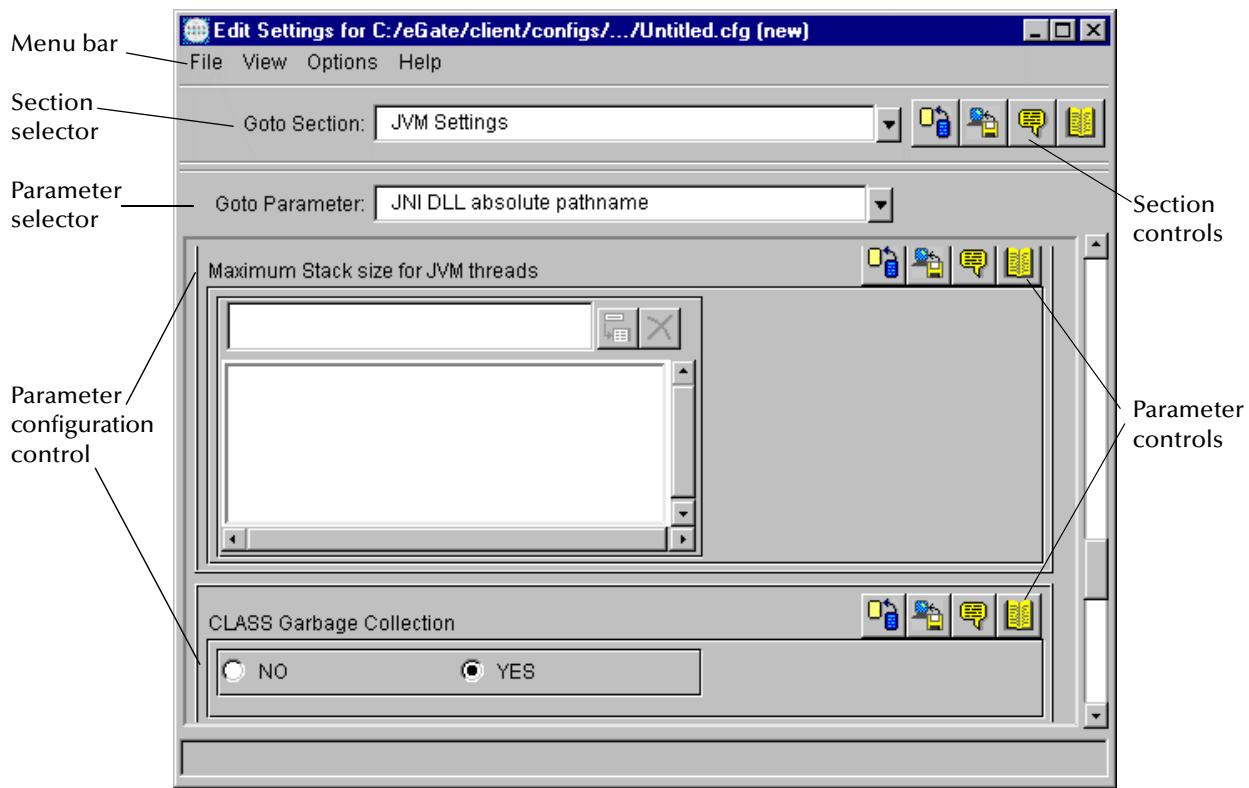
Because every Multi-Mode e*Way functions differently to interface with a specific external application or communications protocol, every e*Way Connection’s configuration parameters are different. See the appropriate e*Way user’s guide for instructions to complete the configuration of an individual e*Way Connection.

The e*Way Connection Editor uses a common set of controls to adjust the various configuration parameters specific to each e*Way Connection. Once you learn how the Editor’s controls work, you can easily edit any e*Way Connection’s configuration.

Controls

Figure 207 below shows a picture of the e*Way Connection Editor window with its most important features indicated.

Figure 207 e*Way Connection Editor Window Controls



The controls for the e*Way Connection Editor fall into the following categories:

- **Menu Bar:** This feature provides access to basic operations: saving the configuration file, view a summary of all parameter settings, starting the Help system, and so on.
- **Section Selector:** The e*Way Connection's configuration parameters are grouped into sections. The Section selector (labeled **Goto Section**) enables you to select the section whose parameters you want to edit.
- **Section Controls:** The Section controls enable you to restore the default settings, restore the last saved settings, display tips, or enter comments for the currently selected section.
- **Parameter Selector:** Each section contains a list of parameters. Use the scroll bar to scroll through the entire list of parameters, or use the Parameter selector to jump to a specific parameter.
- **Parameter Controls:** The Parameter controls enable you to restore the default settings, restore the last saved settings, display tips, or enter comments for the currently selected parameter.

Note: Notice the similarity between the parameter controls and the section controls. The available operations are the same, but the scope differs.

- **Parameter Configuration Controls:** The Parameter configuration controls enable you to set the e*Way Connection's various operating parameters.

Section and Parameter Controls

The section and parameter controls are shown in Table 72 below.

Table 72 Parameter and Section Controls

Button	Function
	Restores default values. There is no undo; however, the system issues a warning before issuing the command.
	Restores saved values. There is no Undo; however, the system issues a warning before issuing the command.
	Allows you to enter user notes.
	Displays tips.

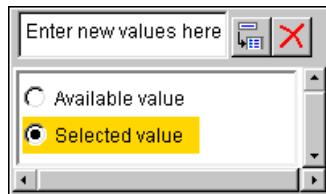
The Section controls for the **Goto Section** selector affect all parameters in the selected section; the controls within the parameter-editing section of the editor affect only the selected parameter.

Parameter Configuration Controls

Parameter configuration controls fall into one of two categories:

- Simple option buttons as shown in [Figure 199 on page 458](#).
- Selection lists as shown in Figure 208 below.

Figure 208 Sample Selection List Controls



Selection lists have two controls, described in Table 73 below.

Table 73 Selection List Controls

Button	Function
	Adds the value in the text box to the list of available values.

Table 73 Selection List Controls (Continued)

Button	Function
	Displays a Delete Items dialog box, used to delete items from the list.

Using the e*Way Connection Editor

This section describes basic procedures you can use when working with the e*Way Connection Editor GUI.

To open the e*Way Connection Editor

- 1 In the Enterprise Manager GUI, select the Navigator's Components tab.
- 2 Open the e*Way Connections folder and, in the Editor pane (on the right), double-click the e*Way Connection you want to edit.
- 3 In the e*Way Connection's Properties dialog box, **General** tab, in the e*Way Connection Configuration File area, click **Edit**.

The Configuration Editor opens.

Navigating Through the Editor

To display a list of all e*Way Connection parameters and their settings

On the **View** menu, click **Summary**.

Use the controls on the Summary list to go to a selected parameter or print a list of parameter settings. When you have finished using the Summary list, click **Cancel**.

To view or edit a section of the e*Way Connection configuration

In the **Goto Section** list, select a section.

To view or edit a selected parameter

- 1 In the **Goto Section** list, select the section that contains the parameter you want to edit.
- 2 In the **Goto Parameter** list, select a parameter. If all the parameters are not visible, use the scroll bars to scroll to the desired parameter.

To navigate using the summary list

- 1 On the **View** menu, click **Summary**.
- 2 Select a parameter and click **Go To**.

Saving Configuration Settings

To save the current configuration settings

On the **File** menu, click **Save**.

To promote the current configuration file to run time

On the File menu, click **Promote to Run Time**.

Modifying Configuration Settings

To modify a configuration setting

- 1 Navigate to the parameter whose options you want to change.
- 2 Select the parameter from those that are provided. To add new options to a list of parameters (see [Figure 208 on page 479](#)), enter the option in the parameter's data-entry box and click .
- 3 To remove options from the list, click . A new dialog box displays, in which you do the actual deletions.

Restoring Default Settings

To reload the default current configuration settings for a parameter

- 1 Navigate to the parameter whose defaults you want to restore.
- 2 On the parameter-editing toolbar, click . To reload the default current configuration settings for a section:
 - 1 Navigate to the section whose defaults you want to restore.
 - 2 On the Section toolbar, click .

Restoring Saved Settings

To reload the saved current configuration settings for a parameter

- 1 Navigate to the parameter whose defaults you want to restore.
- 2 On the Parameter-editing toolbar, click .

To reload the saved current configuration settings for a section

- 1 Navigate to the section whose defaults you want to restore.
- 2 On the Section toolbar, click .

Entering User Notes

User Notes provide you with a means to “comment” your e*Way configuration.

To enter user notes regarding the e*Way's general configuration or operations

On the File menu, click **User Notes**.

To enter user notes regarding a section of the e*Way configuration

On the Section toolbar, click .

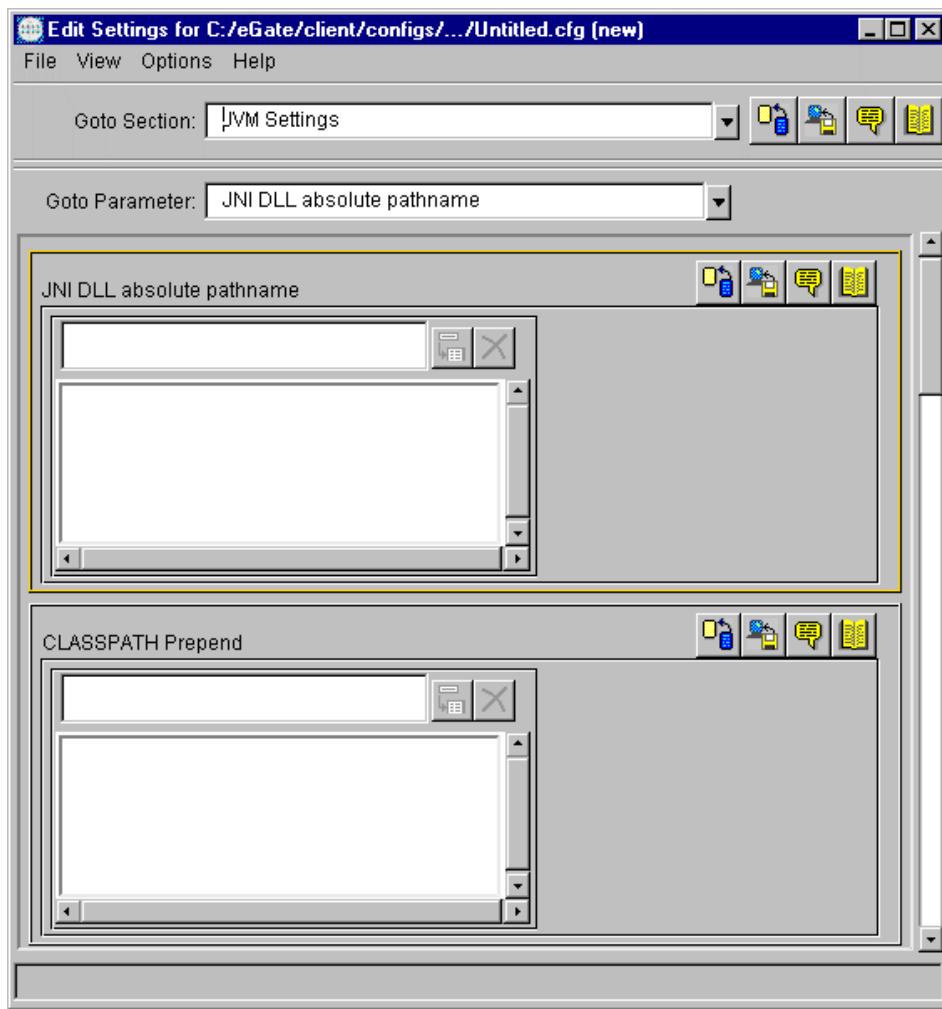
To enter user notes regarding a parameter within the e*Way configuration

On the Parameter-editing toolbar, click .

9.5.7 JVM Settings

JVM settings control the basic Java Virtual Machine (JVM) settings on Multi-Mode e*Ways. These settings are the configuration parameters for this configuration file, and are organized under the **JVM Settings** (in the **Goto Section** at the top of the e*Way Configuration Editor window). See the following figure.

Figure 209 Top Portion of the e*Way Configuration Editor Window



Important: The information entered in the fields on the e*Way Configuration Editor is e*Way-specific. Make sure you see the appropriate e*Way user's guide for specifics about what information should be entered in each field.

You set these settings on the e*Way Configuration Editor (see [“Adding Multi-Mode e*Ways” on page 131](#) for instructions on how to fill out the Editor fields).

The JVM settings that may be entered (depending on the e*Way) on this dialog box include:

- **JNI DLL absolute pathname:** Specifies the absolute pathname to where the JNI DLL installed by the *Java 2 SDK 1.2.2* is located on the Participating Host. This parameter is **mandatory**. This JNI DLL name varies by operating system platform. See Table 74 below).

Table 74 Java 2 JNI DLL Names

Operating System	Java 2 JNI DLL Name
Windows 2000	jvm.dll
Windows NT 4.0	jvm.dll
Solaris 2.6, 7, or 8	libjvm.so
Red Hat Linux 6.2	libjvm.so
Compaq Tru64 UNIX	libjvm.so
HP-UX 11.0 or 11i	libjvm.sl
AIX 4.3.3	libjvm.a

The value assigned can contain a reference to an environment variable, by enclosing the variable name within a pair of % symbols. For example:

%MY_JNIDL%
Such variables can be used when multiple Participating Hosts are used on different platforms.

To ensure that the JNI DLL loads successfully, the Dynamic Load Library search path environment variable must be set appropriately to include all the directories under the Java 2 SDK (or JDK) installation directory that contain shared libraries (UNIX) or DLLs (NT).

- **CLASSPATH Prepend:** Specifies the paths to be prepended to the CLASSPATH environment variable for the Java Virtual Machine settings. This is either an absolute path or an environment variable.

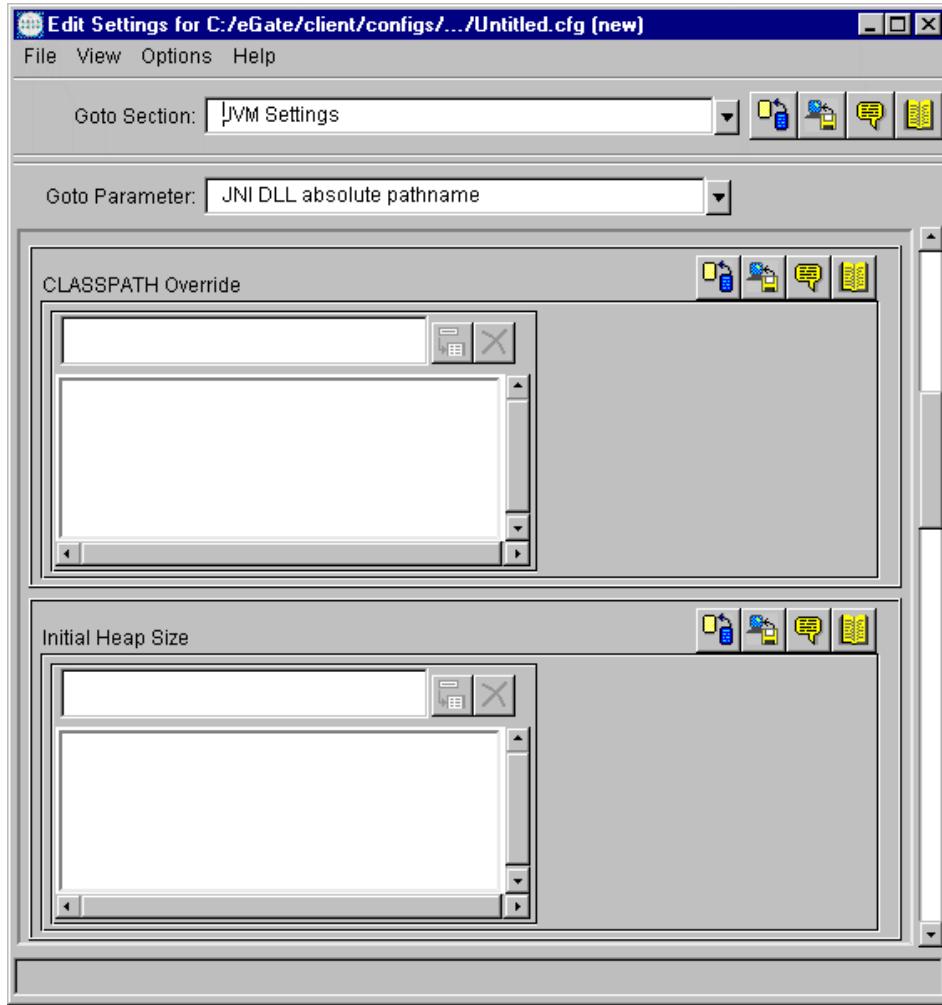
Note: *This parameter is optional.*

If left unset, no paths will be prepended to the CLASSPATH environment variable.

Existing environment variables can be referenced in this parameter by enclosing the variable name in a pair of % signs. For example:

%MY_PRECLASSPATH%

Figure 210 Second Portion of the e*Way Configuration Editor Window



- **CLASSPATH Override:** Specifies the complete CLASSPATH variable to be used by the Java Virtual Machine settings, which is either an absolute path or an environment variable. This parameter is optional. If left unset, an appropriate CLASSPATH environment variable will be set, consisting of required e*Gate components concatenated with the system version of CLASSPATH. See Figure 210.

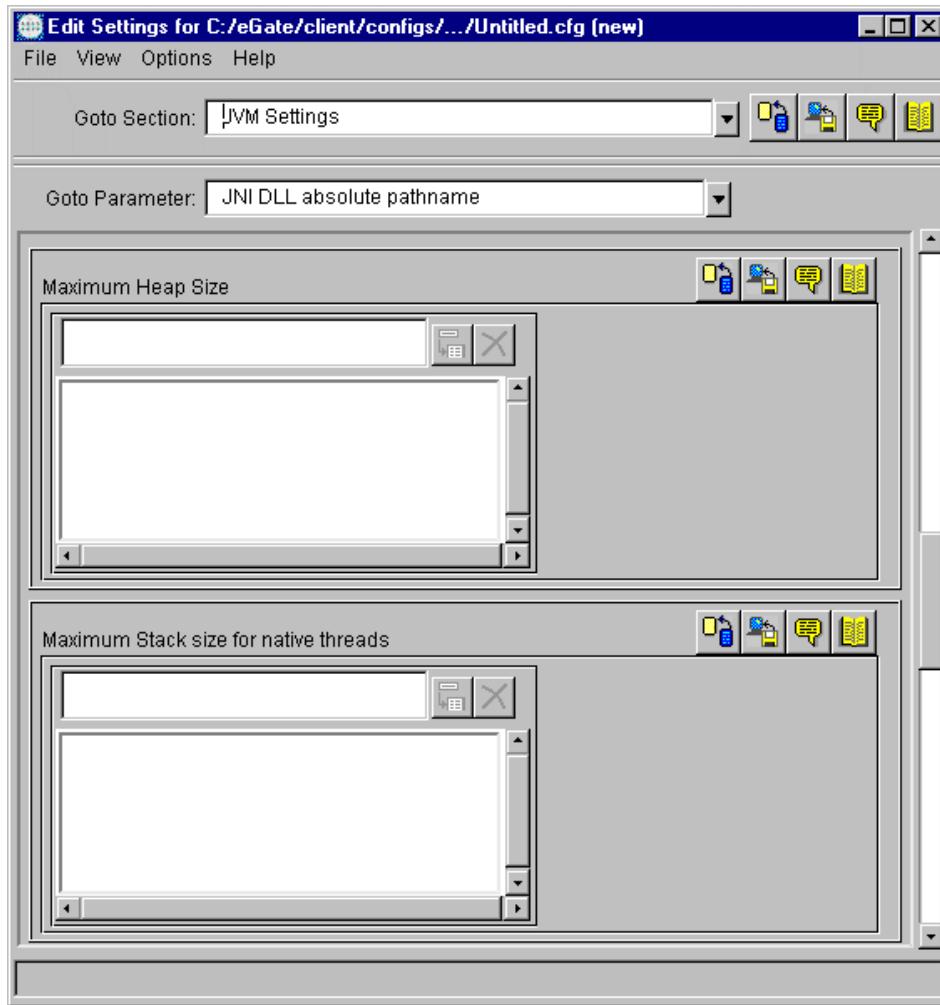
Note: All necessary .jar and .zip files needed by both e*Gate and the Java VM must be included. It is advised that the **CLASSPATH Prepend** parameter should be used.

Existing environment variables can be referenced in this parameter by enclosing the variable name in a pair of % signs. For example:

%MY_CLASSPATH%

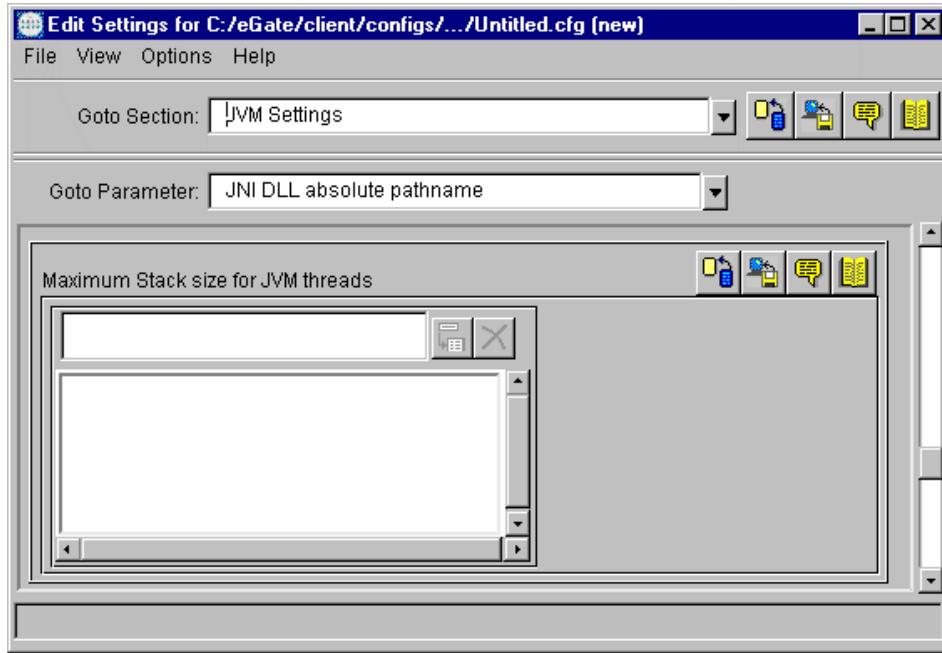
- **Initial Heap Size:** Specifies the value for the initial heap size in bytes. If set to 0 (zero), the preferred value for the initial heap size of the Java Virtual Machine settings will be used. Valid values include an integer between 0 and 2147483647. This parameter is optional. See Figure 210.

Figure 211 Third Portion of the e*Way Configuration Editor Window



- **Maximum Heap Size:** Specifies the value of the maximum heap size in bytes. If set to 0 (zero), the preferred value for the maximum heap size of the Java Virtual Machine settings will be used. Valid values include an integer between 0 and 2147483647. This parameter is optional (see Figure 211 above).
- **Maximum Stack size for native threads:** Specifies the value of the maximum stack size in bytes for native threads. If set to 0 (zero), the default value will be used. Valid values include an integer between 0 and 2147483647. This parameter is optional (see Figure 211).

Figure 212 Fourth Portion of the e*Way Configuration Editor Window



- **Maximum Stack size for JVM threads:** Specifies the value of the maximum stack size in bytes for JVM threads. If set to 0 (zero), the preferred value for the maximum heap size of the Java Virtual Machine settings will be used. Valid values include an integer between 0 and 2147483647. This parameter is optional (see Figure 212).

Note: The rest of the values are either "Yes" or "No."

- **Class Garbage Collection:** Specifies whether the Class Garbage Collection will be done automatically by the Java VM. The selection affects performance issues.
The required value is either YES or NO.
- **Garbage Collection activity reporting:** Specifies whether garbage collection activity will be reported for debugging purposes.
The required value is either YES or NO.
- **Asynchronous Garbage Collection:** Specifies whether asynchronous garbage collection activity will be reported for debugging purposes.
The required value is either YES or NO.
- **Report JVM Info and all class loads:** Specifies whether the Java Virtual Machine information and all class loads will be reported for debugging purposes.
The required value is either YES or NO.
- **Disable JIT:** Specifies whether the Just-In-Time (JIT) compiler will be disabled.
The required value is either YES or NO.

Note: This parameter is not supported for Java Release 1.

- **Allow remote debugging of JVM:** Specifies whether to allow remote debugging of the Java Virtual Machine.

The required value is either **YES** or **NO**.

9.6 e*Insight Business Process Manager Engine

The following button in the Enterprise Manager palette allows you to create one or more instances of the e*Insight Business Process Manager Engine:



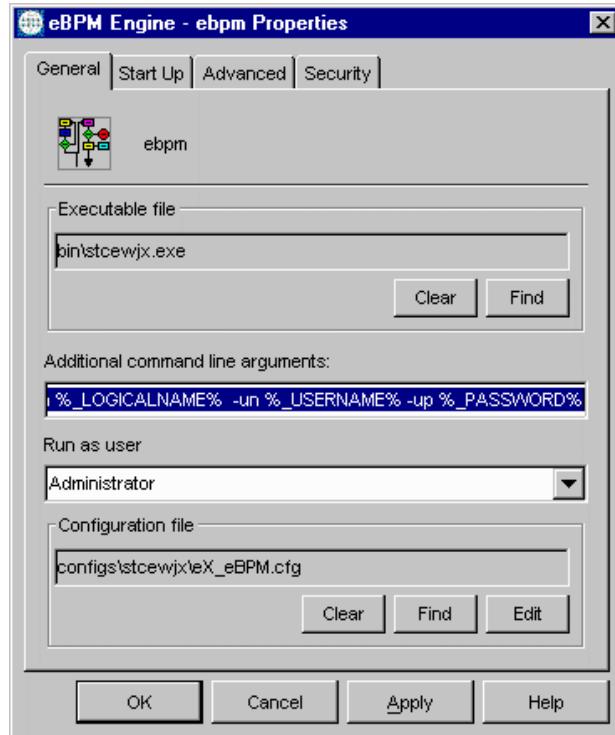
This feature also allows you to create and configure the executable and configuration files necessary to run business processes created in e*Insight. Through this functionality, the files you previously needed to configure for the e*Insight e*Way are now pre-selected for you, simplifying the configuration process.

Note: *This feature is only available with the e*Insight application. If you do not have e*Insight installed, the e*Insight Engine module is not available for use.*

To create and configure a new e*Insight Engine

- 1 Log into the e*Gate Enterprise Manager.
- 2 Open a schema.
- 3 Select the Navigator pane's **Components** tab.
- 4 Open the host on which you want to create the e*Insight Engine.
- 5 Select the Control Broker you want to manage the new e*Insight Engine.
- 6 On the Palette, click . The **New e*Insight Engine Component** dialog box appears.
- 7 Enter the name of the new e*Insight Engine and click **OK**.
- 8 In the Navigator, select the e*Insight Engine component you just created.
- 9 On the toolbar or **Edit** menu, click **Properties**. The **e*Insight Engine Properties** dialog box appears. See Figure 213.

Figure 213 e*Insight Engine Properties Dialog Box



- 10** Under Configuration file, click Edit.

The e*Way Configuration Editor opens.

- 11** Make any desired changes, then save the configuration file.

You are returned to the e*Insight Engine Properties dialog box.

- 12** Click OK to close the dialog box.

Note: For more information on how to use the e*Insight engine feature, see the *e*Insight Business Process Manager Implementation Guide*.

Introduction to e*Gate Monitor

This chapter explains the general operation of the e*Gate Monitor feature and introduces you to its basic operation.

10.1 e*Gate Monitoring Overview

The e*Gate monitoring system provides the following methods to check the status of your e*Gate system:

- **Interactive Monitors:** Display real-time e*Gate status information, allowing you to start and stop e*Gate components.
- **Non-interactive Monitors:** Forward alert and status information through delivery channels, including pager and e-mail, but do not provide means to control e*Gate components. This system also provides an escalation system for unresolved problems and failures, to make sure that all notifications are properly delivered.

10.1.1 Role of the Control Broker

In the e*Gate real-time monitoring feature, system components send messages called Monitoring Events to the Control Broker. These Monitoring Events include an Event code and a description (for example, “10113020: IQ Manager Down Controlled”) plus other information such as time of occurrence and names of possibly affected components.

The e*Gate monitoring feature depends heavily upon the Control Broker, both as a source of information and as intermediary for commands issued to the various e*Gate components. You must have a running Control Broker before you can use any e*Gate monitoring feature. The Control Broker binds a port (this is configurable via a range in the **Control Broker** properties dialog box) and the monitor connects to this port.

Caution: *Because of its importance within the e*Gate monitoring system, you must address any system failures involving the Control Broker as quickly as possible. Both the host and the Control Broker must be active before the e*Gate Monitor can connect to them.*

10.1.2 e*Gate Interactive Monitoring

e*Gate includes two interactive monitors: the e*Gate Monitor and a command-line monitor, **stccmd**. Using **stccmd** entails entering application program interface (API) instructions (in correct syntax) in a DOS-type screen to issue commands and display information.

Note: For more information on **stccmd**, advanced uses of the e*Gate Monitor, and other system control and monitoring operations, see the **e*Gate Integrator Alert and Log File Reference Guide**.

The e*Gate Monitor feature provides a graphical user interface (GUI) that allows you to carry out e*Gate's necessary system-monitoring functions. The rest of this chapter explains the e*Gate Monitor window's basic features and how to use them.

10.2 e*Gate Monitor Basic Operation

Use the e*Gate Monitor window to monitor and control all elements of your operating e*Gate system. This section explains how to access and exit this window as well as its basic features.

To access the e*Gate Monitor

- 1 Click the e*Gate Monitor  icon on your Windows Desktop, or choose **e*Gate Monitor** from the **Start** menu.

The **e*Gate Monitor Login** dialog box appears.

- 2 Enter your user name and password. If necessary, select the appropriate Registry Host name.

- 3 Click **Open** or press ENTER.

The **Open Schema on Registry Host** dialog box appears.

- 4 Select the appropriate schema name then click **Open** or press ENTER.

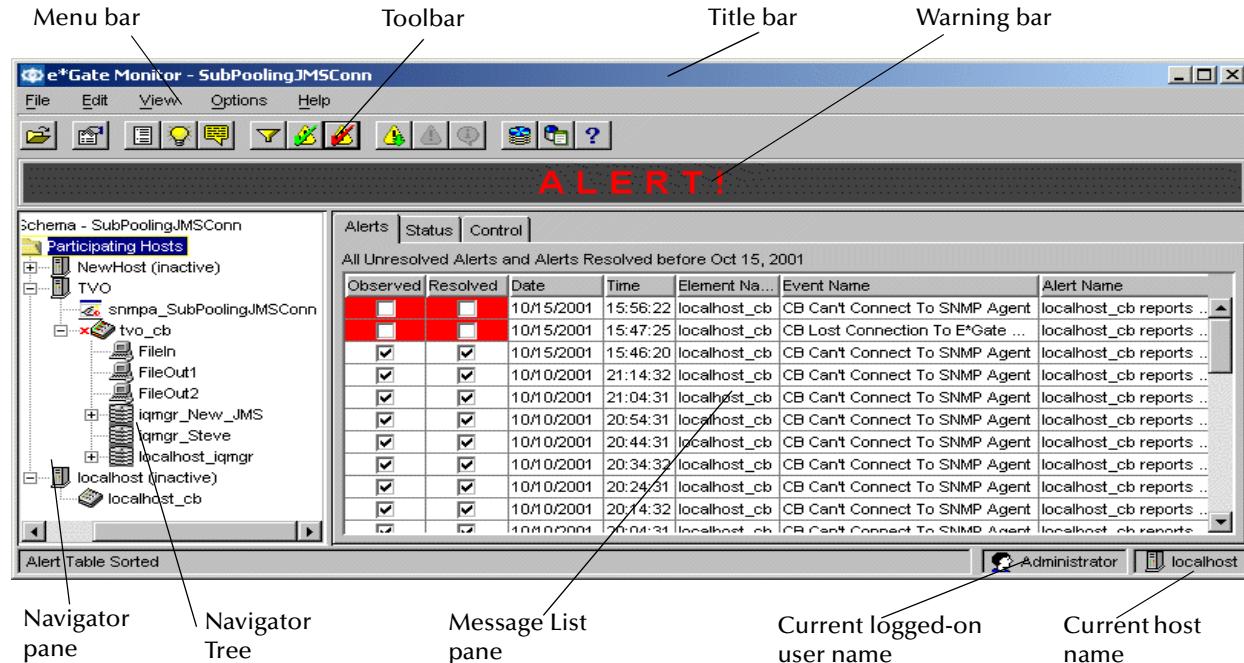
After you briefly see the e*Gate logo with a progress bar, the e*Gate Monitor window appears.

- 5 Size and place the e*Gate Monitor window as desired.

10.2.1 e*Gate Monitor Window

Figure 214 below shows an example of this window (**Alerts** tab).

Figure 214 e*Gate Monitor Window, Alerts Tab



Like the Enterprise Manager, the e*Gate Monitor has two panes. On the left, the navigator pane operates exactly like the same pane within the Enterprise Manager window (see [“Enterprise Manager Window” on page 50](#)).

In the navigator pane, the colors of the component icons show their status at a glance. These color indicators are

- Normal icons, matching the colors of those that appear in the Enterprise Manager, indicate that a component is functioning normally.
 - Red icons indicate that a component is either not functioning or not communicating with the Control Broker.
 - Gray icons indicate that the e*Gate Monitor is not connected to an element's Control Broker.

The Message List pane (right) provides the following tabs:

- **Alerts** — Displays Alert messages (also called “Alerts”) that describe Monitoring Events needing attention and resolution, for example, warnings regarding components that have stopped functioning or system parameters exceeding preset limits.
 - **Status** — Displays status messages that describe conditions other than problems to be solved, for example, news that components are operating normally.
 - **Control** — Presents a console you can use to send commands (such as start up or shut down) to the e*Gate component selected in the Navigator pane.

10.2.2 Toolbar Buttons

Table 75 lists and explains the e*Gate Monitor toolbar buttons. See “[Menu Bar](#)” on [page 493](#) for more information on the buttons’ operation.

Table 75 e*Gate Monitor Toolbar

Button	Name	Function
	Open Schema	Allows you to open another schema.
	Properties	Allows you to configure the Message List pane displays.
	Notification Details	Displays the details of the selected message (same as the Details menu option).
	Notification Tips	Displays tips related to the selected message.
	View or Add Comments	Allows you to view or to enter comments for the selected message (same as the Comments menu option).
	Filter	Changes the date filter for messages.
	Observe All Alerts	Marks all alert messages as “observed.”
	Resolve All Alerts	Marks all alert messages as “resolved.”
	Next Unresolved Alert	Selects the next unresolved alert message.
	Latest Alert Notification	Selects the most recent alert message (same as the Latest Alert menu option).
	Latest Status Notification	Selects the most recent status message (same as the Latest Status menu option).
	Launch IQ Administrator	Starts the IQ Administrator tool, allowing you to view a graphical representation of IQs and the messages (Events) they contain, and to edit the properties of an Event or range of Events.
	JMS Administrator	Starts the JMS Administrator tool, allowing you to view the status of JMS IQ Managers, their Queues and Topics, and messages (Events) within each Queue or Topic. For complete details, see the <i>SeeBeyond JMS Intelligent Queue User's Guide</i> .
	Help	Allows you to access the e*Gate Monitor’s online Help system.

10.2.3 Menu Bar

Table 76, starting below, lists the e*Gate Monitor window menus, including the options contained in each and their functions. For more detailed information about each menu option, see Chapter 2 of the *e*Gate Integrator System Administration and Operations Guide*.

Table 76 e*Gate Monitor Menu Commands

Menu	Option	Function
File	Login	Displays the e*Gate Open Schema Login dialog box, allowing a different user to log in to the system; also allows the current user to log in to a different system.
	Open Schema	Allows you to open another existing schema; you can only have one schema open at a time.
	Exit	Exits the e*Gate Monitor and closes the window (see “ To exit the e*Gate Monitor ” on page 494).
Edit	Filter	Opens the Resolved Alerts Filter dialog box for Alerts or the Status Filter dialog box for status messages, allowing you to display only those resolved Alerts or status messages that originated after a date and time you specify; select the Alerts or Status tab in the Message List pane to display the desired dialog box.
	Properties	Displays the Table Properties Configuration properties dialog box, allowing you to configure and/or change the Message List pane display format; also configures the number of messages to be displayed.
View	Details	Opens the Issue & Recipient Information dialog box, displaying detailed information on the selected alert or status message. Use the Event tab to access information about the Event that triggered the Alert, and the Recipients tab to view contact information for users who are notified when the Alert occurred.
	Notification Tips	Opens the Help window, providing possible cause and problem-resolution information on the selected alert message.
	Comments	Opens the Comments dialog box, allowing you to view previous comments entered about the selected message and/or enter and save new comments of your own.
	Latest Status	Selects the most recent status message issued by the system, in the Message List pane, under the Status tab.
	Latest Alert	Selects the most recent alert message issued by the system, in the Message List pane, under the Alerts tab.
	Next Unresolved Alert	Selects the next unresolved alert message issued by the system, in the list under the Alerts tab.
	Observe All Alerts	Allows you to mark all alert messages as “observed.”
	Resolve All Alerts	Allows you to mark all alert messages as “resolved.”

Table 76 e*Gate Monitor Menu Commands (Continued)

Menu	Option	Function
Options	Flashing Alert Warning	Enables the word "Alert" to flash in the Warning bar near the top of the e*Gate Monitor window when there are any unobserved alerts; acts as a toggle.
	Resolved Notification Message	Enables you to display a pop-up window giving information on alert resolutions; acts as a toggle.
	Connect All Available Control Brokers	Causes the system to automatically attempt to connect to all relevant Control Brokers within the system upon opening a schema; acts as a toggle.
	Enable Schema Access Checking	Acts as a toggle. When selected (in other words, when there is a check mark in front of this option), access to the schema can be granted on a role-by-role basis. Once such access has been granted to one or more roles, all users who do not have a role that has been granted access are denied access to this schema.
	Toolbar Text	Applies text labels to all Toolbar buttons; acts as a toggle between labeled and unlabeled states.
	Roll Over Toolbar	Removes the shadow boxes around Toolbar buttons; acts as a toggle between boxed and unboxed states.
Help	e*Gate Help Topics	Allows you to access the e*Gate Monitor's online Help system and to open its Help window; see " "Online Help Systems" on page 73 for details.
	About e*Gate	Provides e*Gate version information. If you have installed patches, additional information on the patches is available here.

Note: *Menu options also enabled by buttons display, to the left of each option name, smaller versions of the appropriate buttons.*

To exit the e*Gate Monitor

- 1 On the **File** menu, click **Exit**.
- 2 When you exit, a message appears asking if you want to exit. Click **OK** to close the e*Gate Monitor window and exit the program.

10.3 Controlling e*Gate

The e*Gate interactive monitors enable you to control the e*Gate system. The tables below list the commands that monitors can issue to e*Gate components (see Table 77 and Table 78 below).

Table 77 Monitor Commands

Component	Command
All executable components	Start, Shutdown, Status, Version
The following components support these commands in addition to the four commands above:	
Control Brokers	Connect, Disconnect (from current monitor)
Business Object Brokers (BOBs) and e*Ways	Reload, Suspend, Activate (resume from suspended state)
The following components support only these commands:	
IQs	Detach “unplugs” the IQ from any Collaboration that is publishing to that IQ. A detached IQ cannot receive any more Events. Attach reconnects the IQ to its Collaborations, so that the IQ can resume receiving Events.

Note: The **suspend** command effectively takes a component “off line” but does not end the executing process; when the command is received, the component finishes processing any in-process Events, then goes into a “wait” state (e*Ways will also close any open connections to external systems).

The **activate** command brings the component back on-line. Events cannot be published to suspended components, but will remain in their source location (another IQ or an external system) until the subscribing component is reactivated or another subscriber obtains them.

The **reload** command only affects one or two types of e*Ways for which you must explicitly reload configuration changes. Most types of e*Way automatically reload their configuration whenever the configuration is changed, and for them, this command has no effect.

The basic commands described above can be sent to e*Gate components using either the **stccmd** command or the e*Gate Monitor. See “Viewing Alerts Using Command-line Monitoring” in the *e*Gate Integrator Alert and Log File Reference Guide* for more information on sending commands using the text-based monitor **stccmd**.

Table 78 Available Control Tab Commands

Command	Function
Activate	Restart a suspended element.
Debug	Activate Debug logging.

Table 78 Available Control Tab Commands (Continued)

Command	Function
Detach	Detach an IQ from the IQ Manager.
Disconnect	Disconnect the selected Control Broker.
Reload	Reload configuration information (e*Ways only).
Sequence	Query or set the sequence number if the external application protocol requires it (e*Ways only).
Shutdown	Shut down the selected element.
Shutdown All Modules	Shut down all elements associated with the selected Participating Host or Control Broker.
Shutdown CB	Shut down the selected Control Broker.
Start	Start the selected element.
Start All Modules	Start all elements associated with the selected Participating Host or Control Broker.
Status	Query the selected element for its status.
Suspend	Suspend the selected element's execution (but leave the process running).
Update All Status	Query the selected Participating Host or Control Broker and all elements associated with it for their status.
User	Generic command for future use.
Version	Display version information for the selected element.

To issue a command to an e*Gate component using the e*Gate Monitor

- 1 Use the Navigator to select the component you want to control.
- 2 Select the **Control** tab.
- 3 From the **Command** list underneath the Console window, select the command you want to issue.
- 4 Click **Run**.

Note: You cannot use an e*Gate monitor to *start* a Control Broker. The Control Broker must be running before any commands can be sent to any components. You can, however, shut down a Control Broker from an e*Gate monitor.

10.3.1 Starting and Shutting Down Components

To start a component

- 1 In the Navigator, select the component you want to start.

- 2** Select the **Control** tab.
- 3** From the **Command** list underneath the Console window, select **Start**.
- 4** Click **Run**.

To shut down a component

- 1** In the Navigator, select the component you want to shut down.
- 2** Select the **Control** tab.
- 3** From the **Command** list underneath the Console window, select **Shutdown**.
- 4** Click **Run**.

When you change a component's state (for example, starting a down component), the change should be reflected in the Navigator, and an alert or status message should appear within a few moments on the appropriate tab.

Note: Components that are configured as "Start automatically" in their properties dialog boxes (under the **Startup** tab) will not automatically restart after you shut them down manually. However, they will restart on schedule if a schedule has been defined, and there is a call for a scheduled startup.

If you shut down a component that has been configured to start automatically, it will restart automatically if you use the Enterprise Manager to apply any configuration changes to the component.

- 5** Click **Run**.

To shut down all components

- 1** In the Navigator, select the Control Broker managing the components you want to shut down.
- 2** Select the **Control** tab.
- 3** From the **Command** list underneath the Console window, select **Shutdown All Modules**.
- 4** Click **Run**.

When you change a component's state (for example, starting a down component), the change should be reflected in the Navigator, and an alert or status message should appear within a few moments on the appropriate tab.

10.3.2 Displaying Status and Version Information

To display a component's status

- 1** In the Navigator, select a component.
- 2** Select the **Control** tab.
The selected component's status automatically displays in the Console window.
- 3** To update the status, select **Status** from the **Command** list.

4 Click Run.

To display status for all components

- 1** In the Navigator, select a Control Broker.
- 2** Select the **Control** tab.
- 3** From the **Command** list, select **Update All Status**.
- 4** Click **Run**.

To display a component's version information

- 1** In the Navigator, select a component.
- 2** Select the **Control** tab.
- 3** From the **Command** list, select **Version**.
- 4** Click **Run**.

Note: You can also display version information for any e*Gate executable file or .dll file using command-line utilities. Using the `--ver` flag (important: use two dashes) will cause any e*Gate executable file to display its version information (for example, `stccb --ver` or `stcregd --ver`). To display version information for a .dll, you must use the `stcutil -vi` utility. See “System Testing and Support: stcutil” in the *e*Gate Integrator System Administration and Operations Guide* for more information.

On Windows systems, you can also display .dll version information through the .dll file’s properties dialog box. See the Windows Help system for more information about displaying file properties.

10.3.3 Suspending and Activating Components

Caution: Do not try edit an ETD on a machine while it is running an e*Way or BOB that uses that ETD. Trying to edit an ETD on the same machine that is using the ETD in a running e*Gate module can destabilize both the Editor and the module.

To activate a component

- 1** In the Navigator, select the component you want to start.
- 2** Select the **Control** tab.
- 3** From the **Command** list, select **Activate** to restart the suspended component.
- 4** Click **Run**.

To suspend a component

- 1** In the Navigator, select the component you want to stop.
- 2** Select the **Control** tab.
- 3** From the **Command** list, select **Suspend** to suspend the component.
- 4** Click **Run**.

10.4 Non-interactive Monitoring

In addition to the interactive monitors, e*Gate includes a system that enables you to notify support staff of system conditions via pager, e-Mail, fax, or printout. You can also interface with other monitoring agents such as the e*Gate Integrator SNMP Agent or your own custom-written agents (referred to in this guide as *user agents*).

10.4.1 Notification Channels

e*Gate supports notification through the channels shown in Table 79. For a complete list of notification channels, including the supported operating systems for each channel, see the *e*Gate Integrator Alert Agent User's Guide*.

Table 79 Notification Channels and Delivery Systems

Channel	Notification Delivered By
Monitor	Interactive monitors
e-Mail	Alert Agent
Pager	Alert Agent
Fax	Alert Agent
Printer	Alert Agent
SNMP	SNMP Agent
Script	Note: See "Using Scripts" below this table.
User Agent	User agent

Using Scripts

Unlike all other notification channels, scripts are executed directly by the Control Broker; no additional components are required. The script must be supported by the operating system on the host running the Control Broker that executes the script.

For example, if the Notification is configured to execute a Perl script, Perl must be installed on the Participating Host that is executing the notification routing. e*Gate imposes no restrictions of its own on the type of script to be executed.

10.4.2 e*Gate Alert Agent

The e*Gate Alert Agent routes notifications through e-Mail, pager, fax, and printer channels, and is installed separately from the basic configuration of e*Gate. For more information about the e*Gate Alert Agent, see the *e*Gate Integrator Alert Agent User's Guide*.

10.4.3 e*Gate SNMP Agent

The e*Gate system has an additional add-on feature available, the e*Gate SNMP Agent. This feature routes monitoring information to your existing SNMP-based management system. For more information about the e*Gate SNMP Agent, see the *e*Gate Integrator SNMP Agent User's Guide*.

10.4.4 Custom User Agents

In future releases of e*Gate, you will be able to create your own custom alert agent to interface with the e*Gate monitoring system. Contact SeeBeyond for more information.

10.5 Monitoring Resources and Performance

In addition to monitoring component status, e*Gate can monitor system resources and component performances. Using the Enterprise Manager, you can set a threshold level for disk usage on the Participating Host, and high and low thresholds for Event processing for BOBs and e*Ways. When these thresholds are exceeded, the Control Broker will send an appropriate Monitoring Event to notify you of the condition and enable you to take appropriate action.

10.5.1 Setting Disk-usage Thresholds

Disk-usage thresholds cause e*Gate to send a Notification when disk usage on a selected drive or partition exceeds a certain level. You can set different thresholds for each disk or partition installed on the Participating Host.

To set a disk-usage threshold

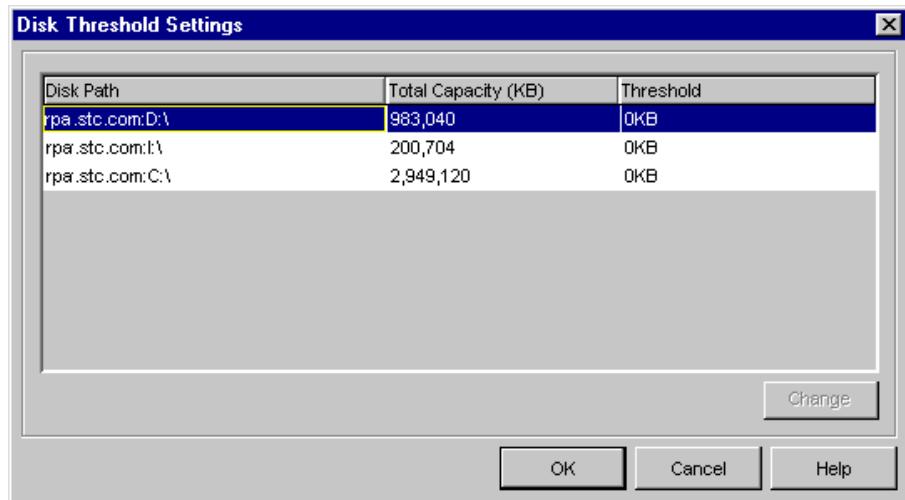
- 1 Start the Enterprise Manager and select the **Components** tab.
- 2 In the Navigator, select the host for which you want to set disk-usage thresholds.
- 3 On the toolbar or **Edit** menu, click  **Properties**.

The **Participating Host Properties** dialog box appears.

- 4 Select the **Advanced** tab, then click **Threshold Setup**.

The Threshold Setup dialog box appears. See Figure 215.

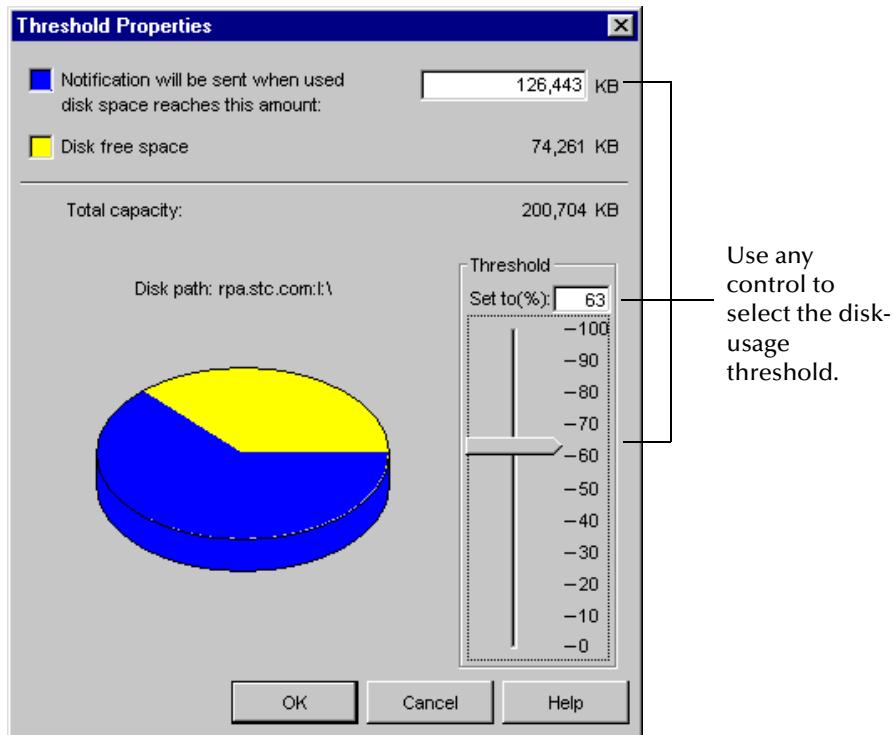
Figure 215 Threshold Setup



- 5 Select a disk or partition to monitor and click **Change** (if no disks appear on the list, see the [Note on page 502](#)).

The Change Disk Threshold dialog box appears (see Figure 216 below).

Figure 216 Change Disk Threshold



- 6 Use the controls to select the disk-usage level at which you want to be notified. The default, zero usage, instructs the Control Broker not to send any disk-threshold notifications.
- 7 Click **OK** on each dialog box until you return to the Enterprise Manager main screen.

Note: If no disks appear in the “threshold” list, you must update the schema’s internal resource table. Log in to the Participating Host, then issue this command at the shell prompt:

stcregutil -rh RHost -rs Schema -un User -up Pass -ur

where:

RHost is the Registry Host name,

Schema is the schema name, and

User and **Pass** are the valid e*Gate user name and password.

The **-ur** flag sets the disk-usage threshold.

For more information about **stcregutil** and its command arguments, see “Registry Utility: **stcregutil**” in the **e*Gate Integrator System Administration and Operations Guide**.

When disk usage exceeds the threshold you set, the Control Broker issues Monitoring-Event code 106K1480. See the **e*Gate Integrator Alerts and Log File Reference Guide** for more information on Alert notifications.

10.5.2 Disk-space Quota Limitations

In UNIX, the “egate” user, or any user that you use to install the e*Gate system, must have *no* disk-quota limitations imposed. e*Gate calculates “available disk space” in terms of total disk space available on the system, and does no quota checking. If you impose a disk-quota restriction on the “egate” user, you risk losing data when IQ-storage demands exceed the user’s quota.

Controlling Disk Space Usage

In e*Gate, you control how much space the system uses on the hard drive in the following ways:

- **Log-file Size and Number** — Make sure you set the log files to record only as much information as you need. Recording excess information only takes up more space on the disk. Also, be sure to delete or back up log files on a regularly scheduled basis to make sure they do not accumulate on the disk.
- **IQ Data** — Be sure to schedule the journaling and expiration (removal) of Events from IQs on a fast enough basis that unneeded data does not build up in the IQs.

10.5.3 Setting Event-processing Thresholds

Event-processing thresholds enable you to monitor the number of data Events being processed by components within the e*Gate system. You can use these notifications to determine whether a component is processing too many or too few Events.

Note: The CNTS debug flag must be turned on before the 'status' display in the monitor will show how many Events a given e*Way/BOB has processed. If the debug level is also set to TRACE, a count will be written to the log file for every 100 Events processed.

Each Event-processing component (BOB or e*Way) has independently configurable thresholds. You must configure each component separately; there is no way to set the thresholds for more than one component at a time.

To set Event-processing thresholds for an e*Way or a BOB

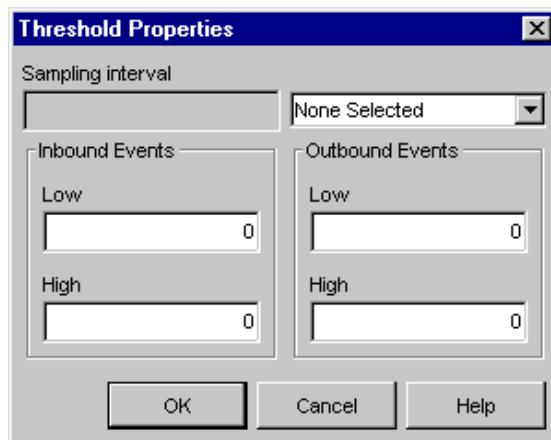
- 1 Start the Enterprise Manager and select the **Components** tab.
- 2 Using the Navigator, select the component you want to configure.
- 3 On the toolbar or **Edit** menu, click  **Properties**.

The selected component's properties dialog box appears.

- 4 Select the **Advanced** tab and then click **Thresholds**.

The **Threshold Properties** dialog box appears. The values shown in Figure 217 are the defaults for both e*Ways and BOBs.

Figure 217 Threshold Properties Dialog Box



- 5 Under **Sampling interval**, select the interval at which to sample the number of Events being processed. For example, to select an interval of ten minutes, enter **10**, then select **Minutes** from the interval list. The default interval (0) disables Event-threshold monitoring for this component.
- 6 Under **Inbound Events**, enter the upper and lower threshold of Events that this component receives as input during the selected interval. The values you enter are the lowest and highest *acceptable* values; numbers that exceed those values will generate notifications. For example, if you enter a lower limit of 5, a notification will be generated if *fewer than* five Events are processed within the sampling interval.

- 7 Under **Outbound Events**, enter the upper and lower threshold of Events that this component publishes as output during the selected interval. The values you enter are the lowest and highest *acceptable* values; numbers that exceed those values will generate notifications. For example, if you enter an upper limit of 500, a notification will be generated if *more than* 500 Events are published within the sampling interval.
- 8 Click **OK** until you return to the Enterprise Manager's main screen.

Note: If you change the *Sampling interval* to a value greater than zero (thus activating threshold monitoring for this component), be sure that you do not leave any other threshold values set to zero unless you wish to use zero as a threshold value. If you forget to change any of the remaining default "zero" values, you may receive erroneous notifications.

Once Event-threshold monitoring has been configured, the Control Broker will send the Events shown in Table 80 below, when thresholds are exceeded.

Table 80 Event-threshold Monitoring-Event Codes

Code	Meaning
105Ba060	e*Way input below threshold
105Ba070	e*Way input above threshold
105Ca060	e*Way output below threshold
105Ca070	e*Way output above threshold
105Bb060	BOB input below threshold
105Bb070	BOB input above threshold
105Cb060	BOB output below threshold
105Cb070	BOB output above threshold

Note: See the *e*Gate Integrator Alert and File Log Reference Guide* for more information on Alert notifications and the *e*Gate Integrator System Administration and Operations Guide* for more information on the e*Gate Monitor feature.

e*Gate Java Debugger

The e*Gate Java Debugger is a GUI that allows remote debugging of Java Collaborations that are run in e*Gate.

11.1 Overview of e*Gate Java Debugger Operation

The e*Gate Integrator includes a source-level debugger for debugging Java Collaborations running within Multi-Mode e*Ways. The e*Gate Java Debugger can debug multiple Collaborations running in different threads. It attaches to the Java Virtual Machine (JVM) using the **Remote debugging port number** value specified in the JVM Settings of the configuration file for the Multi-Mode e*Way. You can even debug an e*Way that fails immediately upon startup, by setting its Suspend option for debugging setting to Yes. For more information, see [“Configuring Multi-Mode e*Ways with e*Way Connections” on page 477](#).

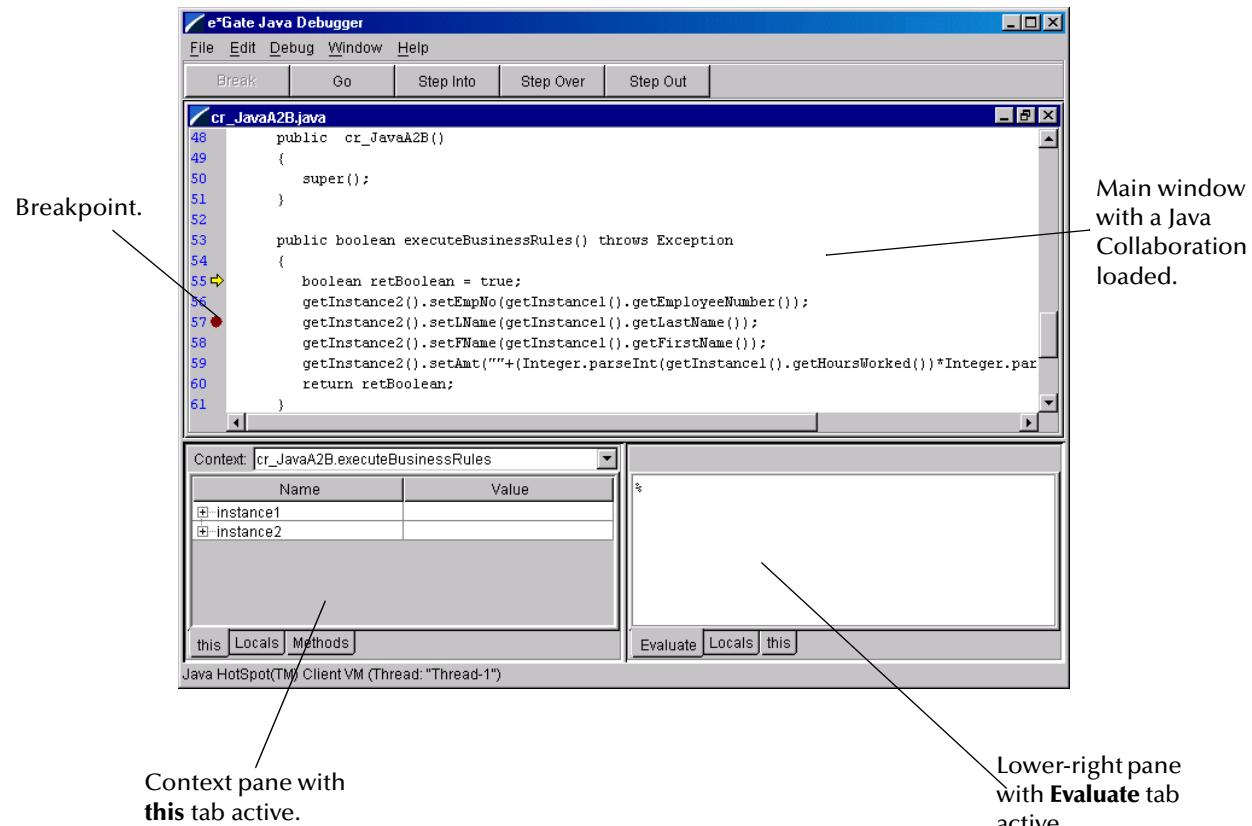
The e*Gate Java Debugger provides facilities to:

- Control execution.
- Set and clear breakpoints.
- Go to a specific statement.
- Step into, over, or out of specific blocks of code.
- Stop in a specific class or method.
- Break on a specified exception.
- Save, open, resume, or create new Debugger sessions.
- Run a specified Java class with specified options.
- Attach to a Java Virtual Machine (JVM).
- Add and modify source path and class path parameters.
- Set context options.
- View or modify current values of instance and local variables.

The purpose of this chapter is to help you understand how to run the Debugger and troubleshoot your Java Collaborations.

To access the Debugger: In e*Gate Monitor, right-click the e*Way and, on the shortcut menu, click **Debugger**.

Figure 218 e*Gate Java Debugger Window



11.1.1 Main Menu

Table 81 lists and describes the commands available from the main menu of the e*Gate Java Debugger.

Table 81 e*Gate Java Debugger Main Menu

Menu	Command	Function
File	New Debugger	Creates a new independent debugger.
	Open	Opens a Java source file (.java) in the e*Gate Java Debugger.
	Resume Session	Opens a previously saved debugging session (.ejdb).
	Save Session	Saves your currently open Java Collaboration with an .ejdb file extension. You can resume running this debugging session at a later time.
	Run	Not used when remote debugging e*Gate Java Collaborations. Runs a Java program from the e*Gate Java Debugger as a stand-alone program. When the Run dialog box opens: 1 Enter the Main Class , which is the main class name and program arguments. 2 JVM Options are populated with -classic . You may add additional JVM options. 3 Java Home This path should point to the root of your JDK installation. 4 Click Run .
	Detach	Detaches your current debugging session from the remote Java Virtual Machine.
	Attach to JVM	Attaches to a remotely executing Java virtual machine. Specify the host and port of the remote Java Virtual Machine. When the debugger is started from the e*Gate Monitor it automatically attaches to the host and port of the Multi-Mode e*Way you selected. The port is set in the configuration file of the e*Way.
	Close	Detaches from the remote JVM and closes the current debugger
	Exit	Closes all debugger windows and exits the e*Gate Java Debugger.
Edit	Cut	Cuts the currently highlighted selection.
	Copy	Copies the highlighted code to the Clipboard.
	Paste	Pastes the code that has been copied to the Clipboard at the current location of the cursor.
	Find	Opens the Find dialog box. Enter the text string to search on and click Find Next . You may perform a case-sensitive search by selecting the Match Case .
	Options	Opens the Options dialog box, which allows you to change parameters and options by selecting the Source Path , Class Path , or Context tabs.

Table 81 e*Gate Java Debugger Main Menu (Continued)

Menu	Command	Function
Debug	Break*	<p>Stops all running Collaborations and gives control to the Debugger.</p> <p>The following also allow you to perform this function:</p> <ul style="list-style-type: none"> ▪ On the toolbar, click Break. ▪ Press the Pause/Break key on the keyboard.
	Go*	<p>Continues execution of a Collaboration. Execution resumes until a breakpoint is hit or the Collaboration completes.</p> <p>The following also allow you to perform this function:</p> <ul style="list-style-type: none"> ▪ On the toolbar, click Go ▪ Press the F5 key on the keyboard.
	Step Into*	<p>Controls the execution of the Collaboration you are debugging with single-step entry into any method call. Execution resumes. If the current line in the Collaboration contains a method call, control returns to the Debugger upon entry into the method. Otherwise, control returns to the Debugger at the next line in the current method.</p> <p>The following also allow you to perform this function:</p> <ul style="list-style-type: none"> ▪ On the toolbar, click Step Into. ▪ Press the F11 key on the keyboard.
	Step Over*	<p>Controls the execution of the Collaboration you are debugging with a single step to the next line in the current method. Execution resumes, with control returning to the Debugger at the next line in the current method.</p> <p>The following also allow you to perform this function:</p> <ul style="list-style-type: none"> ▪ On the toolbar, click Step Over. ▪ Press the F7 key on the keyboard.
	Step Out*	<p>Continues execution until the current method returns. Execution resumes until the current method returns or a breakpoint is encountered.</p> <p>The following also allow you to perform this function:</p> <ul style="list-style-type: none"> ▪ On the toolbar, click Step Out. ▪ Press the F9 key on the keyboard.
	Stop in Class	Adds a class to stop at during the debugging.
	Stop in Method	Adds a method to stop at during the debugging.
	Break on Exception	Adds an exception to break on during the debugging.
Window	Cascade	Cascades the open windows on your monitor.
	Tile	Tiles the open windows on your monitor.
	Console	Returns you to the Console window of the e*Gate Java Debugger.

Note: Note that the toolbar buttons function as described under the **Debug** menu commands of the same names.

11.1.2 Activating the e*Gate Java Debugger

The e*Gate Java debugger is activated from the e*Gate Monitor.

Note: *Currently, the only component that has the capability of opening the e*Gate Java debugger are Multi-Mode e*Ways.*

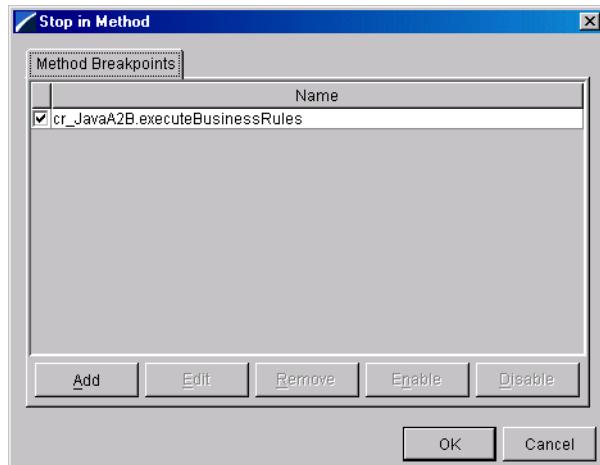
To activate the e*Gate Java debugger

- 1 With the e*Gate Monitor open, select a Multi-Mode e*Way.

Note: *Make sure that the **Remote debugging port number** is set in the Multi-Mode e*Way's configuration file.*

- 2 Right-click the Multi-Mode e*Way, and select **Debugger** from the shortcut menu. The e*Gate Java Debugger window opens.

Figure 219 Stop in Method Dialog Box



When the debugger appears, the **Stop in Method** dialog box as shown in Figure 219 is displayed. Breakpoints are automatically set on the **executeBusinessRules()** method of each Java Collaboration defined in the Multi-Mode e*Way you are debugging. If you want to debug *all* Collaborations, simply click **OK**. Alternatively, you can remove or disable the breakpoints on selected Collaborations, and then click **OK** to connect to the remote Java Virtual Machine (JVM).

11.1.3 Using the e*Gate Java Debugger

After loading a Java Collaboration in the e*Gate Java Debugger you are ready to debug the Collaboration using the Debugger's features to control the execution of the Collaborations you are debugging. Use the menu commands and toolbar buttons to maneuver through the Collaboration.

Controlling execution of the Collaboration

The Debugger provides the following facilities for you to control the execution of Collaborations you are debugging

Moving Up and Down the Stack

The lower-left (dockable) pane in the e*Gate Java Debugger's main window contains a combo-box labeled **Context**, which displays the current stack of the executing Collaboration thread.

To move up or down the stack, select an entry in the **Context** box. When you select a stack frame, the variables windows are updated to reflect the names and values of the fields and local variables visible at that scope.

Setting, Clearing, and Disabling Breakpoints

The e*Gate Java Debugger contains file windows which display the source code of each Collaboration you are debugging. Set a breakpoint in a Collaboration by doing one of the following:

- Place the cursor on the line on which you want to set a breakpoint and right-click. Then, on the shortcut menu, click **Set Breakpoint**.
- Single-click the line number of the line at which you want to set a breakpoint.

If the selected line contains executable code a red dot will appear next to the line number and a breakpoint is set at that location.

Clear breakpoints in a Collaboration by doing one of the following:

- Place the cursor on the line on which you want to clear a breakpoint and right-click. Then, on the shortcut menu, click **Clear Breakpoint**.
- Single-click on the red dot or the line number of the line at which you want to clear a breakpoint.

The red dot disappears and the breakpoint at that location is cleared.

Disable or enable breakpoints in a Collaboration by doing one of the following:

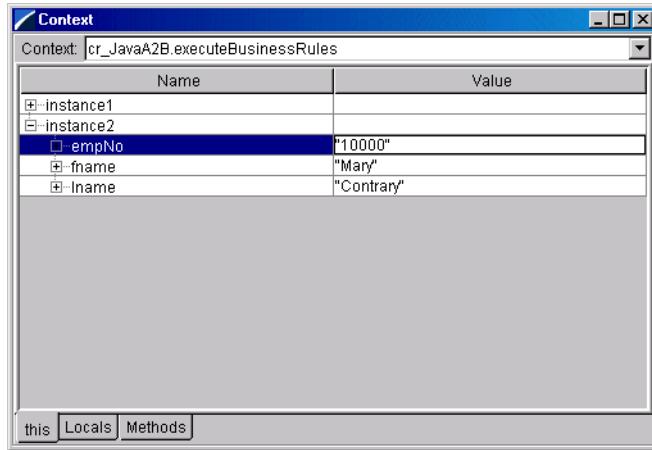
- Place the cursor on the line on which you want to clear a breakpoint and right-click. Then, on the shortcut menu, click **Disable Breakpoint** or **Enable Breakpoint** as appropriate.
- Click the red dot to disable a breakpoint (or the white dot to enable it).

The dot turns white if the breakpoint is disabled or red if enabled.

Viewing Variables

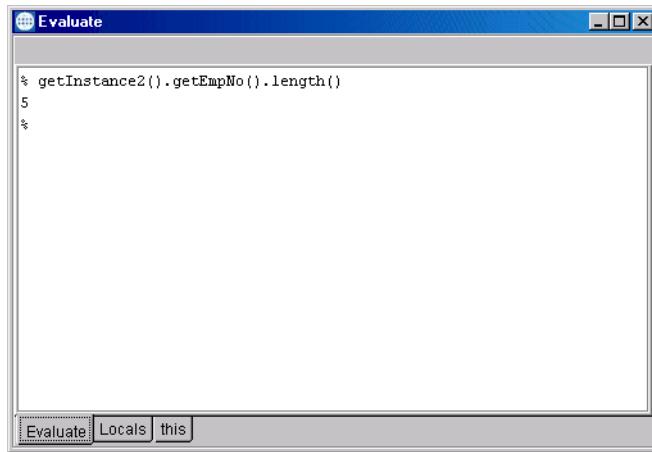
The Context pane in the Debugger main window is dockable. It contains a tab-pane with three tabs, labeled **this**, **Locals**, and **Methods**. The first two panes contain tree-tables that displays the properties of the current object and currently visible local variables, respectively.

Figure 220 Context Pane with this Tab Selected



- **this** tab: The properties of the current object are displayed in the **this** table. If a property is itself a Java object, the property can be expanded to show its sub-properties. The **this** table is updated each time control returns to the Debugger or when you change the stack location in the **Context** pane.
- **Locals** tab: The local variables of the current method are displayed in the **Locals** table. If a variable is itself a Java object, the variable can be expanded to show its sub-properties. The **Locals** table is updated each time control returns to the Debugger or when you change the stack location in the **Context** pane.
- **Methods** tab: The methods of the current object are displayed in the **Methods** table. Double-clicking on a method takes you to the definition of the method in the source window (if its source code is available). The **Methods** table is updated each time control returns to the Debugger or when you change the stack location in the **Context** pane.

Figure 221 Lower Right Pane with Evaluate Tab Selected



- **Evaluate tab:** The **Evaluate** tab is located in the dockable lower-right pane in the Debugger main window. It contains an editable command line where you can enter arbitrary Java expressions. The code is evaluated in the context of the current stack frame. The pane maintains a history of the commands you enter. Use the Up and Down arrow keys on the keyboard to move backward or forward through the command buffer.
- **Locals tab:** There is also a **Locals** tab located in the dockable lower-right pane in the Debugger main window. You may also use this tab to view or edit the local variables that are displayed in the **Locals** table in the **Context** pane.
- **this tab:** There is also a **this** tab located in the dockable lower-right pane in the Debugger main window. You may also use this tab to edit the properties of the current object that are displayed in the **this** table in the **Context** pane.

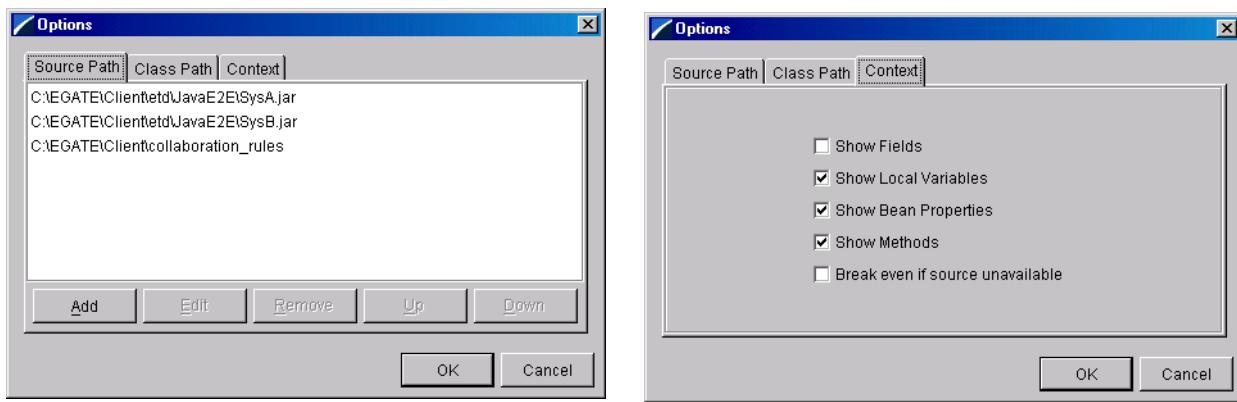
11.1.4 Options Dialog

To set options for the debugger

- On the **Edit** menu, click **Options**.

The **Options** dialog box opens. See Figure 222.

Figure 222 Options Dialog Box



Source Path tab. When you start the debugger, the directories and .jar files that contain the source code of your Java Collaborations and Event Type Definitions are automatically added to the source path used by the debugger to locate source code. You can add, edit, or remove source code locations or change the search order using the **Source Path** tab of the **Options** dialog box.

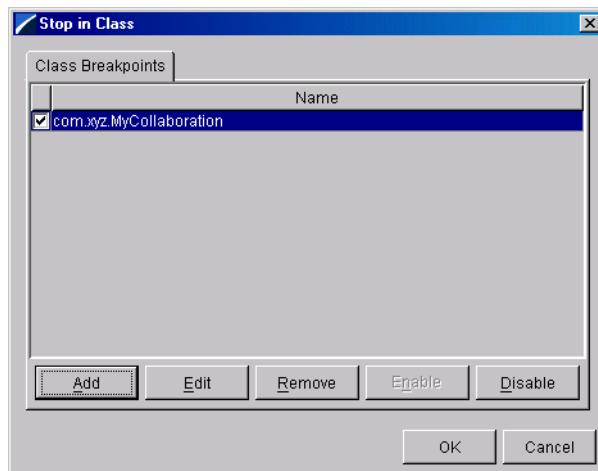
The **Class Path** tab is not used when remote debugging e*Gate Java Collaborations.

Context tab. By default, the debugger displays Java “Bean” properties of the current object and of the local variables of the current method. These are Java class methods that start with **get** or **set**. You may also display all of the fields of the current object and local variables by selecting the **Show Fields** check box in the **Context** tab of the **Options** dialog box. By default the debugger only breaks if source code for the current method is available; to force the debugger to stop even if source code is unavailable, select the **Break even if source unavailable** check box.

To set or modify breakpoints on all of the methods of a Java class

- 1 On the **Debug** menu, click **Stop in Class**. The **Stop in Class** dialog box is displayed. See Figure 223.

Figure 223 Stop in Class Dialog Box



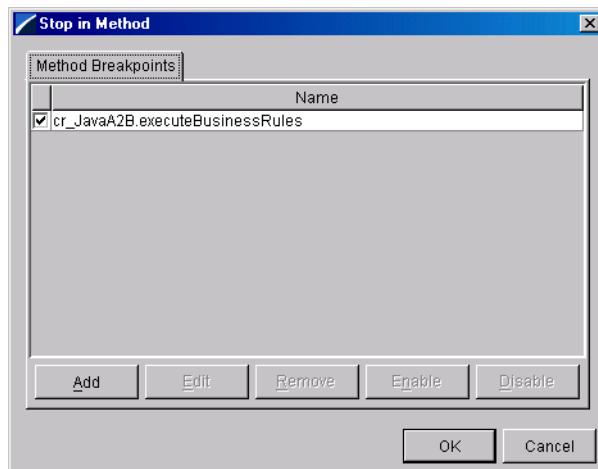
2 You can do one or more of the following:

- To add a class to the breakpoint list, click **Add** and then enter the fully qualified name of a Java class, such as **java.lang.String** or **java.util.Vector**.
Whenever a method of the class is called, control will return to the debugger.
- To modify an existing entry, click **Edit**.
- To delete an entry, click **Remove**.
- To enable or disable a breakpoint, click **Enable** or **Disable** (or select or clear the corresponding checkbox).

To set or modify a breakpoint on a specific method of a Java class

- 1** On the **Debug** menu, click **Stop in Method**. The **Stop in Class** dialog box is displayed. See Figure 224.

Figure 224 Stop in Method Dialog Box



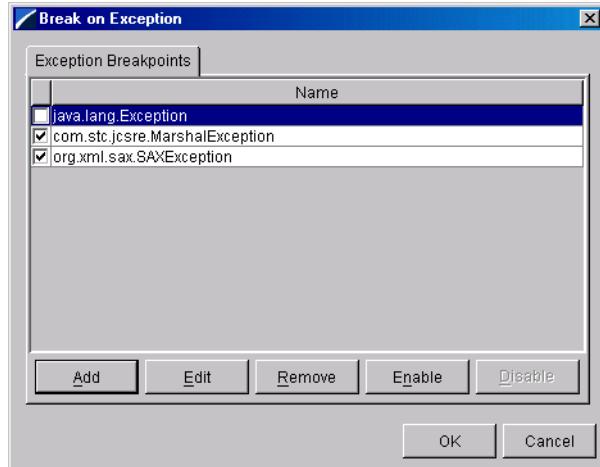
2 You can do one or more of the following:

- To add methods to the list, click **Add** and then enter the name of a method with its fully qualified Java class name, such as **java.lang.String.toUpperCase** or **java.util.Vector.add**.
Whenever the method is called, control will return to the debugger.
- To modify an existing entry, click **Edit**.
- To delete an entry, click **Remove**.
- To enable or disable a method breakpoint, click **Enable** or **Disable** (or select or clear the corresponding checkbox).

To trap Java exceptions

- 1 On the **Debug** menu, click **Break on Exception**. The **Break on Exception** dialog box is displayed. See Figure 225.

Figure 225 Break on Exception Dialog Box



- 2 You can do one or more of the following:
 - To add methods to the list, click **Add** and then enter the fully qualified name of a Java exception, such as **java.lang.Exception** or **java.lang.NullPointerException**. Whenever the exception (or an exception derived from it) is thrown, control will return to the debugger.
 - To modify an existing entry, click **Edit**.
 - To delete an entry, click **Remove**.
 - To enable or disable an exception breakpoint, click **Enable** or **Disable** (or select or clear the corresponding checkbox).

Event Linking and Sequencing (ELS)

In some situations, you want to impose conditions on *a set of Events*, or process *a group of Events* together, or make a decision contingent on the receipt or nonreceipt of *all Events of a certain sort*. For Java Collaboration Rules, you can handle such situations using **Event Linking and Sequencing (ELS)**.

This chapter:

- Explains ELS terminology and concepts.
- Shows you how ELS fits within the rest of the e*Gate system.
- Introduces you to two of the basic triggering criteria and the ELS Wizard.
- Lists and explains the ELS methods of the **ELSController** interface.
- Leads you through a sample implementation where ELS solves a business problem.

12.1 Learning About ELS

ELS is an optional preprocessor for Java Collaboration Rules. When you first start the Collaboration Rules Editor, it is turned off; to turn it on, use **File** command **Enable ELS**. You use ELS when you want to sort Events into separate labeled “buckets” and execute your business rules on a bucket-by-bucket basis rather than an Event-by-Event basis. The bucket label is called a Link Identifier, or Link ID; Events in the same bucket are linked.

When you enable ELS in the Collaboration Rules Editor, three placeholders appear:

- **retrieveLinkIdentifier()** holds the user-written code for setting up buckets.
- **isLinkingComplete()** passes the contents of a full bucket to be processed by the **executeBusinessRules()** section of your Collaboration Rule.
- **onExpire()** holds user-written code for handling buckets on a timer—for example, what to do if a bucket still isn’t full after a certain time has elapsed.

ELS uses virtual memory to keep track of the buckets as they are filled and as their contents age. Because of this, you may need to change the values of certain virtual memory parameters (such as maximum heap size) according to the data you expect, so as to avoid running out of virtual memory. When control is passed to **executeBusinessRules()** for a certain bucket, the bucket is emptied and the system frees all virtual memory occupied by the contents of the bucket.

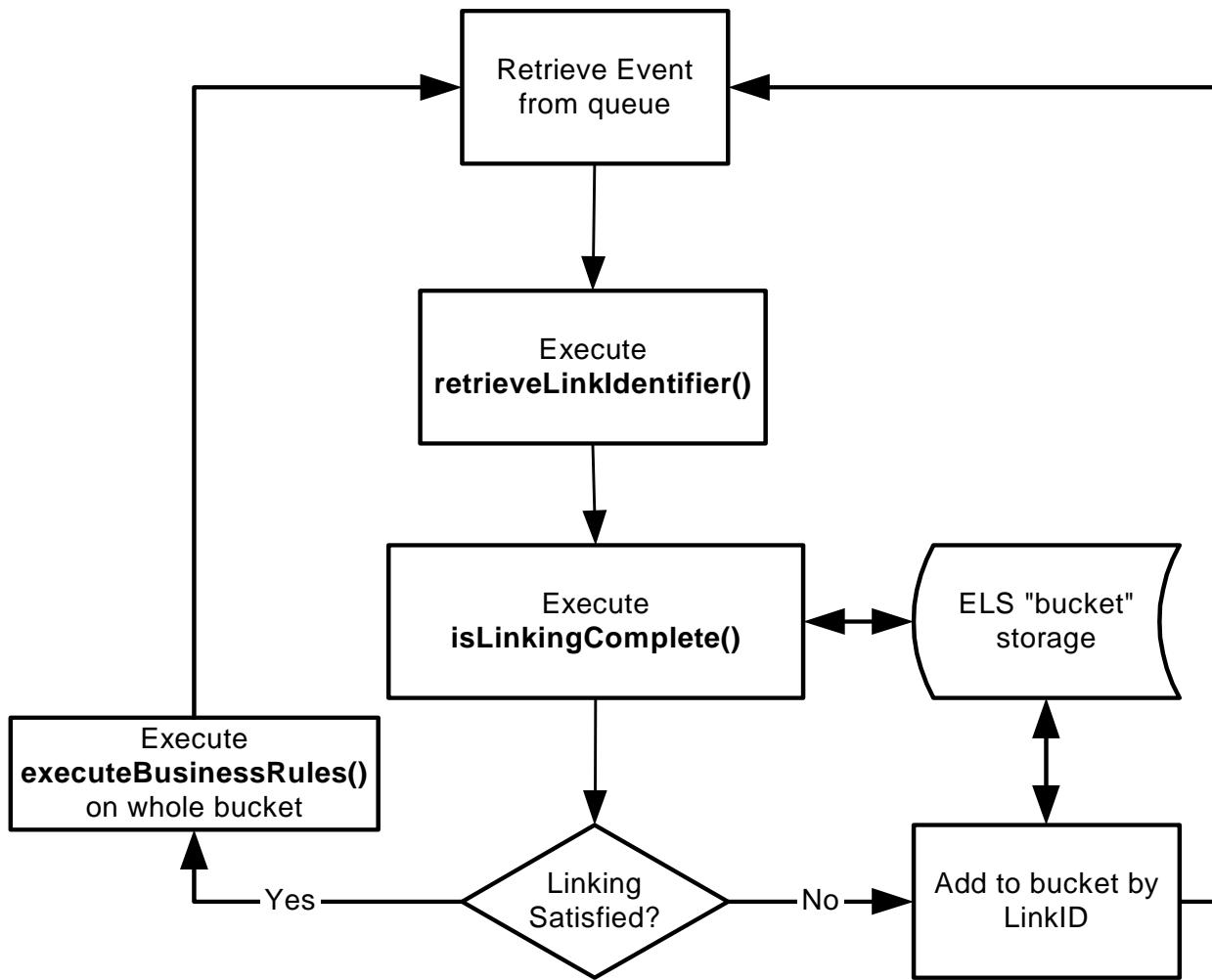
12.1.1 How Does ELS Operate Within e*Gate?

ELS reads messages from input IQs—including STC_Standard, SeeBeyond_JMS, and database IQs—and organizes them in virtual memory according to a user-defined key: the *Link Identifier*, also called the *Link ID*.

Note: To ensure proper processing, take care to use a Link ID key that uniquely identifies the Event—for example, a Social Security Number, not a firstname-lastname.

Figure 226 illustrates the interaction of the methods of an ELS collaboration.

Figure 226 ELS Processing Model

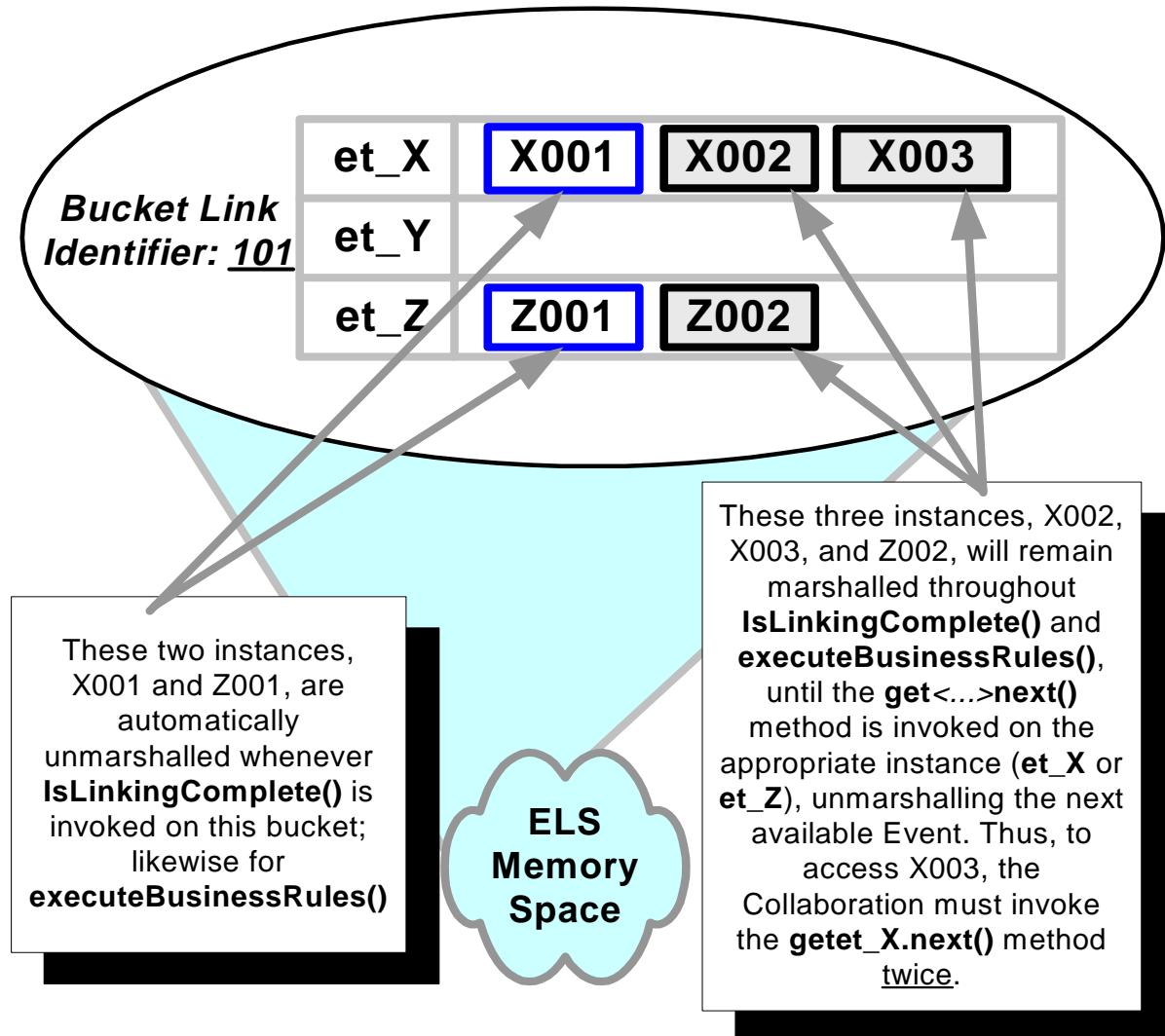


Overview of a simple ELS-enabled Collaboration

Your Collaboration, col_MyELS, links three Event Types named et_X, et_Y, and et_Z. After it has been running for a while, a particular bucket (whose LinkID is 101) holds a total of four Events: three of them (Events X001, X002, and X003) are of Event Type et_X; none of them are Event Type et_Y; and one of them (Event Z001) is of Event Type et_Z.

A new Event arrives. Because your user-written code under `retrieveLinkIdentifier()` was designed to handle the arrival of any of the three Event Types, the first thing it does is call the `available()` method for each inbound Collaboration instance to see if there is a match on the newly arrived Event. As it happens, the new Event, Z002, matches `et_Z`. Next, `isLinkingComplete()` is called. When this occurs, the Collaboration invokes a standard method, `this.prepareInputData()`, which unmarshals the first available Event for each of the inbound Event Type instances in the Collaboration. See Figure 227.

Figure 227 ELS Unmarshalling of Events



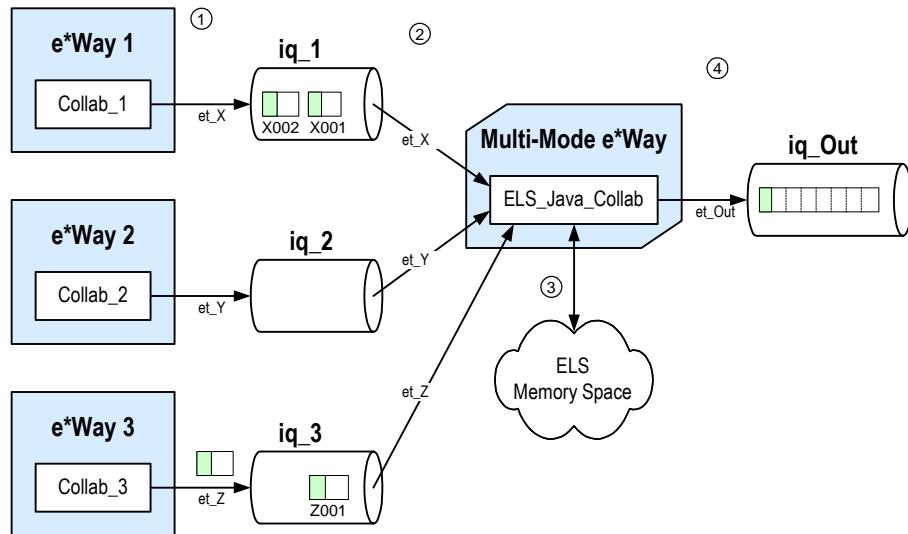
This has the following results:

- For `et_X`, Event X001 is unmarshalled, and calling `available()` will now return true. Subsequent Events of Event Type `et_X`—in this case, X002—can be accessed by calling the instance's `next()` method.
- For `et_Y`, there is no data, and calling `available()` will return false.
- For `et_Z`, Event Z001 is unmarshalled, and calling `available()` will now return true. If `next()` were called, it would unmarshal Z002.

ELS Operation

Upon the firing of a trigger, ELS instantiates all Events in the bucket and starts the main processing defined in the Collaboration Rule. Figure 228 shows the ELS process as an adjunct to a Collaboration hosted by a Multi-Mode e*Way.

Figure 228 ELS in Context With IQs and non-ELS Collaborations



12.1.2 About the SeeBeyond-supplied ELS Methods

The following methods are supplied through the **ELSController** interface.

Methods for buckets and their names

- Iterator **getLinkIdentifiers()** lists all the buckets that have been given names.
- String **getCurrentLinkIdentifier()** tells you the name of the current bucket.
- boolean **isLinkIdentifierExists(aLinkIdentifier)** tells you if a certain bucket exists.

Methods for buckets and their contents

- boolean **hasHappened(aLinkIdentifier)** tells you if the bucket contains anything.
- int **getNumberOfMessages(aLinkIdentifier)** tells you how full the bucket is.
- int **getNumberOfMessages(aLinkIdentifier, aTopicName)** tells you how many Events of a certain Event Type are in this bucket.
- int **getNumberOfMessagesForInstance(aLinkIdentifier, aInstanceName)** tells you how many Events in the bucket were supplied by a particular instance.

Methods for buckets and their timers

- void **setELSEpiration(aLinkIdentifier, aExpirationTime)** sets a timer on a bucket.
- long **getELSEpiration(aLinkIdentifier)** tells you the time setting on a bucket.
- boolean **isCurrentELSEpired()** tells you if the current bucket is past due.
- boolean **isELSEpired(aLinkIdentifier)** tells you if a specified bucket is past due.

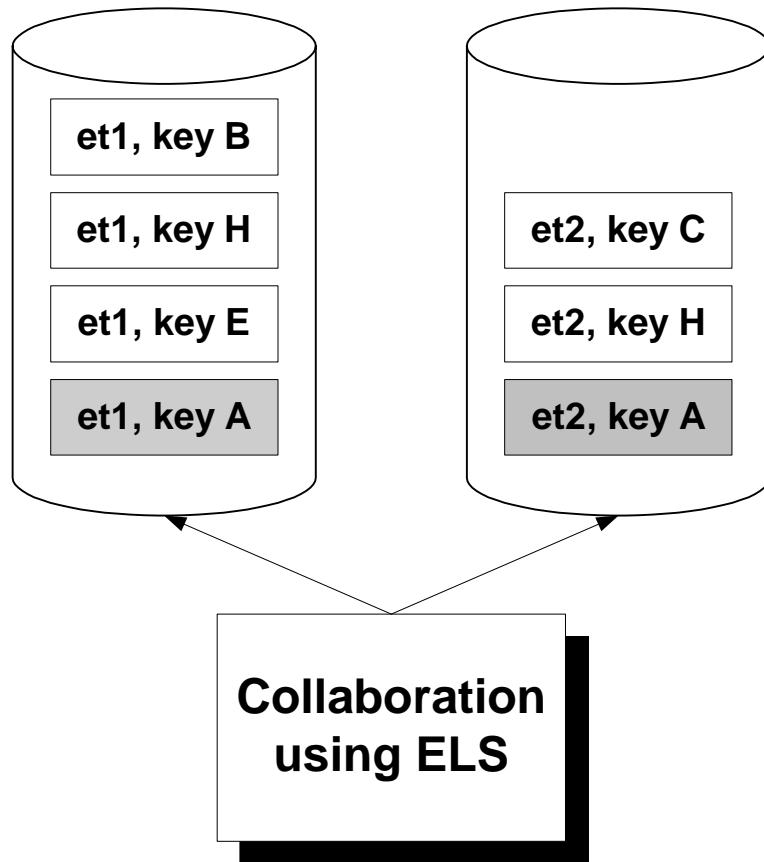
These methods are described in detail in “[ELSController Interface Methods](#)” on [page 589](#).

12.2 Count-Based Triggers

The **getNumberOfMessagesForInstance()** method and the two **getNumberOfMessages()** methods answer the question, “How many Events of a certain key (and possibly other qualifiers) have been received?” In addition, the **hasHappened()** method answers the question, “Has any Event of this key been received?” These four methods are used to set up triggers based on count.

In the schematic shown in Figure 229, the ELS Collaboration has two source Event Types, **et1** and **et2**. Instead of just reading two Events from two IQs and executing business rules on each Event as it arrives, ELS retrieves an Event from each IQ and determines whether they have the same key. Using ELS, these Events can be processed together by the same Collaboration, but only when their keys match (as is the case for the two grey Events with key **A**).

Figure 229 ELS Schematic



In this example, each unmatched Event is held in memory until its mate is retrieved. When the second A Event is received, the Collaboration empties bucket A—in other words, it sends both of the A Events to the `executeBusinessRules()` section of the Collaboration.

The next Event of type **et1** is retrieved, and a call to `getCurrentLinkIdentifier()` reveals that its key value is E. This value is passed to `getNumberOfMessages(E)`, which informs the Collaboration that the E bucket contains only one E Event; the Event is therefore stored in memory until a second E Event is retrieved.

The Events in this example are set up so that the ELSController will detect and pass the two A Events first, bypass the E event, and then pass the two H Events. If the code has never called `setELSExpiration()`, or if it has no code triggered by `onExpire()` becoming true, the three unmatched Events (keys E, C, and B) will wait in memory indefinitely.

Static and Dynamic Count

Count-based triggers can be *static* or *dynamic*. The example in Figure 229 uses a static count: When the count reaches a certain predefined value (in this case 2), the contents of the bucket are emptied into business rules portion of the Collaboration. A more complex Collaboration could use dynamic count—for example, a value passed in by an end user for the number of “hits” to display on a Web page, or a value learned by querying the current content of a cell in a spreadsheet.

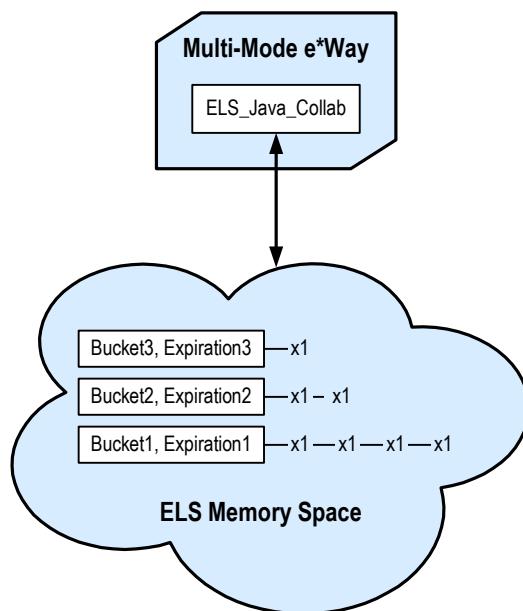
12.3 Timer-Based Triggers

The `isCurrentELSExpired()` method answers the question, “Has the current key reached its expiration time?” The `isELSExpired()` method answers the question, “Has the specified key reached its expiration time?” These are used to set up triggers based on timers.

If a timer has been set and the bucket still isn’t full when reaches its expiration time, control is passed to the user-written code under the `onExpire()` placeholder. If this block of code returns Boolean `true`, the bucket is emptied into the `executeBusinessRules` section in the main portion of the Collaboration. However, if there is no code in the `onExpire()` section, or if the code always returns Boolean `false`, the ELSController loops infinitely. Therefore, any time you use a timer-based trigger, be sure to add some code under `onExpire()` that will eventually return `true`.

Figure 230 shows an example of three buckets with timers.

Figure 230 ELS Buckets With Timers



In Figure 230 above, ELS has pulled seven `x1` Events from the IQs and has created three buckets. This would be the case if the seven messages were scattered with three key values in ELS subtree. When ELS receives the triggering Event, it looks in the ELS subtree for the trigger to receive the key for the trigger. If the ELS Collaboration never receives a key that allows the count-based trigger to fire, eventually one of the timers will expire. If the expired bucket is detected by a call to `isCurrentELSExpired()` or `isELSExpired(aLinkIdentifier)`, the timer can be reset, or any other user-defined action can occur—for example, the expiration of Bucket1 can be used to start the timer on Bucket2. If it not caught, then control passes to the block of code under `onExpire()`.

12.4 The ELS Wizard

12.4.1 About the ELS Wizard

The ELS Wizard is available only when ELS has been enabled. In three steps, it prompts you to specify a type and value for the Link Identifier, a value for message count, and an expiration time in milliseconds—in other words, a bucket name, capacity, and lifetime—and then it generates the corresponding code under the corresponding placeholders. You can then add to this code or modify it using the ELS methods supplied by SeeBeyond.

12.4.2 ELS Wizard Operation

The ELS Wizard leads you through the process of creating a Link Identifier (see [Figure 231 on page 524](#)), specifying a message count (see [Figure 232 on page 525](#)), and then specifying an expiration time in milliseconds (see [Figure 233 on page 526](#)). The code it generates under the `retrieveLinkIdentifier()` placeholder uses the field you specified as a Link Identifier to set a variable named `temp`. The code it generates under the `isLinkingComplete()` placeholder adds calls to `getNumberOfMessagesForInstance()` and `getELSExpiration()`.

Figure 231 shows the results of dragging the **Order** field from the Source Events pane into the **Linking Identifier Type** box of ELS Wizard Step 1.

Figure 231 ELS Wizard Step 1 - Specify Field for Link Identifier

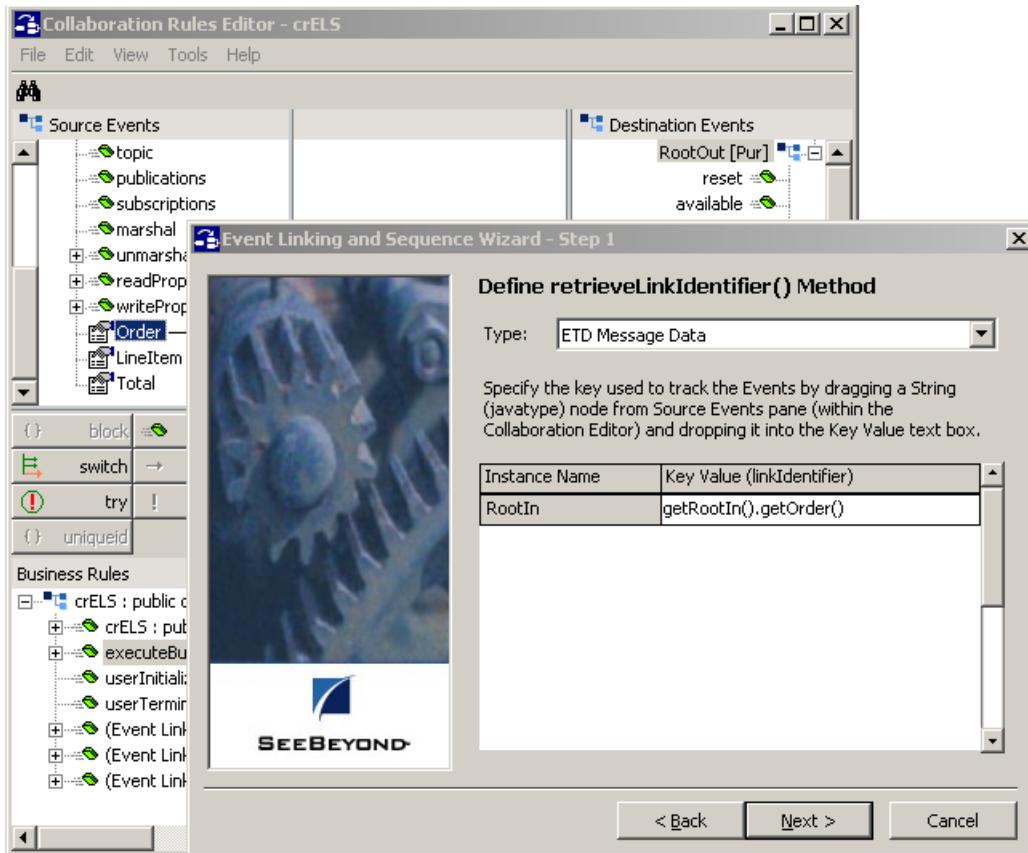


Figure 232 shows step 2 of the wizard, where you supply a message count for the instance name. The number you supply will become a parameter in a call to the `getNumberOfMessagesForInstance()` method for the Link Identifier you specified in step 1, using the current instance name.

Figure 232 ELS Wizard Step 2 - Specify Message Count

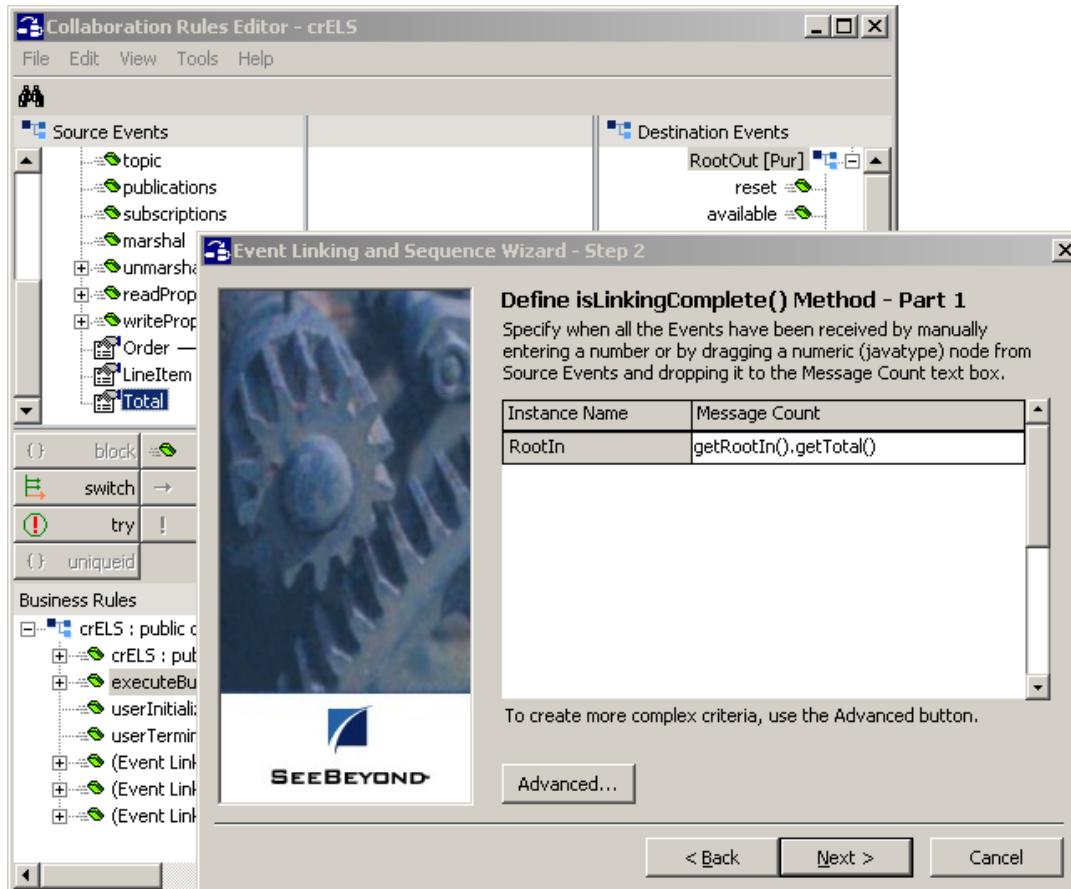


Figure 233 shows step 3 of the wizard, where you supply an expiration time in milliseconds. The number you supply will become a parameter in a call to the **setELSExpiration method()** for the Link Identifier you specified in step 1.

Figure 233 ELS Wizard Step 3 - Specify Expiration

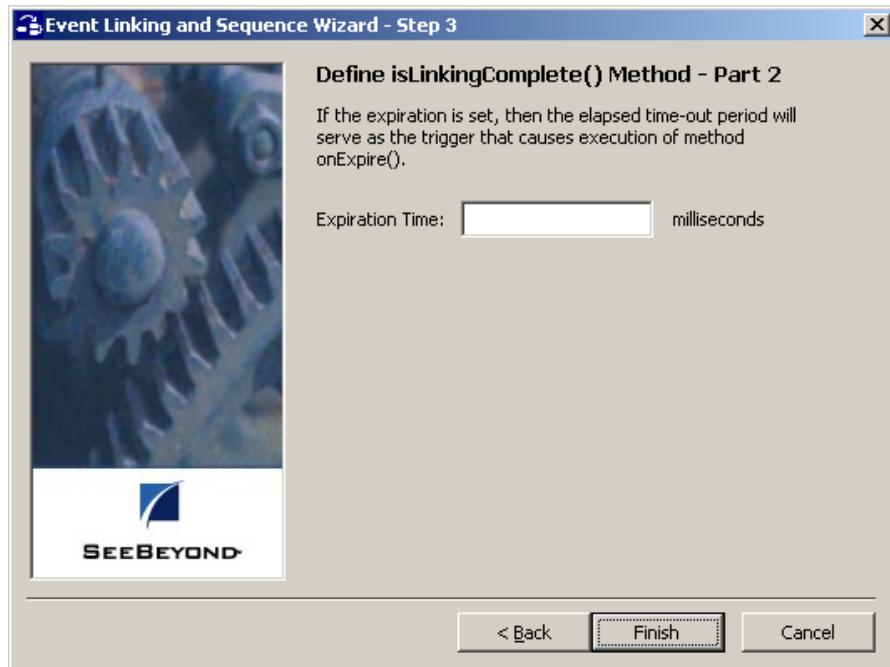
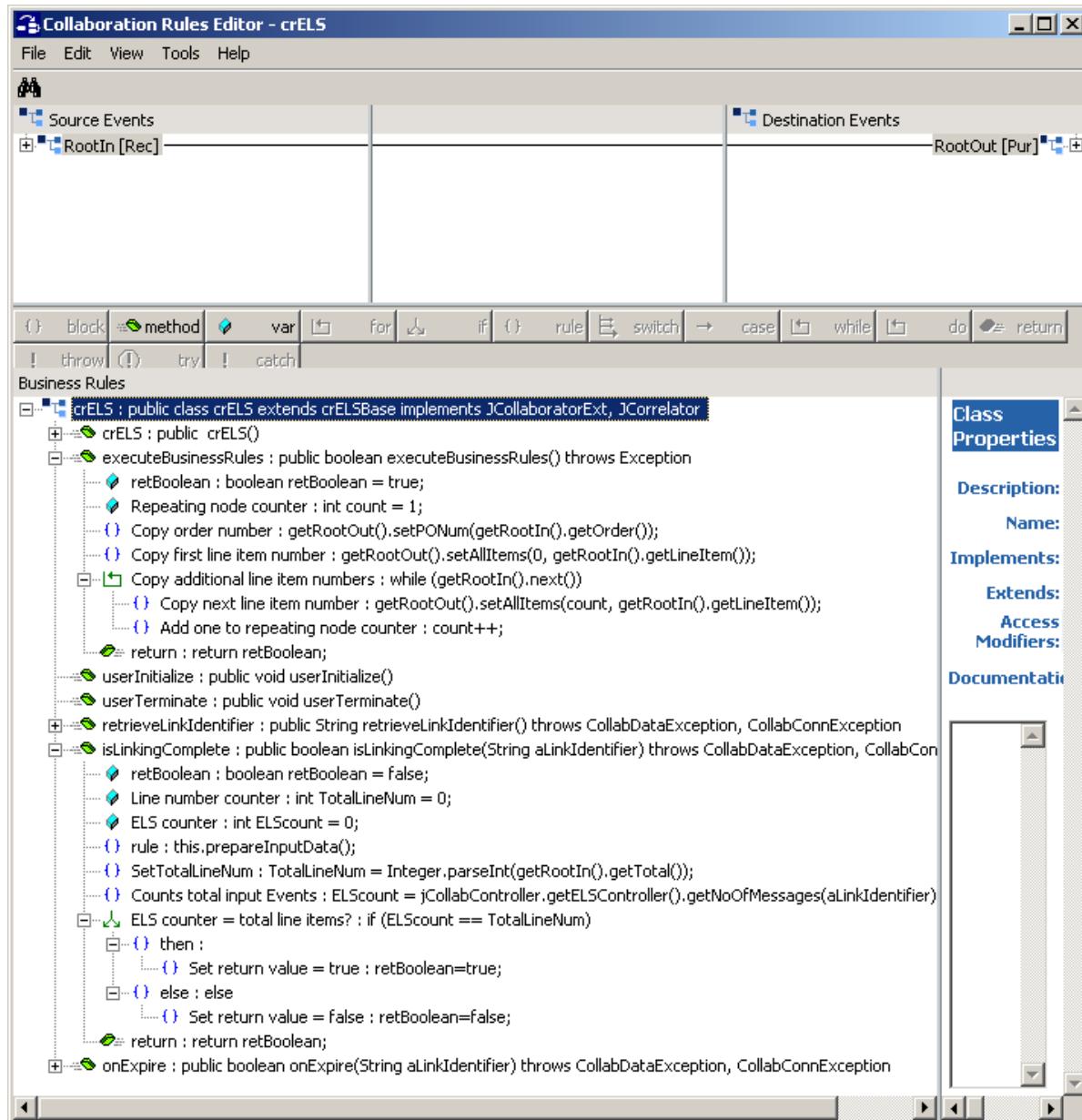


Figure 234 shows the result—notice the new code that the wizard has generated under the `executeBusinessRules()`, `isLinkingComplete()`, and `onExpire()` placeholders

All these methods are discussed in detail in “[ELSController Interface Methods](#)” on [page 589](#).

Figure 234 Code Generated by the ELS Wizard



12.5 Sample Implementation

The following implementation moves you quickly through the key steps of setting up an ELS-enabled Collaboration.

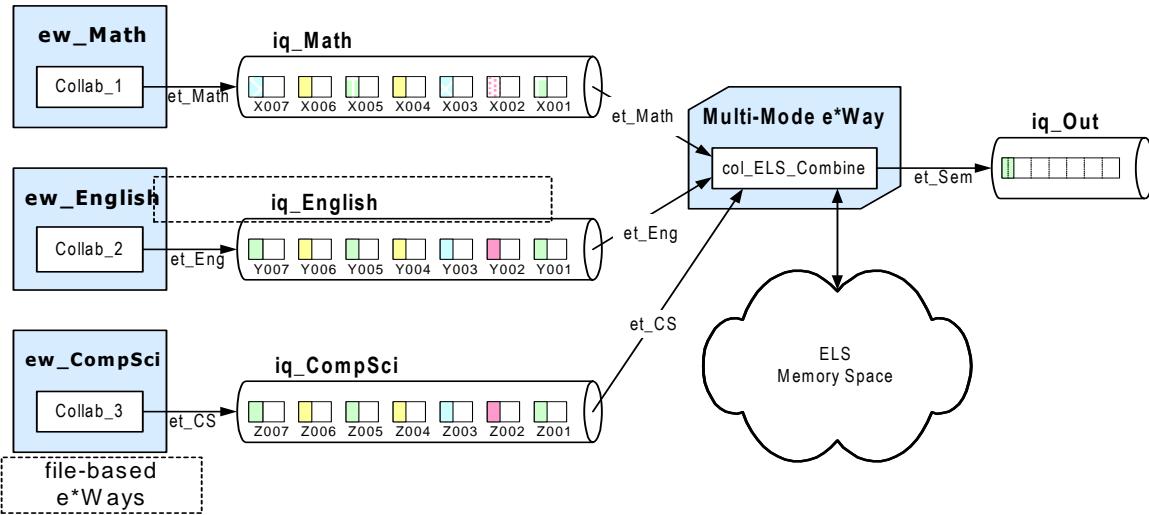
12.5.1 Overview

The sample implementation supposes that a school requires students to pass two courses in each of three subjects—Math, English, and Computer Science—before they are qualified to enroll in advanced studies. The semester grade is withheld until results are received from all six courses. This sample implementation shows you how to create a Collaboration that:

- Receives Events of three different types, each using the same simple ETD (**Course.xsc**), originating from three file-based e*Ways and published to three separate IQs.
- Uses ELS to sort incoming Events according to a specified key: **StudentNum**.
- For each matched set of six incoming Events—two of type **et_Math_Grade**, two of type **et_English_Grade**, and two of type **et_CompSci_Grade**, and all with the same key—transforms the data into one outbound Event: **et_Semester_Grade**.

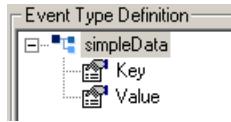
Where [Figure 228 on page 519](#) showed the ELS Collaboration in the context of other e*Gate components, Figure 235 focuses on the details of the ELS_Java_Collaboration itself.

Figure 235 ELS Collaboration



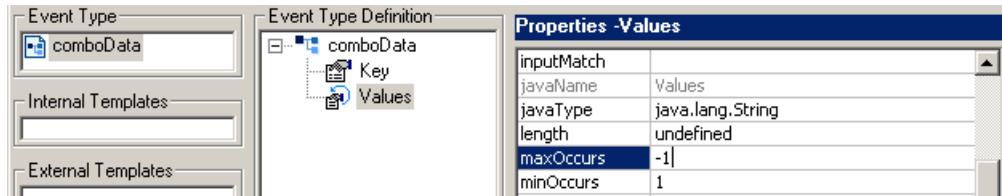
The inbound Event Type instances come from three file-based e*Ways. The three Event Type instances each use an ETD of the following form:

```
{ key | value }
```



When two Events of each Event Type have accumulated, the **executeBusinessRules()** method is triggered. This combines the six Events into a single outbound Event whose ETD has the following form:

```
{ key | [ val_X1 | val_X2 | val_Y1 | val_Y2 | val_Z1 | val_Z2 ] }
```



12.5.2 Steps

For this sample implementation, you will perform the following steps:

- Create the schema and define the Event Types
- Build the ETDs.
- Create the Collaboration Rules.
- Create the business rules for the ELS-enabled Collaboration Rule.

In this book, the steps are abbreviated and incomplete. As a training exercise it would be worthwhile to finish the implementation by creating IQs and e*Ways, building Collaborations, testing, and promoting the schema.

For complete step-by-step instructions taking you from business problem through validation and troubleshooting, see the chapter “e*Gate ELS End-to-End Scenario” in the *Creating an End-to-End Scenario with e*Gate Integrator* guide.

Creating the schema and defining the Event Types

- 1 If necessary, log in to Enterprise Manager with Administrator privileges.
- 2 Create a new schema named **School**.
- 3 Click the **Event Types** folder and create the following four Event Types:
 - et_Math_Grade
 - et_English_Grade
 - et_CompSci_Grade
 - et_Semester_Grade

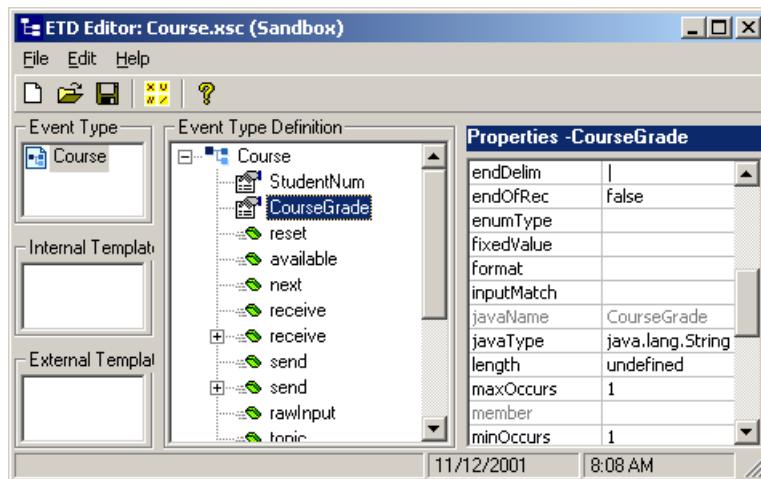
Building the ETDs

You will use the Standard ETD wizard twice, to build the following two ETDs:

- **Course.xsc** for the three Event Types et_Math_Grade, et_English_Grade and et_CompSci_Grade
- **Semester.xsc** for the Event Type et_Semester_Grade

- 1 Edit the properties of et_Math_Grade and define **etd_Course** as follows:
 - ♦ Root node name = **Course**
 - ♦ Package name = **edu.school.credits**
 - ♦ Two nodes, **StudentNum** and **CourseGrade**, with the following properties:
 - ♦ Both nodes are end-delimited with the | (pipe) character.
 - ♦ Both nodes keep the defaults **minOccurs=1** and **maxOccurs=1**
- 2 Compile the ETD and save it in **etd\School\Course.xsc**. See Figure 236.

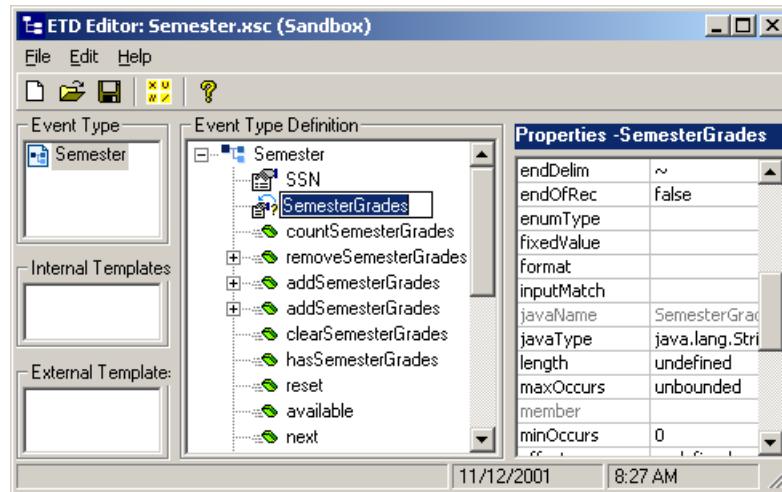
Figure 236 School Example: **etd\School\Course.xsc**



- 3 Assign **Course.xsc** as the ETD for et_English_Grade and et_CompSci_Grade.
- 4 Edit the properties of et_Semester and define **etd_Semester** as follows:
 - ♦ Root node name = **Semester**
 - ♦ Package name = **edu.school.credits**
 - ♦ The first node, named **SSN**, has the following properties:
 - ♦ End-delimited with the | (pipe) character.
 - ♦ Keeps the defaults **minOccurs=1** and **maxOccurs=1**

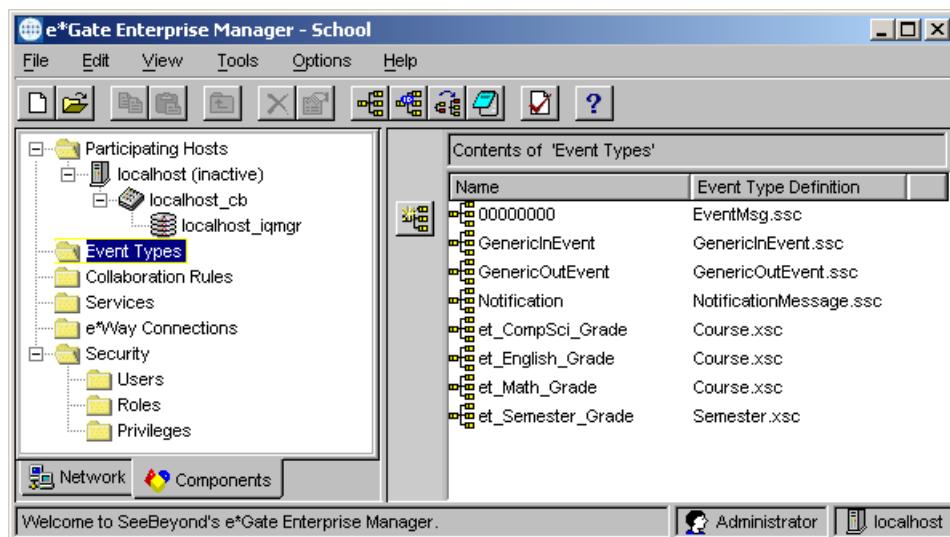
- ♦ The second node, named **SemesterGrades**, has the following properties:
 - ♦ End-delimited with the ~ (tilde) character.
 - ♦ **minOccurs=0** (in other words, data might not be present)
 - ♦ **maxOccurs=-1** (in other words, there is no upper bound on node repetitions)
- 5 Compile the ETD and save it in **etd\School\Semester.xsc**. See Figure 237.

Figure 237 School Example: etd\School\Semester.xsc



After step 5, your schema should look like Figure 238.

Figure 238 School Example: Schema After Creating Event Types and ETDs



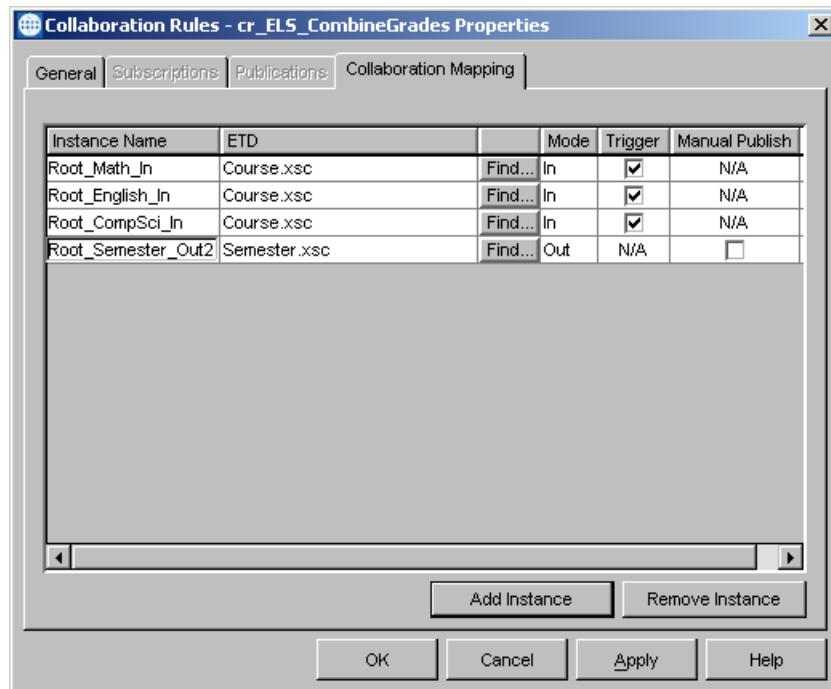
Creating the Collaboration Rules

You will create five Collaboration Rules.

- **cr_Math_In**, **cr_English_In**, and **cr_CompSci_In** are simple Pass Through rules to pull the data in.
- **cr_Semester_Out** is a simple Pass Through rule to push the data out.
- **cr_ELS_CombineGrades** uses the Java Collaboration Service and is ELS-enabled to combine and transform the three inbound data streams into the outbound data stream when all criteria have been met.

- 1 In Enterprise Manager, click **Collaboration Rules** and create the following:
 - **cr_Math_In**—Change the Service from **Java** to **Pass Through**, with subscription and publication to **et_Math_Grade**.
 - **cr_English_In**—Change the Service from **Java** to **Pass Through**, with subscription and publication to **et_English_Grade**.
 - **cr_CompSci_In**—Change the Service from **Java** to **Pass Through**, with subscription and publication to **et_CompSci_Grade**.
 - **cr_Semester_Out**—Change the Service from **Java** to **Pass Through**, with subscription and publication to **et_Semester_Grade**.
- 2 Create **cr_ELS_CombineGrades** keeping the **Java** Collaboration Service.
- 3 Edit its properties, click the **Collaboration Mapping** tab, add four instances, and modify them so that it looks like Figure 239.

Figure 239 School Example: Properties of cr_ELS_CombineGrades



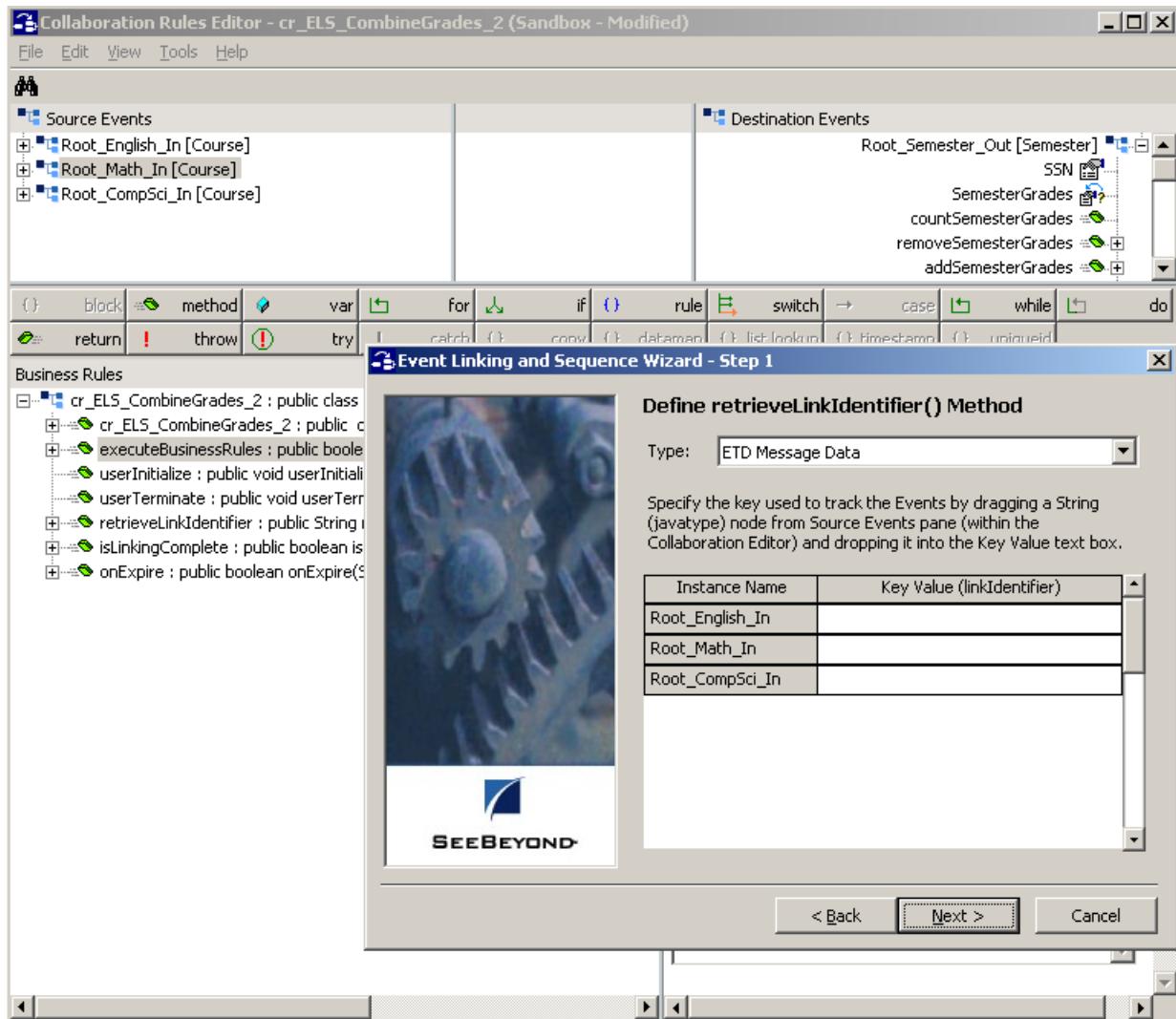
4 Click **Apply**, click the **General** tab, and in the Collaboration Rules area, click **New**.

The Java Collaboration Rules Editor opens.

Creating the ELS Business Rules for cr_ELS_CombineGrades

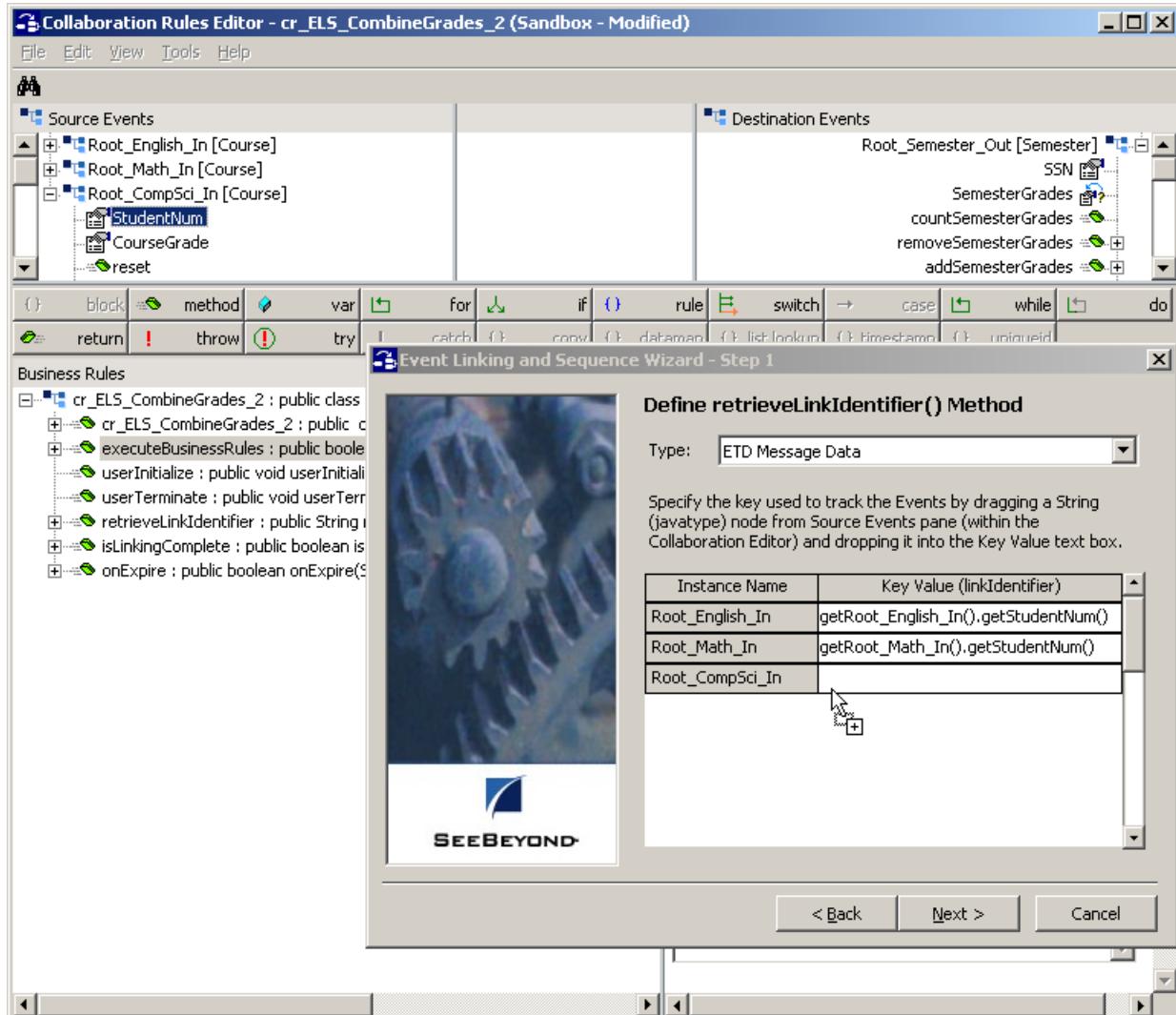
- 1** On the **File** menu, click **Enable ELS**. On the **View** menu, click **Display Code**.
- 2** On the **Tools** menu, click **ELS Wizard**. Click **Next >** to reach Step 1 of the Wizard. See Figure 240.

Figure 240 School Example: cr_ELS_CombineGrades Before Modification



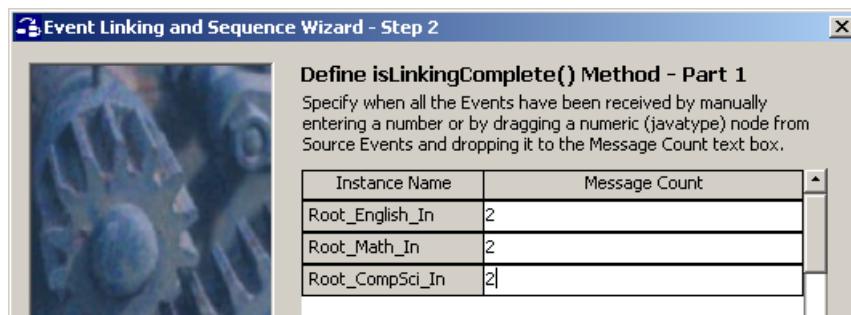
- 3** Expand the **Root_Math_In** instance and drag its **StudentNum** node into the **Field Name** cell for **Root_Math_In**.
- 4** Collapse **Root_Math_In**, expand the **Root_English_In** instance, and then do the equivalent operation for both it and the **Root_CompSci_In** instance. See Figure 241.

Figure 241 School Example: Setting the ELS Link Identifier



- Click Next and then, in Step 2 of the wizard, key in a Message Count of 2 for each of the three Event Type instances. See Figure 242.

Figure 242 School Example: Setting the ELS Message Count

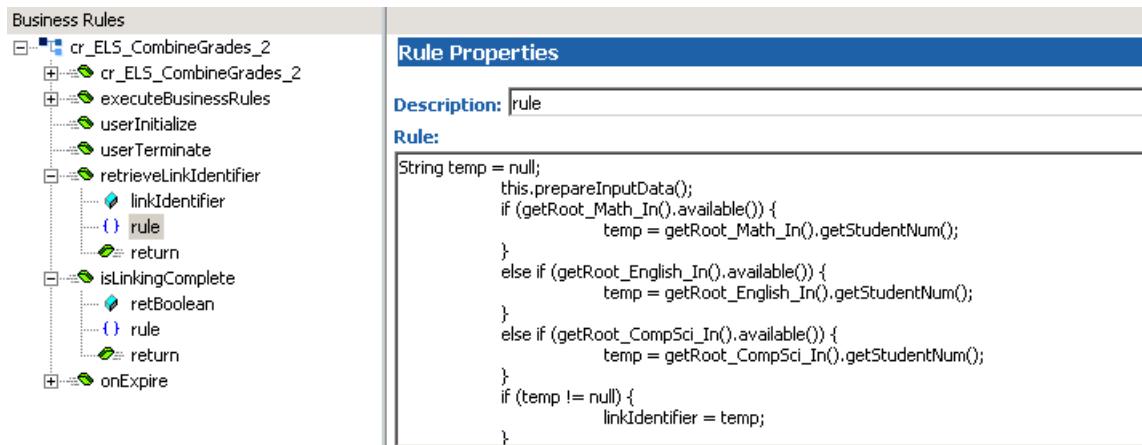


- Click Next. In Step 3, leave the Expiration Time blank and click Finish.

The Wizard closes after it generates Java code according to the values you specified.

- 7 In the Business Rules pane, expand the **retrieveLinkIdentifier()** and **isLinkingComplete()** placeholders and see the Wizard-generated Java code under the **this.prepare.InputData()** rules. As can be seen in Figure 242, the Wizard used the values you specified in Step 1 to set up the common Link Identifier.

Figure 243 School Example: Generated Code Under **retrieveLinkIdentifier()**



At this point, the backbone of the Link Identifier-based ELS code has been created by the Wizard. If appropriate for a more complex Collaboration, you might add to this code or modify it. Notice the usefulness of the **get<instance-name>.available()** method for testing which instance has received data; however, in most cases using ELS, you will want to avoid using the **get<instance-name>.next()** method, since it consumes the Event before it is processed.

Other methods that are particularly useful within the **retrieveLinkIdentifier()** and **isLinkingComplete()** blocks include the **count...()** methods for repeating nodes, the **has...()** methods for nodes that might be empty. In a more complex Collaboration, you might also set up additional criteria under the **onExpire()** section.

In addition to setting up the ELS criteria that must be satisfied before the regular business rules can be run, you need to add logic under the **executeBusinessRules()** block to specify the data transformation that should occur when **isLinkingComplete()** returns **true**.

Creating the Data Transformation Logic Under **executeBusinessRules()**

Within the **executeBusinessRules()** block, knowing that this block of code will only be run when there are two Events stacked up for each of the three inbound Event Type instances, you will create three **while** loops—one for each of the instances—to combine the contents into a single outbound Event Type instance with one key and six values. In this case, the key represents an identifier for the student and the six values represent the grades in the six required courses.

- 1 In the Business Rules pane, expand the **executeBusinessRules()** block. Also, if necessary, expand the **Root_Math_In** and **Root_Semester_Out** instances.

- 2 Click the **retBoolean** variable. Then, on the Business Rules toolbar, click **var** to add a new variable.
- 3 Modify the variable as follows:

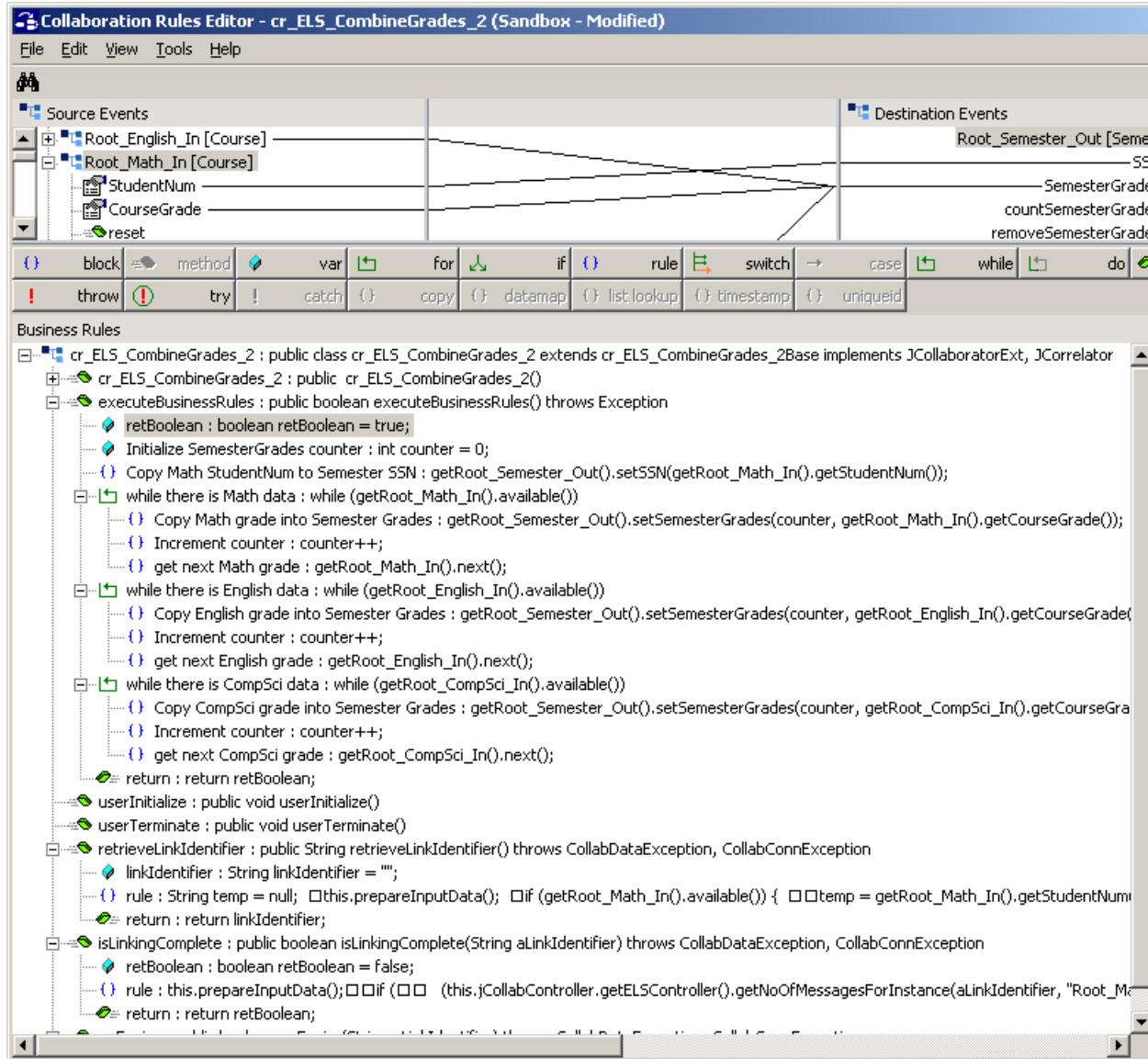
Description	Count Math grades
Name	count_Math
Type	int
Initial Value	0

- 4 In the Business Rules pane, click the newly created variable to see the code added after it. Then, on the Business Rules toolbar, click **while** to add a new while loop. Edit its Description field to read: **While there is Math data**.
- 5 Drag the **CourseGrade** node to the **SemesterGrades** node, creating a new rule. In response to the Select Repetition Instance dialog box, edit the cell for SemesterGrades to **count_Math**. In response to the **Sibling or Child** dialog box, click **Child** to indicate that this rule occurs inside the **while** loop. Edit the Description field of the new rule to read: **Copy Math grade into SemesterGrades**.
- 6 On the Business Rules toolbar, click **rule** to add a new rule. Edit the Description field of the new rule to read: **Increment counter for Math**. In the **Rule** text box, enter the following code:

```
count_Math++
```
- 7 Click the **while** rule (the one you named **While there is Math data**). Then, on the Business Rules toolbar, click **var** to add a new variable. Make the new variable a sibling of the **while** loop.
- 8 Repeat steps 3 through 7, but substitute **English** for **Math**.
- 9 Repeat steps 3 through 6, but substitute **CompSci** for **Math**.

See Figure 244 for the completed ELS-enabled Collaboration Rule.

Figure 244 School Example: Completed ELS-enabled Collaboration Rule



- 10 Compile the completed Collaboration Rule, saving it into a new folder named **collaboration_rules\School**, and then close the Java Collaboration Rules Editor.
- 11 In the Properties dialog box for **cr_ELS_CombineGrades**, click OK.
- 12 Resolve any problems found by the compiler as needed.

If you want to take this sample implementation to completion by creating e*Ways, building Collaborations, and testing your schema using real data, you may want to refer to the *Creating an End-to-End Scenario with e*Gate Integrator* guide for tips on troubleshooting and validation.

XA Transaction Processing

This chapter explains the mechanism that is used in e*Gate 4.5 and later to offer XA-compliant transaction processing with external systems. It summarizes the architecture and mechanism in place to accomplish the goal of Guaranteed Exactly Once Delivery (GEOD) with XA, and it provides scenarios in applying the mechanism.

13.1 Introduction

XA compliance is achieved when cooperating software systems contain sufficient logic to ensure that the transfer of a single unit of data between those systems is neither lost nor duplicated because of a failure condition in one or more of the cooperating systems. e*Gate 4.5 and later satisfies this requirement via utilization of the XA Protocol, from the X/Open Consortium. This vendor-neutral protocol was devised to solve the following problem:

- How to manage transactions between multiple client application programs and multiple database systems.

References

Distributed Transaction Processing: The XA Specification. December, 1991. X/Open Company Limited, Reading, Berkshire, United Kingdom.

Transaction Processing: Concepts and Techniques. 1993. Gray and Reuter, Morgan Kaufmann Publishers, San Francisco, California.

- http://www.unik.no/~mdbase/OS_doc_cc/user1/10_xa.htm
(URL contents © 1997 Object Design, Inc.)
- <http://www.subrahmanyam.com/articles/transactions/NutsAndBoltsOfTP.html>
(URL contents © 1999 Subrahmanyam Allamaraju. Cited with author's consent.)

13.2 Architectural Review

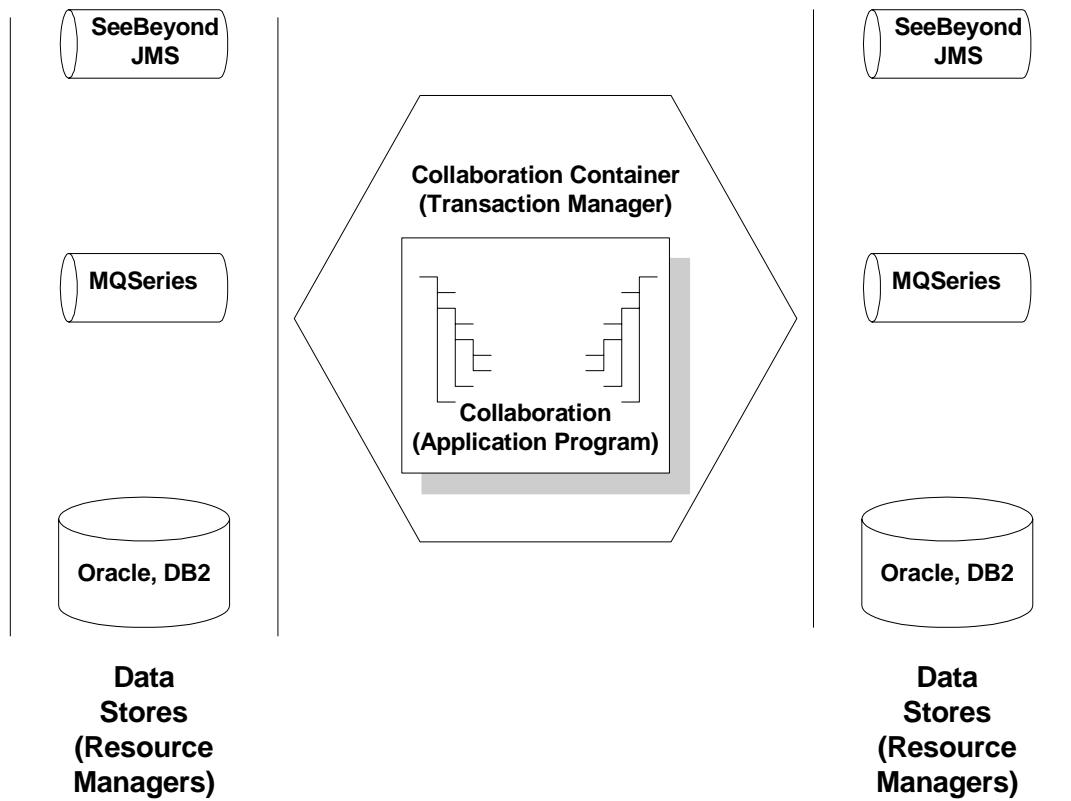
e*Gate manages the publication and subscription of *Events* (messages) to and from *IQs* (internal sources) and external systems. The customized interaction between sources—the *Collaboration*—is built by the e*Gate configuration engineer. Collaborations are executed by *BOBs* and *e*Ways* (core modules) using logic in the *Collaboration container* within the module. The Collaboration container calls the user-created Collaboration.

According to the X/Open specification of XA, Distributed Transaction Processing (DTP) systems are composed of three parties:

- Application Programs (AP) - User applications
- Resource Managers (RM) - Managers of the storage of data
- Transaction Managers (TM) - Coordinates activity across Resource Managers in order to guarantee the integrity of the data across multiple data storage systems

Figure 245 shows the equivalent parties in e*Gate.

Figure 245 e*Gate in a Distributed Transaction Processing (DTP) Context



13.3 Operational Summary

The Collaboration container searches the Collaboration for instances of e*Way Connections and Message Service publications. Each one it finds is treated as a Resource Manager (RM). The Collaboration establishes a connection with each RM. After connections are established, the Collaboration executes its recovery logic to guarantee that no partially complete transactions exist, and then executes its transaction processing model for each event that triggers a Collaboration.

13.4 Working with XA-enabled Collaborations

For a Collaboration to be XA-enabled, all of its sources and destinations must be e*Way Connections that support XA, and XA must be activated in all of its e*Way Connections. If only a subset of the e*Way Connections are XA-compliant, then only those e*Way Connections will be guaranteed to participate in the two-phase commit process.

At this release, the following e*Way Connections support XA:

- Oracle e*Way Connection
- MQSeries e*Way Connection
- DB2 Universal Database e*Way Connection
- SeeBeyond JMS e*Way Connection

For information on configuring a particular e*Way that supports XA, see the user's guide for that e*Way.

To activate XA on a JMS e*Way Connection

- 1 In the Enterprise Manager GUI, with the Components tab active, open the **e*Way Connections** folder and, in the Editor pane (on the right), double-click the e*Way Connection you want to edit.
A Properties dialog box for the e*Way Connection opens to the **General** tab.
- 2 In the **e*Way Connection Configuration File** area, click **Edit**.
The Configuration Editor opens. Initially, the **General Settings** section is active.
- 3 In the **Transaction Type** area, click the XA-compliant option button.
- 4 After you finish setting parameters for the e*Way Connection, close the editor.
- 5 In the Properties dialog box for the e*Way Connection, click **OK**.

When all sources and destinations for a Collaboration are XA-compliant e*Way Connections, the Collaboration container and the external Resource Managers take care of all other details, such as logging, exchanging **prepare**, **commit** and **rollback** calls, and managing recovery.

Mixing XA-Compliant and XA-Noncompliant e*Way Connections

As noted above, a Collaboration can be XA-enabled if and only if all its sources and destinations are XA-compliant e*Way Connections. However, XA-related advantages can accrue to a Collaboration that uses XA-compliant e*Way Connections and also uses exactly one e*Way Connection that is transactional but not XA-compliant—in other words, a mostly XA-enabled Collaboration that also connects to an external system that supports commit/rollback (and is thus transactional) but does not support two-phase commit (and is thus not XA-compliant).

Sequence of operations when exactly one e*Way Connection is XA-noncompliant

When a After the Controller has already started the XA Transaction Manager (TM), the sequence should look like the following.

- 1 A message is received; control passes to the Collaboration.
- 2 The Collaboration runs the code in its `executeBusinessRules()` block and returns.
- 3 XA: The controller causes the XA-enabled Resource Managers (RMs) to Prepare.
- 4 XA: The XA RMs confirm the Prepare. (This is phase one of the two-phase commit.)
- 5 non-XA: The controller causes the non-XA RM to Commit.
- 6 XA: One of the following occurs. (This is phase two of the two-phase commit.)
 - ♦ Upon learning of a successful Commit from the non-XA RM, the controller causes the XA RMs to Commit.
 - ♦ Otherwise, the controller causes the RMs to Rollback.

In this sequence, the only window of potential failure is immediately after a successful Commit by the non-XA RM at step 5, as follows:

- If the XA-noncompliant system fails, then the controller is unaware of the successful Commit that occurred at step 5, and so it needlessly causes the XA-compliant systems to Rollback. But since it can no longer cause the XA-noncompliant system to Rollback, the controller will attempt to redeliver the message to the XA-noncompliant system.
- If an XA-compliant system fails, then the controller correctly causes all XA-compliant systems to Rollback. But since it can no longer cause the XA-noncompliant system to Rollback, the controller will attempt to redeliver the message to the XA-noncompliant system.

Note: *Collaborations whose sources or destinations include two or more XA-noncompliant external systems gain no advantage from enabling XA on those e*Way Connections that support XA.*

Appendix A

Java Classes and Methods

Java classes and methods available to all standard ETDs are as follows.

Table 82 Java Classes and Methods

For information on methods of this type...	... refer to:
Alert	JCollaboration Class (com.stc.jcsre) on page 599.
Base64	Base64Utils Class (com.stc.eways.util) on page 549.
Collaboration	CollabUtils Class (com.stc.eways.util) on page 552; JCollabController Class (com.stc.common.collabService) on page 573.
Date manipulation	SimpleDateFormat class (java.text) — http://java.sun.com/products/jdk/1.2/docs/api/java/text/SimpleDateFormat.html ; DateUtils Class (com.stc.eways.util) on page 559.
ELS	ELSController Interface Methods on page 589.
Encryption	ScEncrypt Class (com.stc.common.utils) on page 628.
File	FileUtils Class (com.stc.eways.util) on page 570.
Internationalization	Character Encoding and Internationalization Methods on page 580.
Logging	EGate Class (com.stc.common.collabService) on page 562.
Mainframe	Mainframe Class (com.stc.eways.util) on page 616.
Mapping	MapUtils Class (com.stc.eways.util) on page 621.
Numeric manipulation	BigDecimal class (java.math) — http://java.sun.com/products/jdk/1.2/docs/api/java/math/BigDecimal.html .
Registry	JCollabController Class (com.stc.common.collabService) on page 573.
Sorting	Arrays class (java.util) — http://java.sun.com/products/jdk/1.2/docs/api/java/util/Arrays.html ; QSort Class (com.stc.common.utils) on page 627.
String manipulation	String class (java.lang) — http://java.sun.com/products/jdk/1.2/docs/api/java/lang/String.html ; StringUtils Class (com.stc.eways.util) on page 656.
Subcollaboration Rules	JCollabController Class (com.stc.common.collabService) on page 573; JSubCollabMapInfo Class (com.stc.common.collabService) on page 603.
Type conversion	STCTypeConverter Class (com.stc.eways.util) on page 630

A.1 Index to Methods for Standard Java-enabled ETDs

- [a2e\(\)](#) on page 616 (in package com.stc.eways.util, class **Mainframe**)
- [asHex\(\)](#) on page 552 (in package com.stc.eways.util, class **CollabUtils**)
- [available\(\)](#) on page 334 (in Java Collaboration Rules Editor GUI)
- [base64Decode\(\)](#) on page 549 (in package com.stc.jcsre.Base64, class **Base64Utils**)
- [base64DecodeToByte\(\)](#) on page 550 (in package com.stc.jcsre.Base64, class **Base64Utils**)
- [byteToBase64String\(\)](#) on page 550 (in package com.stc.jcsre.Base64, class **Base64Utils**)
- [collabDebug\(\)](#) on page 562 (in package com.stc.common.collabService, class **EGate**)
- [collabError\(\)](#) on page 563 (in package com.stc.common.collabService, class **EGate**)
- [collabFatal\(\)](#) on page 563 (in package com.stc.common.collabService, class **EGate**)
- [collabInfo\(\)](#) on page 564 (in package com.stc.common.collabService, class **EGate**)
- [collabTrace\(\)](#) on page 565 (in package com.stc.common.collabService, class **EGate**)
- [collabWarning\(\)](#) on page 565 (in package com.stc.common.collabService, class **EGate**)
- [copyProperties\(\)](#) on page 575 (in package com.stc.common.collabService, class **JCollabController**)
- [count_MyNode_0](#) on page 329 (in Java Collaboration Rules Editor GUI)
- [createSubCollabMapInfo\(\)](#) on page 575 (in package com.stc.common.collabService, class **JCollabController**)
- [decrypt\(\)](#) on page 628 (in package com.stc.common.utils, interface **ScEncrypt**)
- [doMap\(\)](#) on page 622 (in package com.stc.eways.util, class **MapUtils**)
- [doOffsetTrunc\(\)](#) on page 553 (in package com.stc.eways.util, class **CollabUtils**)
- [e2a\(\)](#) on page 617 (in package com.stc.eways.util, class **Mainframe**)
- [e2S\(\)](#) on page 617 (in package com.stc.eways.util, class **Mainframe**)
- [empty\(\)](#) on page 656 (in package com.stc.eways.util, class **StringUtils**)
- [encrypt\(\)](#) on page 628 (in package com.stc.common.utils, interface **ScEncrypt**)
- [eventSend\(\)](#) on page 599 (in package com.stc.jcsre, class **Alerter**)
- [executeBusinessRules\(\)](#)—see “**Methods Presupplied When You Start the Editor**” on page 312.
- [flushAllLinkIdentifiers\(\)](#) on page 590 (in package com.stc.common.collabService, class **JCollabController**)
- [format\(\)](#) on page 559 (in package com.stc.eways.util, class **DateUtils**)
- [get_MyNode_0](#) on page 330 (in Java Collaboration Rules Editor GUI)

- [getCallingCollaboration\(\)](#) on page 603 (in package com.stc.common.collabService, class **JSubCollabMapInfo**)
- [getClassFullPath\(\)](#) on page 604 (in package com.stc.common.collabService, class **JSubCollabMapInfo**)
- [getClassName\(\)](#) on page 605 (in package com.stc.common.collabService, class **JSubCollabMapInfo**)
- [getCollaborationName\(\)](#) on page 576 (in package com.stc.common.collabService, class **JCollabController**)
- [getCtlFileFullPath\(\)](#) on page 605 (in package com.stc.common.collabService, class **JSubCollabMapInfo**)
- [getCtlFileName\(\)](#) on page 606 (in package com.stc.common.collabService, class **JSubCollabMapInfo**)
- [getCurrentLinkIdentifier\(\)](#) on page 590 (in package com.stc.common.collabService, class **JCollabController**)
- [getEgateBaseDirectory\(\)](#) on page 576 (in package com.stc.common.collabService, class **JCollabController**)
- [getELSExpiration\(\)](#) on page 591 (in package com.stc.common.collabService, class **ELSController**)
- [getEventTypeDefinition\(\)](#) on page 607 (in package com.stc.common.collabService, class **JSubCollabMapInfo**)
- [getEventTypeDefinitionPath\(\)](#) on page 607 (in package com.stc.common.collabService, class **JSubCollabMapInfo**)
- [getIncomingEncoding\(\)](#) on page 584 (in package com.stc.common.collabService, class **JCollabController**)
- [getInputData\(\)](#) on page 608 (in package com.stc.common.collabService, class **JSubCollabMapInfo**)
- [getInputTopicName\(\)](#) on page 609 (in package com.stc.common.collabService, class **JSubCollabMapInfo**)
- [getLinkIdentifiers\(\)](#) on page 592 (in package com.stc.common.collabService, class **JCollabController**)
- [getMarshalEncoding\(\)](#) on page 584 (in package com.stc.common.collabService, class **JCollabController**)
- [getNoOfMessagesForInstance\(\)](#) on page 593 (in package com.stc.common.collabService, class **JCollabController**)
- [getNumberOfMessages\(\)](#) on page 593 (in package com.stc.common.collabService, class **JCollabController**)
- [getOutgoingEncoding\(\)](#) on page 585 (in package com.stc.common.collabService, class **JCollabController**)
- [getOutputData\(\)](#) on page 609 (in package com.stc.common.collabService, class **JSubCollabMapInfo**)

- [getParentReferenceETD\(\)](#) on page 610 (in package com.stc.common.collabService, class **JSubCollabMapInfo**)
- [getParentReferenceInstanceName\(\)](#) on page 611 (in package com.stc.common.collabService, class **JSubCollabMapInfo**)
- [getPropertyNames\(\)](#) on page 577 (in package com.stc.common.collabService, class **JCollabController**)
- [getRuleName\(\)](#) on page 611 (in package com.stc.common.collabService, class **JSubCollabMapInfo**)
- [getUnmarshalEncoding\(\)](#) on page 585 (in package com.stc.common.collabService, class **JCollabController**)
- [has_MyNode_\(\)](#) on page 331 (in Java Collaboration Rules Editor GUI)
- [hasHappened\(\)](#) on page 594 (in package com.stc.common.collabService, class **JCollabController**)
- [invoke\(\)](#) on page 578 (in package com.stc.common.collabService, class **JCollabController**)
- [isCurrentELSExpired\(\)](#) on page 595 (in package com.stc.common.collabService, class **JCollabController**)
- [isELSExpired\(\)](#) on page 595 (in package com.stc.common.collabService, class **JCollabController**)
- [isFlushMode\(\)](#) on page 596 (in package com.stc.common.collabService, class **JCollabController**)
- [isLinkIdentifierExists\(\)](#) on page 596 (in package com.stc.common.collabService, class **JCollabController**)
- [isLinkingComplete\(\)](#) on page 597 (in package com.stc.common.collabService, class **JCollabController**)
- [isManualPublish\(\)](#) on page 612 (in package com.stc.common.collabService, class **JSubCollabMapInfo**)
- [isMonkDatePattern\(\)](#) on page 554 (in package com.stc.eways.util, class **CollabUtils**)
- [isPublisher\(\)](#) on page 613 (in package com.stc.common.collabService, class **JSubCollabMapInfo**)
- [isSubCollaboration\(\)](#) on page 579 (in package com.stc.common.collabService, class **JCollabController**)
- [isTrigger\(\)](#) on page 613 (in package com.stc.common.collabService, class **JSubCollabMapInfo**)
- [marshal\(\)](#) on page 335 (in Java Collaboration Rules Editor GUI)
- [next\(\)](#) on page 336 (in Java Collaboration Rules Editor GUI)
- [onExpire\(\)](#) on page 597 (in package com.stc.common.collabService, class **JCollabController**)
- [padCenter\(\)](#) on page 657 (in package com.stc.eways.util, class **StringUtils**)

- [padLeft\(\)](#) on page 658 (in package com.stc.eways.util, class **StringUtils**)
- [padRight\(\)](#) on page 659 (in package com.stc.eways.util, class **StringUtils**)
- [parseMap\(\)](#) on page 623 (in package com.stc.eways.util, class **MapUtils**)
- [publications\(\)](#) on page 337 (in Java Collaboration Rules Editor GUI)
- [qsort\(\)](#) on page 627 (in package com.stc.common.utils, interface **QSort**)
- [rawInput\(\)](#) on page 338 (in Java Collaboration Rules Editor GUI)
- [readBytes\(\)](#) on page 570 (in package com.stc.eways.util, class **FileUtils**)
- [readMap\(\)](#) on page 624 (in package com.stc.eways.util, class **MapUtils**)
- [readProperty\(\)](#) on page 339 (in Java Collaboration Rules Editor GUI)
- [readString\(\)](#) on page 570 (in package com.stc.eways.util, class **FileUtils**)
- [receive\(\)](#) on page 341 (in Java Collaboration Rules Editor GUI)
- [renderMap\(\)](#) on page 625 (in package com.stc.eways.util, class **MapUtils**)
- [reset\(\)](#) on page 342 (in Java Collaboration Rules Editor GUI)
- [retrieveLinkIdentifier\(\)](#) on page 598 (in package com.stc.common.collabService, class **JCollabController**)
- [retrieveRegistryFile\(\)](#) on page 580 (in package com.stc.common.collabService, class **JCollabController**)
- [send\(\)](#) on page 343 (in Java Collaboration Rules Editor GUI)
- [set_MyNode_\(\)](#) on page 332 (in Java Collaboration Rules Editor GUI)
- [setELSExpiration\(\)](#) on page 598 (in package com.stc.common.collabService, class **JCollabController**)
- [setIncomingEncoding\(\)](#) on page 586 (in package com.stc.common.collabService, class **JCollabController**)
- [setInstanceMap\(\)](#) on page 614 (in package com.stc.common.collabService, class **JSubCollabMapInfo**)
- [setMarshalEncoding\(\)](#) on page 587 (in package com.stc.common.collabService, class **JCollabController**)
- [setOutgoingEncoding\(\)](#) on page 587 (in package com.stc.common.collabService, class **JCollabController**)
- [setUnmarshalEncoding\(\)](#) on page 588 (in package com.stc.common.collabService, class **JCollabController**)
- [sprintf\(\)](#) on page 555 (in package com.stc.eways.util, class **CollabUtils**)
- [string2Base64\(\)](#) on page 551 (in package com.stc.jcsre.Base64, class **Base64Utils**)
- [subscriptions\(\)](#) on page 344 (in Java Collaboration Rules Editor GUI)
- [swapInt\(\)](#) on page 618 (in package com.stc.eways.util, class **Mainframe**)
- [swapLong\(\)](#) on page 619 (in package com.stc.eways.util, class **Mainframe**)
- [swapShort\(\)](#) on page 619 (in package com.stc.eways.util, class **Mainframe**)

- [timeStamp\(\)](#) on page 560 (in package com.stc.eways.util, class **DateUtils**)
- [toBoolean\(\)](#) on page 632 (in package com.stc.eways.util, class **STCTypeConverter**)
- [toBooleanPrimitive\(\)](#) on page 631 (in package com.stc.eways.util, class **STCTypeConverter**)
- [toByte\(\)](#) on page 635 (in package com.stc.eways.util, class **STCTypeConverter**)
- [toByteArray\(\)](#) on page 636 (in package com.stc.eways.util, class **STCTypeConverter**)
- [toBytePrimitive\(\)](#) on page 634 (in package com.stc.eways.util, class **STCTypeConverter**)
- [toCharacter\(\)](#) on page 639 (in package com.stc.eways.util, class **STCTypeConverter**)
- [toCharPrimitive\(\)](#) on page 638 (in package com.stc.eways.util, class **STCTypeConverter**)
- [toDouble\(\)](#) on page 642 (in package com.stc.eways.util, class **STCTypeConverter**)
- [toDoublePrimitive\(\)](#) on page 640 (in package com.stc.eways.util, class **STCTypeConverter**)
- [toFloat\(\)](#) on page 644 (in package com.stc.eways.util, class **STCTypeConverter**)
- [toFloatPrimitive\(\)](#) on page 643 (in package com.stc.eways.util, class **STCTypeConverter**)
- [toHex\(\)](#) on page 556 (in package com.stc.eways.util, class **CollabUtils**)
- [toInteger\(\)](#) on page 647 (in package com.stc.eways.util, class **STCTypeConverter**)
- [toIntegerPrimitive\(\)](#) on page 646 (in package com.stc.eways.util, class **STCTypeConverter**)
- [toJavaDatePattern\(\)](#) on page 556 (in package com.stc.eways.util, class **CollabUtils**)
- [toLong\(\)](#) on page 650 (in package com.stc.eways.util, class **STCTypeConverter**)
- [toLongPrimitive\(\)](#) on page 648 (in package com.stc.eways.util, class **STCTypeConverter**)
- [toShort\(\)](#) on page 652 (in package com.stc.eways.util, class **STCTypeConverter**)
- [toShortPrimitive\(\)](#) on page 651 (in package com.stc.eways.util, class **STCTypeConverter**)
- [toString\(\)](#) on page 653 (in package com.stc.eways.util, class **STCTypeConverter**)
- [topic\(\)](#) on page 345 (in Java Collaboration Rules Editor GUI)
- [traceln\(\)](#) on page 566 (in package com.stc.common.collabService, class **EGate**)
- [transformDate\(\)](#) on page 560 (in package com.stc.eways.util, class **DateUtils**)
- [trimBoth\(\)](#) on page 660 (in package com.stc.eways.util, class **StringUtils**)
- [trimLeft\(\)](#) on page 660 (in package com.stc.eways.util, class **StringUtils**)
- [trimRight\(\)](#) on page 661 (in package com.stc.eways.util, class **StringUtils**)

- [uniqueId\(\)](#) on page 557 (in package com.stc.eways.util, class **CollabUtils**)
- [unmarshal\(\)](#) on page 346 (in Java Collaboration Rules Editor GUI)
- userInitialize()—see “[Methods Presupplied When You Start the Editor](#)” on [page 312](#).
- userTerminate()—see “[Methods Presupplied When You Start the Editor](#)” on [page 312](#).
- [write\(\)](#) on page 571 (in package com.stc.eways.util, class **FileUtils**)
- [writeMap\(\)](#) on page 626 (in package com.stc.eways.util, class **MapUtils**)
- [writeProperty\(\)](#) on page 347 (in Java Collaboration Rules Editor GUI)

For the Business Rules methods, see [Methods Presupplied When You Start the Editor](#) on page 312.

Note: *Additional methods are available to ETDs created by specialized ETD builders. For example, an ETD created by the IDocWizard contains methods specific to SAP implementations. For a list of the methods for a specific e*Way or ETD builder, see the user's guide for the corresponding e*Way, ETD builder library, or toolkit.*

A.2 Base64Utils Class (com.stc.eways.util)

The SeeBeyond class com.stc.eways.util.**Base64Utils** extends SeeBeyond class com.stc.jcsre.**Base64** to provide the following public methods for working with Multipurpose Internet Mail Extension (MIME) Base64 encodings:

- [base64Decode\(\)](#) on page 549
- [base64DecodeToByte\(\)](#) on page 550
- [byteToBase64String\(\)](#) on page 550
- [string2Base64\(\)](#) on page 551

References

- http://www.w3.org/Protocols/rfc1341/5_Content-Transfer-Encoding.html
-

base64Decode()

The two **base64Decode()** methods are provided in the **Base64Utils** class, supplied in SeeBeyond class com.stc.jcsre.**Base64** (via com.stc.eways.util.**Base64Utils**):

```
package com.stc.eways.util;  
  
public class Base64Utils  
  
extends com.stc.jcsre.Base64
```

Syntax

```
public static java.lang.String Base64Utils.base64Decode(  
            byte[] _bytes)  
  
public static java.lang.String Base64Utils.base64Decode(  
            java.lang.String _str)
```

Description

Decodes a byte array or string supplied in MIME Base64-encoded format and returns the decoded data as a string. (Compare [base64DecodeToByte\(\)](#) on page 550.)

Parameters

Name	Type	Description
_bytes	byte[]	The array of bytes to be decoded.
_str	java.lang.String	The data to be decoded.

Return Type

java.lang.String

Throws

java.lang.Exception—thrown if the input byte array or text string cannot be decoded.

base64DecodeToByte()

The **base64DecodeToByte()** method is provided in the **Base64Utils** class, supplied in SeeBeyond class **com.stc.jcsre.Base64** (via **com.stc.eways.util.Base64Utils**):

```
package com.stc.jcsre;
public class Base64Utils
    extends com.stc.jcsre.Base64
```

Syntax

```
public static byte[] base64DecodeToByte(java.lang.String data)
```

Description

Decodes a string supplied in MIME Base64-encoded format and returns the decoded data as a byte array. (Compare [base64Decode\(\)](#) on page 549.)

Parameters

Name	Type	Description
data	java.lang.String	The data to be decoded.

Return Type

byte[]—in other words, an array of bytes.

Throws

java.io.IOException—thrown if there are input/output errors.

byteToBase64String()

The **byteToBase64()** method is provided in the **Base64Utils** class, supplied in SeeBeyond class **com.stc.jcsre.Base64** (via **com.stc.eways.util.Base64Utils**):

```
package com.stc.jcsre;
public class Base64Utils
    extends com.stc.jcsre.Base64
```

Syntax

```
public static java.lang.String byteToBase64String(byte[] data)
```

Description

Encodes a byte array into MIME Base64 encoding and returns the encoded data as a text string. (Compare [string2Base64\(\)](#) on page 551.)

Parameters

Name	Type	Description
data	byte[]	The array of bytes to be encoded.

Return Type

java.lang.String

Throws

java.io.IOException—thrown if there are input/output errors.

string2Base64()

The two **string2Base64()** methods are provided in the **Base64Utils** class, supplied in SeeBeyond class **com.stc.jcsre.Base64** (via **com.stc.eways.util.Base64Utils**):

```
package com.stc.eways.util;
public class Base64Utils
    extends com.stc.jcsre.Base64
```

Syntax

```
public static byte[] string2Base64(byte[] _bytes)
public static java.lang.String string2Base64(java.lang.String _str)
```

Description

Encodes a byte array or text string into MIME Base64 encoding and returns the encoded data in the same data type. (Compare **byteToBase64String()** on page 550.)

Parameters

Name	Type	Description
_bytes	byte[]	The array of bytes to be encoded.
_str	java.lang.String	The text string to be encoded.

Return Types

bytes[]—returned when the input argument is a byte array.

java.lang.String—returned when the input argument is a text string.

Throws

java.lang.Exception—thrown if the input text string cannot be encoded.

A.3 CollabUtils Class (com.stc.eways.util)

The SeeBeyond class com.stc.eways.util.CollabUtils provides the following public methods:

- [asHex\(\)](#) on page 552
 - [doOffsetTrunc\(\)](#) on page 553
 - [isMonkDatePattern\(\)](#) on page 554
 - [sprintf\(\)](#) on page 555
 - [toHex\(\)](#) on page 556
 - [toJavaDatePattern\(\)](#) on page 556
 - [uniqueId\(\)](#) on page 557
-

asHex()

Access to the five **asHex()** methods are provided via the **CollabUtils** class, supplied in the SeeBeyond package **com.stc.eways.util**:

```
package com.stc.eways.util;  
  
public class CollabUtils  
  
    public java.lang.String CollabUtils.asHex(byte _b)  
    public java.lang.String CollabUtils.asHex(short _s)  
    public java.lang.String CollabUtils.asHex(int _i)  
    public java.lang.String CollabUtils.asHex(long _l)  
    public java.lang.String CollabUtils.asHex(bytes _bytes,  
                                              int _perline)
```

Description

The first four methods converts the input (byte, short, int, or long) into a text string of (two, four, eight, or sixteen) hexadecimal digits. The fifth method converts a byte array into a list of hexadecimal numbers.

Parameter

Name	Type	Description
_b	byte	Single byte to be converted to a 2-byte hex string.
_s	short	Short integer to be converted to a 4-byte hex string.
_i	int	Full integer to be converted to an 8-byte hex string.
_l	long	Long integer to be converted to a 16-byte hex string.
_bytes	byte[]	Array of bytes to be converted to hexadecimal representation
_perLine	int	A count of the number of bytes to emit before writing out a newline.

Return Type

java.lang.String

Throws

None.

doOffsetTrunc()

Access to the **doOffsetTrunc()** method is provided via the **CollabUtils** class, supplied in the SeeBeyond package **com.stc.eways.util**:

```
package com.stc.eways.util;
public class CollabUtils
```

Syntax

```
public java.lang.String CollabUtils.doOffsetTrunc(int _offset,
                                                int _trunc, java.lang.String _str)
```

Description

Creates a substring from the input string by skipping its first *_offset* characters and then truncating the result to *_trunc* characters altogether.

Parameters

Name	Type	Description
_offset	int	Number of initial characters to skip. Can be any value, including zero. If _offset exceeds the length of _str, then the null string is output.
_trunc	int	Number of characters to output. If _trunc exceeds the length of the beheaded string, then only the beheaded string is returned — in other words, no padding is done.
_str	java.lang.String	Text string to be beheaded and/or curtailed.

Return Type

java.lang.String

Throws

None.

isMonkDatePattern()

Access to the **isMonkDatePattern()** method is provided via the **CollabUtils** class, supplied in the SeeBeyond package **com.stc.eways.util**:

```
package com.stc.eways.util;
public class CollabUtils
```

Syntax

```
public boolean CollabUtils.isMonkDatePattern(
    java.lang.String _pattern)
```

Description

Inquires whether the input string is a valid C (or, equivalently, Monk) date format.

Parameter

Name	Type	Description
_pattern	java.lang.String	The string to be evaluated.

Return Type

boolean

Returns **true** if the specified string is a valid date pattern; otherwise returns **false**.

Throws

None.

sprintf()

Access to the two **sprintf()** methods is provided via the **CollabUtils** class, supplied in the SeeBeyond package **com.stc.eways.util**:

```
package com.stc.eways.util;  
  
public class CollabUtils
```

Syntax

```
public static java.lang.String CollabUtils.printf(  
                      java.lang.String _pattern,  
                      java.lang.Object _arg)  
  
public static java.lang.String CollabUtils.printf(  
                      java.lang.String _pattern,  
                      java.lang.Object[] _args)
```

Description

Outputs a text string formatted according to the specified pattern, using the specified argument or arguments.

Parameters

Name	Type	Description
_pattern	java.lang.String	The format specification code; see “Formatting of Output Text” on page 662 .
_arg	java.lang.Object	The Java language object to be formatted into a text string.
_args	java.lang.Object[]	An array of Java language objects to be formatted into a text string.

Return Type

java.lang.String

Throws

None.

toHex()

Access to the two **toHex()** methods is provided via the **CollabUtils** class, supplied in the SeeBeyond package **com.stc.eways.util**:

```
package com.stc.eways.util;  
  
public class CollabUtils
```

Syntax

```
public java.lang.String CollabUtils.toHex(byte[] _bytes)  
public java.lang.String CollabUtils.toHex(java.lang.String _str)
```

Description

Transforms the input byte array or string into a formatted string showing both the hexadecimal and character representation.

Parameter

Name	Type	Description
_bytes	byte[]	Array of bytes to be re-represented in hex and character format.
_str	java.lang.String	Text string to be re-represented in hex and character format.

Return Type

java.lang.String

Throws

None.

toJavaDatePattern()

Access to the **toJavaDatePattern()** method is provided via the **CollabUtils** class, supplied in the SeeBeyond package **com.stc.eways.util**:

```
package com.stc.eways.util;  
  
public class CollabUtils
```

Syntax

```
static java.lang.String CollabUtils.toJavaDatePattern(  
                                java.lang.String _pattern)
```

Description

Converts a C-language date pattern into a Java date pattern—in other words, a date pattern suitable for use by the `java.text.SimpleDateFormat` class.

Parameter

Name	Type	Description
<code>_pattern</code>	<code>java.lang.String</code>	Date pattern, in Monk format.

Return Type

`java.lang.String`

Throws

`java.text.ParseException`—thrown if the conversion cannot be done.

uniqueId()

Note: For convenience, the `uniqueID()` method is also provided as an alternate spelling. `uniqueID()` and `uniqueId()` are identical.

Access to the `uniqueId()` method is provided via the **CollabUtils** class, supplied in the SeeBeyond package `com.stc.eways.util`:

```
package com.stc.eways.util;
public class CollabUtils
```

Syntax

```
public static long CollabUtils.uniqueId()
public static java.lang.String CollabUtils.uniqueId(int _offset,
                                                    int _trunc)
```

Description

Generates a unique identifier, based on system time (including milliseconds). The `java.lang.String` version uses the same truncation and shifting as the Monk **Unique ID** rule.

Parameters

Name	Type	Description
_offset	int	The number of characters to skip from the start of the string.
_trunc	int	The total number of characters to use from the string. -1 means keep going to the end , and should be used with delimited nodes whose length is unknown.

Return Types

long—in other words, an integer like 20021231235959999.

java.lang.String—in other words, the current system time, using the format yyyyMMddHHmmssuuu, using four digits for the year, two digits each for month, day, hour, minute, and second, and three digits for the milliseconds.

Throws

None.

A.4 DateUtils Class (com.stc.eways.util)

The SeeBeyond class com.stc.eways.util.DateUtils provides the following public methods:

- [format\(\)](#) on page 559
 - [timeStamp\(\)](#) on page 560
 - [transformDate\(\)](#) on page 560
-

format()

Access to the **format()** method is provided via the **DateUtils** class, supplied in the SeeBeyond package **com.stc.eways.util**:

```
package com.stc.eways.util;  
public class DateUtils
```

Syntax

```
public java.lang.String DateUtils.format(java.util.Date _date,  
                                         java.lang.String _pattern)
```

Description

Reformats the supplied Date object using the specified pattern.

Parameters

Name	Type	Description
_date	java.util.Date	The Date object to be reformatted.
_pattern	java.lang.String	A code specifying how to reformat the Date object; see Table 42, "Date and Time Format Codes for the timeStamp Rule" on page 319.

Return Type

java.lang.String

Throws

None.

timeStamp()

Access to the **timeStamp()** method is provided via the **DateUtils** class, supplied in the SeeBeyond package **com.stc.eways.util**:

```
package com.stc.eways.util;  
  
public class DateUtils
```

Syntax

```
public java.lang.String DateUtils.timestamp(java.lang.String pattern)
```

Description

Provides a snapshot of the current date and/or time according to the system clock of the Participating Host, and formatted for output using the specified pattern.

Parameters

Name	Type	Description
_pattern	java.lang.String	A code specifying how to format the date/timestamp; see Table 42, “ Date and Time Format Codes for the timeStamp Rule ” on page 319.

Return Type

java.lang.String

Throws

None.

transformDate()

Access to the **transformDate()** method is provided via the **DateUtils** class, supplied in the SeeBeyond package **com.stc.eways.util**:

```
package com.stc.eways.util;  
  
public class DateUtils
```

Syntax

```
public java.lang.String DateUtils.transformDate(  
    java.lang.String _date,  
    java.lang.String _from,  
    java.lang.String _to)
```

Description

Reformats the supplied date/timestamp from one representation to another.

Parameters

Name	Type	Description
_date	java.lang.String	The date/timestamp to be reformatted.
_from	java.lang.String	The current date/timestamp code.
_to	java.lang.String	The date/timestamp code of the desired output.

Return Type

java.lang.String

Throws

None.

Example

transformDate(“10/31/02”, “%D”, “%d-%b-%Y”) informs the system that “10/31/02” is currently in mm/dd/yy format (American style) and is to be transformed into a format consisting of a two-digit day, a hyphen, an abbreviated month, another hyphen, and a four-digit year—in other words, “31-Oct-2002”.

For a complete list of date/timestamp format codes, see Table 42, [“Date and Time Format Codes for the timeStamp Rule” on page 319](#).

A.5 EGate Class (com.stc.common.collabService)

The SeeBeyond class com.stc.common.collabService.EGate provides the following public methods:

- [collabDebug\(\)](#) on page 562
 - [collabError\(\)](#) on page 563
 - [collabFatal\(\)](#) on page 563
 - [collabInfo\(\)](#) on page 564
 - [collabTrace\(\)](#) on page 565
 - [collabWarning\(\)](#) on page 565
 - [traceln\(\)](#) on page 566
-

collabDebug()

Access to the **collabDebug()** method is provided via the **EGate** class, supplied in the SeeBeyond package **com.stc.common.collabService**:

```
package com.stc.common.collabService;  
  
public class EGate
```

Syntax

```
public static void collabDebug(java.lang.String message)
```

Description

Adds a trace entry into the log file; an end-of-line is automatically appended after the entry. The trace entry is shown only if the Logging Level is set to DEBUG and the Debugging flag is set to TRACE_COLLABSERVICE (COL) in the component's LOG properties. See [Table 83 on page 566](#) and [Table 84 on page 567](#).

Parameters

Name	Type	Description
message	java.lang.String	Information to write to the log file.

Return Type

None.

Throws

None.

collabError()

Access to the **collabError()** method is provided via the **EGate** class, supplied in the SeeBeyond package **com.stc.common.collabService**:

```
package com.stc.common.collabService;  
  
public class EGate
```

Syntax

```
public static void collabError(java.lang.String message)
```

Description

Adds a trace entry into the log file; an end-of-line is automatically appended after the entry. The trace entry is shown only if the Logging Level is set to ERROR and the Debugging flag is set to TRACE_COLLABSERVICE (COL) in the component's LOG properties. See [Table 83 on page 566](#) and [Table 84 on page 567](#).

Parameters

Name	Type	Description
message	java.lang.String	Information to write to the log file.

Return Type

None.

Throws

None.

collabFatal()

Access to the **collabFatal()** method is provided via the **EGate** class, supplied in the SeeBeyond package **com.stc.common.collabService**:

```
package com.stc.common.collabService;  
  
public class EGate
```

Syntax

```
public static void collabFatal(java.lang.String message)
```

Description

Adds a trace entry into the log file; an end-of-line is automatically appended after the entry. The trace entry is shown only if the Logging Level is set to FATAL and the Debugging flag is set to TRACE_COLLABSERVICE (COL) in the component's LOG properties. See [Table 83 on page 566](#) and [Table 84 on page 567](#).

Parameters

Name	Type	Description
message	java.lang.String	Information to write to the log file.

Return Type

None.

Throws

None.

collabInfo()

Access to the **collabInfo()** method is provided via the **EGate** class, supplied in the SeeBeyond package **com.stc.common.collabService**:

```
package com.stc.common.collabService;
public class EGate
```

Syntax

```
public static void collabInfo(java.lang.String message)
```

Description

Adds a trace entry into the log file; an end-of-line is automatically appended after the entry. The trace entry is shown only if the Logging Level is set to INFO and the Debugging flag is set to TRACE_COLLABSERVICE (COL) in the component's LOG properties. See [Table 83 on page 566](#) and [Table 84 on page 567](#).

Parameters

Name	Type	Description
message	java.lang.String	Information to write to the log file.

Return Type

None.

Throws

None.

collabTrace()

Access to the **collabTrace()** method is provided via the **EGate** class, supplied in the SeeBeyond package **com.stc.common.collabService**:

```
package com.stc.common.collabService;  
  
public class EGate
```

Syntax

```
public static void collabTrace(java.lang.String message)
```

Description

Adds a trace entry into the log file; an end-of-line is automatically appended after the entry. The trace entry is shown only if the Logging Level is set to TRACE and the Debugging flag is set to TRACE_COLLABSERVICE (COL) in the component's LOG properties. See [Table 83 on page 566](#) and [Table 84 on page 567](#).

Parameters

Name	Type	Description
message	java.lang.String	Information to write to the log file.

Return Type

None.

Throws

None.

collabWarning()

Access to the **collabWarning()** method is provided via the **EGate** class, supplied in the SeeBeyond package **com.stc.common.collabService**:

```
package com.stc.common.collabService;  
  
public class EGate
```

Syntax

```
public static void collabWarning(java.lang.String message)
```

Description

Adds a trace entry into the log file; an end-of-line is automatically appended after the entry. The trace entry is shown only if the Logging Level is set to WARNING and the Debugging flag is set to TRACE_COLLABSERVICE (COL) in the component's LOG properties. See [Table 83 on page 566](#) and [Table 84 on page 567](#).

Parameters

Name	Type	Description
message	java.lang.String	Information to write to the log file.

Return Type

None.

Throws

None.

traceln()

Access to the two **traceln()** methods is provided via the **EGate** class, supplied in the SeeBeyond package **com.stc.common.collabService**:

```
package com.stc.common.collabService;
public class EGate
    public static void traceln(long tid, long event,
                               java.lang.String message)
    public static void traceln(long tid, long event,
                               byte[] blob, java.lang.String tracestr)
```

Description

Adds a trace entry into the log file; an end-of-line is automatically appended after the entry. The trace entry is shown only if the Trace ID (debugging flag) is active and the Trace Event (Logging Level) is in effect at run time. See Table 83.

Table 83 Trace Events (Logging Levels)

Logging Level	TRACE_EVENT_...					
	...TRACE	...DEBUG	...INFORMATION	...WARNING	...LOGERROR	...FATAL
TRACE	shown	shown	shown	shown	shown	shown
DEBUG	not shown	shown	shown	shown	shown	shown
INFO	not shown	not shown	shown	shown	shown	shown
WARNING	not shown	not shown	shown	shown	shown	shown
ERROR	not shown	not shown	shown	shown	shown	shown
FATAL	not shown	not shown	shown	shown	shown	shown
NONE	not shown	not shown	shown	shown	shown	shown

Parameters

Name	Type	Description
tid	long	Debugging flag (trace ID), such as TRACE_EWAY . For a complete list of trace IDs, see Table 84 below.
event	long	Trace Event, such as TRACE_EVENT_INFORMATION . For a complete list of Trace Events, see Table 85 on page 569 .
message	java.lang.String	Information to write to the log file.
blob	byte[]	Status or error code generated by the operating system or the application generating the Event.
tracestr	java.lang.String	Text string to write to the log file.

Table 84 Trace ID Parameters (for *tid* Parameter)

Note: All *tid* parameters are declared in the following way:

```
public static final long <traceparmname>
```

For example:

```
public static final long TRACE_STCAPIS_VERBOSE
```

Trace ID Parameter	Logged as	Explanation
TRACE VERY VERBOSE		All Verbose type debugging flags.
TRACE COUNTS	(CNTS)	Debugging flag / Trace ID: Display from message counts.
TRACE_GDCOMMON_VERBOSE	(COMV)	Debugging flag / Trace ID: Verbose display from SeeBeyond Common utilities.
TRACE_STCAPIS_VERBOSE	(APIV)	Debugging flag / Trace ID: Verbose display from e*Gate Core APIs.
TRACE_REGISTRY_VERBOSE	(REGV)	Debugging flag / Trace ID: Verbose display from Registry activities.
TRACE_IQ_VERBOSE	(IQV)	Debugging flag / Trace ID: Verbose display from Internal Queue activities.
TRACE_DB_VERBOSE	(DBV)	Debugging flag / Trace ID: Verbose display from Database activities.
TRACE_IP_VERBOSE	(IPV)	Debugging flag / Trace ID: Verbose display from TCPIP activities.
TRACE_MONK_VERBOSE	(MNKV)	Debugging flag / Trace ID: Verbose display from Monk utilities.
TRACE_CB_VERBOSE	(CBV)	Debugging flag / Trace ID: Verbose display from Control Broker activities.
TRACE_EWAY_VERBOSE	(EWYV)	Debugging flag / Trace ID: Verbose display from e*Way activities.

Table 84 Trace ID Parameters (for *tid* Parameter) (Continued)

Note: All *tid* parameters are declared in the following way:
`public static final long <traceparmname>`

For example:

`public static final long TRACE_STCAPIS_VERBOSE`

Trace ID Parameter	Logged as	Explanation
TRACE_MESSAGE_VERBOSE	(MSGV)	Debugging flag / Trace ID: Verbose display from Message activities.
TRACE_COLABSERVICE_VERBOSE	(COLV)	Debugging flag / Trace ID: Verbose display from Collaboration Service activities.
TRACE_APPS_ALL		All applications type Debugging flag / Trace ID.
TRACE_CB	(CB)	Debugging flag / Trace ID: Display from Control Broker activities.
TRACE_EWAY	(EWY)	Debugging flag / Trace ID: Display from e*Way activities.
TRACE_MSGPARSE	(MSGP)	Debugging flag / Trace ID: Display from Message Parsing activities.
TRACE_CONFIGURATION	(CFG)	Debugging flag / Trace ID: Display from Configuration utilities.
TRACE_STATECHANGE	(ST)	Debugging flag / Trace ID: Display from State Transition activities.
TRACE_COLABSERVICE	(COL)	Debugging flag / Trace ID: Display from Collaboration Service activities.
TRACE_LIBS_ALL		All libraries type Debugging flag / Trace ID.
TRACE_GDCOMMON	(COM)	Debugging flag / Trace ID: Display from SeeBeyond Common utilities.
TRACE_DB	(DB)	Debugging flag / Trace ID: Display from Database activities.
TRACE_REGISTRY	(REG)	Debugging flag / Trace ID: Display from Registry activities.
TRACE_IQ	(IQ)	Debugging flag / Trace ID: Display from Internal Queue activities.
TRACE_STCAPIS	(API)	Debugging flag / Trace ID: Display from e*Gate Core APIs.
TRACE_MESSAGE	(MSG)	Debugging flag / Trace ID: Display from Message activities.
TRACE_IP	(IP)	Debugging flag / Trace ID: Display from TCPIP activities.
TRACE_MONK	(MNK)	Debugging flag / Trace ID: Display from Monk utilities.

Table 85 Trace Event Parameters (for *event* Parameter)

Note: All *event* parameters are declared in the following way:

```
public static final long <traceeventparname>
```

For example:

```
public static final long TRACE_EVENT_DEBUG
```

Trace Event Parameter	Logged as	Explanation
TRACE_EVENT_ALL		Logging level / Trace Event: Select All levels.
TRACE_EVENT_DEBUG	(D)	Logging level / Trace Event: Active when DEBUG is selected.
TRACE_EVENT_TRACE	(T)	Logging level / Trace Event: Active when TRACE is selected.
TRACE_EVENT_INFORMATION	(I)	Logging level / Trace Event: INFO is always active.
TRACE_EVENT_WARNING	(W)	Logging level / Trace Event: WARNING is always active.
TRACE_EVENT_APIERROR	(A)	Logging level / Trace Event: Active when ERROR is selected.
TRACE_EVENT_LOGERROR	(E)	Logging level / Trace Event: Active when ERROR is selected. Entry is also placed in System log.
TRACE_EVENT_FATAL	(F)	Logging level / Trace Event: FATAL is always active.
TRACE_EVENT_ERROR	(E)	Logging level / Trace Event: Active when ERROR is selected.
TRACE_APP	(EWY)	Debugging flag / Trace ID: Display from e*Way activities.
TRACE_APP_VERBOSE	(EWYV)	Debugging flag / Trace ID: Verbose display from e*Way activities.
TRACE_EVENT_PUTINLOGMASK		Logging levels / Trace Events which are always placed into the module log file.
TRACE_EVENT_PUTINSYSLOG		Logging levels / Trace Events which makes entries into the System log file.

Return Type

None.

Throws

None.

A.6 FileUtils Class (com.stc.eways.util)

The SeeBeyond class com.stc.eways.util.FileUtils provides the following public methods:

- [readBytes\(\)](#) on page 570
 - [readString\(\)](#) on page 570
 - [write\(\)](#) on page 571
-

readBytes()

The **readBytes()** method is provided in the **FileUtils** class, supplied in SeeBeyond package **com.stc.eways.util** (in **stcutil.jar**):

Syntax

```
static byte[] readBytes(java.lang.String _fileName)
```

Description

Reads the contents of the specified file into an array of bytes.

Parameter

Name	Type	Description
_fileName	java.lang.String	Name of the file to be read.

Return Type

byte[]—in other words, an array of bytes.

Throws

java.io.IOException—If bytes cannot be read from the specified file.

readString()

The two **readString()** methods are provided in the **FileUtils** class, supplied in SeeBeyond package **com.stc.eways.util** (in **stcutil.jar**):

Syntax

```
static java.lang.String readstring(java.lang.String _fileName)  
static java.lang.String readstring(java.lang.String _fileName,  
                               java.lang.String _encoding)
```

Description

Reads the contents of the specified file into a Java string (Unicode). The first method (with only a single parameter) assumes the default character encoding; the second method (with two parameters) allows you to specify the encoding.

Parameters

Name	Type	Description
_fileName	java.lang.String	Name of the file to be read.
_encoding	java.lang.String	The character encoding used in the file being read.
		Basic encoding sets are contained in rt.jar ; extended encoding sets are contained in i18n.jar . For a complete list, see http://java.sun.com/j2se/1.3/docs/guide/intl/encoding.doc.html

Return Type

java.lang.String

Throws

java.io.IOException—If the specified file cannot be read.

write()

The three **write()** methods are provided in the **FileUtils** class, supplied in SeeBeyond package **com.stc.eways.util** (in **stcutil.jar**):

Syntax

```
static void write(java.lang.String _fileName, byte[] _bytes)

static void write(java.lang.String _fileName,
                 java.lang.String _string)

static void write(java.lang.String _fileName,
                 java.lang.String _string,
                 java.lang.String _encoding)
```

Description

Writes to the specified file. The first method writes a specified byte array to a file; the second method writes a specified string to a file and assumes the default character encoding; the third method (with three parameters) allows you to specify the encoding.

Parameters

Name	Type	Description
_fileName	java.lang.String	Name of the file to be written.
_bytes	byte[]	Array of bytes to be written.
_string	java.lang.String	String to be written
_encoding	java.lang.String	The character encoding used in the file being written.
		Basic encoding sets are contained in rt.jar ; extended encoding sets are contained in i18n.jar . For a complete list, see http://java.sun.com/j2se/1.3/docs/guide/intl/encoding.doc.html

Return Type

None.

Throws

java.io.IOException—If the system cannot write the bytes or string to the specified file.

A.7 JCollabController Class (com.stc.common.collabService)

The SeeBeyond class com.stc.common.collabService.**JCollabController** provides a wide variety of methods, in three categories.

General system methods:

- ♦ [copyProperties\(\)](#) on page 575
- ♦ [createSubCollabMapInfo\(\)](#) on page 575
- ♦ [getCollaborationName\(\)](#) on page 576
- ♦ [getEgateBaseDirectory\(\)](#) on page 576
- ♦ [getModuleName\(\)](#) on page 577
- ♦ [getPropertyNames\(\)](#) on page 577
- ♦ [invoke\(\)](#) on page 578
- ♦ [isSubCollaboration\(\)](#) on page 579
- ♦ [retrieveRegistryFile\(\)](#) on page 580

Character encoding/Internationalization methods:

- ♦ [getIncomingEncoding\(\)](#) on page 584
- ♦ [getMarshalEncoding\(\)](#) on page 584
- ♦ [getOutgoingEncoding\(\)](#) on page 585
- ♦ [getUnmarshalEncoding\(\)](#) on page 585
- ♦ [setIncomingEncoding\(\)](#) on page 586
- ♦ [setMarshalEncoding\(\)](#) on page 587
- ♦ [setOutgoingEncoding\(\)](#) on page 587
- ♦ [setUnmarshalEncoding\(\)](#) on page 588
- ♦ [getIncomingEncoding\(\)](#) on page 584

ELS methods (accessed via the ELSController interface; see “[ELSController Interface Methods](#)” on page 589)

- ELS methods for querying Link Identifiers:
 - ♦ [getCurrentLinkIdentifier\(\)](#) on page 590
 - ♦ [getLinkIdentifiers\(\)](#) on page 592
 - ♦ [isLinkIdentifierExists\(\)](#) on page 596
 - ♦ [isLinkingComplete\(\)](#) on page 597
 - ♦ [retrieveLinkIdentifier\(\)](#) on page 598

- **ELS methods for querying message count:**
 - ♦ [hasHappened\(\)](#) on page 594
 - ♦ [getNumberOfMessages\(\)](#) on page 593
 - ♦ [getNoOfMessagesForInstance\(\)](#) on page 593
- **ELS methods for setting and getting timers:**
 - ♦ [setELSExpiration\(\)](#) on page 598
 - ♦ [getELSExpiration\(\)](#) on page 591
 - ♦ [isCurrentELSExpired\(\)](#) on page 595
 - ♦ [isELSExpired\(\)](#) on page 595
 - ♦ [onExpire\(\)](#) on page 597

A.7.1 General System Control Methods

The SeeBeyond class com.stc.common.collabService.**JCollabController** provides the following public methods for querying the system, querying and controlling the Subcollaboration Rule environment, and retrieving files from the registry:

- [copyProperties\(\)](#) on page 575
- [createSubCollabMapInfo\(\)](#) on page 575
- [getCollaborationName\(\)](#) on page 576
- [getEgateBaseDirectory\(\)](#) on page 576
- [getModuleName\(\)](#) on page 577
- [getPropertyNames\(\)](#) on page 577
- [invoke\(\)](#) on page 578
- [isSubCollaboration\(\)](#) on page 579
- [retrieveRegistryFile\(\)](#) on page 580

The general system control methods are available to a *particular instance* of the **JCollabController** class, and are thus accessed inside a Collaboration with the following types of method invocation:

`this.jCollabController.get<...>()`

or

`this.jCollabController.methodName(parm1[, ..., parm2])`

copyProperties()

Access to the **copyProperties()** method is provided via the SeeBeyond package **com.stc.common.collabService**, class **JCollabController**:

```
public class JCollabController;
```

Syntax

```
public void this.jCollabController.copyProperties(  
    com.stc.jcsre.ETD aFromETD,  
    com.stc.jcsre.ETD aToETD)
```

Description

For marshallable ETDs only: Copies the ETD properties from the specified **FromETD** to the specified **ToETD**.

Parameters

Name	Type	Description
aFromETD	com.stc.jcsre.ETD	The ETD object whose properties are to be copied. Must be a marshallable ETD.
aToETD	com.stc.jcsre.ETD	The ETD object to which the properties are to be copied. Must be a marshallable ETD.

Return Type

None.

Comments

Also see “[getPropertyNames\(\)](#) on page 577”, “[readProperty\(\)](#) on page 339”, and “[writeProperty\(\)](#) on page 347”.

createSubCollabMapInfo()

Access to the **createSubCollabMapInfo()** method is provided via the SeeBeyond package **com.stc.common.collabService**, class **JCollabController**:

```
public class JCollabController;
```

Syntax

```
JSubCollabMapInfo this.jCollabController.createSubCollabMapInfo(  
    java.lang.String aCollabRuleName)
```

Description

Creates mapping information (a **JSubCollabMapInfo** object) for the named Subcollaboration Rule. After it is created, the **JSubCollabMapInfo** object can be queried

and manipulated by methods dealing with Subcollaboration Rules—see “[JSubCollabMapInfo Class \(com.stc.common.collabService\) on page 603](#)”—and eventually passed as a parameter when calling the `invoke()` method.

For information on working with Subcollaboration Rules, see “[Subcollaboration Rules](#)” on page 348.

Parameter

Name	Type	Description
aCollabRuleName	java.lang.String	The name of the Subcollaboration Rule to be mapped.

Return Type

JSubCollabMapInfo object—mapping information for the Subcollaboration Rule.

Throws

None.

getCollaborationName()

Access to the `getCollaborationName()` method is provided via the SeeBeyond package **com.stc.common.collabService**, class **JCollabController**:

Syntax

```
public java.lang.String this.jCollabController.getCollaborationName()
```

Description

Retrieves the name of the current Collaboration, as defined in the Enterprise Manager.

Parameters

None.

Return Type

java.lang.String

Throws

None.

getEgateBaseDirectory()

Access to the `getEgateBaseDirectory()` method is provided via the SeeBeyond package **com.stc.common.collabService**, class **JCollabController**:

```
public class JCollabController;
```

Syntax

```
public java.lang.String  
this.jCollabController.getEgateBaseDirectory()
```

Description

Retrieves the absolute path and folder name of the e*Gate base directory as defined in the **.egate.store** file.

Parameters

None.

Return Type

java.lang.String—for example:

```
C:\eGate\client
```

getModuleName()

Access to the **getModuleName()** method is provided via the SeeBeyond package **com.stc.common.collabService**, class **JCollabController**:

```
public class JCollabController;
```

Syntax

```
public java.lang.String this.jCollabController.getModuleName()
```

Description

Retrieves the name of the module in which the current Collaboration is running, as defined in the Enterprise Manager.

Parameters

None.

Return Type

java.lang.String

getPropertyNames()

Access to the **getPropertyNames()** method is provided via the SeeBeyond package **com.stc.common.collabService**, class **JCollabController**:

```
public class JCollabController;
```

Syntax

```
public java.lang.String[] this.jCollabController.getPropertyNames(  
    com.stc.jcsre.ETD anETD)
```

Typical Usage

```
String[] propertyNames = this.jCollabController.getPropertyNames(getXXXX())
```

(where XXXX represents the name of the ETD instance, dragged in from the GUI)

Description

For marshallable ETDs only: Lists the names of all properties defined for the specified ETD instance.

Parameters

Name	Type	Description
anETD	com.stc.jcsre.ETD	The ETD instance whose property names are to be listed. Must be a marshallable ETD.

Return Type

java.lang.String[]—in other words, an array of strings. If the specified ETD is not marshallable, or if it has no properties, the return value is **null**.

Comments

Also see “[copyProperties\(\) on page 575](#), “[readProperty\(\) on page 339](#), and “[writeProperty\(\) on page 347](#).

invoke()

Access to the **invoke()** method is provided via the SeeBeyond package **com.stc.common.collabService**, class **JCollabController**:

```
public class JCollabController;
```

Syntax

```
this.jCollabController.invoke(  
    java.lang.String aCollaborationRuleName,  
    com.stc.jcsre.JCollaboration aCallingCollaboration)
```

Description

Creates a **JSubCollabMapInfo** object. For other methods dealing with Subcollaboration Rules, see “[JSubCollabMapInfo Class \(com.stc.common.collabService\) on page 603](#). For information on working with Subcollaboration Rules, see “[Subcollaboration Rules on page 348](#).

Parameters

Name	Type	Description
aCollaborationRuleName	java.lang.String	Name of the Subcollaboration Rule.
aCallingCollaboration	com.stc.jcsre.JCollaboration	this (see below)

When used as the second parameter in the invocation of the `this.jCollabController.invoke()` method, **this** denotes a value that is a reference to the JSubCollabMap Info object.
For more information on the **this** keyword in general, see http://java.sun.com/docs/books/jls/second_edition/html/expressions.doc.html section 15.8.3.

Return Type

`com.stc.common.collabService.JSubCollabMapInfo` object.

Throws

None.

isSubCollaboration()

Access to the `isSubCollaboration()` method is provided via the SeeBeyond package `com.stc.common.collabService`, class `JCollabController`:

```
public class JCollabController;
```

Syntax

```
public boolean this.jCollabController.isSubCollaboration()
```

Description

Determines whether the current Collaboration Rule is being used as a Subcollaboration Rule. For other Subcollaboration Rule methods, see “[JSubCollabMapInfo Class \(com.stc.common.collabService\)](#)” on page 603. For information on working with Subcollaboration Rules, see “[Subcollaboration Rules](#)” on page 348.

Parameters

None.

Return Type

boolean

Returns **true** if and only if the current environment is a Subcollaboration Rule—in other words, a Collaboration Rule that is being called from within a Collaboration Rule.

Throws

None.

retrieveRegistryFile()

Access to the `retrieveRegistryFile()` method is provided via the SeeBeyond package `com.stc.common.collabService`, class **JCollabController**:

```
public class JCollabController;
```

Syntax

```
public java.lang.String this.jCollabController.retrieveRegistryFile(  
                      java.lang.String aFileName,  
                      java.lang.String aRegistryFilePath)
```

Description

Retrieves the specified file from the e*Gate registry repository.

Parameters

Name	Type	Description
aFileName	java.lang.String	Name of the file to be retrieved.
aRegistryFilePath	java.lang.String	Location of the folder containing the file.

Return Type

`java.lang.String`

A.7.2 Character Encoding and Internationalization Methods

Character encoding methods are specific to each *instance* of the **JCollabController** class, and are thus accessed inside a Collaboration with the following method invocation:

```
this.jCollabController.<methodname>(java.lang.String aInstanceName)
```

About the Encoding Methods

For character encoding, e*Gate provides eight methods and four levels of defaulting. The system can impose or retrieve a character encoding at any of the following phases:

- Before the inbound data is unmarshalled (parsed), the Collaboration can learn its native character encoding via `getIncoming Encoding()`, or it can impose a new character encoding via `setIncomingEncoding()`.
- As the inbound data is being unmarshalled, the Collaboration can retrieve character encoding via `getUnmarshalEncoding()`, or specify it via `setUnmarshalEncoding()`.
- As the outbound data is being marshalled, Collaboration can retrieve character encoding via `getMarshalEncoding()`, or specify it via `setMarshalEncoding()`.

- After the outbound data is marshalled, the Collaboration can retrieve its character encoding via `getOutgoingEncoding()`, or impose a new outbound character encoding via `setOutgoingEncoding()`.

For the two `[g | s]etUnmarshalEncoding()` and the two `[g | s]etUnmarshalEncoding()` methods, several levels of defaults apply:

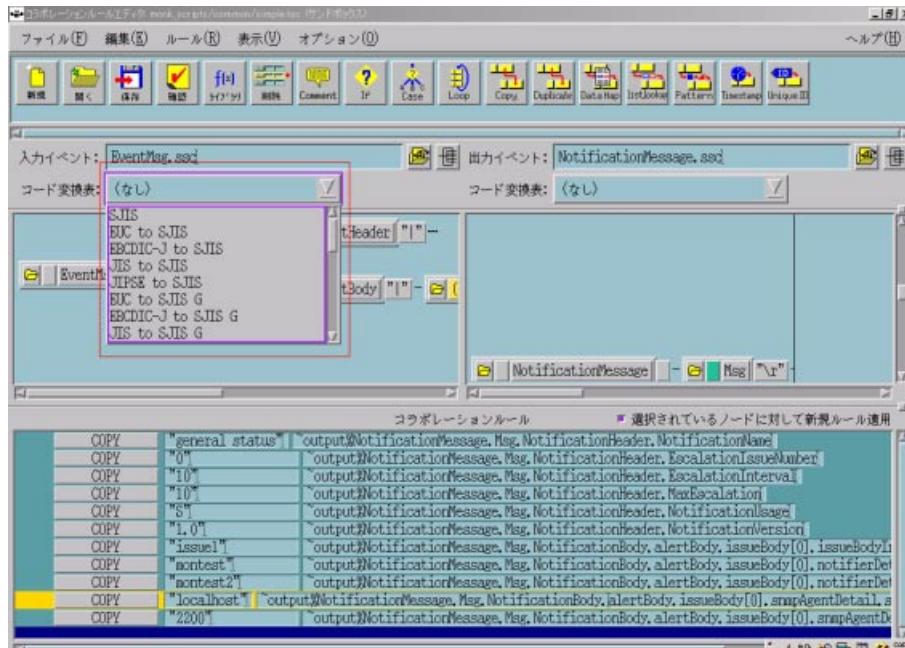
- At each node, the **encoding** property of that node can be used to specify a character encoding for the data of just that node. If the `nodeName.encoding` property is not specified, then:
- For each ETD, the **dataEncoding** property of the ETD can be used to specify a character encoding for all ETD nodes lacking an **encoding** property. If the `etdName.dataEncoding` property is not specified, then:
- The **sscEncoding** property of the ETD specifies the character encoding for all ETD nodes not specified at a finer level. If neither the `etdName.dataEncoding` nor the `etdName.encoding` property is specified, the character encoding defaults to:
- ASCII.**

Character Encodings in the Java Collaboration Rules Editor

Localization in a multi-byte environment

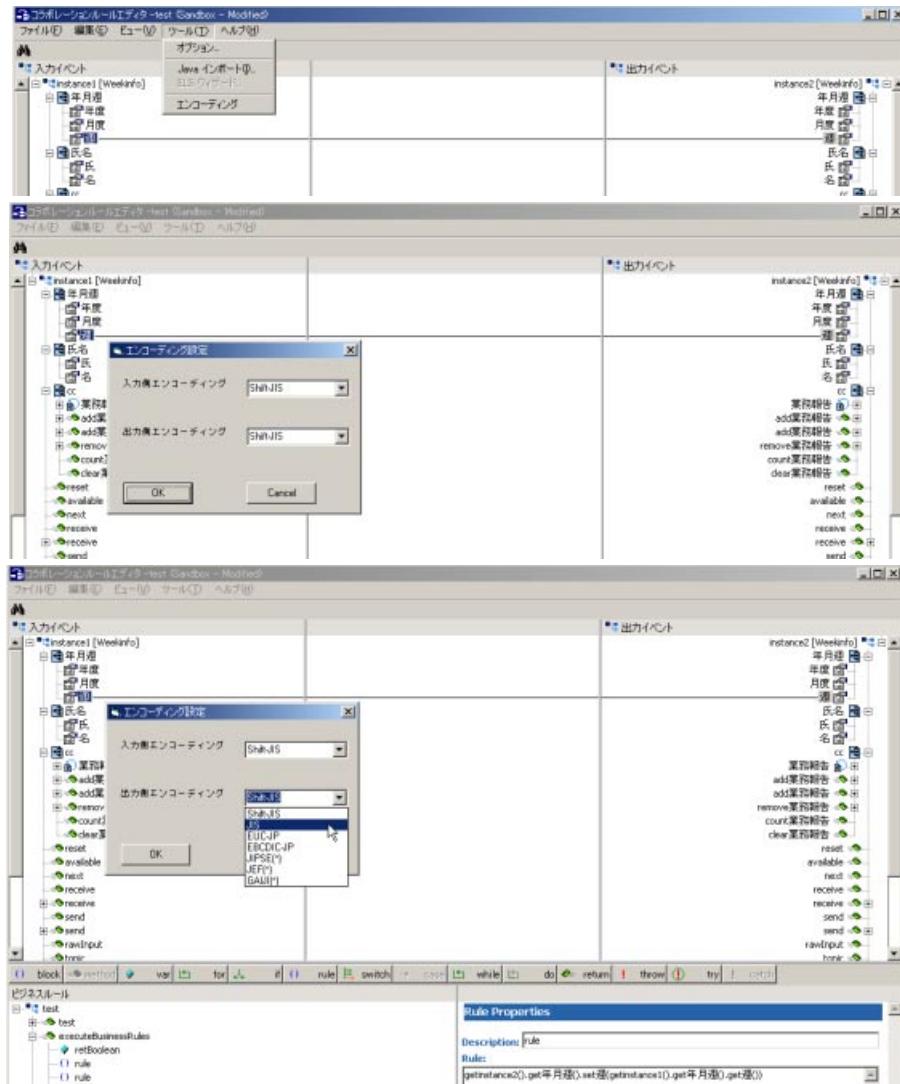
Special considerations apply when Collaborations run in Japanese or Korean operating systems, whose native character encoding is not single-byte. For e*Gate to parse Events and apply string functions correctly, the encoding method for inbound and outbound Event Types must match the native code. To support encoding methods other than the native code, use a code converter to convert the inbound or outbound Event Type instance to the appropriate native code. For Monk Collaborations, see Figure 246.

Figure 246 Monk Collaboration Rules Editor in a Japanese Environment



Similarly, for Java Collaborations that transport and transform data between two multi-byte nodes in a local environment, you can use simple drag-and-drop or **Find and Map** operations. See Figure 247 for three simple examples.

Figure 247 Java Collaboration Rules Editor in a Japanese Environment



Supported languages

Starting with release 4.5.2, e*Gate adds localized support for the following languages:

- **Traditional Chinese.** The native code for Java for Traditional Chinese is BIG5. This release does not support Monk Editors or Collaborations in Chinese, and supports BIG5 only. At this time, the e*Gate GUI is not translated into Chinese.
- **Japanese.** For both Java and Monk, the internal code for Japanese is SJIS. e*Gate supports EUC-JP, EBCDIC-JP, JIS, JIPS, JEF, and Gaiji.
- **Korean.** For Java, the native code for Korean is KSC-5601. e*Gate supports EUC-KR, EBCDIC-KR, and Johab. For Monk, the native code for Korean is UHC. e*Gate supports EUC-KR, KSC-5601, and EBCDIC-KR.

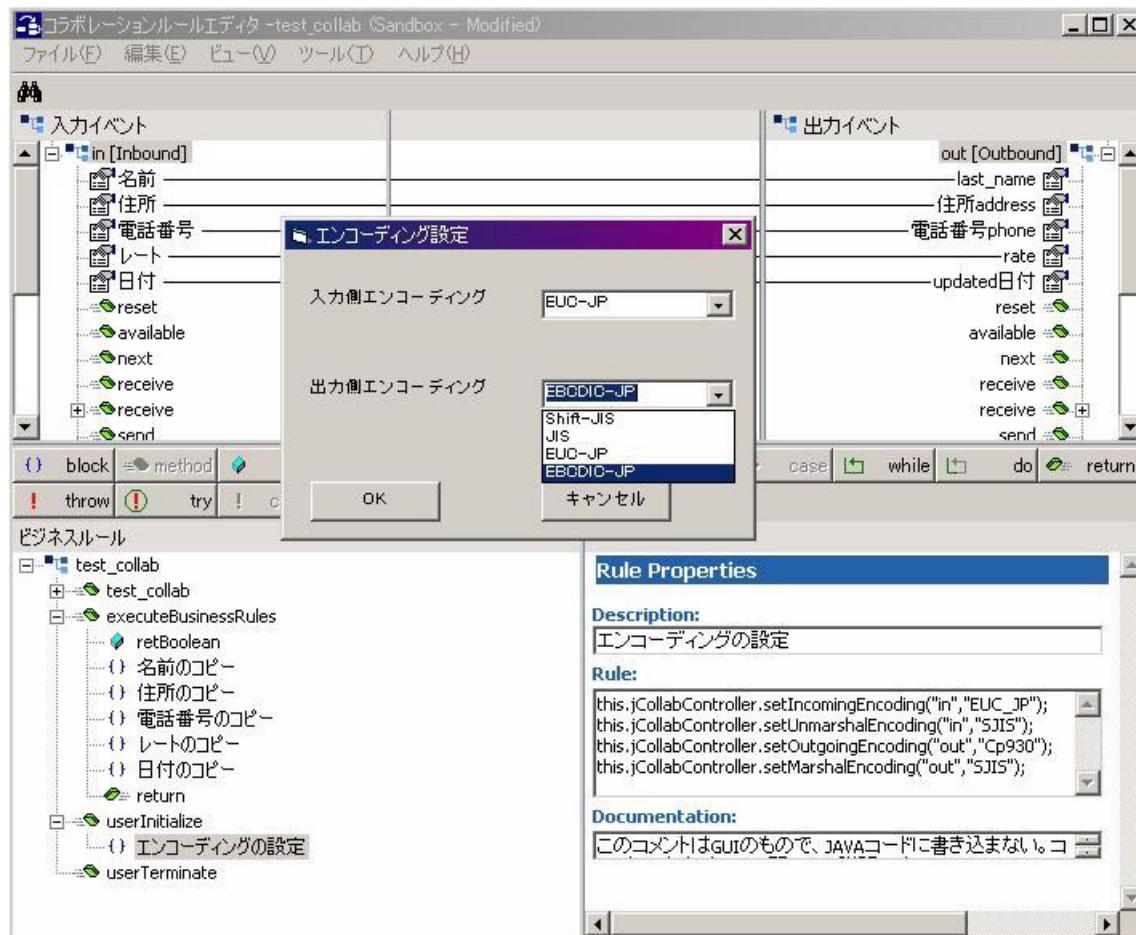
Translating data between single-byte and multi-byte environments

For Java-enabled Collaborations that translate and transform data between single-byte and multi-byte environments, you must create a rule under the `userInitialize()` placeholder method. Use this rule to set:

- The appropriate unmarshal/incoming encoding for inbound Event Type instances. See “[setUnmarshalEncoding\(\)](#) on page 588” and “[setIncomingEncoding\(\)](#) on page 586.”
- The appropriate marshal/outgoing encoding for outbound Event Type instances. See “[setMarshalEncoding\(\)](#) on page 587” and “[setOutgoingEncoding\(\)](#) on page 587.”

Figure 248 shows a Collaboration whose encoding for the inbound Event Type instance (named “in”) is EUC-JP for IncomingEncoding and SJIS for UnmarshalEncoding, and whose encoding for the outbound Event Type instance (named “out”) is set to Cp930 (sometimes called EBCDIK) for OutgoingEncoding and to SJIS for MarshalEncoding.

Figure 248 Setting the Character Encoding in `userInitialize()`



The eight [get/]set...Encoding() methods are described in more detail in the following pages.

getIncomingEncoding()

Syntax

```
public java.lang.String this.jCollabController.getIncomingEncoding(  
                      java.lang.String aInstanceName)
```

Description

Retrieves the inbound character encoding of the specified Event Type instance.

Parameters

Name	Type	Description
aInstanceName	java.lang.String	Name of the Event Type instance in the current Collaboration.

Return Type

java.lang.String

Throws

None.

getMarshalEncoding()

Syntax

```
java.lang.String this.jCollabController.getMarshalEncoding(  
                      java.lang.String aInstanceName)
```

Description

Retrieves the encoding method of the specified Event Type instance as it is being marshalled (serialized into a one-dimensional BLOB).

Parameters

Name	Type	Description
aInstanceName	java.lang.String	Name of the Event Type instance in the current Collaboration.

Return Type

java.lang.String

Throws

None.

Comments

For a complete explanation of how this encoding method fits with the other seven, see [“About the Encoding Methods” on page 580](#).

getOutgoingEncoding()

Syntax

```
java.lang.String this.jCollabController.getOutgoingEncoding(  
                      java.lang.String aInstanceName)
```

Description

Retrieves the outbound character encoding of the specified Event Type Instance.

Parameters

Name	Type	Description
aInstanceName	java.lang.String	Name of the Event Type instance in the current Collaboration.

Return Type

java.lang.String

Throws

None.

Comments

For a complete explanation of how this encoding method fits with the other seven, see [“About the Encoding Methods” on page 580](#).

getUnmarshalEncoding()

Syntax

```
java.lang.String this.jCollabController.getUnmarshalEncoding(  
                      java.lang.String aInstanceName)
```

Description

Retrieves the character encoding of the specified Event Type instance as it is being unmarshalled (parsed).

Parameters

Name	Type	Description
aInstanceName	java.lang.String	Name of the Event Type instance in the current Collaboration.

Return Type

java.lang.String

Throws

None.

Comments

For a complete explanation of how this encoding method fits with the other seven, see [“About the Encoding Methods” on page 580](#).

setIncomingEncoding()**Syntax**

```
void this.jCollabController.setIncomingEncoding(  
    java.lang.String aInstanceName,  
    java.lang.String aEncodeType)
```

Description

Specifies a certain character encoding for the specified Event Type instance before its data is unmarshalled (parsed).

Parameters

Name	Type	Description
aInstanceName	java.lang.String	Name of the Event Type instance in the current Collaboration.
aEncodeType	java.lang.String	The character encoding to be set. ▪ For Japanese, use one of the following: ♦ SJIS ♦ EUC-JP ♦ ISO-2022-JP ♦ CP930 ▪ For Korean, use: ♦ ksc_5601

Return Type

None.

Throws

None.

Comments

For a complete explanation of how this encoding method fits with the other seven, see [“About the Encoding Methods” on page 580](#).

setMarshalEncoding()

Syntax

```
void this.jCollabController.setMarshalEncoding(  
    java.lang.String aInstanceName,  
    java.lang.String aEncodeType)
```

Description

Specifies a particular character encoding for the specified Event Type instance as its data is being marshalled (serialized into flat BLOB form).

Parameters

Name	Type	Description
aInstanceName	java.lang.String	Name of the Event Type instance in the current Collaboration.
aEncodeType	java.lang.String	The character encoding to be set: ▪ For Japanese, use SJIS ▪ For Korean, use ksc_5601 This specifies the native code for the Java Collaboration.

Return Type

None.

Throws

None.

Comments

For a complete explanation of how this encoding method fits with the other seven, see [“About the Encoding Methods” on page 580](#).

setOutgoingEncoding()

Syntax

```
void this.jCollabController.setOutgoingEncoding(java.lang.String  
    aInstanceName,  
    java.lang.String  
    aEncodeType)
```

Description

Specifies a particular outbound character encoding for marshalled data (serialized one-dimensional BLOB) of the specified Event Type instance.

Parameters

Name	Type	Description
aInstanceName	java.lang.String	Name of the Event Type instance in the current Collaboration.
aEncodeType	java.lang.String	The character encoding to be set. <ul style="list-style-type: none"> ▪ For Japanese, use one of the following <ul style="list-style-type: none"> ◆ SJIS ◆ EUC-JP ◆ ISO-2022-JP ◆ CP930 ▪ For Korean, use ksc_5601

Return Type

None.

Throws

None.

Comments

For a complete explanation of how this encoding method fits with the other seven, see [“About the Encoding Methods” on page 580](#).

setUnmarshalEncoding()

Syntax

```
void this.jCollabController.setUnmarshalEncoding(
    java.lang.String aInstanceName,
    java.lang.String aEncodeType)
```

Description

Specifies a particular character encoding for the specified Event Type instance as its data is being unmarshalled (parsed).

Parameters

Name	Type	Description
aInstanceName	java.lang.String	Name of the Event Type instance in the current Collaboration.
aEncodeType	java.lang.String	The character encoding to be set: <ul style="list-style-type: none"> ▪ For Japanese, use SJIS ▪ For Korean, use ksc_5601 This specifies the native code for the Java Collaboration.

Return Type

None.

Throws

None.

Comments

For a complete explanation of how this encoding method fits with the other seven, see [“About the Encoding Methods” on page 580](#).

A.7.3 ELSController Interface Methods

ELS methods, such as methods to query, get, and set values, are available via the ELSController interface of a *particular instance* of the **JCollabController** class, and are therefore accessed inside a Collaboration with the following method invocation:

```
this.jCollabController.getELSController()
```

▪ **Methods for Link Identifiers:**

- ◆ [this.jCollabController.getELSController\(\).flushAllLinkIdentifiers\(\) on page 590](#)
- ◆ [this.jCollabController.getELSController\(\).getLinkIdentifiers\(\) on page 592](#)
- ◆ [this.jCollabController.getELSController\(\).getCurrentLinkIdentifier\(\) on page 590](#)
- ◆ [this.jCollabController.getELSController\(\).isFlushMode\(\) on page 596](#)
- ◆ [this.jCollabController.getELSController\(\).isLinkIdentifierExists\(\) on page 596](#)

▪ **Methods for message count:**

- ◆ [this.jCollabController.getELSController\(\).hasHappened\(\) on page 594](#)
- ◆ [this.jCollabController.getELSController\(\).getNumberOfMessages\(\) on page 593](#)
- ◆ [this.jCollabController.getELSController\(\).getNumberOfMessagesForInstance\(\) on page 593](#)

▪ **Methods for setting and getting timers:**

- ◆ [this.jCollabController.getELSController\(\).setELSExpiration\(\) on page 598](#)
- ◆ [this.jCollabController.getELSController\(\).getELSExpiration\(\) on page 591](#)
- ◆ [this.jCollabController.getELSController\(\).isCurrentELSExpired\(\) on page 595](#)
- ◆ [this.jCollabController.getELSController\(\).isELSExpired\(\) on page 595](#)

flushAllLinkIdentifiers()

Syntax

```
int  
this.jCollabController.getELSController().flushAllLinkIdentifiers()
```

Description

Sets the flush-mode flag to **true** (thus temporarily suspending the processing of all inbound messages) and runs the code in **executeBusinessRules()** for all Events, regardless of Link Identifier. This operation is called *flushing* the Link Identifiers.

After all Link Identifiers have been flushed, the flush-mode flag is reset to **false**.

Note: While the flush-mode flag is **true**, no further incoming messages are accepted.

Parameters

None.

Return Type

int

The number of Link Identifiers left to be flushed.

Throws

None.

Notes

Typically, you would have your Collaboration invoke **flushAllLinkIdentifiers()** when a condition occurs that renders further ELS processing irrelevant or unnecessary.

For example, you might want to reinitialize the system at a particular time every week, or if a particular kind of Event is received, or if a particular kind of exception is thrown.

Also see [isFlushMode\(\)](#) on page 596.

getCurrentLinkIdentifier()

Syntax

```
String  
this.jCollabController.getELSController().getCurrentLinkIdentifier()
```

Description

Retrieves the name of the current Link Identifier.

Parameters

None.

Return Type

String

Throws

None.

Comments

The text string returned by this method is the key to passing an *aLinkIdentifier* argument to any of the following methods:

```
getELSExpiration(String aLinkIdentifier);
setELSExpiration(String aLinkIdentifier, long aExpirationTime);
isELSExpired(String aLinkIdentifier);
isLinkIdentifierExists(String aLinkIdentifier);
hasHappened(String aLinkIdentifier);
getNumberOfMessages(String aLinkIdentifier);
getNumberOfMessages(String aLinkIdentifier, String aTopicName);
getNoOfMessagesForInstance(String aLinkIdentifier, String aInstanceName)
```

getELSExpiration()

Syntax

```
long this.jCollabController.getELSController().getELSExpiration(
    aLinkIdentifier)
```

Description

Retrieves the expiration time (in milliseconds) of the specified Link Identifier.

The default expiration time is 0—in other words, no expiration.

Parameters

Name	Type	Description
aLinkIdentifier	java.lang.String	Name of the Link Identifier.

Return Type

long

The expiration time set for the specified Link Identifier, in milliseconds.

Throws

None.

Notes

getELSExpiration() always returns **0** if there has been no call to **setELSExpiration()** for the specified Link Identifier.

getLinkIdentifiers()

Syntax

```
Iterator  
this.jCollabController.getELSController().getLinkIdentifiers()
```

Description

Retrieves a list of all Link Identifiers that have been defined.

Parameters

None.

Return Type

java.util.Iterator

Throws

None.

Notes

The Iterator returned by this method contains *aLinkIdentifier* text strings that are key for invoking any of the following methods:

- **getELSExpiration(String *aLinkIdentifier*);**
- **setELSExpiration(String *aLinkIdentifier*, long *aExpirationTime*);**
- **isELSExpired(String *aLinkIdentifier*);**
- **isLinkIdentifierExists(String *aLinkIdentifier*);**
- **hasHappened(String *aLinkIdentifier*)**
- **getNumberOfMessages(String *aLinkIdentifier*);**
- **getNumberOfMessages(String *aLinkIdentifier*, String *aTopicName*);**
- **getNoOfMessagesForInstance(String *aLinkIdentifier*, String *aInstanceName*);**

getNumberOfMessages()

Syntax

```
int this.jCollabController.getELSController().getNumberOfMessages(  
    aLinkIdentifier)  
  
int this.jCollabController.getELSController().getNumberOfMessages(  
    aLinkIdentifier,  
    aTopicName)
```

Description

These two methods retrieve the number of Events currently available for the specified Link Identifier. If the *aTopicName* argument is also specified, this method retrieves only the number of Events for the specified topic (same as Event Type) and Link Identifier.

Parameters

Name	Type	Description
aLinkIdentifier	java.lang.String	Name of the Link Identifier.
aTopicName	java.lang.String	Name of the Event Type.

Return Type

int

The number of Events currently available. If there are no Events, or if no such Link Identifier exists, returns 0.

Throws

None.

getNoOfMessagesForInstance()

Syntax

```
int  
this.jCollabController.getELSController().getNoOfMessagesForInstance(  
    aLinkIdentifier,  
    aInstanceName)
```

Description

Retrieves the number of Events currently available for the specified Link Identifier for the specified Event instance.

Parameters

Name	Type	Description
aLinkIdentifier	java.lang.String	Name of the Link Identifier.
aInstanceName	java.lang.String	Name of the Event instance in the current Collaboration.

Return Type**int**

The number of Events meeting the criteria. If there are no Events, or no such Link Identifier exists, returns 0.

Throws

None.

hasHappened()**Syntax**

```
boolean this.jCollabController.getELSController().hasHappened(  
    aLinkIdentifier)
```

Description

Inquires whether the specified Link Identifier has already been processed by this run of the Collaboration.

Parameters

Name	Type	Description
aLinkIdentifier	java.lang.String	Name of the Link Identifier.

Return Type**boolean**

Returns **true** if and only if the specified Link Identifier has already been processed by this run of the Collaboration—in other words, if **executeBusinessRules()** was called previously after successful execution of either **isLinkingComplete()** or **onExpire()**.

Throws

None.

isCurrentELSExpired()

Syntax

```
boolean  
this.jCollabController.getELSController().isCurrentELSExpired(  
    aLinkIdentifier)
```

Description

Inquires whether the specified Link Identifier is expired.

Parameters

Name	Type	Description
aLinkIdentifier	java.lang.String	Name of the Link Identifier.

Return Type

boolean

Returns **true** if the specified Link Identifier is expired; otherwise returns **false**.

Throws

None.

isELSExpired()

Syntax

```
boolean this.jCollabController.getELSController().isELSExpired(  
    aLinkIdentifier)
```

Description

Inquires whether the specified Link Identifier is expired.

When this method returns **true**, if the Message Count for the specified Link Identifier has not been reached, control passes to the block of user-written code under the **onExpire()** placeholder.

Parameters

Name	Type	Description
aLinkIdentifier	java.lang.String	Name of the Link Identifier.

Return Type

boolean

Returns **true** if the specified Link Identifier is expired; otherwise returns **false**.

Throws

None.

isFlushMode()

Syntax

```
boolean this.jCollabController.getELSController().isFlushMode()
```

Description

Inquires whether the system is in flush mode (and thus not accepting any incoming messages).

Parameters

None.

Return Type

boolean

Returns **true** if the system is in flush mode; otherwise returns **false**.

Throws

None.

Notes

Also see [flushAllLinkIdentifiers\(\)](#) on page 590.

isLinkIdentifierExists()

Syntax

```
boolean  
this.jCollabController.getELSController().isLinkIdentifierExists(  
aLinkIdentifier)
```

Description

Inquires whether the specified Link Identifier already exists.

Parameters

Name	Type	Description
aLinkIdentifier	java.lang.String	Name of the Link Identifier.

Return Type

boolean

Returns **true** if the specified Link Identifier already exists; otherwise returns **false**.

Throws

None.

isLinkingComplete()**Syntax**

```
boolean isLinkingComplete(aLinkIdIdentifier)
```

Description

Placeholder method for the main block of ELS code.

If this method's **retBoolean** variable returns **true**, the ELS portion of the code ends, and control passes to the main portion of the Java Collaboration Rule—that is, to the block of code under **executeBusinessRules()**.

Parameters

Name	Type	Description
aLinkIdIdentifier	java.lang.String	Name of the Link Identifier; see " getCurrentLinkIdentifier() " on page 590.

Return Type

boolean

Throws

CollabDataException, **CollabConnException**

onExpire()**Syntax**

```
boolean onExpire(aLinkIdIdentifier)
```

Description

Placeholder method for the block of code that you want to execute when a Link Identifier has expired before its Message Count has been reached.

If this method's **retBoolean** variable returns **true**, the ELS portion of the code ends, and control passes to the main portion of the Java Collaboration Rule—that is, the block of code under **executeBusinessRules()**.

ELS loops until the block of code under this placeholder becomes **true**.

Parameters

Name	Type	Description
aLinkIdentifier	java.lang.String	Name of the Link Identifier.

Return Type**boolean**

retrieveLinkIdentifier()**Syntax**

```
String retrieveLinkIdentifier()
```

Description

Placeholder method for the block of code that you want to execute to fetch and process a Link Identifier (for example, to read and manipulate a certain field).

Parameters

None.

Return Type**String**

setELSExpiration()**Syntax**

```
void this.jCollabController.getELSController().setELSExpiration(  
    aLinkIdentifier,  
    aExpirationTime)
```

Description

Sets the expiration time (in milliseconds) of the specified Link Identifier.

Parameters

Name	Type	Description
aLinkIdentifier	java.lang.String	Name of the Link Identifier.
aExpirationTime	long	Expiration time, in milliseconds.

Return Type

None.

Throws

None.

A.8 JCollaboration Class (com.stc.jcsre)

eventSend()

Access to the **eventSend()** methods is provided indirectly via the **JCollaboration** class, supplied in the SeeBeyond package **com.stc.jcsre.JCollaboration**, from which all Java Collaboration are subclassed (extended):

```
public abstract class JCollaboration
extends java.lang.Object
```

However, **eventSend()** is a member method of the **Alerter** class, supplied in the SeeBeyond package **com.stc.common.collabService**, class **JCollabController**:

```
public class JCollabController;
public class Alerter
extends java.lang.Object
```

Syntax

```
public boolean eventSend(alertCategory, alertSubcategory,
                        alertInfoCode, reasonCode, reasonName,
                        eventInfo, additionalInfo)

public boolean eventSend(severityLevel, alertCategory,
                        alertSubcategory, elemType, reasonCode,
                        reasonName, eventInfo, additionalInfo)

eventSend(com.stc.common.collabService.JCollabController jController,
          alertCategory, alertSubcategory, alertInfoCode, reasonCode,
          reasonName, eventInfo, additionalInfo)
```

Description

Sends Alert Events to the Control Broker. These can be viewed in the e*Gate Monitor or the Alert Agent, if so configured.

Parameters

Name	Type	Description
alertCategory	java.lang.String	Alert-Category Constant (see “ alertCategory Constants ”).
alertSubcategory	java.lang.String	Alert-Subcategory Constant (see “ alertSubcategory Constants ”).
alertInfoCode	java.lang.String	Info-Code Constant (see “ alertInfoCode Constants ”).
reasonCode	int	Status or error code generated by the operating system or the application generating the Event.
reasonName	java.lang.String	Reason why the Event occurred.
eventInfo	java.lang.String	Reserved for user agents or other applications using the SeeBeyond API to create monitoring Events that use this field. It can be an empty string ("").
additionalInfo	java.lang.String	Reserved for future use. It can be an empty string ("").

Name	Type	Description
severityLevel	java.lang.String	Severity-level Constant (see “ severityLevel Constants ”).
elemType	java.lang.String	Element-type Constant (see “ elemType Constants ”).

alertCategory Constants

Name	Description
Alerter.ALERTCAT_STATE_ELEM	Element state
Alerter.ALERTCAT_MESSAGE_CONTENT	Message content
Alerter.ALERTCAT_STATE_EXTERNAL	External state
Alerter.ALERTCAT_OPERATIONAL	Operational
Alerter.ALERTCAT_PERFORMANCE	Performance
Alerter.ALERTCAT_RESOURCE	Resource
Alerter.ALERTCAT_USERDEFINED	User-defined

alertSubcategory Constants

Name	Description
Alerter.ALERTSUBCAT_CUSTOM	Custom category
Alerter.ALERTSUBCAT_DOWN	Down
Alerter.ALERTSUBCAT_UP	Up
Alerter.ALERTSUBCAT_UNRESP	Unresponsive
Alerter.ALERTSUBCAT_RESP	Respond
Alerter.ALERTSUBCAT_CANTCONN	Unable to connect
Alerter.ALERTSUBCAT_CONN	Connected
Alerter.ALERTSUBCAT_LOSTCONN	Lost connection
Alerter.ALERTSUBCAT_UNUSABLE	Unusable/cannot ID
Alerter.ALERTSUBCAT_INTEREST	Interest
Alerter.ALERTSUBCAT_EXPIRED	Expired
Alerter.ALERTSUBCAT_INTHRESH	Input threshold
Alerter.ALERTSUBCAT_OUTTHRESH	Output threshold
Alerter.ALERTSUBCAT_USERAUTH	User authentication
Alerter.ALERTSUBCAT_DELIVERY	Alert delivery
Alerter.ALERTSUBCAT_UNQUEUEABLE	Unqueueable
Alerter.ALERTSUBCAT_DISKTHRESH	Disk threshold
Alerter.ALERTSUBCAT_IQLIMIT	IQ limit
Alerter.ALERTSUBCAT_STATUS	Status
Alerter.ALERTSUBCAT_TIMER	Timer

alertInfoCode Constants

Name	Description
Alerter.ALERTINFO_NONE	None
Alerter.ALERTINFO_FATAL	Fatal
Alerter.ALERTINFO_CONTROLLED	Controlled
Alerter.ALERTINFO_USER	User
Alerter.ALERTINFO_LOW	Low
Alerter.ALERTINFO_HIGH	High
Alerter.ALERTINFO_IOFAILED	I/O failure
Alerter.ALERTINFO_BELOW	Below
Alerter.ALERTINFO_ABOVE	Above

severityLevel Constants

Name	Description
Alerter.SEVERITY_LEVEL_UNDEFINED	Undefined
Alerter.SEVERITY_LEVEL_TRACE	Trace
Alerter.SEVERITY_LEVEL_DEBUG	Debugging
Alerter.SEVERITY_LEVEL_INFO	Information
Alerter.SEVERITY_LEVEL_WARNING	Warning
Alerter.SEVERITY_LEVEL_ERROR	Error
Alerter.SEVERITY_LEVEL_FATAL	Fatal

elementType Constants

Name	Description
Alerter.ALERTCAT_STATE_ELEM	Element state
Alerter.ALERTCAT_MESSAGE_CONTENT	Message content
Alerter.ALERTCAT_STATE_EXTERNAL	External state
Alerter.ALERTCAT_OPERATIONAL	Operational
Alerter.ALERTCAT_PERFORMANCE	Performance
Alerter.ALERTCAT_RESOURCE	Resource
Alerter.ALERTCAT_USERDEFINED	User-defined

Return Type

boolean—Returns **true** when an Alert Event is sent successfully.

Throws

None.

Example

```
eventSend(Alerter.ALERTCAT_MESSAGE_CONTENT,  
          Alerter.ALERTSUBCAT_USERAUTH,  
          Alerter.ALERTINFO_IOFAILED,  
          35827, "Disk Full", "", "");
```

A.9 JSubCollabMapInfo Class (com.stc.common.collabService)

The SeeBeyond class com.stc.common.collabService.**JSubCollabMapInfo** provides the following public methods:

- [getCallingCollaboration\(\)](#) on page 603
 - [getClassFullPath\(\)](#) on page 604
 - [getClassName\(\)](#) on page 605
 - [getCtlFileFullPath\(\)](#) on page 605
 - [getCtlFileName\(\)](#) on page 606
 - [getEventTypeDefinition\(\)](#) on page 607
 - [getEventTypeDefinitionPath\(\)](#) on page 607
 - [getInputData\(\)](#) on page 608
 - [getInputTopicName\(\)](#) on page 609
 - [getOutputData\(\)](#) on page 609
 - [getParentReferenceETD\(\)](#) on page 610
 - [getParentReferenceInstanceName\(\)](#) on page 611
 - [getRuleName\(\)](#) on page 611
 - [isManualPublish\(\)](#) on page 612
 - [isPublisher\(\)](#) on page 613
 - [isTrigger\(\)](#) on page 613
 - [setInstanceMap\(\)](#) on page 614
-

getCallingCollaboration()

Access to the **getCallingCollaboration()** method is provided via the **JSubCollabMapInfo** class, supplied in the SeeBeyond package **com.stc.common.collabService**:

```
package com.stc.common.collabService;  
public class JSubCollabMapInfo
```

Syntax

```
public JSubCollabMapInfo.getCallingCollaboration()
```

Description

Retrieves the JCollaboration—in other words, the Collaboration object—of the Collaboration Rule that is invoking the current Subcollaboration Rule.

Parameters

None.

Return Type

com.stc.jcsre.JCollaboration — the parent object of this Subcollaboration Rule.

Throws

None.

getFullPath()

Access to the **getFullPath()** method is provided via the **JSubCollabMapInfo** class, supplied in the SeeBeyond package **com.stc.common.collabService**:

```
package com.stc.common.collabService;  
  
public class JSubCollabMapInfo
```

Syntax

```
public java.lang.String JSubCollabMapInfo.getFullPath()
```

Description

Retrieves the location of the class corresponding to the current Subcollaboration Rule.

Parameters

None.

Return Type

java.lang.String

Throws

None.

getClassName()

Access to the **getClassName()** method is provided via the **JSubCollabMapInfo** class, supplied in the SeeBeyond package **com.stc.common.collabService**:

```
package com.stc.common.collabService;  
  
public class JSubCollabMapInfo
```

Syntax

```
public java.lang.String JSubCollabMapInfo.getClassName()
```

Description

Retrieves the name of the class corresponding to the current Subcollaboration Rule.

Parameters

None.

Return Type

java.lang.String

Throws

None.

getCtlFileFullPath()

Access to the **getCtlFileFullPath()** method is provided via the **JSubCollabMapInfo** class, supplied in the SeeBeyond package **com.stc.common.collabService**:

```
package com.stc.common.collabService;  
  
public class JSubCollabMapInfo
```

Syntax

```
public java.lang.String JSubCollabMapInfo.getCtlFileFullPath()
```

Description

Retrieves the location of the control file (.ctl) for the current Subcollaboration Rule.

Parameters

None.

Return Type

java.lang.String

Throws

None.

getCtlFileName()

Access to the **getCtlFileName()** method is provided via the **JSubCollabMapInfo** class, supplied in the SeeBeyond package **com.stc.common.collabService**:

```
package com.stc.common.collabService;  
  
public class JSubCollabMapInfo
```

Syntax

```
public java.lang.String JSubCollabMapInfo.getCtlFileName()
```

Description

Retrieves the name of the control file (.ctl) for the current Subcollaboration Rule.

Parameters

None.

Return Type

java.lang.String

Throws

None.

getEventTypeDefinition()

Access to the **getEventTypeDefinition()** method is provided via the **JSubCollabMapInfo** class, supplied in the SeeBeyond package **com.stc.common.collabService**:

```
package com.stc.common.collabService;  
  
public class JSubCollabMapInfo
```

Syntax

```
public java.lang.String JSubCollabMapInfo.getEventTypeDefinition(  
                      java.lang.String aInstanceName)
```

Description

Retrieves the name of the ETD file (.xsc) for the specified Event Type instance.

Parameter

Name	Type	Description
aInstanceName	java.lang.String	Name of the Event Type instance in the current Subcollaboration Rule.

Return Type

java.lang.String

Throws

None.

getEventTypeDefinitionPath()

Access to the **getEventTypeDefinitionPath()** method is provided via the **JSubCollabMapInfo** class, supplied in the SeeBeyond package **com.stc.common.collabService**:

```
package com.stc.common.collabService;  
  
public class JSubCollabMapInfo
```

Syntax

```
public java.lang.String JSubCollabMapInfo.getEventTypeDefinitionPath(  
                      java.lang.String aInstanceName)
```

Description

Retrieves the location of the ETD file (.xsc) for the specified Event Type instance.

Parameter

Name	Type	Description
aInstanceName	java.lang.String	Name of the Event Type instance in the current Subcollaboration Rule.

Return Type

java.lang.String

Throws

None.

getInputData()

Access to the **getInputData()** method is provided via the **JSubCollabMapInfo** class, supplied in the SeeBeyond package **com.stc.common.collabService**:

```
package com.stc.common.collabService;
public class JSubCollabMapInfo
```

Syntax

```
public byte[] JSubCollabMapInfo.getInputData(  
                                java.lang.String aInstanceName)
```

Description

Retrieves the marshalled (unparsed) input data for the specified Event Type instance.

Parameter

Name	Type	Description
aInstanceName	java.lang.String	Name of the Event Type instance in the current Subcollaboration Rule.

Return Type

byte[]—in other words, an array of bytes.

Throws

None.

getInputTopicName()

Access to the **getInputTopicName()** method is provided via the **JSubCollabMapInfo** class, supplied in the SeeBeyond package **com.stc.common.collabService**:

```
package com.stc.common.collabService;  
  
public class JSubCollabMapInfo
```

Syntax

```
public java.lang.String JSubCollabMapInfo.getInputTopicName(  
                java.lang.String aInstanceName)
```

Description

Retrieves the Event Type (topic) name for the specified Event Type instance.

Parameter

Name	Type	Description
aInstanceName	java.lang.String	Name of the Event Type instance in the current Subcollaboration Rule.

Return Type

java.lang.String

Throws

None.

getOutputData()

Access to the **getOutputData()** method is provided via the **JSubCollabMapInfo** class, supplied in the SeeBeyond package **com.stc.common.collabService**:

```
package com.stc.common.collabService;  
  
public class JSubCollabMapInfo
```

Syntax

```
public byte[] JSubCollabMapInfo.getOutputData(  
                java.lang.String aInstanceName)
```

Description

Retrieves the marshalled (unparsed) output data for the specified Event Type instance.

Parameter

Name	Type	Description
aInstanceName	java.lang.String	Name of the Event Type instance in the current Subcollaboration Rule.

Return Type

`byte[]`—in other words, an array of bytes.

Throws

None.

getParentReferenceETD()

Access to the `getParentReferenceETD()` method is provided via the **JSubCollabMapInfo** class, supplied in the SeeBeyond package **com.stc.common.collabService**:

```
package com.stc.common.collabService;
public class JSubCollabMapInfo
```

Syntax

```
public JSubCollabMapInfo.getParentReferenceETD(  
                      java.lang.String aInstanceName)
```

Description

Retrieves the ETD object that is the parent of the specified Event Type instance.

Parameter

Name	Type	Description
aInstanceName	java.lang.String	Name of the Event Type instance in the current Subcollaboration Rule.

Return Type

`com.stc.jcsre.ETD`—in other words, an ETD object.

Throws

None.

getParentReferenceInstanceName()

Access to the **getParentReferenceInstanceName()** method is provided via the **JSubCollabMapInfo** class, supplied in the SeeBeyond package **com.stc.common.collabService**:

```
package com.stc.common.collabService;  
  
public class JSubCollabMapInfo
```

Syntax

```
public java.lang.String  
JSubCollabMapInfo.getParentReferenceInstanceName(  
    java.lang.String aInstanceName)
```

Description

Retrieves the node name of the parent of the specified Event Type instance.

Parameter

Name	Type	Description
aInstanceName	java.lang.String	Name of the Event Type instance in the current Subcollaboration Rule.

Return Type

java.lang.String

Throws

None.

getRuleName()

Access to the **getRuleName()** method is provided via the **JSubCollabMapInfo** class, supplied in the SeeBeyond package **com.stc.common.collabService**:

```
package com.stc.common.collabService;  
  
public class JSubCollabMapInfo
```

Syntax

```
public java.lang.String JSubCollabMapInfo.getRuleName( )
```

Description

Retrieves the name of the current Subcollaboration Rule.

Parameters

None.

Return Type

java.lang.String

Throws

None.

isManualPublish()

Access to the **isManualPublish()** method is provided via the **JSubCollabMapInfo** class, supplied in the SeeBeyond package **com.stc.common.collabService**:

```
package com.stc.common.collabService;  
  
public class JSubCollabMapInfo
```

Syntax

```
public boolean JSubCollabMapInfo.isManualPublish(  
                                java.lang.String aInstanceName)
```

Description

Inquires whether the specified Event Type instance is set to **Manual Publish**.

Parameter

Name	Type	Description
aInstanceName	java.lang.String	Name of the Event Type instance in the current Subcollaboration Rule.

Return Type

boolean

Returns **true** if the specified Event Type instance is set to **Manual Publish**; otherwise returns **false**.

Throws

None.

isPublisher()

Access to the **isPublisher()** method is provided via the **JSubCollabMapInfo** class, supplied in the SeeBeyond package **com.stc.common.collabService**:

```
package com.stc.common.collabService;  
  
public class JSubCollabMapInfo
```

Syntax

```
public boolean JSubCollabMapInfo.isPublisher(  
                                java.lang.String aInstanceName)
```

Description

Inquires whether the specified Event Type instance is a publisher—in other words, if its mode is set to either **OUT** or **IN/OUT**.

Parameter

Name	Type	Description
aInstanceName	java.lang.String	Name of the Event Type instance in the current Subcollaboration Rule.

Return Type

boolean

Returns **true** if the specified Event Type instance is set to **OUT** or **IN/OUT**; returns **false** if it is set to **IN**.

Throws

None.

isTrigger()

Access to the **isTrigger()** method is provided via the **JSubCollabMapInfo** class, supplied in the SeeBeyond package **com.stc.common.collabService**:

```
package com.stc.common.collabService;  
  
public class JSubCollabMapInfo
```

Syntax

```
public boolean JSubCollabMapInfo.isTrigger(  
                                java.lang.String aInstanceName)
```

Description

Inquires whether the specified Event Type instance is set to **Trigger**.

Parameter

Name	Type	Description
aInstanceName	java.lang.String	Name of the Event Type instance in the current Subcollaboration Rule.

Return Type**boolean**

Returns **true** if the specified Event Type instance is set to **Trigger**; otherwise, **false**.

Throws

None.

setInstanceMap()

Access to the **setInstanceMap()** method is provided via the **JSubCollabMapInfo** class, supplied in the SeeBeyond package **com.stc.common.collabService**:

```
package com.stc.common.collabService;  
public class JSubCollabMapInfo
```

Syntax

```
public void JSubCollabMapInfo.setInstanceMap(  
    java.lang.String aInstanceName,  
    java.lang.String aParentReferenceETD,  
    java.lang.String aInputData,  
    java.lang.String aInputTopicName)
```

Description

For an *inbound* Event Type instance: Populates the instance with data, either from an ETD node (requires a queuing operation) or from the byte array.

For an *outbound* Event Type instance: Harvests data into the specified parent ETD node.

Parameters

Name	Type	Description
aInstanceName	java.lang.String	Name of the Event Type instance in the current Subcollaboration Rule.
aParentReferenceETD	com.stc.jcsre.ETD — in other words, an ETD object.	The parent ETD node; typically, this is a node that is dragged in from the GUI. For inbound Event Type instances, this can be null if no queuing operations are needed.
aInputData	byte[]	For inbound instances: The byte array with which to populate the instance. For outbound instances: null
aInputTopicName	java.lang.String	For inbound instances: The Event Type to be populated. For outbound instances: null

Return Type

None.

Throws

None.

A.10 Mainframe Class (com.stc.eways.util)

The SeeBeyond class com.stc.eways.util.**Mainframe** provides methods for converting to and from ASCII, EBCDIC, and PACDEC formats and converting to and from big-endian and little-endian formats.

- [a2e\(\)](#) on page 616
 - [e2a\(\)](#) on page 617
 - [e2S\(\)](#) on page 617
 - [swapInt\(\)](#) on page 618
 - [swapLong\(\)](#) on page 619
 - [swapShort\(\)](#) on page 619
-

a2e()

The two **a2e()** methods are provided in the **Mainframe** class, supplied in SeeBeyond package **com.stc.eways.util** (in **stcutil.jar**):

```
public class Mainframe
```

Syntax

```
static byte a2e(byte _asciichar)  
static byte[] a2e(byte[] _asciitext)
```

Description

These two methods convert an ASCII character or text string into its EBCDIC equivalent.

Parameters

Name	Type	Description
_asciichar	byte	The ASCII character to be converted.
_asciitext	byte[]	An array of ASCII characters to be converted.

Return Types

byte—in other words, a single byte.

byte[]—in other words, an array of bytes.

Throws

None.

e2a()

The two **e2a()** methods are provided in the **Mainframe** class, supplied in SeeBeyond package **com.stc.eways.util** (in **stcutil.jar**):

```
public class Mainframe
```

Syntax

```
static byte e2a(byte _ebcdicchar)  
static byte[] e2a(byte[] _ebcdictext)
```

Description

These two methods convert an EBCDIC character or text string into its ASCII equivalent.

Parameters

Name	Type	Description
_ebcdicchar	byte	The EBCDIC character to be converted.
_ebcdictext	byte[]	An array of EBCDIC characters to be converted.

Return Types

byte—in other words, a single byte.

byte[]—in other words, an array of bytes.

Throws

None.

e2S()

The **e2S()** method is provided in the **Mainframe** class, supplied in SeeBeyond package **com.stc.eways.util** (in **stcutil.jar**):

```
public class Mainframe
```

Syntax

```
static java.lang.String e2S(byte[] _ebcdictext)
```

Description

This method converts EBCDIC text into its Java (Unicode) equivalent.

Parameters

Name	Type	Description
_ebcdictext	byte[]	An array of EBCDIC characters to be converted.

Return Type

`java.lang.String`

Throws

None.

swapInt()

The **swapInt()** method is provided in the **Mainframe** class, supplied in SeeBeyond package **com.stc.eways.util** (in **stcutil.jar**):

```
public class Mainframe
```

Syntax

```
static int swapInt(int _number)
```

Description

Changes the endian-ness of the specified integer:

- Big-endian treats the leftmost byte as the most significant byte.
- Little-endian treats the rightmost byte as the most significant byte.

Parameter

Name	Type	Description
_number	int	The number to be converted

Return Type

`int`

Throws

None.

swapLong()

The **swapLong()** method is provided in the **Mainframe** class, supplied in SeeBeyond package **com.stc.eways.util** (in **stcutil.jar**):

```
public class Mainframe
```

Syntax

```
static long swapLong(long _number)
```

Description

Changes the endian-ness of the specified long integer:

- Big-endian treats the leftmost byte as the most significant byte.
- Little-endian treats the rightmost byte as the most significant byte.

Parameter

Name	Type	Description
_number	long	The number to be converted

Return Type

long

Throws

None.

swapShort()

The **swapShort()** method is provided in the **Mainframe** class, supplied in SeeBeyond package **com.stc.eways.util** (in **stcutil.jar**):

```
public class Mainframe
```

Syntax

```
static short swapShort(short _number)
```

Description

Changes the endian-ness of the specified short integer:

- Big-endian treats the leftmost byte as the most significant byte.
- Little-endian treats the rightmost byte as the most significant byte.

Parameter

Name	Type	Description
_number	short	The number to be converted

Return Type

int (if the endian-ness of an int is being swapped)

long (if the endian-ness of a long is being swapped)

short (if the endian-ness of a short is being swapped)

Throws

None.

A.11 MapUtils Class (com.stc.eways.util)

The SeeBeyond class com.stc.eways.util.**MapUtils** provides methods for creating, querying, and manipulating Map objects and their contents.

- [doMap\(\)](#) on page 622
- [parseMap\(\)](#) on page 623
- [readMap\(\)](#) on page 624
- [renderMap\(\)](#) on page 625
- [writeMap\(\)](#) on page 626

Usage Example

Here is an example of how the **parseMap()**, **doMap()**, and **renderMap()** methods combine and interact:

```
String mapAsString = "do,Sun|lu,Mon|ma,Tue|mi,Wed|ju,Thu|vi,Fri|sa,Sat";
String fieldSeparator = ",";
String recordSeparator = "|";

java.util.Map daySp2En3Map = MapUtils.parseMap(mapAsString,
                                                fieldSeparator, recordSeparator);

String luEn3Day = MapUtils.doMap(daySp2En3Map, "lu");
String saEn3Day = MapUtils.doMap(daySp2En3Map, "sa");

String copyOfMapAsString = MapUtils.renderMap(daySp2En3Map,
                                               fieldSeparator, recordSeparator);
```

In this example, a Map object named `daySp2En3Map` holds a mapping between Spanish two-character day-of-week abbreviations and English three-character day-of-week abbreviations:

- To set things up, data for the map is stored in a variable named `mapAsString`.
- Next, **parseMap()** is called variable was parsed this data into the Map object.
- Next, **doMap()** is called twice to look up the English equivalents of “lu” (lunes, Monday) and “sa” (sabado, Saturday).
- Finally, **renderMap()** is called to create a new string from the Map object. The new string created by **renderMap()** has exactly the same data as the string that was used to set things up in the first place.

doMap()

Access to the two **doMap()** methods is provided via the **MapUtils** class, supplied in the SeeBeyond package **com.stc.eways.util**:

```
package com.stc.eways.util;  
  
public class MapUtils
```

Syntax

```
public java.lang.String MapUtils.doMap(java.util.Map _map,  
                                java.lang.String _key)  
  
public java.lang.String MapUtils.doMap(java.lang.String _fileName,  
                                java.lang.String _key)
```

Description

Looks in the specified Map object or file and finds the String corresponding to the specified key.

Parameters

Name	Type	Description
_map	java.util.Map	The Map object to be consulted.
_fileName	java.lang.String	The name of the file containing the mapping to be consulted.
_key	java.lang.String	The item to be looked up.

Return Type

java.lang.String

Throws

java.io.IOException—If the map cannot be read from the file specified by *_fileName*.

Comments

See the [Usage Example](#) on page 621 for a short sample of how the MapUtils methods are used.

parseMap()

Access to the **parseMap()** method is provided via the **MapUtils** class, supplied in the SeeBeyond package **com.stc.eways.util**:

```
package com.stc.eways.util;  
  
public class MapUtils
```

Syntax

```
public java.util.Map MapUtils.parseMap(java.lang.String _str,  
                                      java.lang.String _fieldSep,  
                                      java.lang.String _recSep)
```

Description

Creates a Map object, using the specified field-separators and record-separators to parse the specified text string.

Parameters

Name	Type	Description
_str	java.lang.String	The text string to be parsed into a Map object.
_fieldSep	java.lang.String	The text string used to delimit each field from the next. This is usually a single character; the default is , (comma).
_recSep	java.lang.String	The text string used to delimit each record from the next. This is usually a single character; the default is (vertical bar, or “pipe” character).

Return Type

java.util.map

Throws

None.

Comments

See the [Usage Example](#) on page 621 for a short sample of how the MapUtils methods are used.

readMap()

Access to the two `readMap()` methods is provided via the **MapUtils** class, supplied in the SeeBeyond package **com.stc.eways.util**:

```
package com.stc.eways.util;  
  
public class MapUtils
```

Syntax

```
public java.util.Map MapUtils.readMap(java.lang.String _fileName)  
  
public java.util.Map MapUtils.readMap(java.lang.String _fileName,  
                                     java.lang.String _fieldSep,  
                                     java.lang.String _recSep)
```

Description

Creates a Map object, using the specified file and default or user-specified field-separators and record-separators.

Parameters

Name	Type	Description
_fileName	java.lang.String	Name of the file to be mapped.
_fieldSep	java.lang.String	The text string used to delimit each field from the next. This is usually a single character; the default is , (comma).
_recSep	java.lang.String	The text string used to delimit each record from the next. This is usually a single character; the default is (vertical bar, or “pipe” character).

Return Type

java.util.map

Throws

java.io.IOException—If the map cannot be read from the specified file.

renderMap()

Access to the **renderMap()** method is provided via the **MapUtils** class, supplied in the SeeBeyond package **com.stc.eways.util**:

```
package com.stc.eways.util;  
  
public class MapUtils
```

Syntax

```
public java.lang.String MapUtils.renderMap(java.util.Map _map,  
                                         java.lang.String _fieldSep,  
                                         java.lang.String _recSep)
```

Description

Creates a well-ordered output string whose fields and records are delimited by the specified strings (usually characters), allowing the text string to be parsed back into a Map object later if needed; see [parseMap\(\)](#) on page 623.

Parameters

Name	Type	Description
_map	java.util.Map	The Map object to be rendered into a delimited text string.
_fieldSep	java.lang.String	The text string used to delimit each field from the next. This is usually a single character; the default is , (comma).
_recSep	java.lang.String	The text string used to delimit each record from the next. This is usually a single character; the default is (vertical bar, or “pipe” character).

Return Type

java.lang.String

Throws

None.

Comments

See the [Usage Example](#) on page 621 for a short sample of how the MapUtils methods are used.

writeMap()

Access to the two **writeMap()** methods is provided via the **MapUtils** class, supplied in the SeeBeyond package **com.stc.eways.util**:

```
package com.stc.eways.util;  
  
public class MapUtils
```

Syntax

```
public void MapUtils.writeMap(java.lang.String _fileName,  
                           java.util.Map _map)  
  
public void MapUtils.writeMap(java.lang.String _fileName,  
                           java.util.Map _map,  
                           java.lang.String _fieldSep,  
                           java.lang.String _recSep)
```

Description

Writes the specified Map object to the specified file, using default or user-specified field-separators and record-separators.

Parameters

Name	Type	Description
_fileName	java.lang.String	The name of the file to be written.
_map	java.util.Map	The Map object to be written out to a file.
_fieldSep	java.lang.String	The text string used to delimit each field from the next. This is usually a single character; the default is , (comma).
_recSep	java.lang.String	The text string used to delimit each record from the next. This is usually a single character; the default is (vertical bar, or “pipe” character).

Return Type

None.

Throws

java.io.IOException—If the system cannot write the map to the specified file.

A.12 QSort Class (com.stc.common.utils)

qsort()

Access to the four **qsort()** methods is provided via the SeeBeyond package **com.stc.common.utils**, class **Qsort**:

```
public interface Qsort;
```

Syntax

```
public static void qsort(data[], compare)
public static void qsort(data, compare)
public static void qsort(data[], compare, case_sensitive)
public static void qsort(data, compare, case_sensitive)
```

Description

Puts the contents of an array or Vector into a well-ordered ranking—for example, alphabetizing characters or strings, or sorting integers or reals by size.

Parameters

Name	Type	Description
data[]	Object	Array of Objects to be sorted.
data	java.util.Vector	Vector of Objects to be sorted.
compare	CompareFunction	Function to be used for comparing the Objects in <i>data</i> or <i>data[]</i> , such stringCompare(Obj1 Obj2), which returns <0 if Obj1 < Obj2; 0 if obj1==obj2; and >0 if obj1 > obj2.
case_sensitive	boolean	When true, specifies that the sort distinguishes uppercase from lowercase—in other words, { A, B, ..., Z, a, b, ..., z } rather than { A, a, B, b, ..., Z, z }

Return Type

None.

Throws

None.

A.13 ScEncrypt Class (com.stc.common.utils)

The SeeBeyond class com.stc.common.utils.ScEncrypt provides methods for encrypting and decrypting strings using a decryption key.

- [decrypt\(\)](#) on page 628
 - [encrypt\(\)](#) on page 628
-

decrypt()

Access to the **decrypt()** method is provided via the SeeBeyond package **com.stc.common.utils**, class **ScEncrypt**:

```
package com.stc.common.utils;
public class ScEncrypt
```

Syntax

```
public static java.lang.String decrypt(java.lang.String key,
                                         java.lang.String data)
```

Description

Decrypts the input data using the decryption key: The data is transformed from a scrambled text string into plaintext form. If the string was encrypted multiple times using different keys, it can only be decrypted by applying the keys in reverse order.

Parameters

Name	Type	Description
key	java.lang.String	Decryption key
data	java.lang.String	Text string to be decrypted.

Return Type

java.lang.String

encrypt()

Access to the **encrypt()** method is provided via the SeeBeyond package **com.stc.common.utils**, class **ScEncrypt**:

```
package com.stc.common.utils;
public class ScEncrypt
```

Syntax

```
public static java.lang.String encrypt(java.lang.String key,  
                                     java.lang.String data)
```

Description

Encrypts the input data using the encryption key. The data is transformed into a scrambled text string that can only be deciphered using the **decrypt()** method and the correct key.

Parameters

Name	Type	Description
key	java.lang.String	Encryption key
data	java.lang.String	Text string to be encrypted.

Return Type

java.lang.String

A.14 STCTypeConverter Class (com.stc.eways.util)

The SeeBeyond class com.stc.common.utils.STCTypeConverter provides data type conversion methods for primitives and objects of the most common data types:

- [toBooleanPrimitive\(\)](#) on page 631
- [toBoolean\(\)](#) on page 632
- [toBytePrimitive\(\)](#) on page 634
- [toByte\(\)](#) on page 635
- [toByteArray\(\)](#) on page 636
- [toCharPrimitive\(\)](#) on page 638
- [toCharacter\(\)](#) on page 639
- [toDoublePrimitive\(\)](#) on page 640
- [toDouble\(\)](#) on page 642
- [toFloatPrimitive\(\)](#) on page 643
- [toFloat\(\)](#) on page 644
- [toIntegerPrimitive\(\)](#) on page 646
- [toInteger\(\)](#) on page 647
- [toLongPrimitive\(\)](#) on page 648
- [toLong\(\)](#) on page 650
- [toShortPrimitive\(\)](#) on page 651
- [toShort\(\)](#) on page 652
- [toString\(\)](#) on page 653

General Notes

- To convert to a primitive type, use the appropriate `to<Type>Primitive()` method. For example:

```
public boolean STCTypeConverter.toBooleanPrimitive(_value)
```

- To convert to an object type, use a `to<Object>()` method. For example:

```
public java.lang.String STCTypeConverter.toString(_value)
```

- **java.lang.IllegalArgumentException** is thrown whenever the `_value` to be converted is a null object.

- **java.lang.IllegalArgumentException** is also thrown in other circumstances when the specified `_value` badly mismatches the data type of the method. For example:

- Using `toByte()` on a `String` or `double _value` if it is too large to fit in one byte.
- Using `toShort()` on a `_value` that cannot fit into the range of a short integer.

toBooleanPrimitive()

Access to the sixteen **toBooleanPrimitive()** methods is provided via the **STCTypeConverter** class, supplied in the SeeBeyond package **com.stc.eways.util**:

```
package com.stc.eways.util;  
public class STCTypeConverter
```

Syntax

```
public boolean STCTypeConverter.toBooleanPrimitive(_value)
```

This method has sixteen possible signatures, corresponding to the following possible data types for *_value*:

- **Boolean** object
- **byte** primitive
- **Byte** object
- **char** primitive
- **Character** object
- **double** primitive
- **Double** object
- **float** primitive
- **Float** object
- **int** primitive
- **Integer** object
- **long** primitive
- **Long** object
- **short** primitive
- **Short** object
- **String** object

Description

Converts the specified primitive or object to a **boolean** primitive.

Parameter

Name	Type	Description
_value	(any of sixteen primitive or object types; see above)	Primitive or object to be converted to boolean .

Return Type**boolean**

- For the following primitives and objects—byte, short, int, long, float, double—returns **false** if and only if `_value == 0`; otherwise, returns **true**.
- For char primitives and Character objects:
 - ◆ Returns **true** if and only if `_value == TRUE_CHAR`
 - ◆ Returns **false** if and only if `_value == FALSE_CHAR`
- For String objects:
 - ◆ Returns **true** if and only if `_value == TRUE_STRING`
 - ◆ Returns **false** if and only if `_value == FALSE_STRING`

Throws**java.lang.IllegalArgumentException** thrown if specified object `_value` is null, or if it is:

- (for Character objects) neither **TRUE_CHAR** nor **FALSE_CHAR**
- (for String objects) neither **TRUE_STRING** nor **FALSE_STRING**

toBoolean()

Access to the sixteen **toBoolean()** methods is provided via the **STCTypeConverter** class, supplied in the SeeBeyond package **com.stc.eways.util**:

```
package com.stc.eways.util;  
public class STCTypeConverter
```

Syntax

```
public Boolean STCTypeConverter.toBoolean(_value)
```

This method has sixteen possible signatures, corresponding to the following possible data types for `_value`:

- **boolean** primitive
- **byte** primitive
- **Byte** object
- **char** primitive
- **Character** object

- **double** primitive
- **Double** object
- **float** primitive
- **Float** object
- **int** primitive
- **Integer** object
- **long** primitive
- **Long** object
- **short** primitive
- **Short** object
- **String** object

Description

Converts the specified primitive or object to a **Boolean** object.

Parameter

Name	Type	Description
<code>_value</code>	(any of sixteen primitive or object types; see above)	Primitive or object to be converted to a Boolean object.

Return Type

Boolean (object)

- For the following primitives and objects—byte, short, int, long, float, double—returns **false** if and only if `_value == 0`; otherwise, returns **true**.
- For char primitives and Character objects:
 - ◆ Returns **true** if and only if `_value == TRUE_CHAR`
 - ◆ Returns **false** if and only if `_value == FALSE_CHAR`
- For String objects:
 - ◆ Returns **true** if and only if `_value == TRUE_STRING`
 - ◆ Returns **false** if and only if `_value == FALSE_STRING`

Throws

java.lang.IllegalArgumentException thrown if specified object `_value` is null, or if it is:

- (for Character objects) neither **TRUE_CHAR** nor **FALSE_CHAR**
- (for String objects) neither **TRUE_STRING** nor **FALSE_STRING**

toBytePrimitive()

Access to the seventeen **toBytePrimitive()** methods is provided via the **STCTypeConverter** class, supplied in the SeeBeyond package **com.stc.eways.util**:

```
package com.stc.eways.util;  
public class STCTypeConverter
```

Syntax

```
public byte STCTypeConverter.toBytePrimitive(_value)
```

This method has seventeen possible signatures, corresponding to the following possible data types for *_value*:

- **boolean** primitive
- **Boolean** object
- **byte** primitive
- one-element byte array
- **Byte** object
- **char** primitive
- **Character** object
- **double** primitive
- **Double** object
- **float** primitive
- **Float** object
- **int** primitive
- **Integer** object
- **long** primitive
- **Long** object
- **short** primitive
- **Short** object
- **String** object

Description

Converts the specified primitive or object to a **byte** primitive.

Parameter

Name	Type	Description
_value	(any of seventeen primitive or object types; see above)	Primitive or object to be converted to a byte primitive.

Return Type**byte** (primitive)

- For boolean primitives and Boolean objects: Returns **1** if and only if `_value == true`; returns **0** if and only if `_value == false`.
- For char primitives and Character objects: Returns the ASCII code of the character.

Throws**java.lang.IllegalArgumentException** thrown if specified object `_value` is null, or if it is:

- too large to fit in a byte
- (for `byte[]` arrays) not a one-element byte array

toByte()

Access to the seventeen **toByte()** methods is provided via the **STCTypeConverter** class, supplied in the SeeBeyond package **com.stc.eways.util**:

```
package com.stc.eways.util;  
public class STCTypeConverter
```

Syntax

```
public Byte STCTypeConverter.toByte(_value)
```

This method has seventeen possible signatures, corresponding to the following possible data types for `_value`:

- **boolean** primitive
- **Boolean** object
- **byte** primitive
- one-element byte array
- **char** primitive
- **Character** object
- **double** primitive (must be small enough to fit in one byte)
- **Double** object (must be small enough to fit in one byte)
- **float** primitive (must be small enough to fit in one byte)
- **Float** object (must be small enough to fit in one byte)

- **int** primitive (must be small enough to fit in one byte)
- **Integer** object (must be small enough to fit in one byte)
- **long** primitive (must be small enough to fit in one byte)
- **Long** object (must be small enough to fit in one byte)
- **short** primitive (must be small enough to fit in one byte)
- **Short** object (must be small enough to fit in one byte)
- **String** object

Description

Converts the specified primitive or object to a **Byte** object.

Parameter

Name	Type	Description
<code>_value</code>	(any of seventeen primitive or object types; see above)	Primitive or object to be converted to a Byte object.

Return Type

Byte (object)

- For boolean primitives and Boolean objects: Returns **1** if and only if `_value == true`; returns **0** if and only if `_value == false`.
- For char primitives and Character objects: Returns the ASCII code of the character.

Throws

java.lang.IllegalArgumentException thrown if specified object `_value` is null, or if it is:

- too large to fit in a byte
- (for byte[] arrays) not a one-element byte array

toByteArray()

Access to the sixteen **toByteArray()** methods is provided via the **STCTypeConverter** class, supplied in the SeeBeyond package **com.stc.eways.util**:

```
package com.stc.eways.util;  
public class STCTypeConverter
```

Syntax

```
public byte[] STCTypeConverter.toByte(_value)  
public byte[] STCTypeConverter.toByte(java.lang.String _value  
                                     java.lang.String _encoding)
```

This method has sixteen possible signatures, corresponding to the following possible data types for `_value`:

- `byte` primitive
- `Byte` object
- `char` primitive
- `Character` object
- `double` primitive
- `Double` object
- `float` primitive
- `Float` object
- `int` primitive
- `Integer` object
- `long` primitive
- `Long` object
- `short` primitive
- `Short` object
- `String` object, default character encoding
- `String` object, specified character encoding

Description

Converts the specified primitive or object to a byte array.

Parameter

Name	Type	Description
<code>_value</code>	(any of fifteen primitive or object types; see above)	Primitive or object to be converted to a byte array.
<code>_encoding</code>	<code>java.lang.String</code>	The character encoding to use when converting a String object. Basic encoding sets are contained in rt.jar ; extended encoding sets are contained in i18n.jar . For a complete list, see http://java.sun.com/j2se/1.3/docs/guide/intl/encoding.doc.html

Return Type

`byte[]`—in other words, an array of bytes.

- For char and short primitives and Character and Short objects: Returns a two-element byte array containing the Unicode bytes for the character.
- For int and float primitives and Integer and Float objects: Returns a four-element byte array with the most significant bits in element 0.

- For double and long primitives and Double and Long objects: Returns an eight-element byte array with the most significant bits in element 0.
- For String objects: If *_encoding* is not specified, uses the default character encoding.

Throws

java.lang.IllegalArgumentException thrown if the specified object *_value* is null.

java.lang.UnsupportedEncodingException thrown if the specified character encoding *_encoding* is not supported.

toCharPrimitive()

Access to the seventeen **toCharPrimitive()** methods is provided via the **STCTypeConverter** class, supplied in the SeeBeyond package **com.stc.eways.util**:

```
package com.stc.eways.util;  
public class STCTypeConverter
```

Syntax

```
public char STCTypeConverter.toCharPrimitive(_value)
```

This method has seventeen possible signatures, corresponding to the following possible data types for *_value*:

- **boolean** primitive
- **Boolean** object
- **byte** primitive
- one-element or two-element byte array
- **Byte** object
- **Character** object
- **double** primitive
- **Double** object
- **float** primitive
- **Float** object
- **int** primitive
- **Integer** object
- **long** primitive
- **Long** object
- **short** primitive
- **Short** object

- **String** object (must be a one-character string)

Description

Converts the specified primitive or object to a **char** primitive.

Parameter

Name	Type	Description
_value	(any of seventeen primitive or object types; see above)	Primitive or object to be converted to a char primitive.

Return Type

char (primitive)

- For boolean primitives and Boolean objects: Returns **FALSE_CHAR** if and only if *_value* is **false**; returns **TRUE_CHAR** if and only if *_value* is **true**.

Throws

- java.lang.IllegalArgumentException** thrown if specified object *_value* is null, or if it is:
- a byte array that has zero elements or more than two elements
 - a String object that is empty or is longer than one character in length

toCharacter()

Access to the seventeen **toCharacter()** methods is provided via the **STCTypeConverter** class, supplied in the SeeBeyond package **com.stc.eways.util**:

```
package com.stc.eways.util;
public class STCTypeConverter
```

Syntax

```
public Character STCTypeConverter.toCharacter(_value)
```

This method has seventeen possible signatures, corresponding to the following possible data types for *_value*:

- **boolean** primitive
- **Boolean** object
- **byte** primitive
- one-element or two-element byte array
- **Byte** object
- **char** primitive
- **double** primitive
- **Double** object

- **float** primitive
- **Float** object
- **int** primitive
- **Integer** object
- **long** primitive
- **Long** object
- **short** primitive
- **Short** object
- **String** object (must be a one-character string)

Description

Converts the specified primitive or object to a **Character** object.

Parameter

Name	Type	Description
<code>_value</code>	(any of seventeen primitive or object types; see above)	Primitive or object to be converted to a Character object.

Return Type

Character (object)

- For boolean primitives and Boolean objects: Returns **FALSE_CHAR** if and only if `_value` is **false**; returns **TRUE_CHAR** if and only if `_value` is **true**.

Throws

java.lang.IllegalArgumentException thrown if specified object `_value` is null, or if it is:

- a byte array that has zero elements or more than two elements
- a String object that is empty or longer than one character in length

toDoublePrimitive()

Access to the seventeen **toDoublePrimitive()** methods is provided via the **STCTypeConverter** class, supplied in the SeeBeyond package **com.stc.eways.util**:

```
package com.stc.eways.util;  
  
public class STCTypeConverter
```

Syntax

```
public double STCTypeConverter.toDoublePrimitive(_value)
```

This method has seventeen possible signatures, corresponding to the following possible data types for `_value`:

- **boolean** primitive
- **Boolean** object
- **byte** primitive
- eight-element byte array
- **Byte** object
- **char** primitive
- **Character** object
- **double** primitive
- **Double** object
- **float** primitive
- **Float** object
- **int** primitive
- **Integer** object
- **long** primitive
- **Long** object
- **short** primitive
- **Short** object
- **String** object

Description

Converts the specified primitive or object to a **double** primitive.

Parameter

Name	Type	Description
<code>_value</code>	(any of seventeen primitive or object types; see above)	Primitive or object to be converted to a double primitive.

Return Type

double (primitive)

- For boolean primitives and Boolean objects: Returns **1** if and only if `_value == true`; returns **0** if and only if `_value == false`.

Throws

java.lang.IllegalArgumentException thrown if specified object `_value` is null, or if it is:

- (for byte[] arrays) not an eight-element byte array

Notes

When an eight-element byte array is converted, the most significant bits are taken from element 0.

toDouble()

Access to the seventeen **toDouble()** methods is provided via the **STCTypeConverter** class, supplied in the SeeBeyond package **com.stc.eways.util**:

```
package com.stc.eways.util;  
public class STCTypeConverter
```

Syntax

```
public Double STCTypeConverter.toDouble(_value)
```

This method has seventeen possible signatures, corresponding to the following possible data types for *_value*:

- **boolean** primitive
- **Boolean** object
- **byte** primitive
- eight-element byte array
- **Byte** object
- **char** primitive
- **Character** object
- **double** primitive
- **float** primitive
- **Float** object
- **int** primitive
- **Integer** object
- **long** primitive
- **Long** object
- **short** primitive
- **Short** object
- **String** object

Description

Converts the specified primitive or object to a **Double** object.

Parameter

Name	Type	Description
_value	(any of seventeen primitive or object types; see above)	Primitive or object to be converted to a Double object.

Return Type

Double (object)

- For boolean primitives and Boolean objects: Returns **1** if and only if `_value == true`; returns **0** if and only if `_value == false`.
- For char primitives and Character objects: Returns the ASCII code of the character.

Throws

java.lang.IllegalArgumentException thrown if specified object `_value` is null, or if it is:

- (for byte[] arrays) not an eight-element byte array

Notes

When an eight-element byte array is converted, the most significant bits are taken from element 0.

toFloatPrimitive()

Access to the seventeen **toFloatPrimitive()** methods is provided via the **STCTypeConverter** class, supplied in the SeeBeyond package **com.stc.eways.util**:

```
package com.stc.eways.util;
public class STCTypeConverter
```

Syntax

```
public float STCTypeConverter.toFloatPrimitive(_value)
```

This method has seventeen possible signatures, corresponding to the following possible data types for `_value`:

- **boolean** primitive
- **Boolean** object
- **byte** primitive
- four-element byte array
- **Byte** object
- **char** primitive
- **Character** object
- **double** primitive (must be in range)

- **Double** object (must be in range)
- **float** primitive
- **Float** object
- **int** primitive
- **Integer** object
- **long** primitive
- **Long** object
- **short** primitive
- **Short** object
- **String** object

Description

Converts the specified primitive or object to a **float** primitive.

Parameter

Name	Type	Description
_value	(any of seventeen primitive or object types; see above)	Primitive or object to be converted to a float primitive.

Return Type

float (primitive)

Throws

java.lang.IllegalArgumentException thrown if specified object *_value* is null, or if:

- (for byte[] arrays) *_value* is not a four-element byte array
- (for double primitives or Double objects) the result would be out of range

toFloat()

Access to the seventeen **toFloat()** methods is provided via the **STCTypeConverter** class, supplied in the SeeBeyond package **com.stc.eways.util**:

```
package com.stc.eways.util;  
public class STCTypeConverter
```

Syntax

```
public float STCTypeConverter.toFloat(_value)
```

This method has seventeen possible signatures, corresponding to the following possible data types for *_value*:

- **boolean** primitive
- **Boolean** object
- **byte** primitive
- four-element byte array
- **Byte** object
- **char** primitive
- **Character** object
- **double** primitive
- **Double** object
- **float** primitive
- **Float** object
- **int** primitive
- **Integer** object
- **long** primitive
- **Long** object
- **short** primitive
- **Short** object
- **String** object

Description

Converts the specified primitive or object to a **Float** object.

Parameter

Name	Type	Description
<code>_value</code>	(any of seventeen primitive or object types; see above)	Primitive or object to be converted to a Float object.

Return Type

Float (object)

Throws

java.lang.IllegalArgumentException thrown if specified object `_value` is null, or if:

- (for byte[] arrays) `_value` is not a four-element byte array
- (for double primitives or Double objects) the result would be out of range

toIntegerPrimitive()

Access to the seventeen **toIntegerPrimitive()** methods is provided via the **STCTypeConverter** class, supplied in the SeeBeyond package **com.stc.eways.util**:

```
package com.stc.eways.util;  
public class STCTypeConverter
```

Syntax

```
public int STCTypeConverter.toIntegerPrimitive(_value)
```

This method has seventeen possible signatures, corresponding to the following possible data types for *_value*:

- **boolean** primitive
- **Boolean** object
- **byte** primitive
- four-element byte array
- **Byte** object
- **char** primitive
- **Character** object
- **double** primitive
- **Double** object
- **float** primitive
- **Float** object
- **Integer** object
- **long** primitive
- **Long** object
- **short** primitive
- **Short** object
- **String** object

Description

Converts the specified primitive or object to an **int** primitive.

Parameter

Name	Type	Description
_value	(any of seventeen primitive or object types; see above)	Primitive or object to be converted to an int primitive.

Return Type

int (primitive)

Throws

java.lang.IllegalArgumentException thrown if specified object *_value* is null, or if:

- (for byte[] arrays) *_value* is not a four-element byte array
- (for long, float, or double primitives or objects) the result would be out of range

toInteger()

Access to the seventeen **toInteger()** methods is provided via the **STCTypeConverter** class, supplied in the SeeBeyond package **com.stc.eways.util**:

```
package com.stc.eways.util;
public class STCTypeConverter
```

Syntax

```
public Integer STCTypeConverter.toInteger(_value)
```

This method has seventeen possible signatures, corresponding to the following possible data types for *_value*:

- **boolean** primitive
- **Boolean** object
- **byte** primitive
- four-element byte array
- **Byte** object
- **char** primitive
- **Character** object
- **double** primitive (must be in range)
- **Double** object (must be in range)
- **float** primitive (must be in range)
- **Float** object (must be in range)
- **int** primitive

- **Integer** object
- **long** primitive (must be in range)
- **Long** object (must be in range)
- **short** primitive
- **Short** object
- **String** object

Description

Converts the specified primitive or object to an **Integer** object.

Parameter

Name	Type	Description
<code>_value</code>	(any of seventeen primitive or object types; see above)	Primitive or object to be converted to an Integer object.

Return Type

Integer (object)

Throws

`java.lang.IllegalArgumentException` thrown if specified object `_value` is null, or if:

- (for byte[] arrays) `_value` is not a four-element byte array
- (for long, float, or double primitives or objects) the result would be out of range

toLongPrimitive()

Access to the seventeen **toLongPrimitive()** methods is provided via the **STCTypeConverter** class, supplied in the SeeBeyond package **com.stc.eways.util**:

```
package com.stc.eways.util;  
public class STCTypeConverter
```

Syntax

```
public long STCTypeConverter.toLongPrimitive(_value)
```

This method has seventeen possible signatures, corresponding to the following possible data types for `_value`:

- **boolean** primitive
- **Boolean** object
- **byte** primitive
- eight-element byte array

- **Byte** object
- **char** primitive
- **Character** object
- **double** primitive (must be in range)
- **Double** object (must be in range)
- **float** primitive (must be in range)
- **Float** object (must be in range)
- **int** primitive
- **Integer** object
- **long** primitive
- **Long** object
- **short** primitive
- **Short** object
- **String** object

Description

Converts the specified primitive or object to a **long** primitive.

Parameter

Name	Type	Description
<code>_value</code>	(any of seventeen primitive or object types; see above)	Primitive or object to be converted to a long primitive.

Return Type

long (primitive)

Throws

java.lang.IllegalArgumentException thrown if specified object `_value` is null, or if:

- (for byte[] arrays) `_value` is not an eight-element byte array
- (for float or double primitives or objects) the result would be out of range

toLong()

Access to the seventeen **toLong()** methods is provided via the **STCTypeConverter** class, supplied in the SeeBeyond package **com.stc.eways.util**:

```
package com.stc.eways.util;  
public class STCTypeConverter
```

Syntax

```
public Long STCTypeConverter.toLong(_value)
```

This method has seventeen possible signatures, corresponding to the following possible data types for *_value*:

- **boolean** primitive
- **Boolean** object
- **byte** primitive
- eight-element byte array
- **Byte** object
- **char** primitive
- **Character** object
- **double** primitive (must be in range)
- **Double** object (must be in range)
- **float** primitive (must be in range)
- **Float** object (must be in range)
- **int** primitive
- **Integer** object
- **long** primitive
- **short** primitive
- **Short** object
- **String** object

Description

Converts the specified primitive or object to a **Long** object.

Parameter

Name	Type	Description
_value	(any of seventeen primitive or object types; see above)	Primitive or object to be converted to a Long object.

Return Type**Long** (object)**Throws****java.lang.IllegalArgumentException** thrown if specified object *_value* is null, or if:

- (for byte[] arrays) *_value* is not an eight-element byte array
- (for float, or double primitives or objects) the result would be out of range

toShortPrimitive()

Access to the seventeen **toShortPrimitive()** methods is provided via the **STCTypeConverter** class, supplied in the SeeBeyond package **com.stc.eways.util**:

```
package com.stc.eways.util;  
public class STCTypeConverter
```

Syntax

```
public short STCTypeConverter.toShortPrimitive(_value)
```

This method has seventeen possible signatures, corresponding to the following possible data types for *_value*:

- **boolean** primitive
- **Boolean** object
- **byte** primitive
- two-element byte array
- **Byte** object
- **char** primitive
- **Character** object
- **double** primitive (must be in range)
- **Double** object (must be in range)
- **float** primitive (must be in range)
- **Float** object (must be in range)
- **int** primitive (must be in range)

- **Integer** object (must be in range)
- **long** primitive (must be in range)
- **Long** object (must be in range)
- **Short** object
- **String** object

Description

Converts the specified primitive or object to a **short** primitive.

Parameter

Name	Type	Description
_value	(any of seventeen primitive or object types; see above)	Primitive or object to be converted to a short primitive.

Return Type

short (primitive)

Throws

java.lang.IllegalArgumentException thrown if specified object *_value* is null, or if:

- (for byte[] arrays) *_value* is not a two-element byte array
- (for int, long, float, or double primitives or objects) the result would be out of range

toShort()

Access to the seventeen **toShort()** methods is provided via the **STCTypeConverter** class, supplied in the SeeBeyond package **com.stc.eways.util**:

```
package com.stc.eways.util;  
public class STCTypeConverter
```

Syntax

```
public Short STCTypeConverter.toShort(_value)
```

This method has seventeen possible signatures, corresponding to the following possible data types for *_value*:

- **boolean** primitive
- **Boolean** object
- **byte** primitive
- two-element byte array
- **Byte** object

- **char** primitive
- **Character** object
- **double** primitive (must be in range)
- **Double** object (must be in range)
- **float** primitive (must be in range)
- **Float** object (must be in range)
- **int** primitive (must be in range)
- **Integer** object (must be in range)
- **long** primitive (must be in range)
- **Long** object (must be in range)
- **short** primitive
- **String** object

Description

Converts the specified primitive or object to a **Short** object.

Parameter

Name	Type	Description
_value	(any of seventeen primitive or object types; see above)	Primitive or object to be converted to a Short object.

Return Type

Short (object)

Throws

java.lang.IllegalArgumentException thrown if specified object *_value* is null, or if:

- (for byte[] arrays) *_value* is not a two-element byte array
- (for int, long, float, or double primitives or objects) the result would be out of range

toString()

Access to the eighteen **toString()** methods is provided via the **STCTypeConverter** class, supplied in the SeeBeyond package **com.stc.eways.util**:

```
package com.stc.eways.util;  
public class STCTypeConverter
```

Syntax

```
public java.lang.String STCTypeConverter.toString(_value)
```

This method has eighteen possible signatures, corresponding to the following possible data types for `_value`:

- **boolean** primitive
- **Boolean** object
- **byte** primitive
- byte array, using default character encoding
- byte array, using specified character encoding
- **Byte** object
- **char** primitive
- **Character** object
- **double** primitive
- **Double** object
- **float** primitive
- **Float** object
- **int** primitive
- **Integer** object
- **long** primitive
- **Long** object
- **short** primitive
- **Short** object
- **String** object

Description

Converts the specified primitive or object to a **String** object.

Parameter

Name	Type	Description
_value	(any of seventeen primitive or object types; see above)	Primitive or object to be converted to a byte array.
_encoding	java.lang.String	The character encoding to use when converting a byte[] array. Basic encoding sets are contained in rt.jar ; extended encoding sets are contained in i18n.jar . For a complete list, see http://java.sun.com/j2se/1.3/docs/guide/intl/encoding.doc.html

Return Type**String** (object)**Throws**

java.lang.IllegalArgumentException thrown if the object specified for `_value` is null or if `_encoding` is null.

java.lang.UnsupportedEncodingException—If the specified character encoding is not supported.

A.15 StringUtils Class (com.stc.eways.util)

The SeeBeyond class com.stc.eways.util.StringUtils provides the following public methods for querying, padding, and trimming text strings:

- [empty\(\)](#) on page 656
 - [padCenter\(\)](#) on page 657
 - [padLeft\(\)](#) on page 658
 - [padRight\(\)](#) on page 659
 - [trimBoth\(\)](#) on page 660
 - [trimLeft\(\)](#) on page 660
 - [trimRight\(\)](#) on page 661
-

empty()

The **empty()** method is provided in the **StringUtils** class, supplied in SeeBeyond package **com.stc.eways.util** (in **stcutil.jar**):

```
public class StringUtils
```

Syntax

```
static boolean empty(java.lang.String _string)
```

Description

Inquires whether the specified string is either null or empty.

Parameters

Name	Type	Description
_string	java.lang.String	The string to be queried.

Return Type

boolean

Returns **true** if the specified String is either null or empty; returns **false** if the String contains one or more non-null characters.

Throws

None.

padCenter()

The two **padCenter()** methods are provided in the **StringUtils** class, supplied in SeeBeyond package **com.stc.eways.util** (in **stcutil.jar**):

```
public class StringUtils
```

Syntax

```
static java.lang.String padCenter(java.lang.String _string,  
                                  int _length)  
  
static java.lang.String padCenter(java.lang.String _string,  
                                  int _length, char _padchar)
```

Description

These two methods prepend and append sufficient characters so that the returned string is centered in the specified length. The first method (with only two parameters) pads the string using blank spaces (\u0020); the second method (with three parameters) allows you to specify the padding character.

The length of the right padding equals that of the left or exceeds it by 1. For example, if the eleven-character string **This String** were centered to 40 characters with ^ (circumflex) characters, fourteen ^'s would be prepended and fifteen ^'s appended:

```
12345----1-----2----+---3----+---4  
^~~~~~^~~~~~This String^~~~~~^~~~~~^
```

Important: None of the padding methods truncate data: If the specified string length is insufficient to hold the entire string, the value of **_length** is ignored.

Parameters

Name	Type	Description
_string	java.lang.String	The string to be centered.
_length	int	The length of the centered string to be returned.
_padchar	char	If specified, the string can be padded on the left and right with nonblank characters.

Return Type

java.lang.String

Throws

None.

padLeft()

The **padLeft()** methods are provided in the **StringUtils** class, supplied in SeeBeyond package **com.stc.eways.util** (in **stcutil.jar**):

```
public class StringUtils
```

Syntax

```
static java.lang.String padLeft(java.lang.String _string,  
                                int _length)  
  
static java.lang.String padLeft(java.lang.String _string,  
                                int _length, char _padchar)
```

Description

These two methods prepend sufficient characters so that the returned string is right-justified in the specified length. The first method (with only two parameters) pads the string using blank spaces (\u0020); the second method (with three parameters) allows you to specify the padding character.

For example, the eleven-character string **This String** could be left-padded to 30 characters by prepending nineteen . (period) characters, as follows:

```
12345----1-----2----+---3  
.....This String
```

Important: *None of the padding methods truncate data: If the specified string length is insufficient to hold the entire string, the value of **_length** is ignored.*

Parameters

Name	Type	Description
_string	java.lang.String	The string to be left-padded.
_length	int	The length of the padded string to be returned.
_padchar	char	If specified, the string can be left-padded with nonblank characters.

Return Type

java.lang.String

Throws

None.

padRight()

The two **padRight()** methods are provided in the **StringUtils** class, supplied in SeeBeyond package **com.stc.eways.util** (in **stcutil.jar**):

```
public class StringUtils
```

Syntax

```
static java.lang.String padRight(java.lang.String _string,  
                                int _length)  
  
static java.lang.String padRight(java.lang.String _string,  
                                int _length, char _padchar)
```

Description

These two methods append sufficient characters so that the returned string is left-justified in the specified length. The first method (with only two parameters) pads the string using blank spaces (\u0020); the second method (with three parameters) allows you to specify the padding character.

For example, the eleven-character string **This String** would be right-padded to twenty characters by appending nine - (hyphen) characters, as follows:

```
12345----1-----2  
This String-----
```

Important: *None of the padding methods truncate data: If the specified string length is insufficient to hold the entire string, the value of **_length** is ignored.*

Parameters

Name	Type	Description
_string	java.lang.String	The string to be right-padded.
_length	int	The length of the padded string to be returned.
_padchar	char	If specified, the string can be right-padded with nonblank characters.

Return Type

java.lang.String

Throws

None.

trimBoth()

The **trimBoth()** method is provided in the **StringUtils** class, supplied in SeeBeyond package **com.stc.eways.util** (in **stcutil.jar**):

```
public class StringUtils
```

Syntax

```
public java.lang.String trimBoth(java.lang.String _string,  
                                java.lang.String _chars)
```

Description

Trims the specified character or characters from both the beginning and the end of the specified String.

Parameters

Name	Type	Description
_string	java.lang.String	The string to be trimmed.
_chars	java.lang.String	The character or characters to be removed from the string.

Return Type

java.lang.String

Throws

None.

Example

trimBoth("aababcbcdabcdcbabaa","ab") trims the first five characters and the last five characters, since they consist only of **a**s and **b**s, yielding "cbcddabcdcbc".

trimLeft()

The **trimLeft()** method is provided in the **StringUtils** class, supplied in SeeBeyond package **com.stc.eways.util** (in **stcutil.jar**):

```
public class StringUtils
```

Syntax

```
public java.lang.String trimLeft(java.lang.String _string,  
                                 java.lang.String _chars)
```

Description

Trims the specified character or characters from the beginning of the specified String.

Parameters

Name	Type	Description
_string	java.lang.String	The string to be trimmed.
_chars	java.lang.String	The character or characters to be removed from the string.

Return Type

java.lang.String

Throws

None.

Example

`trimLeft("aababcbddababcdcbabaa","ab")` trims the first five characters, since they consist only of **as** and **bs**, yielding "cbcddababcdcbabaa".

trimRight()

The **trimRight()** method is provided in the **StringUtils** class, supplied in SeeBeyond package **com.stc.eways.util** (in **stcutil.jar**):

```
public class StringUtils
```

Syntax

```
public java.lang.String trimRight(java.lang.String _string,
                                  java.lang.String _chars)
```

Description

Trims the specified character or characters from the end of the specified String.

Parameters

Name	Type	Description
_string	java.lang.String	The string to be trimmed.
_chars	java.lang.String	The character or characters to be removed from the string.

Return Type

java.lang.String

Throws

None.

Example

`trimRight("aababcbcddabcdcbbabaa","ab")` trims the last five characters, since they consist only of **as** and **bs**, yielding "aababcbcddabcdcbc".

A.16 Formatting of Output Text

Several methods, such as **copy()**, **dataMap()**, **lookup()**, and **sprintf()**, use a special set of flags to specify a pattern for text formatting. Here are some examples of these patterns:

- `%-#5.4D`
- `%+012.4o`
- `%-+4i`
- `%B`

Syntax

```
%[flag[flag]][width][.precision]<C>  
%[flag[flag]]#[width].<precision><C>
```

In the syntax pattern given above:

- The % is always required.
- Angle brackets denote a required argument—`<reqArg>`
- Square brackets denote an optional argument—`[optArg]`

Description

This format specification converts arguments from their internal representation to a printable form.

Parameters

Name	Description
[flag]	Formatting option that modifies the <C> conversion character. Up to three flags can be specified. Not all flags can be used with each data type. See "Examples" on page 664 for a list of flags that can be used with each data type. <ul style="list-style-type: none"> - Output is left aligned. + A sign (+ or -) always precedes output. space If the first character to be output is not a sign (+ or -), a space character is prefixed. Only one space is allowed in a format specification. 0 Numbers are right-aligned and padded with leading zeros. # Output includes a decimal point.
<width>	Minimum width of output data. A positive integer no greater than 9,999,999. The total is determined by the length of the formatted data but cannot be less than <width>. If the formatted data is narrower than <width>, then the result is left-padded or right-padded with spaces or zeros depending on other flags.
<.precision>	Number of digits to the right of the decimal place. A nonnegative integer. If used, must be preceded by a . (period) to distinguish it from <width>.
<C>	Conversion character indicating output data type. Data types with capital letters attempt to print that element as Monk-readable text. Lowercase data types print in a normal, text-readable format. Not available for E, F, or *; reserved for future use. Select one of the following: <ul style="list-style-type: none"> a, A Any Monk object. b, B Binary output of a number. d, D Decimal output of a number. Integer (positive or negative). e, E Exponent output of a number. Floating point number formatted with scientific notation [-]n.me+/-xx. f, F Fixed output of a number. Floating point number formatted with decimal notation [-]n.m where - is output for negative numbers. i, I Decimal output of a number. n, N Number of bytes written so far. o, O Octal output of a number. s, S String output. x, X hexadecimal output of a number. *

A literal string can be included in the format. For example

```
printf("Cherries are %s" "red." )      => "Cherries are red."
```

The following table relates conversion characters <C> to format flags.

<C>	Permitted Format Flags for <C>
a, A	-
b, B	0, +, -, space
d, D	+, -, space
e, E	+
f, F	+, .
i, I	+, -
n, N	none
o, O	0, -, +, #, space
s, S	none
x, X	0, -, +, #, space
*	none

Examples

```
printf("%b" "33")          => "100001"
printf("%-8c" "Tiger")      => "Tiger    "
printf("%07o" "33")         => "0000041"
```

These three examples demonstrate, respectively: binary conversion; left-justify using the minus character; and padding with zeros.

Table 86 lists a variety of inputs, formats and the resulting string.

Note: The doublequotes are not part of the data; instead, they show where spaces occur.

Table 86 Sample Inputs and Format Codes and Their Results

Input	Format Code	Result
Floating point format examples		
12.345	%9.0f	" 12"
12.345	%9.1f	" 12.3"
12.345	%9.2f	" 12.34"
12.345	%9.3f	" 12.345"
12.345	%9.4f	" 12.3450"
12.345	%8.4f	" 12.3450"
12.345	%7.4f	"12.3450"
12.345	%6.4f	"12.3450"
12.345	%09.0f	"000000012"
12.345	%09.1f	"0000012.3"
12.345	%09.2f	"000012.34"
12.345	%+-09.2f	"+12.34 "
-12.345	%+-09.2f	"-12.34 "
12.345	%+09.2f	"+00012.34"
-12.345	%+09.2f	"-00012.34"
12.345	%-09.2f	"12.34 "
-12.345	%-09.2f	"-12.34 "

Table 86 Sample Inputs and Format Codes and Their Results (Continued)

Input	Format Code	Result
Integer Format Examples		
123	%i	"123"
123	%8i	" 123"
123	%7i	" 123"
123	%-6i	"123 "
123	%-5i	"123 "
123	%+4i	"+123"
123	%+3i	"+123"
Octal Format Examples		
33	%o	"41"
33	% o	" 41"
33	%09o	"000000041"
33	%08o	"00000041"
33	%8o	" 41"
33	%7o	" 41"
33	%6o	" 41"
33	%5o	" 41"
33	%-9o	"41 "
33	%+09o	"+00000041"
-33	%+09o	"-00000041"
33	%+9o	" +41"
-33	%+9o	" -41"
-33	%#9o	" -41"

Glossary

Access Control List (ACL)

The security feature in e*Gate; a role-based list of information that specifies which users have permission to access e*Gate and its components and what specific access rights the users have.

advisory lock

The lock placed on a file when a user checks it out from the run-time schema. An advisory lock is simply a flag that warns other users that someone is already editing the file; it does not prevent other users from also checking out the file.

agent (Alert, SNMP)

A stand-alone application that monitors processes and resources and sends Notifications to e*Gate system users, informing them of system status (for example, when a preset disk space level is exceeded).

Application Programming Interface (API)

An API is the set of classes, functions, and methods of a particular programming language that developers use to code software. API documentation is the documenting of the syntax and use of the API methods.

business Event

A unit of data sent by an external system to e*Gate representing a change in that system's information.

Business Object Broker (BOB)

The executable component **stcbob.exe**. BOBs use Collaborations to route and transform data within the e*Gate system.

Business Rules pane

Use the **Business Rules** pane in the Java Collaboration Rules Editor to navigate and edit the Java code of a Collaboration.

Business Rules toolbar

Use the buttons on the Business Rules toolbar in the Java Collaboration Rules Editor to add corresponding Java statements to a Collaboration.

byte length

Length in bytes of the string or regular expression to be matched within an Event Type Definition. e*Gate measures fixed-length data from byte 1.

byte offset

The beginning byte location of the string or regular expression to be verified within an Event Type Definition, beginning at byte 0.

child nodes

Nodes that are below a given node within the same branch of the Event Type Definition tree. Child nodes can inherit certain properties, such as delimiters, from their parent nodes.

Collaboration

The component within an e*Way or BOB that performs data transformation and/or routing. It is the business logic that is applied to an Event in the course of delivery from a publisher to a subscriber. Collaboration components do the following functions: Subscriber components receive Events of a known type while publisher components distribute the transformed Events to a specified recipient. See also **Collaboration Rules**.

Collaboration Rules Editors

The graphical user interface (GUI) features used to work with Collaboration Rules scripts in the Java and Monk programming languages. See also **Collaboration Rules script**.

Collaboration-ID Rules Editor

The graphical user interface (GUI) feature used to create Collaboration Rules scripts in the Monk programming language for *e*Gate release 3.6 only*. See also **Collaboration Rules script**.

Collaboration Rules

The program logic that instructs a Collaboration how to execute the business logic required to support e*Gate's data transformation and routing. See also **Collaboration** and **Collaboration script**.

Collaboration Rules script

A Collaboration script (data program) written using the Collaboration Rules Editor feature.

Collaboration script

The data flow and transformation logic contained in and configured by an e*Gate Collaboration and written as a program in any of the following programming languages: Monk, Java, or C.

Collaboration Services

Libraries that provide the low-level facilities by which Collaborations execute Collaboration Rules, for example, issuing system-specific terminate calls.

command line

A tool for monitoring and controlling e*Gate by entering application program interface (API) commands at a DOS or DOS-type prompt.

committing files

Takes them out of the run-time schema and places them in the Sandbox. See also, **Sandbox** and **run time**.

Control Broker

An automatically generated e*Gate component that starts and monitors e*Ways and BOBs. At least one Control Broker must be running on each host within a schema.

delimiter

A special character assigned to mark the boundary of an Event node.

delimiter declaration field

In the HL7 standard, the location within an Event where a character is to be used as a delimiter. Also refers to the Event Type Definition node boundary it marks.

destination

Pertaining to the primary output Event Type Definition within a Collaboration Rules component or Collaboration Rules script.

e*Gate Monitor

A standard e*Gate component that provides graphical access to e*Gate systems and e*Gate status information, state control, and troubleshooting log files and journals.

e*Way Connection

An e*Way Connection is the encoding of the access information for one particular external connection or SeeBeyond JMS IQ Manager. In terms of content, it is similar to an e*Way configuration file, in defining enough information to be able to “login” or connect to the particular system. However, unlike e*Way configuration files, there is no schedule information. The idea is that the e*Way Connection will be information shared across multiple interfaces.

e*Way Configuration Editor

The graphical user interface (GUI) feature used to configure e*Ways.

e*Way Intelligent Adapter

A component that provides a noninvasive point of contact between an e*Gate system and an external business application (often abbreviated as e*Way). e*Ways establish connectivity with applications, using whatever communication protocol is appropriate. e*Ways perform the following main functions: (1) receiving unprocessed data from external components, transforming it into Events, and forwarding it to other components within e*Gate via Intelligent Queues (IQs); and (2) sending processed data to external components (can also include data transformation).

Enterprise Manager

The e*Gate graphical user interface (GUI) that allows you to create, configure, and modify all components of an e*Gate system.

Event

A unit package of data processed by the e*Gate system. This data has a defined structure, for example, a known number of fields with known characteristics and

delimiters. Events are classified by type (Event Type) and exchanged within e*Gate as Event Type Definitions (ETDs).

Event, delimited

A variable-length Event made up of nodes whose boundaries are marked by delimiters.

Event, fixed

An Event of prescribed length. Each node within a fixed Event Type Definition is identified by its length and location within that Event Type Definition.

Event Linking and Sequencing (ELS)

Event Linking and Sequencing is a feature that allows for Events that arrive from independent input streams to be delivered to subscribers as related units. Complex Linking and Sequencing can be configured using the e*Gate 4.5.2 Java Collaboration Rules Editor, so that n different input streams can be linked and sequenced according to rules based on any combination of content or time-out rules.

Event, monitoring

An Event sent from one e*Gate component to another that describes an internal e*Gate condition, such as “component up” or “component down.”

Event Type

A class of Events with common data structure (for example, a known number of fields, with known characteristics and delimiters). An Event Type is also a logical name entry in e*Gate that points to a single Event Type Definition (ETD).

Event Type Definition (ETD)

A programmatic representation of an Event Type that Collaboration Rules can use when parsing, transforming, or routing data.

Event Type Definition Editors

The graphical user interface (GUI) features used to configure Event Type Definitions (ETDs) in the Java and Monk programming languages; abbreviated as ETD Editor. See also [Event Type Definition](#).

Event Type Definition node

A segment of an Event Type Definition (ETD) that is represented graphically as a node in an Event Type Definition tree in the Event Type Definition Editor window, and represents a portion of an Event.

Event Type Definition tree

The graphical or logical representation of the Event Type Definition and its hierarchy.

Extensible Markup Language (XML)

w3.org defines Extensible Markup Language (XML) as the universal format for structured documents and data on the Web.

external system

A system that sends or receives data and is outside of the e*Gate system.

Guaranteed Exactly Once Delivery (GEOD)

Using XA, GEOD guarantees once and only once delivery. Guaranteed Exactly Once Delivery refers to the usage of XA-compliant e*Gate and external components to ensure the delivery occurs once regardless of failures.

ignore

When a file from the run-time schema, which already carries an advisory lock, is checked out. The advisory lock stays with the original user who checked out the file, and does not transfer to the new user.

instance

A specific node within a series of repeating nodes.

Intelligent Queue (IQ)

A standard e*Gate component that manages the exchange of information between components within the e*Gate system, providing nonvolatile storage for data as it passes from one component to another.

IQ Manager

A standard e*Gate component that reorganizes Intelligent Queues (IQs), archives queue information upon request to save disk space, and locks the queues when maintenance is performed.

IQ Service

A utility that provides the transport of components within Intelligent Queues (IQs), handling the low-level implementation of data exchange, such as system calls to initialize or reorganize a database.

Java Message Service (JMS)

See **SeeBeyond JMS** for the e*Gate implementation of JMS.

log file

A text file that contains a record of all actions taken by an e*Way. Use log files to troubleshoot any problems in the system and discover how to solve them.

Monitor

An executable e*Gate component that enables users to view messages that describe the state of e*Gate internal components. Interactive monitors also enable users to send commands to e*Gate components; non-interactive monitors only enable users to view notifications.

monitoring Event

An Event, sent by one e*Gate component to another (usually to the Control Broker) that describes occurrences within the e*Gate system. Monitoring Events include error messages, such as "component down" or "component lost"; status messages such as "component up" or "contact re-established"; system performance messages, such as "event processing below preset threshold" or "disk space low"; and miscellaneous messages such as scheduled timers, configuration changes, or "event content of interest."

Monk

SeeBeyond's event-processing language.

Monk Test Console

A graphical user interface (GUI) test feature for testing Monk functions and Collaboration scripts before introducing them into the run-time environment.

Navigator Tree

The tree-like graphical display in the Navigator/Components pane of the Enterprise Manager window. This display shows the components of the e*Gate system and how they relate to each other in pictorial form using an icon to represent each component.

node

See [Event Type Definition node](#).

node set

A group of associated nodes that are order-independent, or that repeat.

notification

A notification sent to the user by the e*Gate system.

notification routing

The Collaboration Rules script that specifies how monitoring Events are translated into notifications.

parent nodes

Nodes that are above a given node within the same branch of the Event Type Definition tree.

Participating Host

A client computer that supports an e*Gate system, as opposed to the Registry Host, which acts as a server to the Participating Host.

promoting files

Update the run-time schema to use the new file or files. If the file already exists in the run-time schema, that file is replaced with the file from the Sandbox. Promoting a file automatically removes it from the user's Sandbox and, if the user has locked the file, releases the lock. When you delete a file from the Sandbox without promoting it to the run-time schema, you *remove* the file. If the file was locked, the lock is released.

publish

See [publish/subscribe](#).

publish/subscribe

Abbreviated as pub/sub; subscriber components retrieve Events. Publisher components make Events available to other e*Gate components. See also [Collaboration](#).

Registry

The storage place (in a directory) for all e*Gate configuration details, including file containment.

Registry Service

The service that handles all requests for updates to the e*Gate registry and forwards updated files to clients as necessary.

regular expression

A pattern representing a set of strings to be matched.

removing files

Delete a file from the Sandbox without promoting it to the run-time schema. If the user carried the advisory lock for the file, the lock is released.

root node

The highest-level node in the Event Type Definition tree.

run time

The environment in the Registry shared by all users of that Registry. The run time contains parameters that run for each instance of e*Gate unless the controlling user has a parameter in his or her own Sandbox, in which case the Sandbox is overridden. The run time is the production environment of a schema. See also, **Team Registry**.

Sandbox

A user's local development area. Each user has his own Sandbox. Files in a user's Sandbox are available for testing the functions in the file themselves, but they are not available to the run-time schema. In other words, files within a person's Sandbox are not available to the e*Gate components (such as e*Ways or BOBs) that use them. See also, **Team Registry**.

schema

Includes files and associated stores created by e*Gate, which contain the parameters of all the components that control, route, and transform data as it moves through e*Gate in a predefined system configuration.

SeeBeyond JMS

e*Gate implementation of the Java Message Service (JMS) using IQ Managers, IQs, and a special e*Way Connection.

sibling nodes

Nodes that are children of the same parent node.

source

Pertaining to the primary input Event or Event Type Definition within a Collaboration or Collaboration script.

subnode

A node that is connected through parent-child relationships to another node that is higher in the Event Type Definition tree.

subnode set

A set of order-independent or repeating Event Type Definition nodes one level below the currently selected node in the Event Type Definition tree.

subscribe

See [publish/subscribe](#).

Team Registry

Allows multiple users to develop components of a single schema simultaneously by compartmentalizing the e*Gate Registry into work-in-progress and run-time environment areas, implemented by the Sandbox and run-time environments.

Index

Symbols

- .class files (executable Java bytestream) 262
- .ctl files 262
- .dsc files (database script, Monk database e*Ways) 107
- .ejdb files (e*Gate Java debugging session) 507
- .isc files (Monk Collaboration-ID Editor) 107
- .jar files (Java archives) 164, 176
- .java files (Java source code) 262
- .monk files (used by Monk Collaborations) 107
- .ssc files (Monk ETDs) 204, 216
- .tsc files (transformation script for Monk Collaborations) 107
- .xpr files (project file, Java Collaboration Editor) 262
- .xts files (Java Collaboration Rules Editor) 262
- in package com.stc.common.collabService 599
- Alerter (Java class), methods in
 - eventSend() method 599
- Alerts 491
- Allow remote debugging of JVM 487
- application connectivity 37
- Application Connectivity layer 37
- application program interface (API) 35, 37
- stccmd 490
- Application Programs (AP)
 - in Distributed Transaction Processing (DTP) systems 539
- application-specific e*Ways 39
- architecture 32–33, 34
- archive files 54
- ASCII, converting to and from 616, 617
- asHex()
 - Java method of Standard ETD 552
- Asynchronous Garbage Collection 486
- Attach to JVM (debugger commands) 507
- auto-starting e*Ways 124
- available()
 - Java method of Standard ETD 334
- avoidance of data duplication 37

A

- a2e()
 - Java method of Standard ETD 616
- access control 35
- Access Control List (ACL)
 - (defined) 666
- activating components 498
- adding Business Object Brokers (BOBs) 130
- adding Collaborations
 - basic setup 143
 - overview 142
 - troubleshooting 147
- adding e*Ways
 - basic operation 123
 - Multi-Mode e*Ways 131
 - overview 123
- adding Intelligent Queues (IQs)
 - IQ Managers 137
 - IQ Services 136
- Additional command line arguments text box 454, 473
- Advanced tab
 - Control Broker properties 92
- advisory locks 62
- Alert Agents 499
 - creating 53
- Alerter (Java class)
- Base64 (Java class) 549–551
- Base64 (Java class), methods in
 - base64Decode() 549
- Base64 encoding 549
- base64Decode()
 - Java method of Standard ETD 549
- base64DecodeToByte()
 - Java method of Standard ETD 550
- Base64Utils (Java class) 549–551
- Base64Utils (Java class), methods in
 - base64DecodeToByte() 550
 - byteToBase64String() 550
- Base64Utils class
 - byteToBase64String() method 551
- big-endian 618, 619
- block (Business Rule) 291
- BOBs. *See* Business Object Brokers
- Break (debugger command) 508
- Break on Exception (debugger command) 508, 515
- breakpoints
 - clearing 510
 - enabling/disabling 510
 - setting or modifying 510, 513, 514
- business analysis and preparations 43
- business logic 36
- Business Object Brokers (BOBs)
 - (defined) 666

B

- Base64 (Java class) 549–551
- Base64 (Java class), methods in
 - base64Decode() 549
- Base64 encoding 549
- base64Decode()
 - Java method of Standard ETD 549
- base64DecodeToByte()
 - Java method of Standard ETD 550
- Base64Utils (Java class) 549–551
- Base64Utils (Java class), methods in
 - base64DecodeToByte() 550
 - byteToBase64String() 550
- Base64Utils class
 - byteToBase64String() method 551
- big-endian 618, 619
- block (Business Rule) 291
- BOBs. *See* Business Object Brokers
- Break (debugger command) 508
- Break on Exception (debugger command) 508, 515
- breakpoints
 - clearing 510
 - enabling/disabling 510
 - setting or modifying 510, 513, 514
- business analysis and preparations 43
- business logic 36
- Business Object Brokers (BOBs)
 - (defined) 666

- as components under a CB 126
- as data flow controllers 83
- command-line options 454, 473
- configuring 130
- creating and configuring 53, 130, 462
- deprecation of 40
- interaction with Sandbox 61
- moving from one component to another 55
- overview 39, 123
- relationship with IQs 136
- system files for 127
- where to find (in components tree) 51
- Business Rules** 476
 - block 291
 - case 315
 - catch (with try) 321
 - default 315
 - else (with if) 306
 - finally (with try) 321
 - for 304, 305
 - if, then, else 306
 - method 310
 - return 313
 - switch 315
 - then 316
 - then (with if) 306
 - throw 317
 - try, catch, finally 321
 - variable 324
 - while 325
- Business Rules and Data Processing layer 36
- Business Rules pane**
 - described 266
 - illustrated 265
- Business Rules toolbar**
 - (illustrated) 265
 - Java Collaboration Rules Editor 269
- byte offsets 238
- byteToBase64String()
 - Java method of Standard ETD 550, 551
- C**
 - case (Business Rule) 315
 - catch (Business Rule, with try) 321
 - CBs.*See* Control Brokers
 - channels
 - available Notification 499
 - character encoding
 - discussed 580
 - example 583
 - methods for 385, 584, 585, 586, 587, 588
 - character encodings, methods for
 - MIME Base64 549
- child nodes 158
- Choose a method (dialog box)
 - accessing 284
 - illustrated 286
- circular dependencies, in ETD templates 180
- .class files (executable Java bytestream) 262
- Class Garbage Collection 486
- classes, Java
 - Alerter 599
 - Base64 549–551
 - Base64Utils 549–551
 - CollabUtils 552–558
 - com.stc.common.collabService.JSubCollabMapI
 nfo 603, 604, 605, 606, 607, 608, 609, 610, 611,
 612, 613, 614
 - com.stc.common.utils.Qsort 627
 - DateUtils 559–561
 - EGate 562–569
 - FileUtils 570–572
 - JCollabController 573–598
 - character encoding 580
 - JCollaboration 599–602
 - JSubCollabMapInfo 603–615
 - Mainframe 616–620
 - MapUtils 621–626
 - ScEncrypt 628–629
 - STCTypeConverter 630–655
 - StringUtils 656–662
- CLASSPATH Override 468, 484
- CLASSPATH Prepend 467, 483
- codes
 - for text formatting 663–665
- collabDebug()
 - Java method of Standard ETD 562
- collabError()
 - Java method of Standard ETD 563
- collabFatal()
 - Java method of Standard ETD 563
- collabInfo()
 - Java method of Standard ETD 564
- Collaboration 36, 37, 38, 44, 45, 47, 56, 59, 84, 85, 106, 107, 109, 123, 124, 136, 143, 145, 146, 148, 254, 450, 456, 464, 476
 - (defined) 667
 - as necessary component of an e*Way 85
- Collaboration .class files (Java) 106
- Collaboration Editor 108
- Collaboration Rule 27, 36, 37, 106, 107, 109, 116, 117, 118, 121, 122, 144, 148, 448, 456
- Collaboration Rules
 - (defined) 667
 - (folder in Components view) 51
 - as necessary components of Collaborations 142
 - assigning to Collaborations 145

- characteristics of 36
- correspondence with Collaborations 143
- creating 53
- Collaboration Rules Editor 36, 52, 56, 73, 107, 119, 120, 352–447
- Collaboration Rules script 56, 85, 106, 107, 109, 118, 120, 148, 149
- Collaboration Rules scripts (Monk) 106
- Collaboration Rules, Monk
 - creating 116
- Collaboration script 85, 106, 107, 119, 457
- Collaboration Service 36, 51, 109, 110, 117, 144
- Collaboration Service types
 - C 109
 - Java 109
 - Monk 109
 - Monk ID 109
 - Pass Through 109
 - Route Table 109
- Collaboration Services
 - creating 53
- Collaboration-ID Rules Editor 52, 56, 107, 123
- Collaborations
 - adding 142, 143
 - configuring 144
 - configuring in publication order 143
 - correspondence with Collaboration Rules 143
 - creating 53
 - Java file types 108
 - moving from one component to another 55
 - subscription properties of 145
 - where to find (in components tree) 51
- collabTrace()
 - Java method of Standard ETD 565
- CollabUtils (Java class) 552–558
 - in package com.stc.common.collabService 562
 - in package com.stc.eways.util 552
- CollabUtils class
 - asHex() method 552
 - doOffsetTrunc() method 553
 - isMonkDatePattern() method 554
 - sprintf() method 555
 - toHex() method 556
 - toJavaDatePattern() method 556
 - uniqueID() method 557
 - uniqueId() method 557
- collabWarning()
 - Java method of Standard ETD 565
- com.stc.common.collabService package
 - EGate class 562
 - JCollabController class 573
 - JSubCollabMapInfo class 603, 604, 605, 606, 607, 608, 609, 610, 611, 612, 613, 614
- com.stc.common.utils
 - ScEncrypt class 628
 - STCTypeConverter class 630
- com.stc.common.utils package
 - Qsort class 627
- com.stc.eways.util package 552
 - Base64Utils class 549
 - DateUtils class 559
 - FileUtils class 570
 - JSubCollabMapInfo class 603
 - Mainframe class 616
 - MapUtils class 621
 - StringUtils class 656
- com.stc.jcsre package
 - Base64 class 549
 - JCollaboration class 599
- com.stc.jcsre.JCollaboration 283
- command line 95, 121, 129, 147, 245
 - adding new parameters 453, 472
- command-line arguments
 - rp %_PORT% 454, 473
 - rp %_REGPORT% 473
- command-line options
 - adding in BOB 454, 473
 - adding in IQ Manager 454, 473
 - changing in BOB 454, 473
 - changing in IQ Managers 454, 473
- command-line utilities
 - stccmd 490
- committing files 61, 63
 - in Team Registry 62
- Compile pane, Java Collaboration Rules Editor
 - (illustrated) 265
 - described 266
- compiling
 - ETDs 182
- compiling Java-enabled ETDs 102
- components
 - activating 498
 - displaying status 497, 498
 - shutting down 497
 - starting 496
 - suspending 498
 - version, displaying 498
- components tree 52
 - illustrated 51
 - maneuvering upward through 52
 - of Enterprise Manager 51
- configuration files, e*Way 449, 453
- configuration files, Multi-Mode e*Ways 465
- configuring Multi-Mode e*Ways with e*Way Connections 477
- Context pane
 - in e*Gate Java Debugger 510
- Control 491

- Control Broker 36, 125, 126, 455, 474, 476, 494
(defined) 668
interaction with e*Gate Monitor 489
relationship with e*Gate Monitor 491
Control Broker properties
(listed) 87
Advanced 92
General 88
Notification Setup 88
Security 93
Timers 90
Control Brokers (CBs)
and system management 127
as a data-management feature 86
as control layer components 35
configuring 87
creating 53
function of 36
moving components into or from 55
options for starting/stopping e*Ways 454
where to find (in components tree) 51
Control layer 35
conventions, writing in document 28
conversion, methods for
Also see STCTypeConverter
converting endian-ness, methods for 618, 619
converting strings, methods for 617
copy (Business Rule),Business Rules
copy 294
copyProperties()
Java method of Standard ETD 575
CP930 character encoding method 586, 587
createSubCollabMapInfo()
Java method of Standard ETD 575, 578
creating a schema 86
creating a system design
information gathering 44
overview 44
system structure 45
creating Collaboration Rules and scripts
Collaboration Rule properties 109
Collaboration Rules Editor 107
Collaboration Rules scripts 107
Collaboration Services and types 109
configuring Collaboration Rules 117
initializing Collaboration Rules 117
Java and C language scripts 107
listing services 109
Monk language scripts 107
overview 106
using Collaboration scripts 107
creating Event Types and ETDs
assigning definitions to Event Types 105
creating Event Type Definitions (ETDs) 103
creating Event Types 96
overview 95
creating external templates 180
creating internal templates 180
creating Java-enabled ETDs 99
.ctl files 262
CTRL key
to drag a node before another node 178, 181
custom e*Ways 39
- ## D
- data duplication, avoidance of 37
data flows
gathering information on 44
data interactions
gathering information on 44
data type conversion, methods for
See STCTypeConverter
database access e*Ways 39
databases 37, 39
datamap (Business Rule),Business Rules
datamap 296
DateUtils (Java class) 559–561
in package com.stc.eways.util 559
DateUtils class
format() method 559
timeStamp() method 560
transformDate() method 560
decoding and encoding strings 549, 550, 551
decrypt()
Java method of Standard ETD 628
default (Business Rule) 315
default (Business Rule),Business Rules
default 299
default repository (registry) files
exporting 54
default.txt file
editing 454, 473
deleting templates 181
delimited ETDs
adding nodes 228
creating root nodes 222
defining default delimiters 217
definition 215
delimiter syntax 222
Destination Events pane
Java Collaboration Rules Editor 265, 266
Detach (debugger commands) 507
Disable JIT 471, 486
disk space warnings 500
displaying component version information 498
displaying system status 497, 498
Distributed Transaction Processing (DTP) 539

- document
 - conventions 29
 - doMap()
 - Java method of Standard ETD 622
 - doOffsetTrunc()
 - Java method of Standard ETD 553
 - DOS device name
 - ineligibility as node name 174
 - double-byte characters 581
 - in ETD node names 174
 - .dsc files (database script, Monk database e*Ways) 107
- E**
- e*Gate Alert Agent 499
 - e*Gate editors, overview 34
 - e*Gate Java Debugger
 - activating 509
 - clearing breakpoints 510
 - Context pane 510
 - main menu commands 507
 - opening a Java Collaboration 512
 - overview 505
 - setting breakpoints 510
 - setting or modifying breakpoints 513, 514
 - trapping exceptions 515
 - using 510
 - viewing variables 511
 - e*Gate Java Debugger, commands
 - Attach to JVM 507
 - Break 508
 - Break on Exception 508
 - Detach 507
 - Go 508
 - Resume Session 507
 - Save Session 507
 - Step Into 508
 - Step Out 508
 - Step Over 508
 - Stop in Class 508
 - Stop in Method 508
 - e*Gate layers 33
 - Application Connectivity 37
 - Business Rules and Data Processing 36
 - Control 35
 - Intelligent Queuing 37
 - View 34
 - e*Gate Monitor 35, 494
 - (illustrated) 491
 - overview 489
 - relationship with stccmd 490
 - reliance on Control Broker 489, 491
 - e*Gate Monitor operation
 - accessing 490
 - exiting 494
 - menu bar 493
 - toolbar buttons 492
 - window parts 491
 - e*Gate SNMP Agent 500
 - e*Gate system setup, general overview 40
 - e*Gate system, overview 32
 - e*Insight Business Process Manager Module 487
 - e*Insight engines
 - as components under a CB 126
 - e*Way
 - (defined) 668
 - e*Way configuration
 - activating or modifying logging options 456
 - activating or modifying monitoring thresholds 456
 - changing "run as" user name 454
 - changing command-line parameters 453
 - creating or selecting configuration file 453
 - defining components 450
 - modifying properties 451
 - selecting executable file 452
 - setting startup options or schedules 454
 - e*Way Configuration Editor
 - (illustrated) 458
 - controls for 458
 - controls, described 458
 - creating and configuring BOBs 462
 - entering user notes 461
 - modifying configuration settings 461
 - navigating through 460
 - online Help for 462
 - online Help system for 462
 - overview 457
 - parameter configuration controls 459, 479
 - promoting configuration file to run time 461
 - restoring default settings 461
 - restoring saved settings 461
 - saving configuration settings 460
 - section and parameter controls 459
 - e*Way Connection 83, 84, 123, 131, 134, 135
 - (defined) 668
 - e*Way Connection Editor operation
 - entering user notes 481
 - modifying configuration settings 481
 - navigating through 480
 - promoting configuration file to run time 481
 - restoring default settings 481
 - restoring saved settings 481
 - saving configuration settings 480
 - section and parameter controls 479
 - e*Way Connections
 - (folder in Components view) 51

- for JMS 274
 - used with Multi-Mode e*Ways 464
- XA-compliant 540
- e*Way Editor 452, 458
- e*Way operation
 - Enterprise Manager 449
 - functional components 449
 - overview 448
- e*Way properties, modifying 451
- e*Ways
 - and Collaborations 457
 - and connectivity 37
 - as components under a CB 126
 - as named components of a schema 45
 - configuring 128
 - creating and configuring 53, 125, 127
 - gathering information on external systems 44
 - interaction with Sandbox 61
 - moving from one component to another 55
 - purpose of 37
 - starting 124
 - system files for 127
 - where to find (in components tree) 51
- e*Ways, types of
 - application-specific 39
 - custom 39
 - Generic e*Way Extension Kit 39
 - SAP 39
- e2a()
 - Java method of Standard ETD 617
- e2S()
 - Java method of Standard ETD 617
- EBCDIC, converting to and from 616, 617
- editing files 54, 57
 - in Sandbox 63
 - in Team Registry 63
- EGate (Java class) 562–569
- EGate (Java class), methods in
 - collabDebug() 562
 - collabError() 563
 - collabFatal() 563
 - collabInfo() 564
 - collabTrace() 565
 - collabWarning() 565
 - traceln() 566
- EGate class
 - collabDebug() method 562
 - collabError() method 563
 - collabFatal() method 563
 - collabInfo() method 564
 - collabTrace() method 565
 - collabWarning() method 565
 - traceln() method 566
- eInsight Engines
- creating 53
- .ejdb files (e*Gate Java debugging session) 507
- ELS methods
 - and jCollabController prefix 589
 - flushAllLinkIdentifiers() 590
 - getCurrentLinkIdentifier() 590
 - getELSExpiration() 591
 - getLinkIdentifiers() 592
 - getNumberOfMessagesForInstance() 593
 - getNumberOfMessages() 593
 - IsCurrentELSExpired() 595
 - isFlushMode() 596
- ELS placeholder methods
 - isLinkingComplete() 597
 - onExpire() 597
 - retrieveLinkIdentifier() 598
- ELSController interface
 - flushAllLinkIdentifiers() method 590
 - getCurrentLinkIdentifier() method 590
 - getELSExpiration() method 591
 - getLinkIdentifiers() method 592
 - getNumberOfMessagesForInstance() method 593
 - getNumberOfMessages() method 593
 - hasHappened() method 594
 - IsCurrentELSExpired() method 595
 - isFlushMode() method 596
 - isLinkIdentifierExists() method 596
 - setELSExpiration() method 598
- else (Business Rule, with if) 306
- empty()
 - Java method of Standard ETD 656
- encoding and decoding strings 549, 550, 551
- encoding methods
 - discussed 580
 - example 583
 - getIncomingEncoding() 584
 - getMarshalEncoding() 584
 - getOutgoingEncoding() 585
 - getUnmarshalEncoding() 585
 - in Monk 385
 - setIncomingEncoding() 586
 - setMarshalEncoding() 587
 - setOutgoingEncoding() 587
 - setUnmarshalEncoding() 588
- encodings, methods for
 - MIME Base64 549
- encrypt()
 - Java method of Standard ETD 628
- endian-ness, swapping 618, 619
- Enterprise Manager basic operation
 - accessing 49
 - Components Tree 51
 - editor pane 50
 - exiting 59

- File > Edit File option 57
- File > New menu 53
- Menu bar 53
- navigator pane 50
 - navigator pane, Components view 50
 - navigator pane, Network view 50
- Sandbox and run time 57
- toolbar buttons 52
 - View > Summary menu 58
- window areas 50
- Enterprise Manager feature
 - overview 34
 - using 49
- escape character, for delimited ETDs 221
- ETD
 - (defined) 95
 - (discussed) 95
 - properties 191–203
 - specifying node information 205
- ETD Editor feature
 - function in setup 103
 - overview 204
- ETD Editor window
 - accessing 209
- ETD nodes
 - creation 205
 - naming 174, 206
 - parents and children 205
 - structure 205
 - using 205
- ETD operations
 - changing node details 253
 - creating comments 250
 - deleting ETDs 253
 - editing files 250
 - extracting input delimiters 247
 - finding nodes 250
 - modifying internal templates 253
 - moving nodes 251
 - opening 242
 - pruning 252
 - saving ETDs under new names 246
 - testing files 247
 - using cut, copy, paste 251
 - using the Build tool 242
- ETD templates
 - external, breaking links 257
 - external, changing repetition properties 256
 - external, including 255
 - external, overview 254
 - internal, changing repetition properties 258
 - internal, converting to 258
 - internal, creating 257
 - internal, overview 257
- internal, referencing 258
- ETD Tester (Java) 183
- ETD Tree 204
- ETDs
 - compiling 102, 182
 - creating 99
 - defining specific Event Types 158
 - explained 158
 - Java properties of 176
 - opening 102
 - promoting 102
 - testing 183
- ETDs, Monk
 - editing 204
- EUC character encoding methods
 - in Monk 385
- EUC-JP character encoding method 586, 587
- Event
 - (defined) 668
 - Event Type 44, 47, 56, 59, 84, 95, 96, 103, 104, 105, 106, 107, 109, 121, 122, 139, 147, 148, 158, 159, 163, 204, 205, 207, 215, 216, 222, 232
 - (defined) 669
- Event Type Definition (ETD)
 - (defined) 669
 - (discussed) 95
 - defined 95
 - See* ETD
- Event Type Definition (ETD) Editor 52, 56, 99, 103, 104, 105, 204–259
 - Java 27, 159–196
 - Monk 27
- Event Type Definition (ETD) information 44, 85, 174, 205, 206, 207
- Event Type Definition (ETD) node. *See* node.
- Event Type Definition (ETD) pane 102
- Event Type properties 163, 207
- Event Types
 - (folder in Components view) 51
 - as entities received and processed by Collaborations 142
 - as named components of a schema 45
 - creating 53
 - defined 95
 - priority 146
- Events
 - defined 95
 - mapping 37
- eventSend()
 - Java method of Standard ETD 599
- event-threshold warnings 502
- exceptions, trapping 515
- executable files, e*Way 449, 452
- exporting files 54

external system **38, 42, 43, 48, 83, 84, 104, 109, 123, 127, 143, 452**
publishing or subscribing to **85**

external systems

and Multi-Mode e*Ways **464**
gathering information on **44**

external templates

converting .ssc files containing **167**
creating **180**
deleting **181**
renaming **180**

External Templates pane

shortcut menu for **180**

F

F11 key
(in debugger) Step Into **508**

F5 key
(in debugger) Go **508**

F7 key
(in debugger) Step Over **508**

F9 key
(in debugger) Step Out **508**

file types

.class (executable Java bytestream) **262**
.ctl **262**
.dsc (database script, Monk database e*Ways)
107
.ejdb (e*Gate Java debugging session) **507**
.isc (Monk Collaboration-ID Editor) **107**
.java (Java source code) **262**
.monk (text file used by Monk Collaborations)
107
.ssc (Monk ETDs) **204, 216**
.tsc (transformation script for Monk
Collaborations) **107**
.xpr (project file, Java Collaboration Editor) **262**
.xts (Java Collaboration Rules Editor) **262**

files

committing in Team Registry **62**
editing in Team Registry **63**
promoting in Team Registry **63**
removing from Sandbox **62**
removing in Team Registry **64**
saving in Team Registry **63**

files, e*Way **449**

files, methods for reading and writing **570, 571**

FileUtils (Java class) **570–572**

in package com.stc.common.collabService **570**

FileUtils (Java class), methods in

readBytes() **570**
readString() **570**
write() **571**

finally (Business Rule, with try) **321**

fixed ETDs

adding nodes **235**
creating root nodes **232**
definition **215**

flags for stcinstd **95**

flushAllLinkIdentifiers()

ELS method **590**

for (Business Rule)
illustrated **304, 305**

Format Specification **662**

format()

Java method of Standard ETD **559**

formatting, codes for **663–665**

function definitions, e*Way **449**

G

gaiji conversion

in Monk **385**

Garbage Collection activity reporting **486**

Generic e*Way Extension Kit **39**

getCallingCollaboration()

Java method of Standard ETD **603**

getClassFullPath()

Java method of Standard ETD **604**

getClassName()

Java method of Standard ETD **605**

getCollaborationName()

Java method of Standard ETD **576**

getCtlFileFullPath()

Java method of Standard ETD **605**

getCtlFileName()

Java method of Standard ETD **606**

getCurrentLinkIdentifier()

ELS method **590**

getEgateBaseDirectory()

Java method of Standard ETD **576**

getELSExpiration()

ELS method **591**

getEventTypeDefinition()

Java method of Standard ETD **607**

getEventTypeDefinitionPath()

Java method of Standard ETD **607**

getIncomingEncoding()

Java method of Standard ETD **584**

getInputData()

Java method of Standard ETD **608**

getInputTopicName()

Java method of Standard ETD **609**

getLinkIdentifiers()

ELS method **592**

getMarshalEncoding()

Java method of Standard ETD **584**

getModuleName()
 Java method of Standard ETD 577
getNumberOfMessagesForInstance()
 ELS method 593
getNumberOfMessages()
 ELS method 593
getOutgoingEncoding()
 Java method of Standard ETD 585
getOutputData()
 Java method of Standard ETD 609
getParentReferenceETD()
 Java method of Standard ETD 610
getParentReferenceInstanceName()
 Java method of Standard ETD 611
getPropertyNames()
 Java method of Standard ETD 577
getRuleName()
 Java method of Standard ETD 611
getting started
 add-ons and client software 42
 overview 42
getUnmarshalEncoding()
 Java method of Standard ETD 585
Go (debugger command) 508

H

hasHappened()
 ELS method,ELS methods
 hasHappened() 594
HL7 repeating fields 231

I

if, then, else (Business Rule) 306
importing files 54
In/Out Event instances
 and JMS e*Way Connections 274
Initial Heap Size 469, 484
Insert Java Function (shortcut menu command) 284
 cautionary note 284
Intelligent Queue
 Managers. *See* IQ Managers
 Services. *See* IQ Services
Intelligent Queue (IQ) 37, 47, 83, 84, 110, 124, 125, 137, 138, 139, 140, 141, 142, 143, 147, 148, 456, 457, 464, 476
 defined by IQ Service 138
 published to by an inbound Collaboration 85
Intelligent Queues (IQs)
 purpose and function 37
 and BOBs 40
 as named components of a schema 45
 configuring 138

creating 53, 137
moving from one component to another 55
relationship to IQ Manager 86
starting 142
where to find (in components tree) 51
Intelligent Queueing layer 37
intended audience, user's guide 26
interactive monitoring 489
interactive monitoring of e*Gate 490
interactive monitors 34, 35
internal templates 180, 181
 deleting 181
 properties
 as a root node 180
 as instance 181
Internal Templates pane
 shortcut menu for 180, 181
IQ
 (defined) 670
IQ Administrator
 toolbar button for 492
IQ Manager 37, 125, 136, 137, 138, 142, 148
 (defined) 670
 adding command-line options 454, 473
 changing command-line options 454, 473
IQ Managers
 as a data-management feature 86
 as components under a CB 126
 creating 53
 moving from one component to another 55
 where to find (in components tree) 51
IQ Services 136
 creating 53
 defined 37
 documentation for 110
 listed in Enterprise Manager 110
 purpose 136
 specifying 138
 where to find (in components tree) 51
IQs
 adding 136
 creating 53
 moving from one component to another 55
 overview 136
 starting 142
 tips for setting up 136
.isc files (Monk Collaboration-ID Editor) 107
IsCurrentELSExpired()
 ELS method 595
isFlushMode()
 ELS method 596
isLinkIdentifierExists()
 ELS method,ELS methods
 isLinkIdentifierExists() 596

- isLinkingComplete()
 - ELS placeholder method 597
- isManualPublish()
 - Java method of Standard ETD 612
- isMonkDatePattern()
 - Java method of Standard ETD 554
- ISO-2022-JP character encoding method 586, 587
- isPublisher()
 - Java method of Standard ETD 613
- isSubCollaboration()
 - Java method of Standard ETD 579
- isTrigger()
 - Java method of Standard ETD 613
- J**
- Japanese character encoding 581
- Java 108
 - Collaboration file types 108
 - Event Type Definition (ETD) Editor 27
- Java classes
 - CollabUtils 552
 - com.stc.common.collabService.JSubCollabMapInfo 603, 604, 605, 606, 607, 608, 609, 610, 611, 612, 613, 614
 - com.stc.common.utils.Qsort 627
 - DateUtils 559
 - EGate 562
 - FileUtils 570
 - JCollabController
 - character encoding 580
 - JCollaboration 599
 - JSubCollabMapInfo 603
- Java Collaboration Rules Editor
 - (illustrated) 265
 - adding custom rules 115
- Java ETD Editor 159–203
- Java Event Type Definition (ETD)
 - builder wizards 98
- Java Event Type Definition (ETD) Editor 98, 99
- Java Event Type Definition wizard
 - using 100
- .java files (Java source code) 262
- Java letter-or-digit
 - in node names 174
- Java letters
 - in node names 174
- Java Message Service. *See* SeeBeyond JMS
- Java packages
 - com.stc.common.collabService 603, 604, 605, 606, 607, 608, 609, 610, 611, 612, 613, 614
 - com.stc.common.utils 627
 - com.stc.jcsre.JCollaboration 283
- Java properties, of ETDs 176
- Java Virtual Machine (JVM) settings 466, 482
- Java-enabled ETD
 - properties 191–203
- Java-enabled ETDs
 - compiling 102, 182
 - creating 99
 - explained 158
 - opening 102
 - promoting 102
- jCollabController
 - prefix for all ELS methods 589
- JCollabController (Java class) 573–598
 - in com.stc.common.collabService package 573
 - setUnmarshalEncoding() 588
- JCollabController (Java class), methods in
 - (for character encoding) 580
 - (for ELS) 589
 - copyProperties() 575
 - createSubCollabMapInfo() 575
 - flushAllLinkIdentifiers() 590
 - getCollaborationName() 576
 - getCurrentLinkIdentifier() 590
 - getEgateBaseDirectory() 576
 - getELSExpiration() 591
 - getIncomingEncoding() method 584
 - getLinkIdentifiers() 592
 - getMarshalEncoding() method 584
 - getModuleName() 577
 - getNoOfMessages() 593
 - getNoOfMessagesForInstance() 593
 - getOutgoingEncoding() 585
 - getPropertyNames() 577
 - getUnmarshalEncoding() 585
 - hasHappened() 594
 - invoke() 578
 - isCurrentELSExpired() 595
 - isELSExpired() 595
 - isFlushMode() 596
 - isLinkIdentifierExists() 596
 - isLinkingComplete() 597
 - isSubCollaboration() 579
 - onExpire() 597
 - retrieveLinkIdentifier() 598
 - retrieveRegistryFile() 580
 - setELSExpiration() 598
 - setIncomingEncoding() 586
 - setMarshalEncoding() 587
 - setOutgoingEncoding() 587
 - setUnmarshalEncoding() 588
- JCollaboration (Java class) 599–602
 - in package com.stc.jcsre 599
- JCollaboration (Java class), methods in
 - eventSend() method 599
- JIS character encoding methods

in Monk 385
JMS Administrator
toolbar button for 492
JMS e*Way Connections 339, 347
and In/Out Event instances 274
JMS IQs 37, 83, 123, 136
JNI DLL absolute pathname 483
JNI DLL names
for various operating systems 467, 483
JSubCollabMapInfo (Java class) 603–615
in package com.stc.common.collabService 603
JSubCollabMapInfo class
getCallingCollaboration() method 603
getClassFullPath() method 604
getClassName() method 605
getCtlFileFullPath() method 605
getCtlFileName() method 606
getEventTypeDefinition() method 607
getEventTypeDefinitionPath() method 607
getInputData() method 608
getInputTopicName() method 609
getOutputData() method 609
getParentReferenceETD() method 610
getParentReferenceInstanceName() method 611
getRuleName() method 611
isManualPublish() method 612
isPublisher() method 613
isTrigger() method 613
setInstanceMap() method 614
JVM settings 466, 482

K

ksc-5601 character encoding method 586, 587, 588

L

labels on buttons 56
leaf nodes 158
library files, e*Way 449
Link Identifiers, ELS 592
list lookup (Business Rule), Business Rules
list lookup 307
little-endian 618, 619

M

Mainframe (Java class) 616–620
in package com.stc.eways.util 616
Mainframe (Java class), methods in
a2e() 616
e2a() 617
e2S() 617

swapInt() 618
swapLong() 619
swapShort() 619
mapping Events 37
Mapping pane, Java Collaboration Rules Editor
(illustrated) 265
described 266
MapUtils (Java class) 621–626
in package com.stc.eways.util 621
readMap() 624
renderMap() 625
writeMap() 626
MapUtils (Java class), methods in
doMap() 622
parseMap() method 623
MapUtils class
parseMap() method 623
marshal()
Java method of Standard ETD 335
Maximum Heap Size 470, 485
Maximum Stack size for JVM threads 486
Maximum Stack size for native threads 485
menu bar, Java Collaboration Rules Editor
(illustrated) 265
method (Business Rule) 310
Method Browser
accessing 284
methods of Standard ETD
a2e() 616
asHex() 552
available() 334
base64Decode() 549
base64DecodeToByte() 550
byteToBase64String() 550, 551
collabDebug() 562
collabError() 563
collabFatal() 563
collabInfo() 564
collabTrace() 565
collabWarning() 565
copyProperties() 575
createSubCollabMapInfo() 575, 578
decrypt() 628
doMap() 622
doOffsetTrunc() 553
e2a() 617
e2S() 617
empty() 656
encrypt() 628
eventSend() 599
format() 559
getCallingCollaboration() 603
getClassFullPath() 604
getClassName() 605

getCollaborationName() 576
 getCtlFileFullPath() 605
 getCtlFileName() 606
 getEgateBaseDirectory() 576
 getEventTypeDefinition() 607
 getEventTypeDefinitionPath() 607
 getIncomingEncoding() 584
 getInputData() 608
 getInputTopicName() 609
 getMarshalEncoding() 584
 getModuleName() 577
 getOutgoingEncoding() 585
 getOutputData() 609
 getParentReferenceETD() 610
 getParentReferenceInstanceName() 611
 getPropertyNames() 577
 getRuleName() 611
 getUnmarshalEncoding() 585
 isManualPublish() 612
 isMonkDatePattern() 554
 isPublisher() 613
 isSubCollaboration() 579
 isTrigger() 613
 marshal() 335
 next() 336
 padCenter() 657
 padLeft() 658
 padRight() 659
 parseMap() 623
 publications() 337
 qsort() 627
 rawInput() 338
 readBytes() 570
 readMap() 624
 readProperty() 339
 readString() 570
 receive() 341
 renderMap() 625
 reset() 342
 retrieveRegistryFile() 580
 send() 343
 setIncomingEncoding() 586
 setInstanceMap() 614
 setMarshalEncoding() 587
 setOutgoingEncoding() 587
 setUnmarshalEncoding() 588
 sprintf() 555
 subscriptions() 344
 swapInt() 618
 swapLong() 619
 swapShort() 619
 timeStamp() 560
 toBoolean() 632
 toBooleanPrimitive() 631
 toByte() 635
 toByteArray() 636
 toBytePrimitive() 634
 toCharacter() 639
 toCharPrimitive() 638
 toDouble() 642
 toDoublePrimitive() 640
 toFloat() 644
 toFloatPrimitive() 643
 toHex() 556
 toInteger() 647
 toIntegerPrimitive() 646
 toJavaDatePattern() 556
 toLong() 650
 toLongPrimitive() 648
 topic() 345
 toShort() 652
 toShortPrimitive() 651
 toString() 653
 traceIn() 566
 transformDate() 560
 trimBoth() 660
 trimLeft() 660
 trimRight() 661
 uniqueID() 557
 uniqueId() 557
 unmarshal() 346
 write() 571
 writeMap() 626
 writeProperty() 347
 methods, browsing 284
 MIME
 Base64 character encoding 549
 modifying breakpoints 513, 514
 Monitor 27
 monitor messages 491
 monitoring features, overview 35
 monitors
 non-interactive 499
 setting disk-usage thresholds 500
 setting event-processing thresholds 502
 Monk 107, 109, 117, 118, 119, 120, 148, 149, 152, 154, 155, 156, 204, 221, 224, 226, 448, 449, 450
 Event Type Definition (ETD) Editor 27
 Monk Collaboration Rules
 creating 116
 Monk Collaboration Rules Editor 119
 .monk files (used by Monk Collaborations) 107
 Monk Test Console 52, 56, 119, 148, 149–156
 multibyte characters
 in ETD node names 174
 multibyte operating systems 174
 Multi-Mode e*Way configuration
 activating or modifying logging options 476

activating or modifying monitoring thresholds 476
changing "run as" user name 473
changing command-line parameters 472
Multi-Mode e*Ways 132
before creating 131
characteristics 464
configuring 465
creating and configuring 131
defined and described 464
overview 131
properties (illustrated) 132
Multipurpose Internet Mail Extension.
See MIME

N

names
for nodes in ETDs 164
of Java packages 164
naming conventions
for nodes 174
for package names 174
naming system elements
conventions, data files 47
conventions, other system entries 47
conventions, system components 46
making checklists 45
overview 45
navigator tree (left-hand pane)
illustrated 51
levels, explained 126
maneuvering upward through 52
of e*Gate Monitor window 491
new ETD files, creating 215
next()
Java method of Standard ETD 336
node 163, 174, 205, 206, 207, 215, 217, 221, 222, 223, 224, 225, 226, 227, 228, 229, 230, 231, 232, 235, 236, 237, 239, 252, 253, 256
node names
guidelines for 174
node sets
adding 239
adding node subsets 241
overview 239
nodes
(terminology) 158
naming 164
nodes in Java ETDs
(terminology) 158
adding 177
as aliases for templates 180
deleting 178

expanding/collapsing 177
moving 177
names for 164
naming 174
renaming 178
repeating 178, 179, 183
nodes in Monk ETDs 257
acceptable characters for names 206
attaching external templates to 256
converting to internal template 258
cutting, copying, pasting 252
cutting, pasting, copying 252
finding 250
fixed or variable 236
modifying properties 259
moving 251
naming 233
node sets 239
properties of 240
root 234
subset 241
variable repetition options for 230
non-interactive monitoring 489
Notification channels 499
Notification Setup tab
Control Broker properties 88

O

onExpire() ELS placeholder method 597
online Help systems
accessing 74
books and topics 78
Contents Tree 76
entry and exit operation 76
exiting 80
GUI features 76
hypertext links 74
Index operation 78
overview 73
overview of features 77
printing 78
search operation 78
tab features 77
tab operation 77
toolbar buttons 78
using 73
window parts 75
opening ETDs 102
operating systems
various JNI DLL names 467, 483
organization of information, document 27

P

packages

- com.stc.common.utils 630
- com.stc.eways.util 549, 552
- com.stc.jcsre 549

naming conventions for 174

packages, Java

- com.stc.common.collabService 603, 604, 605, 606, 607, 608, 609, 610, 611, 612, 613, 614

- com.stc.common.utils 627

- com.stc.jcsre.JCollaboration 283

padCenter()

Java method of Standard ETD 657

padding strings, methods for 657, 658, 659

padLeft()

Java method of Standard ETD 658

padRight()

Java method of Standard ETD 659

Palette

text labels on 56

panes

in Java ETD Editor 98, 99

parameters

adding to command line 453, 472

parent nodes 158

parseMap()

Java method of Standard ETD 623

Participating Host 67, 87, 125, 126, 131, 141

(defined) 671

moving components into or from 55

relationship to schema and Control Broker (CB)

86

Participating Hosts

(folder in Components view) 51

adding new 67

creating 53

editing properties 68

naming 67

PassThrough

class (Java) 109, 113

Service 109

priority

of Event Types 146

privileges

assigning 94

changing 94

removing 94

product architecture 32–33, 34

promote 62, 63

promoting files 61

in Team Registry 63

promoting Java-enabled ETDs 102

properties of standard ETD

Event Type instance 191

Field (leaf) nodes 197

parent (Element) nodes 192, 201

Properties pane, Java Collaboration Rules Editor

(illustrated) 265

described 266

pub/sub 37, 86, 142, 145

publications()

Java method of Standard ETD 337

publish 37, 59, 85, 122, 123, 124, 139, 147, 456, 457,

464, 476

publishers

priority (of Event Types) 146

publishing and subscribing 85, 86, 142, 145

Q

Qsort interface

qsort() method 627

qsort()

Java method of Standard ETD 627

R

rawInput()

Java method of Standard ETD 338

readBytes()

Java method of Standard ETD 570

readMap()

Java method of Standard ETD 624

readProperty()

Java method of Standard ETD 339

readString()

Java method of Standard ETD 570

receive()

Java method of Standard ETD 341

Registry 56, 57, 86, 87, 153, 154, 155, 452, 476

(defined) 672

as a control layer component 35

as a data-management feature 86

function of 36

registry

retrieving files from 54

registry files

exporting 54

Registry Service 36

(defined) 672

as a control layer component 35

as a data-management feature 86

Remote debugging port number 509

remove 62

removing files

in Team Registry 64

renaming external templates 180

renaming internal templates 181
renderMap()
 Java method of Standard ETD 625
repeating fields, HL7 231
repeating nodes 178, 179, 230
 (example) 183
Report JVM Info and all class loads 486
reset()
 Java method of Standard ETD 342
Resource Managers (RM)
 in Distributed Transaction Processing (DTP)
 systems 539
Resume Session (debugger commands) 507
retrieveLinkIdentifier()
 ELS placeholder method 598
retrieveRegistryFile()
 Java method of Standard ETD 580
return (Business Rule) 313
reviewing and testing system
 Monk Test Console 149
 overview 147
 troubleshooting 148
right-click
 to access shortcut menu 59
right-clicking
 an ETD 102
Root Collaboration Rules 348
root nodes 158
-rp %_PORT%
 optional argument at end of command line 454,
 473
-rp %_REGPORT% 454
 optional argument at end of command line 473
rule (Business Rule), Business Rules
 rule 314
run time 57
 promoting files to 54
running (activating) the host 95
running a schema/Control Broker 147
run-time environment
 defined) 672

S

-sa (flag for stcinstd) 95
Sandbox 36, 57, 62
 (defined) 672
 interaction with e*Ways and BOBs 61
 placing files into 54
 removing files from 54
SAP e*Ways 39
SAP R/3 systems 39
Save Session (debugger commands) 507
saving files
 in Team Registry 63
ScEncrypt (Java class) 628–629
 in package com.stc.common.utils 630
ScEncrypt (Java class), methods in
 decrypt() 628
 encrypt() 628
ScEncrypt class
 decrypt() method 628
 encrypt() method 628
schema 32, 34, 36, 41, 44, 49, 51, 52, 53, 56, 57, 58, 59,
86, 140, 147, 155, 174, 207, 246, 254, 255, 492, 493,
494
schema, views of
 components tree 51
schemas
 creating 54
 defined 86
 opening 54
Security
 (folder in Components view) 51
security features 69
Security tab
 Control Broker properties 93
SeeBeyond JMS
 (defined) 672
SeeBeyond JMS IQ Manager
 e*Way Connections for 83
SeeBeyond JMS IQ Managers 37, 123, 136
 e*Way Connection for 668
 e*Way Connections for 339, 347, 540
SeeBeyond JMS IQ Service 136
SeeBeyond Web site 31, 39
Select Repetition Instance (dialog box) 326
send()
 Java method of Standard ETD 343
Services
 (folder in Components view) 51
setELSExpiration()
 ELS method, ELS methods
 setELSExpiration() 598
setIncomingEncoding()
 Java method of Standard ETD 586
setInstanceMap()
 Java method of Standard ETD 614
setMarshalEncoding()
 Java method of Standard ETD 587
setOutgoingEncoding()
 Java method of Standard ETD 587
setting or modifying breakpoints 513, 514
setUnmarshalEncoding()
 Java method of Standard ETD 588
SHIFT key
 to drag a node before another node 178, 181
shortcut menu

- for creating external templates 180
- for Insert Java Function 284
- for Undo 286
- from right-clicking an ETD 102
- from right-clicking in External Templates pane 180
 - from right-clicking in Internal Templates pane 181
- shortcut menus
 - accessed by right-clicking 59, 181
 - accessed via right-clicking 284, 286
 - in ETD Editor
 - for adding internal templates 180
 - for adding nodes 177
 - for deleting nodes 178
 - for deleting templates 181
 - for renaming nodes 178
- shutting down e*Gate components 497
- SJIS character encoding 583
- SJIS character encoding method 586, 587, 588
- SJIS character encoding methods
 - in Monk 385
- SNMP Agent 500
- SNMP Agents
 - creating 53
- Source Events pane, Java Collaboration Rules Editor
 - (illustrated) 265
 - described 266
- Species 588
- sprintf()
 - Java method of Standard ETD 555
- ss (flag for stcinstd) 95
- .ssc files (Monk ETDs) 204, 216
- standard ETD
 - properties 191–203
- standard ETD Editor 159–203
- starting e*Gate components 496
- starting e*Ways 124
- Status 491
- stccb 454, 473
- stccb command 147
- stccmd
 - command-line monitoring utility 490
- stcinstd
 - flags for 95
- stcinstd command 95
- STCJavaPassThrough class 109
- STCJavaPassThrough.class 113
- STCTypeConverter (Java class) 630–655
 - toBoolean() method 632
 - toByte() method 635
 - toByteArray() method 636
 - toBytePrimitive() method 634
 - toCharacter() method 639
 - toCharPrimitive() method 638
 - toDouble() method 642
 - toDoublePrimitive() method 640
 - toFloat() method 644
 - toFloatPrimitive() method 643
 - toInteger() method 647
 - toIntegerPrimitive() method 646
 - toLong() method 650
 - toLongPrimitive() method 648
 - toShort() method 652
 - toShortPrimitive() method 651
 - toString() method 653
- STCTypeConverter (Java class), methods in
 - toBooleanPrimitive() 631
- STCTypeConverter class
 - toBooleanPrimitive() method 631
- Step Into (debugger command) 508
- Step Out (debugger command) 508
- Step Over (debugger command) 508
- Stop in Class (debugger command) 508, 513
- Stop in Method (debugger command) 508, 514
- strings, methods for padding 657, 658, 659
- StringUtils (Java class) 656–662
- StringUtils (Java class), methods in
 - empty() 656
 - padCenter() 657
 - padLeft() 658
 - padRight() 659
 - trimBoth() 660
 - trimLeft() 660
 - trimRight() 661
- Subcollaboration Rules 348–351
 - characteristics of 348
 - illustrated 351
 - steps for calling 349
- subnodes 158
- subscribe 37, 85, 121, 123, 139, 147, 464
- subscribing and publishing 85, 86, 142, 145
 - and Collaborations 143
- subscriptions()
 - Java method of Standard ETD 344
- supporting documents 30
- suspending components 498
- swapInt()
 - Java method of Standard ETD 618
- swapLong()
 - Java method of Standard ETD 619
- swapShort()
 - Java method of Standard ETD 619
- switch (Business Rule) 315
- system configuration 35
- system design components
 - data flow relationships 83
 - data management relationships 86

logical relationships 85
overview 83

system preparations 48

system setup overview
 e*Gate GUIs 81
 introduction 81
 setup steps 82

system status, displaying 497, 498

T

tab in e*Gate Monitor 491

Team Registry 36, 58
 (defined) 673
 committing files 62
 editing files 63
 file operations 63
 promoting files 63
 removing files 64
 saving files 63
 unedit 63

Team Registry environments
 run time 36
 Sandbox 36

testing ETDs
 Java-enabled 183

text files
 editing 54, 57

text formatting, codes for 663–665

text labels on buttons 56

the 590

then (Business Rule) 316

then (Business Rule, with if) 306

thresholds
 disk 500
 event 502

throw (Business Rule) 317

Timers tab
 Control Broker properties 90

timestamp (Business Rule), Business Rules
 timestamp 318

timeStamp()
 Java method of Standard ETD 560

toBoolean()
 Java method of Standard ETD 632

toBooleanPrimitive()
 Java method of Standard ETD 631

toByte()
 Java method of Standard ETD 635

toByteArray()
 Java method of Standard ETD 636

toBytePrimitive()
 Java method of Standard ETD 634

toCharacter()

Java method of Standard ETD 639

toCharPrimitive()
 Java method of Standard ETD 638

toDouble()
 Java method of Standard ETD 642

toDoublePrimitive()
 Java method of Standard ETD 640

toFloat()
 Java method of Standard ETD 644

toFloatPrimitive()
 Java method of Standard ETD 643

toHex()
 Java method of Standard ETD 556

toInteger()
 Java method of Standard ETD 647

toIntegerPrimitive()
 Java method of Standard ETD 646

toJavaDatePattern()
 Java method of Standard ETD 556

toLong()
 Java method of Standard ETD 650

toLongPrimitive()
 Java method of Standard ETD 648

toolbar
 Business Rules 269
 e*Gate Monitor 492
 Enterprise Manager 52
 ETD Editor 160
 in online Help 77, 78
 Java Collaboration Rules Editor 265, 269
 relationship with menu bar and shortcut menus 59
 text labels on 56

topic()
 Java method of Standard ETD 345

toShort()
 Java method of Standard ETD 652

toShortPrimitive()
 Java method of Standard ETD 651

toString()
 Java method of Standard ETD 653

traceln()
 Java method of Standard ETD 566

Transaction Managers (TM)
 in Distributed Transaction Processing (DTP) systems 539

transaction processing
 XA (architecture) 539
 XA (overview) 538

transformDate()
 Java method of Standard ETD 560

trapping exceptions 515

trimBoth()
 Java method of Standard ETD 660

trimLeft()
 Java method of Standard ETD 660
trimRight()
 Java method of Standard ETD 661
troubleshooting e*Ways 463
 overview 462
 via e*Way log file 463
try (Business Rule) 321
.tsc files (transformation script for Monk Collaborations) 107
type conversion, methods for
 See STCTypeConverter

U

UHC character encoding methods
 in Monk 385
Undo 286
unedit
 Team Registry 63
Unicode, converting to and from 617
uniqueid (Business Rule), Business Rules
 uniqueid 322
uniqueID()
 Java method of Standard ETD 557
uniqueId()
 Java method of Standard ETD 557
unmarshal()
 Java method of Standard ETD 346
user's guide purpose and scope 26
userInitialize()
 used with character encoding methods 583
Users
 creating 53
UTF-8 character encoding methods
 in Monk 385

V

-v (flag for stcinstd) 95
validating ETDs
 Java-enabled 183
variable (Business Rule) 324
version information, displaying 498
View layer 34
viewing variables using the debugger 511

W

while (Business Rule) 325
wizards
 for building Java-enabled ETDs 98, 99
 for building Standard ETDs 100

write()
 Java method of Standard ETD 571
writeMap()
 Java method of Standard ETD 626
writeProperty()
 Java method of Standard ETD 347

X

X/Open Consortium 538
XA-compliant e*Way Connections 540
XA-compliant transaction processing
 activating 540
 architecture 539
 how achieved 538
 overview 538
XML DTD converter 226
.xpr files (project file, Java Collaboration Editor) 262
.xts files (Java Collaboration Rules Editor) 262