Important Information

SOME TIBCO SOFTWARE EMBEDS OR BUNDLES OTHER TIBCO SOFTWARE. USE OF SUCH EMBEDDED OR BUNDLED TIBCO SOFTWARE IS SOLELY TO ENABLE THE FUNCTIONALITY (OR PROVIDE LIMITED ADD-ON FUNCTIONALITY) OF THE LICENSED TIBCO SOFTWARE. THE EMBEDDED OR BUNDLED SOFTWARE IS NOT LICENSED TO BE USED OR ACCESSED BY ANY OTHER TIBCO SOFTWARE OR FOR ANY OTHER PURPOSE.

USE OF TIBCO SOFTWARE AND THIS DOCUMENT IS SUBJECT TO THE TERMS AND CONDITIONS OF A LICENSE AGREEMENT FOUND IN EITHER A SEPARATELY EXECUTED SOFTWARE LICENSE AGREEMENT, OR, IF THERE IS NO SUCH SEPARATE AGREEMENT, THE CLICKWRAP END USER LICENSE AGREEMENT WHICH IS DISPLAYED DURING DOWNLOAD OR INSTALLATION OF THE SOFTWARE (AND WHICH IS DUPLICATED IN THE LICENSE FILE) OR IF THERE IS NO SUCH SOFTWARE LICENSE AGREEMENT OR CLICKWRAP END USER LICENSE AGREEMENT, THE LICENSE(S) LOCATED IN THE “LICENSE” FILE(S) OF THE SOFTWARE. USE OF THIS DOCUMENT IS SUBJECT TO THOSE TERMS AND CONDITIONS, AND YOUR USE HEREOF SHALL CONSTITUTE ACCEPTANCE OF AND AN AGREEMENT TO BE BOUND BY THE SAME.

This document contains confidential information that is subject to U.S. and international copyright laws and treaties. No part of this document may be reproduced in any form without the written authorization of TIBCO Software Inc.

TIBCO, The Power of Now, TIBCO, TIBCO ActiveMatrix, TIBCO ActiveMatrix BusinessWorks, TIBCO Administrator, TIBCO ActiveSpaces, TIBCO Designer, TIBCO Enterprise Message Service, TIBCO Hawk, TIBCO Runtime Agent, TIBCO Rendezvous, are either registered trademarks or trademarks of TIBCO Software Inc. in the United States and/or other countries. EJB, Java EE, J2EE, and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries.

All other product and company names and marks mentioned in this document are the property of their respective owners and are mentioned for identification purposes only.

THIS SOFTWARE MAY BE AVAILABLE ON MULTIPLE OPERATING SYSTEMS. HOWEVER, NOT ALL OPERATING SYSTEM PLATFORMS FOR A SPECIFIC SOFTWARE VERSION ARE RELEASED AT THE SAME TIME. SEE THE README FILE FOR THE AVAILABILITY OF THIS SOFTWARE VERSION ON A SPECIFIC OPERATING SYSTEM PLATFORM.

THIS DOCUMENT IS PROVIDED “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT.

THIS DOCUMENT COULD INCLUDE TECHNICAL INACCURACIES OR TYPOGRAPHICAL ERRORS. CHANGES ARE PERIODICALLY ADDED TO THE INFORMATION HEREIN; THESE CHANGES WILL BE INCORPORATED IN NEW EDITIONS OF THIS DOCUMENT. TIBCO SOFTWARE INC. MAY MAKE IMPROVEMENTS AND/OR CHANGES IN THE PRODUCT(S) AND/OR THE PROGRAM(S) DESCRIBED IN THIS DOCUMENT AT ANY TIME.

THE CONTENTS OF THIS DOCUMENT MAY BE MODIFIED AND/OR QUALIFIED, DIRECTLY OR INDIRECTLY, BY OTHER DOCUMENTATION WHICH ACCOMPANIES THIS SOFTWARE, INCLUDING BUT NOT LIMITED TO ANY RELEASE NOTES AND “READ ME” FILES.

Copyright © 2012-2013 TIBCO Software Inc. ALL RIGHTS RESERVED.

TIBCO Software Inc. Confidential Information
Contents

Figures ........................................................................ 13

Tables ......................................................................... 15

Preface ........................................................................ 17

Related Documentation .................................................. 18
  TIBCO API Exchange Gateway Documentation .................. 18
  Other TIBCO Product Documentation ................................. 18

Typographical Conventions ............................................ 19

Connecting with TIBCO Resources ..................................... 22
  How to Join TIBCOmmunity ............................................ 22
  How to Access TIBCO Documentation ................................. 22
  How to Contact TIBCO Support ........................................ 22

Chapter 1  Introduction to TIBCO API Exchange Gateway .......... 23

Product Overview .......................................................... 24

Functional Components ................................................... 26

Design time components .................................................... 27
  TIBCO API Exchange Gateway Configuration GUI .................. 27
  TIBCO API Exchange Gateway Studio ................................. 27

Run Time Components ..................................................... 28
  Gateway Operational Layer ............................................. 28
  Gateway Management Layer ........................................... 28

Deployment Architecture .................................................. 31
  Single Server Deployment Architecture ............................... 31
  Distributed Deployment Architecture ................................ 35

Chapter 2  Getting Started .............................................. 39

Overview ........................................................................ 40

Prerequisites ................................................................. 40

Examples ........................................................................ 40

Configuring an endpoint operation for API Exchange Gateway........ 42

Working with TIBCO API Exchange Gateway Studio .................. 47
  Starting TIBCO API Exchange Gateway Studio ..................... 47
  Loading default ASG_DefaultImplementation Project ................ 47
<table>
<thead>
<tr>
<th>Contents</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Editing, Validating and Building default ASG_DefaultImplementation Project</td>
<td>48</td>
</tr>
<tr>
<td>Debugging Project in TIBCO API Exchange Gateway Studio</td>
<td>51</td>
</tr>
<tr>
<td><strong>Chapter 3  TIBCO API Exchange Gateway Engine Configuration</strong></td>
<td>55</td>
</tr>
<tr>
<td>TIBCO API Exchange Gateway Engine</td>
<td>56</td>
</tr>
<tr>
<td>Starting TIBCO API Exchange Gateway Engine</td>
<td>56</td>
</tr>
<tr>
<td>Processing Units of TIBCO API Exchange Gateway Engine</td>
<td>57</td>
</tr>
<tr>
<td>asg-validate using asg-tools</td>
<td>59</td>
</tr>
<tr>
<td>Configuring Log Files Settings</td>
<td>60</td>
</tr>
<tr>
<td>Logging Levels of TIBCO API Exchange Gateway Engine</td>
<td>63</td>
</tr>
<tr>
<td>Run Time Properties</td>
<td>65</td>
</tr>
<tr>
<td><strong>Chapter 4  Apache Module For TIBCO API Exchange Gateway</strong></td>
<td>93</td>
</tr>
<tr>
<td>Overview</td>
<td>94</td>
</tr>
<tr>
<td>Installing Apache HTTP Server</td>
<td>95</td>
</tr>
<tr>
<td>Installing Apache HTTP Server With SSL</td>
<td>95</td>
</tr>
<tr>
<td>Configuring Apache HTTP Server Using HTTP Transport</td>
<td>96</td>
</tr>
<tr>
<td>Running Apache HTTP Server</td>
<td>98</td>
</tr>
<tr>
<td>Secure Communications</td>
<td>99</td>
</tr>
<tr>
<td>Two-way (Mutual) SSL Authentication</td>
<td>100</td>
</tr>
<tr>
<td>Generate Keys and Certificates</td>
<td>100</td>
</tr>
<tr>
<td>Configure SSL on Apache HTTP Server</td>
<td>100</td>
</tr>
<tr>
<td>Configure Client Authentication With Digital Certificates On Apache HTTP Server</td>
<td>101</td>
</tr>
<tr>
<td>Configure Client Certificate Identification Details On Apache HTTP Server</td>
<td>102</td>
</tr>
<tr>
<td>Register Partners On API Exchange Gateway User Interface</td>
<td>104</td>
</tr>
<tr>
<td>Configuring SSL Communications</td>
<td>105</td>
</tr>
<tr>
<td>Configuring One Way SSL Authentication</td>
<td>105</td>
</tr>
<tr>
<td>Configuring Two-way (Mutual) SSL Authentication</td>
<td>106</td>
</tr>
<tr>
<td>Configure Apache Server For Basic HTTP Authentication</td>
<td>114</td>
</tr>
<tr>
<td>Configure Apache HTTP server for Basic Authentication</td>
<td>114</td>
</tr>
<tr>
<td>Configuring TIBCO API Exchange Gateway for Basic Authentication</td>
<td>119</td>
</tr>
<tr>
<td>Configure Apache Module For RVRD Setup Through A Firewall (DMZ)</td>
<td>121</td>
</tr>
<tr>
<td><strong>Chapter 5  Transport Communication</strong></td>
<td>125</td>
</tr>
<tr>
<td>Rendezvous Transport Communication</td>
<td>126</td>
</tr>
<tr>
<td>Configuration Setup</td>
<td>128</td>
</tr>
<tr>
<td>Setting Rendezvous Transport Properties</td>
<td>128</td>
</tr>
<tr>
<td>Secure Deployments With TIBCO Rendezvous</td>
<td>135</td>
</tr>
<tr>
<td>Configuration For Secure Rendezvous Daemon</td>
<td>137</td>
</tr>
<tr>
<td>Configuration Tips</td>
<td>137</td>
</tr>
<tr>
<td>Setup and Configure Rendezvous daemons</td>
<td>138</td>
</tr>
</tbody>
</table>
## Contents

Configuration Setup For Apache Module And API Exchange Gateway ................................................................. 138

**JMS Transport Communication** .......................................................................................................................... 147
- SOAP JMS Transport ................................................................................................................................. 148
- Gateway As SOAP JMS Server .................................................................................................................. 148
- Gateway As SOAP JMS Client .................................................................................................................. 149
- Configuring SOAP JMS Transport ........................................................................................................ 150
- ESB Transport/Channel .......................................................................................................................... 157

**Chapter 6 Gateway Configuration User Interface** ........................................................................................................ 163

- Overview .................................................................................................................................................. 164
- Starting GUI ........................................................................................................................................ 166
- Authentication Process For Gateway User Interface ...................................................................................... 168
- Configuration Setup For Authentication Process ....................................................................................... 172
  - LDAP Server Authentication ............................................................................................................. 172
  - File Based Authentication .................................................................................................................. 173
  - Default Authentication ...................................................................................................................... 177
- Enabling Debug Logging For Gateway User Interface .................................................................................... 178
  - Create Properties File ..................................................................................................................... 178
  - Use Properties File In The TRA File ............................................................................................. 179
- Manage A Gateway Project Configuration ................................................................................................... 180
- Validate Configuration ............................................................................................................................ 182
- Project Configuration ............................................................................................................................... 183
- **MAPPING** ............................................................................................................................................ 184
  - Mapping .............................................................................................................................................. 184
  - Schemas ............................................................................................................................................. 185
  - Error Maps .......................................................................................................................................... 185
- **SECURITY** ............................................................................................................................................ 187
  - WSS .................................................................................................................................................... 187
  - KeyStores ........................................................................................................................................ 187
  - Policy Mapping ................................................................................................................................. 188
  - Policy Binding .................................................................................................................................... 189
- **MONITORING** .................................................................................................................................... 190
  - Monitors .............................................................................................................................................. 190
  - KPI Groups ........................................................................................................................................ 192
- **ROUTING** ............................................................................................................................................ 193
  - Facade Operations .............................................................................................................................. 193
  - Target Operations ............................................................................................................................. 195
  - Target Operation Groups .................................................................................................................. 202
  - Routing .............................................................................................................................................. 203
- **PARTNER** ............................................................................................................................................ 205
  - Partners ............................................................................................................................................. 205

TIBCO API Exchange Gateway User's Guide
<table>
<thead>
<tr>
<th>Contents</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Partner Groups</td>
<td>206</td>
</tr>
<tr>
<td>Facade Access</td>
<td>206</td>
</tr>
<tr>
<td>Set RunTime Properties</td>
<td>208</td>
</tr>
<tr>
<td>General Properties</td>
<td>208</td>
</tr>
<tr>
<td>Monitoring Properties</td>
<td>209</td>
</tr>
<tr>
<td>Transport Properties</td>
<td>212</td>
</tr>
<tr>
<td>Database Properties</td>
<td>221</td>
</tr>
<tr>
<td>Security Properties</td>
<td>222</td>
</tr>
<tr>
<td>Chapter 7 Transaction Pipeline Processing</td>
<td>223</td>
</tr>
<tr>
<td>Transaction Pipeline processing</td>
<td>224</td>
</tr>
<tr>
<td>Request pipeline processing</td>
<td>224</td>
</tr>
<tr>
<td>Response pipeline processing</td>
<td>226</td>
</tr>
<tr>
<td>Parsing Step</td>
<td>228</td>
</tr>
<tr>
<td>Set The Partner Identity For Request</td>
<td>228</td>
</tr>
<tr>
<td>Set The Routing Key For Request</td>
<td>229</td>
</tr>
<tr>
<td>Enrich The Audit Trail Log For Request</td>
<td>229</td>
</tr>
<tr>
<td>Validate The Request Content</td>
<td>230</td>
</tr>
<tr>
<td>Set Metric Increment For Content Based Throttles</td>
<td>230</td>
</tr>
<tr>
<td>Set Sticky Key For Load-Balancing With StickyResourceAffinity</td>
<td>231</td>
</tr>
<tr>
<td>Parsing XSLT Documents</td>
<td>232</td>
</tr>
<tr>
<td>Parsing Output Document Schema</td>
<td>236</td>
</tr>
<tr>
<td>Mappings and Transformations</td>
<td>240</td>
</tr>
<tr>
<td>Mapping Types</td>
<td>241</td>
</tr>
<tr>
<td>Mapping Configuration</td>
<td>242</td>
</tr>
<tr>
<td>Transformations (XSLT Mapping)</td>
<td>245</td>
</tr>
<tr>
<td>Mapping Schemas</td>
<td>251</td>
</tr>
<tr>
<td>Mapping Container</td>
<td>251</td>
</tr>
<tr>
<td>Context Document</td>
<td>252</td>
</tr>
<tr>
<td>Chapter 8 Routing</td>
<td>255</td>
</tr>
<tr>
<td>Routing Overview</td>
<td>256</td>
</tr>
<tr>
<td>Routing Key</td>
<td>257</td>
</tr>
<tr>
<td>How To Derive And Configure Routing Key</td>
<td>258</td>
</tr>
<tr>
<td>Routing Use Case</td>
<td>263</td>
</tr>
<tr>
<td>Target Operation Group</td>
<td>266</td>
</tr>
<tr>
<td>Overview</td>
<td>266</td>
</tr>
<tr>
<td>Routing Algorithms For Target Operation Group</td>
<td>268</td>
</tr>
<tr>
<td>LoadBalanced</td>
<td>268</td>
</tr>
<tr>
<td>RoundRobin</td>
<td>268</td>
</tr>
<tr>
<td>Weighted RoundRobin</td>
<td>271</td>
</tr>
</tbody>
</table>
RoundRobin With Failover .............................................................. 274
Weighted RoundRobin With Failover ...................................................... 274
Sticky Resource Affinity ................................................................. 275

Target Operation Group Configuration .......................................................... 278
Configure A Target Operation Group ...................................................... 278
Configure A RoundRobin Target Operation Group ...................................................... 279
Configure A WeightedRoundRobin Target Operation Group ...................................................... 280
Configure A RoundRobinWithFailOver Target Operation Group ...................................................... 281
Configure A WeightedRoundRobinWithFailOver Target Operation Group ...................................................... 283
StickyResourceAffinity Target Operation Group Configuration ...................................................... 284

HealthCheck For Reference ................................................................. 289
Overview ............................................................................. 289
HealthCheck Modes For A Target Operation ...................................................... 289
HealthCheck Methods For Timer Mode .................................................... 290
HealthCheck Configuration For Target Operation .............................................. 292
Configuration for Reset Mode Of HealthCheck .............................................. 292
Configuration for Timer Based HealthCheck .............................................. 292

Chapter 9 Throttles ............................................................................ 297
Overview ............................................................................... 298
Facade Throttles ...................................................................... 298
Service Throttles ...................................................................... 299
Throttle Types ........................................................................ 299
Monitor Time Modifiers ................................................................. 303
Throttle Counter ...................................................................... 304
Throttle Chaining ..................................................................... 305
Configuring Throttles ................................................................ 306
Configuration Parameters for Throttles ..................................................... 306
Create a throttle policy definition ......................................................... 307
Content Based Throttles ................................................................. 310
Configuring Content Based Throttles ..................................................... 311
Configure throttle ..................................................................... 311
Define XSLT File. ..................................................................... 312
Upload XSLT File ..................................................................... 315
Payload Size Throttles ..................................................................... 316
Payload Size Throttle Types ............................................................. 316
Configuring Payload Size Throttles ....................................................... 318
Traffic Shaping ........................................................................ 320
Configuration ........................................................................ 320

Chapter 10 Authentication and Authorization ....................................... 323
User Authentication Overview ............................................................... 324
Transport and Protocol Level Authentication .................................................. 324
WS Security Services Authentication ......................................................... 326
Security Service Providers ........................................................................ 327
Web Services Security (WSS) Properties .................................................. 327
Types of Security Service Providers ....................................................... 327
Configuring LDAP Authentication Service Provider (LDAP ASP) ............... 328
Configuring Trust Identity Provider ....................................................... 335
Configuring Subject Identity Provider ................................................... 337
Configuring WSS Service Provider ....................................................... 338
Limitations .................................................................................. 339
Configure Secure Services With API Exchange Gateway .......................... 344
Define DSS Properties For Services ..................................................... 344
Configuring Services .................................................................. 346
Partner Authorization Overview ............................................................. 348
Operation Identification ................................................................ 348
Partner Identification .................................................................. 348
Partner Authorization .................................................................. 349

Chapter 11 Security Policies ................................................................. 351
Overview .................................................................................. 352
Security Concepts ....................................................................... 354
Types of Security Policies ................................................................. 357
Authentication ........................................................................ 357
Authorization ........................................................................ 358
Confidentiality ........................................................................ 358
Integrity .................................................................................. 359
CredentialMapping .................................................................... 359
Manage Policies ........................................................................ 361
Configure Shared Resource ............................................................ 361
Create Policy ........................................................................ 363
Register Policy ....................................................................... 366
Apply Policy ......................................................................... 367
Policy Use Cases ........................................................................ 368
Authentication Policies ................................................................. 368
Authorization Policies ................................................................. 372
Integrity Policies ....................................................................... 375
Confidentiality Policies ............................................................... 379
Credential Mapping Policies ............................................................ 382
Types of Security Shared Resources .................................................. 387
Shared Resources Properties ............................................................. 388
Configuring LDAP Authentication Shared Resource ............................. 388
Configuring Trust Identity Provider ................................................... 395
## Configuring Subject Identity Provider

- LdapAsp.properties ....................................................... 398
- SubjectIsp.properties .................................................... 399
- TrustIsp.properties ....................................................... 400
- IdentityIsp.properties .................................................... 400
- WssAsp.properties ....................................................... 401

## Chapter 12 OAuth Server

- Overview ........................................................................ 406
- Concepts ........................................................................ 406
- Benefits Of Using API Exchange Gateway OAuth Server 408
- OAuth Server Components Interactions .......................... 409
- Use Cases ....................................................................... 414
  - Authentication Code Flow Use Case .............................. 414
  - Password Credential Flow Use Case .............................. 414
  - Client Credential Flow Use Case ................................. 415
- Configuration Setup of OAuth 2.0 Server ...................... 416

## Chapter 13 OAuth Service Provider Interface

- Overview ........................................................................ 422
- Owner Service Provider Interface .................................. 423
  - Owner Service Provider Interface (SPI) Flow ................. 423
  - Owner Service Provider Interface (SPI) Java API .......... 424
- Client Service Provider Interface .................................. 426
  - Client Service Provider Interface (SPI) Flow ................ 426
  - Client Service Provider Interface (SPI) Java API .......... 426
- Scope Service Provider Interface .................................. 429
  - Scope Service Provider Interface (SPI) Flow ............... 429
  - Scope Service Provider Interface (SPI) Java API .......... 429
- How to Deploy Service Provider Interface .................... 432

## Chapter 14 Gateway Management Features

- Central Logger .............................................................. 434
  - Overview .................................................................... 434
  - Pre-requisites ............................................................. 434
  - Database Setup and Configuration for Central logger 434
  - Run Time Properties For Central Logger ....................... 438
  - Enable Reporting to Central Logger ............................. 438
  - Run Central Logger .................................................... 438
  - Enabling JMS Channel For Central Logger ................... 439
Configuring Load Balancer .............................................................. 496
Configuring Apache Modules For Core Engines ............................................. 499
Cluster Configuration For Runtime Components ............................................. 500
Configuring Fault Tolerance Parameters ................................................... 503
Configure Rendezvous Session Connection Parameters ....................................... 510

Appendix A Appendix .......................................................... 512
Editing Cluster Deployment Descriptor (CDD) File ............................................... 513
Rendezvous Session Connection Parameters For Apache Module ........................... 516
Rendezvous Session Connection Parameters For Gateway Core Engine .................... 517
Rendezvous Session Connection Parameters For Gateway Core Engine and Global Throttle Manager Communication ...................................................................... 519
Rendezvous Session Connection Parameters For Gateway Core Engine and Central Logger Communication 521

Chapter 17 Advanced Features .................................................. 525
Cache Agent ............................................................................ 526
How to run Cache Agent. .............................................................. 526
Hot Deployment Overview .................................................................. 527
Enabling Hot Deployment ............................................................... 527
Invoking Hot Deployment ............................................................... 527
Extension Mechanism ..................................................................... 528

Appendix B Configuration Tasks ................................................ 531
Cross-Origin Resource Sharing(CORS) Filter Properties .......................................... 532
Configure JMS Destinations for Southbound Service Operations .......................... 534
Configure Async Mode for Southbound JMS Service ............................................ 535
Configure Retry parameters for HTTP/HTTP(s) Transport ......................................... 536
Enable detail level logging for Gateway ........................................................ 537
Configure TIBCO Enterprise Message Service .................................................. 538
Configuring JMS northbound transport for XML ................................................. 540
Changing the Stack Size ................................................................... 543
Modify Unicast Discovery URL in CDD file ..................................................... 544
Generate Private Keys And Public Certificates with OpenSSL .............................. 547
Editing Cluster Deployment Descriptor (CDD) File ............................................... 551
Connection Problem to TIBCO DataGrid . ...................................................... 553

Glossary ...................................................................... 555

Index ......................................................................... 557
Figures
## Tables

<table>
<thead>
<tr>
<th>Table 1</th>
<th>General Typographical Conventions ................................................... xix</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table 2</td>
<td>Syntax Typographical Conventions ................................................................ xx</td>
</tr>
</tbody>
</table>
Preface

TIBCO® API Exchange Gateway is a high-performance event-based service-request routing engine. It treats requests and responses as events and implements the logic of a gateway layer using simple event-condition-action rules. TIBCO API Exchange Gateway provides the functionality of service abstraction, back-end integration, routing, request authentication and authorization, orchestration and throttling management with the implementation of policies at each layer of the gateway. To support these core functions, TIBCO API Exchange Gateway offers the centralized logging and Partner & Service Management.

TIBCO API Exchange Gateway architecture has been designed so that when multiple gateway servers are deployed, key management functions including throttle management, cache management and logging are co-ordinated across servers.

Topics

- Related Documentation, page 18
- Typographical Conventions, page 19
- Connecting with TIBCO Resources, page 22
Related Documentation

This section lists documentation resources you may find useful.

TIBCO API Exchange Gateway Documentation

The following documents form the TIBCO API Exchange Gateway documentation set:

- TIBCO API Exchange Gateway Installation Read this manual for instructions on site preparation and installation.
- TIBCO API Exchange Gateway User’s Guide Read this manual for instructions on how to configure and use this product.
- TIBCO API Exchange Gateway Release Notes Read the release notes for a list of new and changed features. This document also contains lists of known issues and closed issues for this release.

Other TIBCO Product Documentation

You may find it useful to read the documentation for the following TIBCO products:

- TIBCO BusinessEvents®
- TIBCO ActiveSpaces®
- TIBCO Rendezvous®
- TIBCO Enterprise Message Service™
- TIBCO ActiveMatrix BusinessWorks™
- TIBCO Spotfire®
Typographical Conventions

The following typographical conventions are used in this manual.

Table 1  General Typographical Conventions

<table>
<thead>
<tr>
<th>Convention</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ENV_NAME</strong></td>
<td>TIBCO products are installed into an installation environment. A product installed into an installation environment does not access components in other installation environments. Incompatible products and multiple instances of the same product must be installed into different installation environments. An installation environment consists of the following properties:</td>
</tr>
<tr>
<td></td>
<td>• <strong>Name</strong>  Identifies the installation environment. This name is referenced in documentation as <strong>ENV_NAME</strong>. On Microsoft Windows, the name is appended to the name of Windows services created by the installer and is a component of the path to the product shortcut in the Windows Start &gt; All Programs menu.</td>
</tr>
<tr>
<td></td>
<td>• <strong>Path</strong>   The folder into which the product is installed. This folder is referenced in documentation as <strong>TIBCO_HOME</strong>.</td>
</tr>
<tr>
<td><strong>TIBCO_HOME</strong></td>
<td>TIBCO API Exchange Gateway installs into a directory within a <strong>TIBCO_HOME</strong>. This directory is referenced in documentation as <strong>ASG_HOME</strong>. For example on linux platform, the value of <strong>ASG_HOME</strong> is /home/asg/tibcoasg/asg/2.0.</td>
</tr>
<tr>
<td><strong>ASG_HOME</strong></td>
<td>TIBCO API Exchange Gateway stores the configuration files in a directory which is separate from the installation directory. This directory is referenced in documentation as <strong>ASG_CONFIG_HOME</strong>. For example on linux platform, the value of <strong>ASG_CONFIG_HOME</strong> is: /home/asg/tibcoasgconfig/tibco/cfgmgmt.</td>
</tr>
<tr>
<td><strong>ASG_CONFIG_HOME</strong></td>
<td>Code font identifies commands, code examples, filenames, pathnames, and output displayed in a command window. For example: Use <strong>MyCommand</strong> to start the foo process.</td>
</tr>
<tr>
<td><strong>code font</strong></td>
<td>Bold code font is used in the following ways:</td>
</tr>
<tr>
<td><strong>bold code font</strong></td>
<td>• In procedures, to indicate what a user types. For example: Type <strong>admin</strong>.</td>
</tr>
<tr>
<td></td>
<td>• In large code samples, to indicate the parts of the sample that are of particular interest.</td>
</tr>
<tr>
<td></td>
<td>• In command syntax, to indicate the default parameter for a command. For example, if no parameter is specified, <strong>MyCommand</strong> is enabled: <strong>MyCommand</strong> [<strong>enable</strong></td>
</tr>
</tbody>
</table>
Typographical Conventions

Table 1  General Typographical Conventions (Cont’d)

<table>
<thead>
<tr>
<th>Convention</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>italic font</em></td>
<td>Italic font is used in the following ways:</td>
</tr>
<tr>
<td></td>
<td>• To indicate a document title. For example: See TIBCO ActiveMatrix</td>
</tr>
<tr>
<td></td>
<td>BusinessWorks Concepts.</td>
</tr>
<tr>
<td></td>
<td>• To introduce new terms For example: A portal page may contain</td>
</tr>
<tr>
<td></td>
<td>several portlets. <em>Portlets</em> are mini-applications that run in a</td>
</tr>
<tr>
<td></td>
<td>portal.</td>
</tr>
<tr>
<td></td>
<td>• To indicate a variable in a command or code syntax that you must</td>
</tr>
<tr>
<td></td>
<td>replace. For example: <em>MyCommand</em> <em>PathName</em></td>
</tr>
<tr>
<td>Key combinations</td>
<td>Key name separated by a plus sign indicate keys pressed simultaneously. For example: Ctrl+C.</td>
</tr>
<tr>
<td></td>
<td>Key names separated by a comma and space indicate keys pressed one after the other. For example: Esc, Ctrl+Q.</td>
</tr>
<tr>
<td>Note icon</td>
<td>The note icon indicates information that is of special interest or importance, for example, an additional action required only in certain circumstances.</td>
</tr>
<tr>
<td>Tip icon</td>
<td>The tip icon indicates an idea that could be useful, for example, a way to apply the information provided in the current section to achieve a specific result.</td>
</tr>
<tr>
<td>Warning icon</td>
<td>The warning icon indicates the potential for a damaging situation, for example, data loss or corruption if certain steps are taken or not taken.</td>
</tr>
</tbody>
</table>

Table 2  Syntax Typographical Conventions

<table>
<thead>
<tr>
<th>Convention</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ ]</td>
<td>An optional item in a command or code syntax.</td>
</tr>
<tr>
<td></td>
<td>For example:</td>
</tr>
<tr>
<td></td>
<td><em>MyCommand</em> [optional_parameter] <em>required_parameter</em></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>be chosen. For example, you can select only one of the following</td>
</tr>
<tr>
<td></td>
<td>parameters:</td>
</tr>
<tr>
<td></td>
<td><em>MyCommand</em> <em>para1</em></td>
</tr>
</tbody>
</table>
Table 2  Syntax Typographical Conventions

<table>
<thead>
<tr>
<th>Convention</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>{ }</code></td>
<td>A logical group of items in a command. Other syntax notations may appear within each logical group. For example, the following command requires two parameters, which can be either the pair <code>param1</code> and <code>param2</code>, or the pair <code>param3</code> and <code>param4</code>. MyCommand `{param1 param2}</td>
</tr>
</tbody>
</table>
Connecting with TIBCO Resources

How to Join TIBCOMmunity

TIBCOMmunity is an online destination for TIBCO customers, partners, and resident experts. It is a place to share and access the collective experience of the TIBCO community. TIBCOMmunity offers forums, blogs, and access to a variety of resources. To register, go to http://www.tibcommunity.com.

How to Access TIBCO Documentation

You can access TIBCO documentation here:

http://docs.tibco.com

How to Contact TIBCO Support

For comments or problems with this manual or the software it addresses, contact TIBCO Support as follows:

- For an overview of TIBCO Support, and information about getting started with TIBCO Support, visit this site:
  http://www.tibco.com/services/support
- If you already have a valid maintenance or support contract, visit this site:
  http://support.tibco.com

Entry to this site requires a user name and password. If you do not have a user name, you can request one.
Chapter 1  Introduction to TIBCO API Exchange Gateway

This chapter provides an overview of TIBCO API Exchange Gateway software, and a brief description of the major functional components.

Topics

- Product Overview, page 24
- Functional Components, page 26
- Deployment Architecture, page 31
Product Overview

TIBCO API Exchange Gateway provides the routing between partners and the service providers operating at the back-end. TIBCO API Exchange Gateway is an event-based routing engine which processes requests and responses at a high performance level. TIBCO API Exchange Gateway exposes these services, wherever they may be located, through a single logical point of ingress so it can not only route service requests, but also protect itself and services from unexpected or undesirable demand.

TIBCO API Exchange Gateway provides the following key features:

- Receives, routes and forwards requests at high speed.
- Routes requests between any requestor and any service endpoint.
- Protects from over-usage or access from unauthorized partners.
- Protects service endpoints from over-usage or unauthorized usage.
- Reports operation activity such as, performance and fault monitoring.
- Assures service level agreements are met.

TIBCO API Exchange Gateway provides an event-driven webservices platform responsible for routing, co-ordinating and managing the partner and customer API requests to the various services exposed by an organizations internal services layer.

TIBCO API Exchange Gateway supports the following policy models:

- Authorization rules
  You can configure authorization policies for partners that determine whose requests are handled.

- Throttling models
  You can configure various types of throttles that determine when the requests are handled.

- Routing rules
  You can configure the routing rules that determine where the requests are handled.

- Light-weight orchestration models determine how requests are handled.
  The following diagram shows the overview of the product functionality:
Figure 1 Functional Overview
Functional Components

This section provides a general overview of each product component used by TIBCO API Exchange Gateway software.
Design time components

TIBCO API Exchange Gateway provides the following design time components:

**TIBCO API Exchange Gateway Configuration GUI**

Partner data, partner operations, partner groups, services, operations, mappings, throttles, errormaps, schemas and routing information required for the various functionality of the software are configured via a web-based GUI.

**TIBCO API Exchange Gateway Studio**

The Gateway Studio is a design time environment which allows you to design and develop the custom extensions. The custom extensions can be integrated with the default implementation to customize the default behavior of the gateway core engine.
Run Time Components

TIBCO API Exchange Gateway provides the following runtime components:

Gateway Operational Layer

The Gateway Operational layer consists of following components:

Module for Apache HTTP Server

The Apache layer is used to terminate the incoming HTTP(s) transport and communicate with Facade component to forward the requests for further processing. Optionally, a JMS Server can be deployed to use the JMS transport.

Core Engine

The core engine is a high-performance event-based service-request routing engine that receives requests as events and uses the rules engine to determine where requests are handled.

The core engine contains the following main sub-components:

- Facade: The Facade provides a public northbound interface for the gateway to receive requests for a given API with a given binding (for example, SOAP over HTTP or SOAP over JMS).
- Router: The Router receives the requests from the Facade and routes it to the appropriate service handler.

Cache Agent

The cache agent stores the cache data for all objects of the cluster.

Gateway Management Layer

The Gateway Management layer provides the activity and request tracking. It has following sub-components.

Central Logger

Central Logger provides the centralized of messages in a database or file.
**Global Throttle Manager**

The Global Throttle Manager manages the Façade Throttle Manager and Service Throttle Manager. This component maintains the state of all global throttles in both Façades (Façade Throttles) and Routers (Service Throttles).

**Cache Clearing Manager**

The Cache Clearing Manager component clears the cache based on the size and age of the cached values.

**Monitoring and Management Server**

The Monitoring and Management Server is the central management component that allows you to monitor the status and manage the operational tasks of all components in the Gateway cluster.

**Gateway Reporting (Optional)**

The Gateway Reporting component generates the various type of reports based on the data logged in by the Central Logger component. This component integrates with the TIBCO Spotfire product to display the metrics.

The primary software components are shown in the following diagram:
Figure 2 Functional Components
Deployment Architecture

TIBCO API Exchange Gateway is deployed as a cluster of engines that together act as a single logical gateway. The engines in the cluster can run on a single server or in a distributed environment across multiple physical or virtual servers, providing fine grained control over the cluster deployment topology.

TIBCO Rendezvous is used for the communication between most of the run-time components, both in the gateway operational layer itself and between the components from the gateway management layer and the gateway operational layer and all components share a single set of configuration files which need to be stored on shared storage device that’s accessible for each of the run-time components. In case the multiple instances of core engine are deployed in a cluster, also multiple cache-agents are instantiated providing a single distributed cache that is shared across all core engines to support association and response cache functionality.

Single Server Deployment Architecture

In its simplest non-high available form, TIBCO API Exchange Gateway can run as a single server. The deployment of the software components as a single server instance are shown below:
This configuration provides full gateway functionality, including the optional operational reporting & analytics provided by the TIBCO Spotfire Server components. The TIBCO Spotfire Professional client software running on a Windows workstation and the TIBCO Spotfire WebPlayer Server software running on a Windows server are not shown in the diagram.
The protocol termination components, Apache HTTP server for HTTP and transports and optionally the TIBCO Enterprise Messaging Service for JMS transport, need to be deployed and managed with their standard operations management tools.

The Module for Apache HTTP Server that is part of the TIBCO API Exchange Gateway is deployed as a normal module for the Apache HTTP server. This module turns the Apache http requests into TIBCO Rendezvous messages for the communications with the gateway core engine. In case the Apache HTTP Server is deployed within a DMZ zone, the TIBCO Rendezvous Routing Deamon can be configured to forward the TIBCO Rendezvous messages from the DMZ network through the firewall to the internal network in which the TIBCO API Exchange Gateway components are deployed.

Run Time Components

The run-time components of the TIBCO API Exchange Gateway are deployed as a single application that can span multiple host servers. The run-time components are as follows:

- Gateway Core Engine
- Central Logger
- Global Throttle Manager
- Cache Clearing Manager

The management layer components, Central Logger and Global Throttle Manager, communicate at run-time with the gateway core engine using Rendezvous messages.

The Central Logger component receives events as the messages from the Rendezvous bus and stores them in the central logger database at appropriate intervals; this reduces the disk load during high transaction rates. These events are published by the gateway core engines during their operation. The Global Throttle Manager reports the throttle usage data to the Central Logger via the same mechanism. Network reduces the operating system load of the gateway core engines, as they are not to a local disk. It provides the mechanism to centralize the, as it is very important to provide a single, consistent view of activities. Finally, ensures that all components are not limited in their performance by the file system.

The Global Throttle Manager controls the throttle allocation for each gateway core engine. The Global Throttle Manager receives throttle reports from the gateway core engines over the Rendezvous bus. It, also, sends the throttle grants back to the gateway core engines over the Rendezvous bus.
The Global Throttle Manager treats the throttle usage events as the heartbeat interval of a core engine. In absence of, a configurable, number of consecutive heartbeats, the Global Throttle Manager treats the gateway core engine as dead and distributes the throttle limits of the dead instance equally to all the alive core engines.

The Cache Clearing Manager does not use the Rendezvous bus to interact with the gateway core engine. It connects directly to one of the Cache agents to clear the cache so that it does not grow to an excess size. This cleanup of the cache is called cache flushing.

To deploy and start the cluster of run-time components, you can use any of the following methods:

- **Command Line:**
  
  At the command-line, you specify the component unit to start and optionally a custom CDD file to use.

- **TIBCO API Exchange Gateway Monitoring and Management Server:**
  
  The run-time components can be deployed and started using the Management and Monitoring Server. This is the recommended method.

It’s recommended to use only one method for the entire gateway cluster you are deploying. Using the Management and Monitoring Server to deploy the runtime components is recommended method.

Both of the deployment methods use two default resources: an EAR file and a cluster deployment descriptor (CDD), which is an XML file.

When the gateway core engine (with or without caching agent), global throttle manager or cache clearing manager are started, they use the `asg_core.ear` and `asg_core.cdd` files in the `ASG_HOME/bin` directory.

When the central logger component is started, it uses `asg_c1.ear` and `asg_c1.cdd` files in the `ASG_HOME/bin` directory.

The Monitoring and Management Server and the GUI Configuration server can be started at the command line.

Any configuration updates that are made through the GUI Configuration server is persisted in the configuration files on the shared storage device. These configuration files needs to be reloaded by the run-time components in order to be effectuated.

The optional operational reporting & analytics provided by the TIBCO Spotfire server components interacts with the Central Logger through the central logger database using a standard JDBC connection.
Distributed Deployment Architecture

TIBCO API Exchange Gateway supports a distributed deployment environment, in which multiple instances of the gateway engines can be deployed. This architecture meets the requirements of high availability and scalability of the gateway components, which is recommended in a production environment.

Scaling and High Availability

TIBCO API Exchange Gateway provides a default site topology file which is configured for a deployment with single instances for each engine of the gateway cluster, all being deployed on a single server host. This configuration allows you to quickly deploy the API Exchange Gateway in a development environment, though it typically does not meet availability and scalability requirements for a production deployment. See High Availability Deployment Of Runtime Components on page 485.

TIBCO API Exchange Gateway Studio can be used to create production site topology configurations for your production environment including load balanced and fault-tolerant setups.

Load Balancing

TIBCO API Exchange Gateway can be rapidly scaled up and down through the addition or removal of additional instances of the core engines to the gateway cluster.

The TIBCO API Exchange Gateway architecture has been designed so that when multiple core engine instances are deployed in a gateway cluster, the key management functions including throttle management, cache management, cache clearing management and central are co-ordinated across all core engine instances. As these management functions are performed out-of-band, the components that provide these functions do not need to be scaled in support of higher transaction volumes.

However, as transaction levels increase, it is likely that this will be accompanied by a corresponding increase in management activity. To avoid the possible impact of management activity upon the core engines of the TIBCO API Exchange Gateway, these management components and the TIBCO Spotfire Servers should be moved onto separate servers.

Following diagram illustrates a simplified view of the scaled solution depicting the deployment of various components in a distributed environment:
Increasing the number of the core engines in TIBCO API Exchange Gateway deployment provides a near linear increase in the maximum number of transactions that can be managed and reduces the impact of the failure of an individual core engine. TIBCO API Exchange Gateway uses a shared nothing model between the active core engines to ensure that there is no shared state.

To support a load-balanced setup, the transport protocol termination components have to be configured appropriately.

For the JMS endpoints, load balancing requests across multiple core engines is achieved by setting up non-exclusive queues in the JMS server which automatically balances the load of incoming messages across the JMS receivers of the core engine instances. For HTTP and endpoints, load balancing the requests
across multiple core engines is handled by the API Exchange Gateway module for Apache HTTP Server. If a comma-separated list of TIBCO Rendezvous subjects is configured for an Apache server location, the module for Apache HTTP server load balances the incoming requests across the list of Rendezvous subjects. For each deployed core engine, a different Rendezvous subject from the list needs to be configured to ensure that requests are only handled once by a single core engine.

When the protocol termination components themselves reach the limits of the scale they can provide, an IP load-balancer can be added to the deployment in front of multiple Apache HTTP servers or JMS servers. The load-balancer should be configured to make these all available on a single IP address.

**High Availability of TIBCO API Exchange Gateway**

For a high available setup of the TIBCO API Exchange Gateway deployment, a different approach is taken for the components in the Gateway Operational Layer and the components in the Gateway Management Layer.

**Gateway Operational Layer**

As the core engine and Apache HTTP server maintain no state, fault tolerance is provided by having multiple instances running across sites and the host servers with the same configuration supporting a load balanced configuration. See Load Balancing, page 35.


Fault tolerance of cache agents is handled transparently by the object management layer. For fault tolerance of cache data, the only configuration task is to define the number of backups you want to keep, and to provide sufficient storage capacity. Use of a backing store is not needed as the cache agents are only used to implement the association cache, which is automatically rebuilt after complete failure as new transactions are handled by the API Exchange Gateway.

**Gateway Management Layer**

The components of the Gateway Management Layer should be deployed once in a primary-secondary group configuration. The central logger and the global throttle manager need to have a single running instance at all times to ensure that the gateway core engine operates without loss of functionality.
Therefore the central logger and global throttle manager need to be deployed in fault tolerant configuration with one active engine and one or more standby agents on a separate host servers. Such fault-tolerant engine setup can be configured in the cluster deployment descriptor (CDD) file by specifying the maximum number of one active agent for either of the agent classes and by creating multiple processing unit configurations for both the global throttle manager and the central logger agent. Deployed standby agents maintain a passive Rete network. They do not listen to events from channels and they do not update working memory. They will take over from the active instance in case it fails.

The other components of the Gateway Management Layer have no direct impact on the functionality of an operating API Exchange Gateway instance and hence they can be deployed with a cold standy configuration. This applies to the following components:

- Cache Clearing Manager
- Configuration GUI Server
- Monitoring and Management Server

It is suggested that multiple versions of these components are deployed across host servers with one running. If the running instance goes down start one of the other instances to regain full gateway functionality.
Chapter 2  Getting Started

This chapter provides an overview of the examples provided with the product.

Topics

- Overview, page 40
- Configuring an endpoint operation for API Exchange Gateway, page 42
- Create a new configuration, page 42
- Configure Partner Group, page 42
- Configure Partner Data, page 43
- Configure a Facade Operation, page 43
- Configure a Target Operation, page 44
- Configure an Authorization Configuration, page 44
- Configure Routing Configuration, page 45
- Test the gateway configuration, page 46
- Working with TIBCO API Exchange Gateway Studio, page 47
Overview

TIBCO API Exchange Gateway provides the following examples:
- GetLocation
- BookQuery
- Caching

Prerequisites

This section describes the list of tasks to be completed before you can run the examples.

Installation

Finish the installation of TIBCO API Exchange Gateway software and post installation tasks, as described, in the TIBCO API Exchange Gateway Installation Guide.

Verify the TIBCO API Exchange Gateway server status

Verify the installation and health of TIBCO API Exchange Gateway server by following the steps listed in Checking TIBCO API Exchange Gateway server status section of the TIBCO API Exchange Gateway Installation Guide.

Examples

This section describes the examples shipped with the TIBCO API Exchange Gateway software.

Following examples require the installation of TIBCO Runtime Agent 5.7.4 and TIBCO BusinessWorks 5.10.

GetLocation

The GetLocation service demonstrates the routing capability of API Exchange Gateway. The gateway interacts with a mock-up of a backend service which returns the coordinates of a device associated with a phone number. The gateway routes the request to a different backend service depending on the input phone number in the request.
This example is shipped with a **GetLocation** project which contains the client operations and back end services to be executed.

Refer to the ASG_HOME/examples/GetLocation/readme.html for the instructions on how to run the example.

**BookQuery**

The BookQuery service queries all the books in a book store by different criteria such as query by author, isbn, publisher, and title. TIBCO API Exchange Gateway example project implements the following four policies:

- Rate Throttles
- Quota Throttles
- High Water Mark Throttles
- Error Throttles

Refer to the ASG_HOME/examples/BookQuery/readme.html for the instructions on how to run the example.

**Caching**

The Caching example demonstrates the caching functionality provided by TIBCO API Exchange Gateway.

It has following main components:

- **ASG_CBA** is a TIBCO BusinessWorks project simulates a east side service, which takes the firstIdentity field as a cross-reference key and translates its value. This project does a simple translation on the key using an XML file. Alternatively, the service can be an adapter call, a lookup in a database, and other webservice.

- **A SOAP service** which is hosted on the TIBCO API Exchange Gateway. This service type is NOOP which means that it just returns an XML document with a received key and a translated value.

After the translated value is received from the service implemented in ASG_CBA BusinessWorks project, this is plugged into a SOAP payload. This payload is used to call a SOAP service running on the TIBCO API Exchange Gateway.
When the CustomStage feature is applied to an operation configuration, it enables a set of rules in the ASG_DefaultImplementation project. When the operation request is invoked, it looks into the cache for a cross-reference before the routing step. It uses the value of firstIdentity as a key for the cross-reference. If a value for that key is present in the cache, it is used in the cross reference. If a value is not found, then it sends a TIBCO Rendezvous message to retrieve that value from the service in the ASG_CBA BusinessWorks project.

Refer to the ASG_HOME/examples/Caching/readme.html for the instructions on how to run the example.

**Configuring an endpoint operation for API Exchange Gateway**

This section explains the steps to configure an service operation on the TIBCO API Exchange Gateway platform. This includes:

- Create a new configuration, page 42
- Configure Partner Group, page 42
- Configure Partner Data, page 43
- Configure a Facade Operation, page 43
- Configure a Target Operation, page 44
- Configure an Authorization Configuration, page 44
- Configure Routing Configuration, page 45
- Test the gateway configuration, page 46

**Create a new configuration**

1. Launch the GUI. See Starting GUI on page 166 for details.
2. Expand the Projects node.
3. Click Add New Project Configuration icon. Verify that the system generates a new configuration with a default configuration name as ASG_Config_uniqueNumId under Projects.
4. Put the mouse over the new project configuration, select the icon to rename the configuration name as ASG_Get_Start and hit Enter.

**Configure Partner Group**

1. Click the ASG_Get_Start configuration.
2. Click on Partner Groups tab.
3. Click **Add property (+)** icon from the top menu bar.

4. Type the values for the fields as shown in the following table:

*Table 3  Partner Group Configuration:*

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Group Name</td>
<td>supportASG</td>
</tr>
<tr>
<td>Email</td>
<td><a href="mailto:support@tibco.com">support@tibco.com</a></td>
</tr>
<tr>
<td>Phone</td>
<td>0019202331999</td>
</tr>
</tbody>
</table>

**Configure Partner Data**

1. Select the **ASG_Get_Start** configuration, if not already selected.
2. Click on **PARTNER** tab on the right hand bar.
3. Click on **Partners** tab on the top menu bar.
4. Click Add property icon to create a new partner.
5. Type the values for the fields as shown in the following table:

*Table 4  Partner Data Configuration:*

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Partner Name</td>
<td>tibcoASG</td>
</tr>
<tr>
<td>Partner Email</td>
<td><a href="mailto:support@tibco.com">support@tibco.com</a></td>
</tr>
<tr>
<td>Partner Phone</td>
<td>0019202331999</td>
</tr>
<tr>
<td>Partner Group</td>
<td>supportASG (select from the drop down list box)</td>
</tr>
</tbody>
</table>

**Configure a Facade Operation**

1. Select the **ASG_Get_Start** configuration, if not already selected.
2. Click on **ROUTING** tab on the right hand bar.
3. Click on **Facade Operations** tab on the top menu bar.
4. Click Add property icon to create a new facade operation.
5. Type the values for the fields as shown in the following table:
Table 5  Facade Operation Configuration:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operation Name</td>
<td>queryBookByAuthorBW</td>
</tr>
<tr>
<td>SOAP Action</td>
<td>&quot;/GetBooksByAuthorEndpoint&quot;</td>
</tr>
<tr>
<td>Operation URI</td>
<td>/ServerProcesses/GetBooksByAuthorEndpoint</td>
</tr>
<tr>
<td>Operation Service Name</td>
<td>MWC</td>
</tr>
</tbody>
</table>

Configure a Target Operation

1. Ensure that the ASG_Get_Start configuration is selected.
2. Click on ROUTING tab on the right hand bar.
3. Click on Target Operations tab on the top menu bar.
4. Click Add property icon to create a new target operation.
5. Type the values for the fields as shown in the following table:

Table 6  Target Operation Configuration:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operation Name</td>
<td>http.GetBooksByAuthor</td>
</tr>
<tr>
<td>Type</td>
<td>HTTP (select from the drop down list box)</td>
</tr>
<tr>
<td>Timeout</td>
<td>30000</td>
</tr>
<tr>
<td>SOAP Action</td>
<td>&quot;/GetBooksByAuthorEndpoint&quot;</td>
</tr>
<tr>
<td>URI</td>
<td>/ServerProcesses/GetBooksByAuthorEndpoint</td>
</tr>
<tr>
<td>Host</td>
<td>127.0.0.1</td>
</tr>
<tr>
<td>Port</td>
<td>9696</td>
</tr>
</tbody>
</table>

Configure an Authorization Configuration

1. Ensure that the ASG_Get_Start configuration is selected.
2. Click on PARTNER tab on the right hand bar.
3. Click on **Facade Access** tab on the top menu bar.
4. Click Add property icon to create a new partner operation.
5. Type in the following values:

### Table 7  Partner Authorization Configuration:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Partner</td>
<td>tibcoASG</td>
</tr>
<tr>
<td></td>
<td>(select from the drop down list box)</td>
</tr>
<tr>
<td>Facade Operation</td>
<td>queryBookByAuthorBW</td>
</tr>
<tr>
<td></td>
<td>(select from the drop down list box)</td>
</tr>
<tr>
<td>Partner Timeout</td>
<td>5000</td>
</tr>
</tbody>
</table>

### Configure Routing Configuration

1. Ensure that the **ASG_Get_Start** configuration is selected.
2. Click on **ROUTING** tab on the right hand bar.
3. Click on **Routing** tab on the top menu bar.
4. Click Add property icon to create a new routing configuration.
5. Type in the following values:

### Table 8  Routing Configuration:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operation Name</td>
<td>queryBookByAuthorBW</td>
</tr>
<tr>
<td></td>
<td>(select from the drop down list box)</td>
</tr>
<tr>
<td>Routing Type</td>
<td>Target Operation</td>
</tr>
<tr>
<td></td>
<td>(select from the drop down list box)</td>
</tr>
<tr>
<td>Routing Key</td>
<td>default</td>
</tr>
<tr>
<td>Target Operation</td>
<td>http.GetBooksByAuthor</td>
</tr>
<tr>
<td></td>
<td>(select from the drop down list box)</td>
</tr>
</tbody>
</table>
Save the gateway configuration

On the menu bar, click the **Save Configuration** icon to save the **ASG_Get_Start** configuration.

Test the gateway configuration

Run Apache HTTP Server, if not running

See [Running Apache HTTP Server, page 98](#) for details.

Run TIBCO API Exchange Gateway Core Server

1. Navigate to the TIBCO API Exchange Gateway installation as below:
   
   ```
   cd ASG_HOME/bin
   ```

2. Start the gateway engine for the **ASG_Get_Start** configuration as below:

   On windows platform, type the following command:
   ```
   asg-engine -u asg-caching-core -a ASG_Get_Start
   ```

   On unix platform, type the following command:
   ```
   ./asg-engine -u asg-caching-core -a ASG_Get_Start
   ```

3. Verify that the gateway engines started successfully and there are no errors.

4. Verify that the configuration for operation, services and partner are loaded correctly in the **asg_engine** engine logs. By default, the engine log file is **asg-caching-core** which is located under ASG_HOME\bin\logs directory.

Test the configured operation and target operation

1. Launch TIBCO Designer and open the project:
   
   ```
   ASG_HOME/examples/BookQuery/BookQuery
   ```

2. Run the following server processes:
   
   — BooksInterface-service1

3. Run the following client process:
   
   — QueryByAuthorClient

4. Verify that the process runs successfully without any errors.
Working with TIBCO API Exchange Gateway Studio

TIBCO API Exchange Gateway Studio provides a design time environment for adding custom extensions to a project. It is an Eclipse-based user interface, which is used to build, maintain, configure and modify deployments for the project. It is integrated into the standard Eclipse menus wherever appropriate, and works with many established Eclipse UI methodologies and plug-ins.

TIBCO API Exchange Gateway Studio is not supported on the solaris platform.

Starting TIBCO API Exchange Gateway Studio

This section describes the steps to run the TIBCO API Exchange Gateway Studio.

1. Navigate to the $ASG_HOME/studio/eclipse directory.
2. Start the TIBCO API Exchange Gateway Studio as follows:
   - On Windows platform, double click the studio.exe executable.
   - You can also type the following command on a command prompt window:
     ```
     cd $ASG_HOME/studio/eclipse
     studio.exe
     ```
   - On Unix platform, type the following command on a command prompt window:
     ```
     ./studio
     ```
3. If you are prompted, select or create the eclipse workspace directory where your project files will be stored. (If you check the option to use this workspace as a default, you are not prompted again.)
4. Click OK.
5. The first time you run TIBCO API Exchange Gateway Studio, a Welcome screen displays. Click the X next to Welcome to close the welcome screen.

Loading default ASG_DefaultImplementation Project

1. Start TIBCO API Exchange Gateway Studio, if not already started. See Starting TIBCO API Exchange Gateway Studio, page 47
2. From the top menu, select File > Import > TIBCO BusinessEvents > Existing TIBCO BusinessEvents Studio Project.

3. Click Next.

4. Type the values as below for following fields:

   - Select root directory: ASG_HOME\asg\1.2\projects.
   - Projects: Select ASG_DefaultImplementation project.
   - Copy projects into workspace: Check this box.

5. Click Finish.

6. In the Studio Explorer, on the left hand side, select the ASG_DefaultImplementation project and expand ASG_DefaultImplementation > DefaultImplementation node to view the channels, rule functions, rules and other resources in the project.

---

**Editing, Validating and Building default ASG_DefaultImplementation Project**

After the ASG_DefaultImplementation project is imported in the TIBCO API Exchange Gateway Studio, you can edit the default project to add the custom rule functions, rules, or any extensions as required to customize the default transaction processing pipeline.

**To Add or Edit a Resource In Project**

You can add or edit a resource in the project as below:

1. In Studio Explorer, select the folder where you want to store the new resource and right click to display menu.
   
   For example, to add a new rule function, do following:
   
   - Select the RuleFunction node, right-click and select New > Rule Function.
   - Input the values in the Rule Function wizard, as appropriate.
   - Enter the code for the rule function in the editor.

   For example, to edit an existing rule function, do following:

   - Select the rule function to be edited.
   - Double click to open it in an editor. Modify the function code as required.
   - Save the rule function.
2. Save your project.
   — To save all changes to all resources in a project (since last save), click **File > Save All** or click **Ctrl+Shift+S**.
   — To save changes in just the currently viewed resource, click **File > Save** or click **Ctrl+S**, or click the **Save** button.

**To Validate a Project, Project Folder, or Project Resource**

In Studio Explorer, do one of the following:

- Right click a project name, folder name, or a project resource name, and select **Validate Project**.
- Select a project name, folder name, or a project resource name and select **Project > Validate Project**.

A pop-up window displays the message, "Validation was successful" or summary information about any problems. Details about problems display in the Problems view.

**To Fix Validation Errors**

Many validation issues can be fixed using the Quick Fix feature. In the Problems view, right click on a problem and select Quick Fix.

**To Build default ASG_DefaultImplementation Project**

You may want to build the EAR file of the project for testing before deployment, as explained below:

1. In the Studio Explorer, highlight the ASG_DefaultImplementation project name, then from the top menus select **Project > Build Enterprise Archive**.
   If you see a message asking you to save all project resources, click Yes. It means an unsaved resource editor is open.

2. At the Build Enterprise Archive dialog, complete values according to guidelines provided in **Table 9, Build Enterprise Archive Reference Parameters**.

3. Click **Apply** to save the configuration details.
   (To revert to the version already saved, click Revert.)
4. Click OK to build the archive.

Table 9  *Build Enterprise Archive Reference Parameters*

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Name of this EAR configuration. (Not the EAR filename.) Default value is the project name.</td>
</tr>
<tr>
<td>Author</td>
<td>Person responsible for the EAR file. Default value is the currently logged-on user name.</td>
</tr>
<tr>
<td>Description</td>
<td>Optional description.</td>
</tr>
<tr>
<td>Archive Version</td>
<td>Increments on each build of the EAR. You can also manually enter a version identifier.</td>
</tr>
<tr>
<td>Generate Debug Info</td>
<td>Check this checkbox if you want to use the debugger. Default setting is checked.</td>
</tr>
<tr>
<td>Include all service level global variables</td>
<td>Check to include service level global variables.</td>
</tr>
<tr>
<td>File Location</td>
<td>Browse to the directory in which you want to store the EAR and enter an EAR filename. For example, for the TIBCO API Exchange Gateway engine the ear file is set as TIBCO_HOME\asg\1.2\bin\asg_core.ear.</td>
</tr>
<tr>
<td>Delete Temporary Files</td>
<td>Before TIBCO API Exchange Gateway packages the EAR file, it generates the Java code in a temporary directory. After the files are packaged in the EAR file, then the temporary files and directory are deleted. You can choose to keep the generated Java files, for example to troubleshoot some problem with an EAR file. To do so, uncheck the Delete Temporary Files checkbox, and specify where to store the Java files in the Compilation Directory field. Default setting is checked, meaning that temporary files are not saved.</td>
</tr>
<tr>
<td>Compilation Directory</td>
<td>If you uncheck the Delete Temporary Files checkbox, specify the directory where you want to save the Java files generated during the process of building the EAR file.</td>
</tr>
</tbody>
</table>
Debugging Project in TIBCO API Exchange Gateway Studio

This section explains how to debug the ASG_DefaultImplementation project within TIBCO API Exchange Gateway Studio.

TIBCO API Exchange Gateway studio debugger allows you to set the breakpoints, stepping through the code, suspending launched programs, examining the contents of variables, providing rule input, and so on.

TIBCO API Exchange Gateway studio debugger integrates with the Eclipse Java development toolkit debugger. You can debug local projects using their CDD and EAR files.

To debug the ASG_DefaultImplementation project in TIBCO API Exchange Gateway Studio, follow the steps as below:

1. Copy $TIBCO_HOME/be/5.1/bin/be-engine.tra to $TIBCO_HOME/be/5.1/bin/be-engine.tra.bak
2. Copy $TIBCO_HOME/asg/1.2/bin/asg-engine.tra to $TIBCO_HOME/be/5.1/bin/be-engine.tra

Steps 1 and 2 are optional steps and are used just to back up the TRA files.

3. Open the ASG_CONFIG_HOME/asg/asg.properties file and copy the whole contents.
4. Open the $TIBCO_HOME/be/5.1/bin/be-engine.tra file, go to the end of the file and append the contents of ASG_CONFIG_HOME/asg/asg.properties file.
5. In the $TIBCO_HOME/be/5.1/bin/be-engine.tra file only, uncomment and set the following property to the project configuration folder.

    tibco.clientVar.ASG/ConfigRoot=C:/ProgramData/TIBCOASG/tibco/cfgmgmt/asg/default

For example, if you want to debug a BookQuery project configuration, set the property as below:

    tibco.clientVar.ASG/ConfigRoot=C:/ProgramData/TIBCOASG/tibco/cfgmgmt/asg/BookQuery

You must change the value of the tibco.clientVar.ASG/ConfigRoot property to the configuration of the project you are working on.

6. Save your changes for the $TIBCO_HOME/be/5.1/bin/be-engine.tra file.
7. Start the TIBCO API Exchange Gateway Studio, if not already started. See Starting TIBCO API Exchange Gateway Studio on page 47.

8. Import the `ASG_HOME/projects/ASG_DefaultImplementation` project in TIBCO API Exchange Gateway Studio. See Loading default ASG_DefaultImplementation Project on page 47.

9. In Studio Explorer, select the `ASG_DefaultImplementation` project. From the menu, select Run > Debug Configurations.

10. On the Debug Configurations dialog box, for local debugging, select and double click the **TIBCO BusinessEvents Application** node at the left hand side. Make sure that the ASG_DefaultImplementation project is selected.

   — Select the **Main** tab and configure values as explained in Debug Configuration Parameters Reference on page 52.

   — Select the **Environment** tab and configure environment variables as needed, to run or debug the project in TIBCO API Exchange Gateway Studio.

   Edit the following variables to set the value as:

   Path: `PATH: TIBCO_HOME/tibrv/8.3/bin`

   where PATH is existing value of the Path variable.

   The **Environment** tab allows you to add new variables. You can select and then edit existing variables. You can append your edited variable to the existing environment variable, or you can replace the existing environment variable with it. For example if a custom function depends on a native library, you can add the path to that library using the `PATH`, `LD_LIBRARY_PATH`, `SHLIB_PATH`, or `LIBPATH` variable, as appropriate for your operating system.

   — Select the **Classpath** tab and configure the classpath for external libraries or custom functions as needed, for example if the project uses Rendezvous or JMS channels.

Table 10: Debug Configuration Parameters Reference

<table>
<thead>
<tr>
<th>Field</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>A descriptive name. It appears in the drop-down list of configurations.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Main Tab</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Project</td>
</tr>
</tbody>
</table>
11. On the Main tab, click **Apply** button to save the debug configuration changes.

12. Click **Debug** button to launch the debugger. Verify that the debugger is launched.

### Set the Debug Perspective within TIBCO API Exchange Gateway Studio

You can set the debug perspective within the gateway studio as follows:

1. As needed, switch to Debug perspective. Select **Window > Open Perspective**, or click the Open Perspective ( ![Open Perspective](image) ) button. Then select **Other > Debug**.
2. Click the down-arrow to the right of the debugger button. You see a drop-down list. Do one of the following:
   — Select a debug configuration from the list.
   — Select Debug Configurations. At the Debug Configurations dialog select a debug configuration and click **Debug**.

3. Verify that the debugger is launched.
Chapter 3  

TIBCO API Exchange Gateway Engine Configuration

This chapter explains the processing units, logging configuration and run time properties of the TIBCO API Exchange Gateway.

Topics

- Starting TIBCO API Exchange Gateway Engine, page 56
- Processing Units of TIBCO API Exchange Gateway Engine, page 57
- Configuring Log Files Settings, page 60
- Logging Levels of TIBCO API Exchange Gateway Engine, page 63
- Run Time Properties on page 65
TIBCO API Exchange Gateway Engine

TIBCO API Exchange Gateway Engine provides the core functionality of the gateway engine at runtime.

Starting TIBCO API Exchange Gateway Engine

This section describes the command and the various command line options used to run the TIBCO API Exchange Gateway engine at the command line.

To run the gateway engine, use the following command format. See Table 11, TIBCO API Exchange Gateway Engine Command Line Options, page 56

```
```

Table 11  TIBCO API Exchange Gateway Engine Command Line Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-h</td>
<td>Displays this help.</td>
</tr>
</tbody>
</table>
| --propFile | Specifies the location of the startup property file (TRA file).  
When you execute asg-engine, by default it looks for a property file of the same name (asg-engine.tra) in the working directory (the directory where you execute the command). This property file provides the startup values and other parameters to the engine executable.  
You can specify the path and filename of a startup property file explicitly using the --propFile parameter. |
| -p       | Specifies the location of a custom property file.  
This parameter allows you to pass one or more custom property files to asg-engine. Specify the path and filename. |
| -n       | Allows you to provide a name for the TIBCO API Exchange Gateway engine.  
The name provided here is used in the console and in log files. If you do not provide a name, the default value is the processing unit name. |
| -d       | Starts the debugger service on the engine for remote debugging. |
Processing Units of TIBCO API Exchange Gateway Engine

Processing units deploy as engines within which the agents run. In the Processing Units tab of the CDD file (See CDD File and Processing Units, page 58), you define which agents to include in the processing unit, and which logging configuration to use.

Table 11  TIBCO API Exchange GatewayEngine Command Line Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-a</td>
<td>Specifies the location of an TIBCO API Exchange Gateway configuration directory.</td>
</tr>
<tr>
<td>-c</td>
<td>Specifies the path and filename for the Cluster Deployment Descriptor (CDD) file you want to use. The default is default.cdd.</td>
</tr>
<tr>
<td>-u</td>
<td>Specifies the processing unit name you want to use for this engine. This processing unit name must exist in the CDD file you refer in the -c option. Following processing unit names are available, See Processing Units of TIBCO API Exchange Gateway Engine on page 57 for processing units details.</td>
</tr>
<tr>
<td></td>
<td>• asg-core - Starts up the core engine.</td>
</tr>
<tr>
<td></td>
<td>• asg-caching-core - Starts up the core engine in cache agent enabled mode.</td>
</tr>
<tr>
<td></td>
<td>• asg-gtm - Starts up the global throttle manager.</td>
</tr>
<tr>
<td></td>
<td>• asg-cache - Starts up the cache agent.</td>
</tr>
<tr>
<td></td>
<td>• asg-cache-cleanup- Starts up the cache cleanup agent.</td>
</tr>
<tr>
<td></td>
<td>• asg-cl - Starts up the central logger.</td>
</tr>
<tr>
<td></td>
<td>The default is asg-caching-core.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>EAR filename and path</th>
<th>Specify the path and filename for the EAR file you want to use.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>If you do not specify the EAR file name and no tibco.repourl property is set, the EAR file defaults to ASG_HOME/bin/asg_core.ear for all processing units of TIBCO API Exchange Gateway except for asg_cl. For asg_cl processing unit the default ear is ASG_HOME/bin/asg_cl.ear.</td>
</tr>
<tr>
<td></td>
<td>If you do not specify the EAR file name and the property tibco.repourl is set, the engine will use the property tibco.repourl as the EAR file path and name. To use this property, add it to the asg-engine.tra file. If you deploy using TIBCO Administrator this property is added to the generated TRA file automatically.</td>
</tr>
</tbody>
</table>
Processing units are configured in the CDD files as below:

Table 12  CDD File and Processing Units

<table>
<thead>
<tr>
<th>CDD File Name</th>
<th>Processing Unit Names</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASG_HOME/bin/asg-core.cdd</td>
<td>asg-core, asg-caching-core, asg-gtm, asg-cache, asg-cache-cleanup</td>
</tr>
<tr>
<td>ASG_HOME/bin/asg_cl.cdd</td>
<td>asg-cl</td>
</tr>
<tr>
<td>ASG_HOME/bin/asg_validator.cdd</td>
<td>asg-validate</td>
</tr>
</tbody>
</table>

By default, TIBCO API Exchange Gateway engine supports the following processing units:

- **asg-core**
  
asg-core is the main processing unit of TIBCO API Exchange Gateway that provides the critical gateway functions at runtime. This requires a separate cache agent running to provide the caching functionality at runtime and function properly.

- **asg-caching-core**
  
asg-caching-core is a processing unit of TIBCO API Exchange Gateway that provides the gateway functions at runtime including the caching functionality. This agent is cache enabled stand alone core engine and does not require a separate cache agent running.

- **asg-gtm**
  
asg-gtm is a processing unit of TIBCO API Exchange Gateway that provides the functionality of a global throttle manager at runtime.

- **asg-cache**
  
asg-cache is a processing unit of TIBCO API Exchange Gateway that provides the caching functionality to the gateway core engine.

- **asg-cache-cleanup**
  
asg-cache-cleanup is a processing unit of TIBCO API Exchange Gateway that starts the cache clearing manager to clear the cache based on the size and age of the cached values.
asg-cl

asg-cl is a processing unit of TIBCO API Exchange Gateway that provides the functionality of a Central Logger component at runtime.

**asg-validate using asg-tools**

This tool checks for a complete and correct configuration set before it can be used by the gateway engine at runtime.

It loads the data from all the configuration (cfg) files for a configuration set into the memory and validates it against each other. It also loads and compiles all the XSL files for a configuration set, so it checks if the XSL files are valid.

This tool ensures that the data in one configuration file which is dependent on other configuration file is valid.

For example, if you have defined a partner \( P_1 \) and assigned to a partner group \( PG_1 \) for a myconfig configuration. When you run asg-validate for myconfig, it will load partner \( P_1 \) data into memory and finds that this is assigned to \( PG_1 \) partner group. At that point it will check if the data of partner group \( PG_1 \) is also available in the memory of the gateway engine. If it doesn't find \( PG_1 \) partner group data, it will throw an error indicating that partner group \( PG_1 \) is not defined in Partner Groups.

If all the data such as partner, services, operations, throttles etc are valid and correct for a configuration set, it displays the successful message as: **The ASG configuration status is OK** otherwise it reports the errors on the console.

You can find the output of the asg-validate tool in a log file. See Log File For asg-validate on page 60.

**Running asg-validate using asg-tools**

asg-tools utility allows users to perform the validation for a gateway project configuration.

To perform the validation for a project configuration, do following:

1. Navigate to ASG_HOME/bin directory.
2. Type the command as below:

   ```
   asg-tools -u asg-validate -a asg_config_name
   ```

   where \( asg_config_name \) is the name of the configuration to be validated.

For example, to validate the BookQueryBE configuration, type the following command:
asg-tools -u asg-validate -a BookQueryBE

Log File For asg-validate

The output of asg-validate tool is stored in a log file named as asg-validate.log. By default, this log file is located in the ASG_CONFIG_HOME/logs directory. You can change the location of the logs directory in the ASG_HOME/bin/asg_validation.cdd file by editing the dir parameter as below:

```
<log-configs>
  <log-config id="logConfig">
    <dir>C:/ProgramData/TIBCO_ASG/tibco/cfgmgmt/logs</dir>
  </log-config>
</log-configs>
```

Limitations of asg-validate

asg-validate mainly validates the integrity of a data stored in the configuration files for a configuration set. This tool does not detect certain configuration issues such as Invalid URL for a transport, Invalid queue or topic name for JMS services.

The Validate icon on the TIBCO API Exchange Gateway configuration user interface (UI) uses the asg-validate tool to validate the configuration data entered on the user interface. It's recommended to run the validation after you have completely entered the data for a configuration set. This helps to find the errors at the design time so that they can be fixed earlier and prevent the errors at runtime for the gateway core engine.

Configuring Log Files Settings

TIBCO API Exchange Gateway engine logs the messages in the files. This section explains the configuration settings required for the log files. The configuration for log files are defined as log configurations for a processing unit. Each processing unit references a log configuration. The log configurations are defined in the Collections tab of the CDD file depending on the processing unit. Use ASG_HOME/bin/asg_cl.cdd for the asg_cl processing unit, and ASG_HOME/bin/asg_core.cdd for rest of the runtime processing units. See CDD File and Processing Units, page 58 to find the CDD file for a processing unit.
For a reference to the settings, See File Properties Settings for a Log Configuration in CDD Collections Tab, page 61

Table 13  File Properties Settings for a Log Configuration in CDD Collections Tab

<table>
<thead>
<tr>
<th>Property</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable</td>
<td>Check the Enable checkbox to enable log files to be written. Configure the settings in this section to specify details. If this checkbox is unchecked, all other properties in this section are ignored.</td>
</tr>
<tr>
<td>Directory</td>
<td>Enter the absolute path to the directory in which you want to store the files. If you do not enter a leading slash, the files are stored relative to the working directory (the directory in which you start the be-engine.exe executable).</td>
</tr>
<tr>
<td>Name</td>
<td>Name of the log file. If no name is set, the default value is the processing unit name.log. For example, for the asg-caching-core processing unit, the log file name defaults to asg-caching-core.log.</td>
</tr>
<tr>
<td>Max number</td>
<td>Number of log files to keep. When the Max size setting value is reached, a new log file is created for the next log entries. Files are created up to the Max number setting size. The oldest file is deleted when a new file is added after this value is reached. Default is 10.</td>
</tr>
<tr>
<td>Max size</td>
<td>Maximum size of one log file in bytes. Default is 10000000.</td>
</tr>
<tr>
<td>Append</td>
<td>If checked then new entries are added to the end of the file. If not checked, the contents of the file are flushed each time the engine starts.</td>
</tr>
</tbody>
</table>
Log File Name and Location

Set the name and location of the log file for a log configuration using the Name and Directory settings. See File Properties Settings for a Log Configuration in CDD Collections Tab, page 61.

If you do not enter a leading slash, the files are stored relative to the working directory (the directory in which you start the asg-engine.exe executable). If you do not specify a name, the processing unit name is used. If no processing unit name is specified, the name defaults to processing unit name.log.

Number and Size of Log Files

You can set the size of a single log file, the number of files to keep, and whether a log file is flushed when an engine starts, or whether entries are appended. See File Properties Settings for a Log Configuration in CDD Collections Tab, page 61.

The maximum size of the file is defined in the bytes, and the default value is configured as 10MB.

Set the Log File configuration Settings in CDD File

You can edit the parameters for log file configuration in the Cluster Deployment Descriptor (asg_core.cdd) file as follows:

1. Navigate to the ASG_HOME/bin directory.
2. Copy the asg_core.cdd file to ASG_HOME/projects/ASG_DefaultImplementation folder.
3. Navigate to the ASG_HOME/studio/eclipse directory.
4. Start the TIBCO API Exchange Gateway Studio as follows:
   - On Windows platform, run the following executable: studio.exe
   - On Unix platform, type the following command: ./studio
5. If you are prompted, select or create the Eclipse workspace directory where your project files will be stored. (If you check the option to use this workspace as a default, you are not prompted again.)
6. Click OK.
7. Close the Welcome screen.
8. From the File menu select Import.
9. In the Import Select wizard, select an import source as General > Existing Projects into Workspace and click Next. You see the Import Projects dialog.

10. In the Import Projects dialog Select root directory field, browse to and select the project: ASG_HOME/projects/ASG_DefaultImplementation.

11. Click Finish.

12. In the Studio Explorer, expand the ASG_DefaultImplementation project node. Verify that you see the asg_core.cdd file.


14. Select and expand Collections tab.

15. Select Log Configurations > logConfig.

16. On the right side, go to Configuration > Files section.

17. Edit the properties for log files configuration as needed. See File Properties Settings for a Log Configuration in CDD Collections Tab, page 61

18. Back up the original asg_core.cdd file in the ASG_HOME/bin directory.

19. Copy the modified asg_core.cdd file from ASG_HOME/projects/ASG_DefaultImplementation to the ASG_HOME/bin directory.

If you select the "Copy projects into workspace" option during the import of the project, then the modified asg_core.cdd file exists in the workspace directory. Make sure to copy the asg_core.cdd file from workspace directory to the ASG_HOME/bin directory.

Logging Levels of TIBCO API Exchange Gateway Engine

You can define a level of logging for the TIBCO API Exchange Gateway engine. A level corresponds to what types of messages are logged to the log files and what types of messages are filtered out.

Levels of Logging

This section explains the various levels of logging available for the gateway engine.
Following logging levels are supported:

<table>
<thead>
<tr>
<th>Level</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>Specifies no logging for the engine. Highest possible rank. Filters out all logging messages (turns logging off).</td>
</tr>
<tr>
<td>3</td>
<td>Specifies Error level of logging. Logs the runtime error messages that might or might not cause the engine to stop running.</td>
</tr>
<tr>
<td>2</td>
<td>Specifies WARNING level of logging. Logs the potentially harmful runtime messages (warnings).</td>
</tr>
<tr>
<td>1</td>
<td>Specifies INFORMATION level of logging. Logs the runtime informational messages of general interest (Information).</td>
</tr>
<tr>
<td>0</td>
<td>Specifies Debug level of logging. Lowest possible rank. Logs the detailed runtime messages, for use in identifying and debugging issues.</td>
</tr>
</tbody>
</table>

The Level property in the CDD Collections Tab should be at least set to 1 (info) for the log levels of TIBCO API Exchange Gateway engine to work.

**How to Set the Logging Level for the Gateway Engine**

Set the logging level using the `tibco.clientVar.ASG/Logging/MinLogLevel` property defined in the `ASG_CONFIG_HOME/asg/asg.properties` file as below:

```
tibco.clientVar.ASG/Logging/MinLogLevel=1
```

The default value is 1, which is Information level of logging.

TIBCO API Exchange Gateway stores the configuration files in a directory which is separate from the installation directory. This directory is referenced as `ASG_CONFIG_HOME`. For example, on windows platform, the value of `ASG_CONFIG_HOME` is `C:\ProgramData\TIBCOASG\tibco\cfgmgmt`. `ASG_CONFIG_HOME` is defined during the installation time.
Run Time Properties

This section explains the runtime properties for the core engine and the central logger component of the gateway. The properties are defined in the `asg.properties` and `asg_cl.properties` files respectively.

The runtime properties can also be set in one of the following ways:

- Using the gateway user interface. The properties are set on the home page by selecting Gateway Engine Properties from the drop down list box next to icon.

- Using a text editor to directly edit `asg.properties` and `asg_cl.properties` files. The files are located in `ASG_CONFIG_HOME` directory.

Run Time Properties of Core Engine

The properties for the gateway core engine are defined in the `asg.properties` file located in `ASG_CONFIG_HOME` directory.

Following properties can be defined:

Table 14  Core Engine Properties

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
</tr>
</thead>
</table>
| tibco.clientVar.ASG/HttpClient/useSynchHttpClient | A boolean field which is used to send a synchronous request from the client at the facade side using the HTTP transport.  
If the value is set to `true`, then the client sends the synchronous request for HTTP transport. If the value is set to `false`, then asynchronous request is sent for HTTP transport.  
Default is `false` |
| tibco.clientVar.ASG/HttpClient/workers | Number of threads used for HTTP Client. This is relevant only if the value of the `useSynchHttpClient` is `false` (i.e when asynchronous request is sent for HTTP transport).  
Default is 10 |
Table 14  Core Engine Properties

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tibco.clientVar.ASG/modRV/facade_request</td>
<td>Specifies the Rendezvous subject name which is used by the gateway core engine to listen for requests from Apache module. Default is _LOCAL.asg.north.request</td>
</tr>
<tr>
<td>tibco.clientVar.ASG/modRV/facade_request_binary</td>
<td>Specifies the Rendezvous subject name which is used by the gateway core engine to listen for binary requests from Apache module. Default is _LOCAL.asg.north.request_binary</td>
</tr>
<tr>
<td>tibco.clientVar.ASG/modRV/facade_response</td>
<td>Specifies the Rendezvous subject to send the response when Apache module is configured not to use the auto generated reply subject. If the Apache module is configured to use the reply subject then this is ignored.</td>
</tr>
<tr>
<td>tibco.clientVar.ASG/modRV/RvDaemon</td>
<td>Specifies the value of Rendezvous daemon for the gateway core engine to connect and listen for the requests from the Apache module.</td>
</tr>
<tr>
<td>tibco.clientVar.ASG/modRV/RvNetwork</td>
<td>Specifies the value of Rendezvous network for the gateway core engine to connect and listen for the requests from the Apache module.</td>
</tr>
<tr>
<td>tibco.clientVar.ASG/modRV/RvService</td>
<td>Specifies the value of Rendezvous service for the gateway core engine to connect and listen for the requests from the Apache module.</td>
</tr>
<tr>
<td>tibco.clientVar.ASG/GTM/RV/RvDaemon</td>
<td>Specifies the value of Rendezvous daemon for the gateway core engine to connect to the global throttle manager.</td>
</tr>
<tr>
<td>tibco.clientVar.ASG/GTM/RV/RvNetwork</td>
<td>Specifies the value of Rendezvous network for the gateway core engine to connect to the global throttle manager.</td>
</tr>
</tbody>
</table>
### Table 14: Core Engine Properties

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tibco.clientVar.ASG/GTM/RV/RvService</td>
<td>Specifies the value of Rendezvous service for the gateway core engine to connect to the global throttle manager.</td>
</tr>
<tr>
<td>tibco.clientVar.ASG/CL/RV/RvDaemon</td>
<td>Specifies the value of Rendezvous daemon for the gateway core engine to send messages to the central logger.</td>
</tr>
<tr>
<td>tibco.clientVar.ASG/CL/RV/RvNetwork</td>
<td>Specifies the value of Rendezvous network for the gateway core engine to send messages to the central logger.</td>
</tr>
<tr>
<td>tibco.clientVar.ASG/CL/RV/RvService</td>
<td>Specifies the value of Rendezvous service for the gateway core engine to send messages to the central logger.</td>
</tr>
<tr>
<td>tibco.clientVar.Common/Connections/RV/SubjectPrefix</td>
<td>Specifies the prefix for all Rendezvous subject names used between gateway core engine and central logger, gateway core engine and global throttle manager components. Default is <code>TIBCO.ASG.INTERNAL</code></td>
</tr>
<tr>
<td>com.tibco.cep.runtime.channel.payload.validation</td>
<td>Set this field as <code>true</code> or <code>false</code> to toggle the validation of event payloads. Default is <code>false</code>.</td>
</tr>
<tr>
<td>tibco.clientVar.ASG/Deployments/AllowHotUpdate</td>
<td>Set this field <code>true</code> or <code>false</code> to enable or disable hot configuration update. Default is <code>false</code>.</td>
</tr>
</tbody>
</table>
Table 14  Core Engine Properties

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tibco.clientVar.ASG/ConfigRoot</td>
<td>Specifies the root directory for an entire configuration.</td>
</tr>
<tr>
<td></td>
<td>• If this property is set, the configuration value specified for this property takes precedence over the configuration specified by asg-engine command line option such as asg-engine.exe -a &lt;Config project name&gt;.</td>
</tr>
<tr>
<td></td>
<td>For example, the root directory for default configuration is:</td>
</tr>
<tr>
<td></td>
<td>C:/ProgramData/ASG200/tibco/cfgmgmt/asg/default</td>
</tr>
<tr>
<td>tibco.clientVar.EXAMPLES_HOME</td>
<td>Specifies the home location for the examples directory shipped with API Exchange Gateway.</td>
</tr>
<tr>
<td></td>
<td>For example, the home directory for the examples location:</td>
</tr>
<tr>
<td></td>
<td>ASG_HOME/examples</td>
</tr>
<tr>
<td>tibco.clientVar.ASG/Logging/reportingEnabled</td>
<td>Specifies if the reporting to common logger is enabled or not. By default, the core engine does not record the transactions to common logger.</td>
</tr>
<tr>
<td></td>
<td>Default is false</td>
</tr>
<tr>
<td></td>
<td>See Enable Reporting to Central Logger, page 438.</td>
</tr>
<tr>
<td>tibco.clientVar.ASG/Logging/interval</td>
<td>Specifies the time interval (in milliseconds) between core engine and central logger to record transactions.</td>
</tr>
<tr>
<td></td>
<td>Default is 30000</td>
</tr>
<tr>
<td>tibco.clientVar.ASG/Logging/MinLogLevel</td>
<td>Set this field to enable the detail level logging for the central logger component. If the value is 1 the central logger records all the details of transaction. If the value is 0 the central logger records high level transcation.</td>
</tr>
<tr>
<td></td>
<td>Default is 0.</td>
</tr>
</tbody>
</table>
Table 14  Core Engine Properties

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tibco.clientVar.ASG/Throttle/UpdateIntervalSec</td>
<td>Specifies the time interval (in seconds) for sending throttle updates to global throttle manager. Default is 10</td>
</tr>
<tr>
<td>tibco.clientVar.ASG/anonymous/PartnerName/Authenticated</td>
<td>Specifies the default partner name for the unauthenticated requests. Default value is =anon</td>
</tr>
<tr>
<td>tibco.clientVar.ASG/Endpoint/ESB0/requestQueue</td>
<td>Specifies the queue name for an ESB channel (one) communication for the target operation request. Default queue name is asg.out.request</td>
</tr>
<tr>
<td>tibco.clientVar.ASG/Endpoint/ESB0/replyQueue</td>
<td>Specifies the queue name for ESB channel (one) communication for the target operation response. Default queue name is asg.out.request.reply.0.0</td>
</tr>
<tr>
<td>tibco.clientVar.ASG/Endpoint/ESB1/requestQueue</td>
<td>Specifies the queue name for an ESB channel (two) communication for the target operation request. Default queue name is asg.out.request</td>
</tr>
<tr>
<td>tibco.clientVar.ASG/Endpoint/ESB1/replyQueue</td>
<td>Specifies the queue name for an ESB channel (two) communication for the target operation response. Default queue name is asg.out.request.reply.0.1</td>
</tr>
<tr>
<td>tibco.clientVar.ASG/Endpoint/ESB2/requestQueue</td>
<td>Specifies the queue name for an ESB channel (three) communication for the target operation request. Default queue name is asg.out.request</td>
</tr>
</tbody>
</table>
**Table 14  Core Engine Properties**

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tibco.clientVar.ASG/Endpoint/ESB2/replyQueue</td>
<td>Specifies the queue name for an ESB channel (three) communication for the target operation response. Default queue name is asg.out.request.reply.0.2</td>
</tr>
<tr>
<td>tibco.clientVar.ASG/Facade/ESB0/requestQueue</td>
<td>Specifies the queue name to be used for the requests sent by the client at the facade side when the inbound channel is ESB (JMS transport with XML). Default queue name is asg.in.request</td>
</tr>
<tr>
<td>tibco.clientVar.ASG/Facade/ESB0/replyQueue</td>
<td>Specifies the queue name to be used for the facade responses received for the client requests when the inbound channel is ESB (JMS transport with XML). Default queue name is asg.in.request.reply.0</td>
</tr>
<tr>
<td>tibco.clientVar.ASG/Facade/SOAPJMS/requestQueue</td>
<td>Specifies the queue name to be used for the facade requests sent by the client when the inbound transport is SOAP JMS. Default queue name is asg.soap.in.request</td>
</tr>
<tr>
<td>tibco.clientVar.ASG/Facade/SOAPJMS/replyQueue</td>
<td>Specifies the queue name to be used for the facade responses received for the client requests when the inbound transport is SOAP JMS. Default queue name is asg.soap.in.request.reply.0</td>
</tr>
<tr>
<td>tibco.clientVar.ASG/Endpoint/SOAPJMS/DefaultTargetRequestQueue</td>
<td>Specifies the queue name to be used for the requests sent to target operations when the inbound transport is SOAP JMS. Default queue name is asg.soap.forward</td>
</tr>
</tbody>
</table>
Table 14  Core Engine Properties

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
</tr>
</thead>
</table>
| tibco.clientVar.ASG/Endpoint/SOAPJMS/TargetResponseQueue | Specifies the queue name to be used for the target responses received from the target operations when the inbound transport is SOAP JMS.  
Default queue name is asg.soap.forward.reply.0 |

**EMS Server Connections (Facade ESB Channel) Properties**

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
</tr>
</thead>
</table>
| tibco.clientVar.Common/Connections/JMS/FacadeEsbConnection0/JMSPublisherURL | Specifies the connection URL for the EMS Server used for facade operation requests from ESB communication domain. ESB communication uses JMS transport with XML.  
Default is tcp://localhost:7222 |
| tibco.clientVar.Common/Connections/JMS/FacadeEsbConnection0/JNDIContextURL | Specifies the URL to the JNDI service provider used for facade operation requests with ESB communication domain.  
Default is tibjmsnaming://localhost:7222 |
| tibco.clientVar.Common/Connections/JMS/FacadeEsbConnection0/TopicConnectionFactoryName | Specifies the name of TopicConnectionFactory object stored in JNDI. This object is used to create a topic connection with ESB services at the facade side.  
Default is TopicConnectionFactory |
| tibco.clientVar.Common/Connections/JMS/FacadeEsbConnection0/QueueConnectionFactoryName | Specifies the name of QueueConnectionFactory object stored in JNDI. This object is used to create a queue connection with ESB services at the facade side.  
Default is QueueConnectionFactory |
Table 14  Core Engine Properties

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tibco.clientVar.Common/Connections/JMS/FacadeEsbConnection0/JNDIUsername</td>
<td>Specifies the user name for logging into the JNDI server in the ESB communication domain at the facade side. If the JNDI provider does not require access control, this field can be empty. Default is admin</td>
</tr>
<tr>
<td>tibco.clientVar.Common/Connections/JMS/FacadeEsbConnection0/JNDIPassword</td>
<td>Specifies the password for logging into the JNDI server in the ESB communication domain at the facade side. If the JNDI provider does not require access control, this field can be empty.</td>
</tr>
<tr>
<td>tibco.clientVar.Common/Connections/JMS/FacadeEsbConnection0/JMSUsername</td>
<td>Specifies the user name for logging into the EMS server in the ESB communication domain at the facade side. Default is admin</td>
</tr>
<tr>
<td>tibco.clientVar.Common/Connections/JMS/FacadeEsbConnection0/JMSPassword</td>
<td>Specifies the password for logging into the EMS server in the ESB communication domain at the facade side.</td>
</tr>
</tbody>
</table>

EMS Server Connections for ESB Channel1 at Target Side.

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tibco.clientVar.Common/Connections/JMS/TargetEsbConnection0/JMSPort</td>
<td>Specifies the connection URL for the EMS Server used for target operation requests sent to backend services in ESB communication domain. ESB communication uses JMS transport with XML. Default is tcp://localhost:7222</td>
</tr>
<tr>
<td>tibco.clientVar.Common/Connections/JMS/TargetEsbConnection0/JNDIContextURL</td>
<td>Specifies the URL to the JNDI service provider used for target operation requests sent to backend services in ESB communication domain. Default is tibjmsnaming://localhost:7222</td>
</tr>
</tbody>
</table>
Table 14  Core Engine Properties

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tibco.clientVar.Common/Connections/JMS/TargetEsbConnection0/TopicConnectionFactoryName</td>
<td>Specifies the name of TopicConnectionFactory object stored in JNDI. This object is used to create a topic connection with ESB services at the target operation side. Default is TopicConnectionFactory</td>
</tr>
<tr>
<td>tibco.clientVar.Common/Connections/JMS/TargetEsbConnection0/QueueConnectionFactoryName</td>
<td>Specifies the name of QueueConnectionFactory object stored in JNDI. This object is used to create a queue connection with ESB services at the target operation side. Default is QueueConnectionFactory</td>
</tr>
<tr>
<td>tibco.clientVar.Common/Connections/JMS/TargetEsbConnection0/JNDIUsername</td>
<td>Specifies the user name for logging into the JNDI server in the ESB communication domain at the target operation side. If the JNDI provider does not require access control, this field can be empty. Default is admin</td>
</tr>
<tr>
<td>tibco.clientVar.Common/Connections/JMS/TargetEsbConnection0/JNDIPassword</td>
<td>Specifies the password for logging into the JNDI server in the ESB communication domain at the target operation side. If the JNDI provider does not require access control, this field can be empty.</td>
</tr>
<tr>
<td>tibco.clientVar.Common/Connections/JMS/TargetEsbConnection0/JMSUsername</td>
<td>Specifies the user name for logging into the EMS server in the ESB communication domain at the target operation side. Default is admin</td>
</tr>
<tr>
<td>tibco.clientVar.Common/Connections/JMS/TargetEsbConnection0/JMSPassword</td>
<td>Specifies the password for logging into the EMS server in the ESB communication domain at the target operation side.</td>
</tr>
</tbody>
</table>
EMS Server Connections for ESB Channel2 at Target Side.

- **tibco.clientVar.Common/Connections/JMS/TargetEsbConnection1/JMSProviderURL**
  Specifies the connection URL for the EMS Server used for target operation requests sent to backend services in ESB communication domain. ESB communication uses JMS transport with XML.
  Default is `tcp://localhost:7222`

- **tibco.clientVar.Common/Connections/JMS/TargetEsbConnection1/JNDIContextURL**
  Specifies the URL to the JNDI service provider used for target operation requests sent to backend services in ESB communication domain.
  Default is `tibjmsnaming://localhost:7222`

- **tibco.clientVar.Common/Connections/JMS/TargetEsbConnection1/TopicConnectionFactoryName**
  Specifies the name of TopicConnectionFactory object stored in JNDI. This object is used to create a topic connection with ESB services at the target operation side.
  Default is `TopicConnectionFactory`

- **tibco.clientVar.Common/Connections/JMS/TargetEsbConnection1/QueueConnectionFactoryName**
  Specifies the name of QueueConnectionFactory object stored in JNDI. This object is used to create a queue connection with ESB services at the target operation side.
  Default is `QueueConnectionFactory`

- **tibco.clientVar.Common/Connections/JMS/TargetEsbConnection1/JNDIUsername**
  Specifies the user name for logging into the JNDI server in the ESB communication domain at the target operation side. If the JNDI provider does not require access control, this field can be empty.
  Default is `admin`

- **tibco.clientVar.Common/Connections/JMS/TargetEsbConnection1/JNDIPassword**
  Specifies the password for logging into the JNDI server in the ESB communication domain at the target operation side. If the JNDI provider does not require access control, this field can be empty.
Table 14  Core Engine Properties

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tibco.clientVar.Common/Connections/JMS/TargetEsbConnection1/JMSUsername</td>
<td>Specifies the user name for logging into the EMS server in the ESB communication domain at the target operation side. Default is admin</td>
</tr>
<tr>
<td>tibco.clientVar.Common/Connections/JMS/TargetEsbConnection1/JMSPassword</td>
<td>Specifies the password for logging into the EMS server in the ESB communication domain at the target operation side.</td>
</tr>
<tr>
<td>tibco.clientVar.Common/Connections/JMS/TargetEsbConnection2/JMSProviderURL</td>
<td>Specifies the connection URL for the EMS Server used for target operation requests sent to backend services in ESB communication domain. ESB communication uses JMS transport with XML. Default is tcp://localhost:7222</td>
</tr>
<tr>
<td>tibco.clientVar.Common/Connections/JMS/TargetEsbConnection2/JNDIContextURL</td>
<td>Specifies the URL to the JNDI service provider used for target operation requests sent to back end services in ESB communication domain. Default is tibjmsnaming://localhost:7222</td>
</tr>
<tr>
<td>tibco.clientVar.Common/Connections/JMS/TargetEsbConnection2/TopicConnectionFactoryName</td>
<td>Specifies the name of TopicConnectionFactory object stored in JNDI. This object is used to create a topic connection with ESB services at the target operation side. Default is TopicConnectionFactory</td>
</tr>
<tr>
<td>tibco.clientVar.Common/Connections/JMS/TargetEsbConnection2/QueueConnectionFactoryName</td>
<td>Specifies the name of QueueConnectionFactory object stored in JNDI. This object is used to create a queue connection with ESB services at the target operation side. Default is QueueConnectionFactory</td>
</tr>
</tbody>
</table>
Table 14  Core Engine Properties

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tibco.clientVar.Common/Connections/JMS/TargetEsbConnection2/JNDIUsername</td>
<td>Specifies the user name for logging into the JNDI server in the ESB communication domain at the target operation side. If the JNDI provider does not require access control, this field can be empty. Default is admin</td>
</tr>
<tr>
<td>tibco.clientVar.Common/Connections/JMS/TargetEsbConnection2/JNDIPassword</td>
<td>Specifies the password for logging into the JNDI server in the ESB communication domain at the target operation side. If the JNDI provider does not require access control, this field can be empty.</td>
</tr>
<tr>
<td>tibco.clientVar.Common/Connections/JMS/TargetEsbConnection2/JMSUsername</td>
<td>Specifies the user name for logging into the EMS server in the ESB communication domain at the target operation side. Default is admin</td>
</tr>
<tr>
<td>tibco.clientVar.Common/Connections/JMS/TargetEsbConnection2/JMSPassword</td>
<td>Specifies the password for logging into the EMS server in the ESB communication domain at the target operation side.</td>
</tr>
</tbody>
</table>

Connection Parameters for SOAP JMS (Facade)

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tibco.clientVar.Common/Connections/JMS/SOAPConnection_Facade/JMSProviderURL</td>
<td>Specifies the connection URL for the EMS Server used for client requests when SOAP JMS transport is used at the facade side. Default is tcp://localhost:7222</td>
</tr>
<tr>
<td>tibco.clientVar.Common/Connections/JMS/SOAPConnection_Facade/JNDIContextURL</td>
<td>Specifies the URL to the JNDI service provider used for client requests when SOAP JMS transport is used at the facade side. Default is tibjmsnaming://localhost:7222</td>
</tr>
</tbody>
</table>
### Table 14  Core Engine Properties

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tibco.clientVar.Common/Connections/JMS/SOAPConnection_Facade/TopicConnectionFactoryName</td>
<td>Specifies the name of TopicConnectionFactory object stored in JNDI. This object is used to create a topic connection with JMS application at facade side. Default is TopicConnectionFactory</td>
</tr>
<tr>
<td>tibco.clientVar.Common/Connections/JMS/SOAPConnection_Facade/QueueConnectionFactoryName</td>
<td>Specifies the name of QueueConnectionFactory object stored in JNDI. This object is used to create a queue connection with JMS application at facade side. Default is QueueConnectionFactory</td>
</tr>
<tr>
<td>tibco.clientVar.Common/Connections/JMS/SOAPConnection_Facade/JNDIUsername</td>
<td>Specifies the user name for logging into the JNDI server used to send client requests at facade side. If the JNDI provider does not require access control, this field can be empty. Default is admin</td>
</tr>
<tr>
<td>tibco.clientVar.Common/Connections/JMS/SOAPConnection_Facade/JNDIPassword</td>
<td>Specifies the password for logging into the JNDI server used to send client requests at facade side. If the JNDI provider does not require access control, this field can be empty.</td>
</tr>
<tr>
<td>tibco.clientVar.Common/Connections/JMS/SOAPConnection_Facade/JMSUsername</td>
<td>Specifies the user name for logging into the EMS server used to send client requests at facade side. Default is admin</td>
</tr>
<tr>
<td>tibco.clientVar.Common/Connections/JMS/SOAPConnection_Facade/JMSPassword</td>
<td>Specifies the password for logging into the EMS server used to send client requests at facade side.</td>
</tr>
</tbody>
</table>
Table 14 Core Engine Properties

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tibco.clientVar.Common/Connections/JMS/SOAPConnection_Target/JMSProviderURL</td>
<td>Specifies the connection URL for the EMS Server when SOAP JMS transport is used at the target operation side. Default is tcp://localhost:7222</td>
</tr>
<tr>
<td>tibco.clientVar.Common/Connections/JMS/SOAPConnection_Target/JNDIContextURL</td>
<td>Specifies the URL to the JNDI service provider when SOAP JMS transport is used at the target operation side. Default is tibjmsnaming://localhost:7222</td>
</tr>
<tr>
<td>tibco.clientVar.Common/Connections/JMS/SOAPConnection_Target/TopicConnectionFactoryName</td>
<td>Specifies the name of TopicConnectionFactory object stored in JNDI. This object is used to create a topic connection with JMS application at target operation side. Default is TopicConnectionFactory</td>
</tr>
<tr>
<td>tibco.clientVar.Common/Connections/JMS/SOAPConnection_Target/QueueConnectionFactoryName</td>
<td>Specifies the name of QueueConnectionFactory object stored in JNDI. This object is used to create a queue connection with JMS application at target operation side. Default is QueueConnectionFactory</td>
</tr>
<tr>
<td>tibco.clientVar.Common/Connections/JMS/SOAPConnection_Target/JNDIUsername</td>
<td>Specifies the user name for logging into the JNDI server when SOAP JMS transport is used at the target operation side. If the JNDI provider does not require access control, this field can be empty. Default is admin</td>
</tr>
<tr>
<td>tibco.clientVar.Common/Connections/JMS/SOAPConnection_Target/JNDIPassword</td>
<td>Specifies the password for logging into the JNDI server when SOAP JMS transport is used at the target operation side. If the JNDI provider does not require access control, this field can be empty.</td>
</tr>
</tbody>
</table>
### Table 14  Core Engine Properties

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tibco.clientVar.Common/Connections/JMS/SOAPConnection_Target/JMSUsername</td>
<td>Specifies the user name for logging into the EMS server when SOAP JMS transport is used at the target operation side. Default is <strong>admin</strong></td>
</tr>
<tr>
<td>tibco.clientVar.Common/Connections/JMS/SOAPConnection_Target/JMSPassword</td>
<td>Specifies the password for logging into the EMS server when SOAP JMS transport is used at the target operation side.</td>
</tr>
<tr>
<td>tibco.clientVar.DefaultImplementation/Connections/HTTP/FacadeHTTPConnection/Port</td>
<td>Specifies the port through which the gateway accepts HTTP requests from client. Default value: <strong>9222</strong></td>
</tr>
<tr>
<td>tibco.clientVar.DefaultImplementation/Connections/HTTP/FacadeHTTPSSLConnection/Port</td>
<td>Specifies the port through which ASG accepts SSL enabled HTTP requests from client. Default value: <strong>9233</strong></td>
</tr>
<tr>
<td>tibco.clientVar.DefaultImplementation/Connections/HTTP/FacadeHTTPSSLConnection/useSSL</td>
<td>This is a boolean field which indicates if ssl should be enabled for accepting HTTPS requests. If set to <strong>true</strong>, SSL is enabled to accept the requests using HTTPS transport.</td>
</tr>
<tr>
<td>tibco.clientVar.DefaultImplementation/Connections/HTTP/FacadeHTTPSSLConnection/Identity</td>
<td>Specifies an identity resource which is used by FacadeHTTPSSLConnection HTTP shared resource to provide SSL properties.</td>
</tr>
</tbody>
</table>
| tibco.clientVar.DefaultImplementation/Connections/HTTP/FacadeHTTPSSLConnection/IdentityFileType | Specifies the type of SSL to be configured for the Identity resource. Supported types of SSL are:                                                                                                             
|                                                                              | — Identify File                                                                                                                                                                                              
|                                                                              | — Certificate/Key URL                                                                                                                            |
### Core Engine Properties

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tibco.clientVar.DefaultImplementation/Connections/HTTP/FacadeHTTPSSLConnection/Identity URL</td>
<td>Specifies the URL to the identity file if the Identity file type is of the type Identity File. For example, C:\keystore.jks</td>
</tr>
<tr>
<td>tibco.clientVar.DefaultImplementation/Connections/HTTP/FacadeHTTPSSLConnection/Identity FilePassword</td>
<td>Specifies the password to the identity file if the Identity file type is of the type Identity File.</td>
</tr>
<tr>
<td>tibco.clientVar.DefaultImplementation/Connections/HTTP/FacadeHTTPSSLConnection/CertificateURL</td>
<td>Specifies the URL to the certificate file if the Identity file type is of the type Certificate/Private Key. For example, C:\mydomain.csr</td>
</tr>
<tr>
<td>tibco.clientVar.DefaultImplementation/Connections/HTTP/FacadeHTTPSSLConnection/KeyURL</td>
<td>Specifies the URL to the private key in certificate file if the Identity file type is of the type Certificate/Private Key. For example, C:\keystore.jks</td>
</tr>
<tr>
<td>tibco.clientVar.DefaultImplementation/Connections/HTTP/FacadeHTTPSSLConnection/KeyPassword</td>
<td>Specifies the password for the private key if the identity file type is of the type Certificate/Private Key.</td>
</tr>
<tr>
<td>tibco.clientVar.DefaultImplementation/Connections/HTTP/FacadeHTTPSSLConnection/Requires ClientAuthentication</td>
<td>Indicates a boolean flag to enable or disable mutual SSL authentication for https transport between the client and the gateway.</td>
</tr>
</tbody>
</table>
### Table 14 Core Engine Properties

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tibco.clientVar.DefaultImplementation/Connections/HTTP/OAuthWebappsConnection/Host</td>
<td>Specifies the host IP address of the API Exchange Gateway OAuth Server.  Default value is 9322.</td>
</tr>
<tr>
<td>tibco.clientVar.DefaultImplementation/Connections/HTTP/OAuthWebappsConnection/Port</td>
<td>Specifies the non-SSL port number of the API Exchange Gateway OAuth Server.  Default value is 9322.</td>
</tr>
</tbody>
</table>
| tibco.clientVar.DefaultImplementation/Connections/HTTP/OAuthWebappsConnection/Webapps | • Specifies the location of the OAuth web application.  
• User should change only if you want to add custom login page or access grant page to the OAuth server. |
| tibco.clientVar.DefaultImplementation/Connections/HTTP/OAuthWebappsSSLConnection/Port | Specifies the SSL port number of the API Exchange Gateway OAuth Server.  Default value is 9333.                                               |
| tibco.clientVar.DefaultImplementation/Connections/HTTP/OAuthWebappsSSLConnection/useSSL | This is a boolean field which indicates if ssl should be enabled for accepting HTTPS requests for OAuth APIs and servlets.  If set to true, SSL is enabled to accept the requests using HTTPS transport for OAuth server.  Default value is true |
| tibco.clientVar.DefaultImplementation/Connections/HTTP/OAuthWebappsSSLConnection/Identity | Specifies an identity resource which is used by OAuthWebappsSSLConnection HTTP shared resource to provide SSL properties for OAuth servlets and APIs.  Default value: /DefaultImplementation/SharedResources/HTTP/OAuthIdentityResource.id |
**Table 14  Core Engine Properties**

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
</tr>
</thead>
</table>
| tibco.clientVar.DefaultImplementation/Connections/HTTP/OAuthWebappsSSLConnection/IdentityFileType | Specifies the type of SSL to be configured for the Identity resource set for OAuth server. Supported types of SSL are:  
  — Identify File
  — Certificate/Key URL
  — Default value is: Certificate/Private |
| tibco.clientVar.DefaultImplementation/Connections/HTTP/OAuthWebappsSSLConnection/IdentityURL | Specifies the URL to the identity file if the Identity file type is of the type Identity File for OAuth server SSL connection.  
  For example, C:\keystore.jks |
| tibco.clientVar.DefaultImplementation/Connections/HTTP/OAuthWebappsSSLConnection/IdentityFilePassword | Specifies the password to the identity file used for OAuth server SSL connection if the Identity file type is of the type Identity File. |
| tibco.clientVar.DefaultImplementation/Connections/HTTP/OAuthWebappsSSLConnection/CertificateURL | Specifies the URL to the certificate file used for OAuth server SSL connection if the Identity file type is of the type Certificate/Private Key. |
| tibco.clientVar.DefaultImplementation/Connections/HTTP/OAuthWebappsSSLConnection/KeyURL | Specifies the URL to the private key in certificate file used for OAuth server SSL connection if the Identity file type is of the type Certificate/Private Key. |
| tibco.clientVar.DefaultImplementation/Connections/HTTP/OAuthWebappsSSLConnection/KeyPassword | Specifies the password for the private key used for OAuth server SSL connection if the identity file type is of the type Certificate/Private Key. |
Table 14 Core Engine Properties

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>tibco.clientVar.DefaultImplementation/Connections/HTTP/OAuthWebappsSSLConnection/RequiresClientAuthentication</code></td>
<td>Indicates a boolean flag to enable or disable mutual SSL authentication for https transport used for OAuth server requests from the requestor.</td>
</tr>
</tbody>
</table>

Enable JMS Transport for Central Logger Properties

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>tibco.clientVar.ASG/Logging/transport</code></td>
<td>Specifies the transport to be used between the core engine and central logger component to log the transactions.</td>
</tr>
<tr>
<td><code>tibco.clientVar.Common/Connections/JMS/CL_JMSConnection/JMSProviderURL</code></td>
<td>Specifies the URL to connect to the Enterprise Message Service (EMS) or a JMS server.</td>
</tr>
<tr>
<td>Example: tcp://localhost:7222</td>
<td></td>
</tr>
<tr>
<td><code>tibco.clientVar.Common/Connections/JMS/CL_JMSConnection/JNDIContextURL</code></td>
<td>Specifies a JNDI connection URL to look up a JMS server.</td>
</tr>
<tr>
<td>Example: tibjmsnaming://localhost:7222</td>
<td></td>
</tr>
<tr>
<td><code>tibco.clientVar.Common/Connections/JMS/CL_JMSConnection/TopicConnectionFactoryName</code></td>
<td>Specifies the name of TopicConnectionFactory object stored in JNDI. This object is used to create a topic connection with JMS server for the Central logger.</td>
</tr>
<tr>
<td>Default is TopicConnectionFactory</td>
<td></td>
</tr>
<tr>
<td><code>tibco.clientVar.Common/Connections/JMS/CL_JMSConnection/QueueConnectionFactoryName</code></td>
<td>Specifies the name of QueueConnectionFactory object stored in JNDI. This object is used to create a queue connection with JMS server for the Central logger.</td>
</tr>
<tr>
<td>Default is QueueConnectionFactory</td>
<td></td>
</tr>
</tbody>
</table>
### Table 14  Core Engine Properties

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tibco.clientVar.Common/Connections/JMS/CL_JMSSConnection/JNDIUsername</td>
<td>Specifies the user name to use when logging into the JNDI server. If the JNDI provider does not require access control, this field can be empty. Example, admin</td>
</tr>
<tr>
<td>tibco.clientVar.Common/Connections/JMS/CL_JMSSConnection/JNDIPassword</td>
<td>Specifies the password for logging into the JNDI server. If the JNDI provider does not require access control, this field can be empty.</td>
</tr>
<tr>
<td>tibco.clientVar.Common/Connections/JMS/CL_JMSSConnection/JMSUsername</td>
<td>Specifies the user name to use to authenticate to the JMS server.</td>
</tr>
<tr>
<td>tibco.clientVar.Common/Connections/JMS/CL_JMSSConnection/JMSPassword</td>
<td>Specifies the password to use to authenticate to the JMS server.</td>
</tr>
<tr>
<td>tibco.clientVar.Common/Connections/JMS/CL_JMSSConnection/TransactionReportDestinationName</td>
<td>Specifies the name of the JMS destination to which the transaction reports are sent to the Central Logger by the gateway core engine. For example, asg.cl.transaction.queue</td>
</tr>
<tr>
<td>tibco.clientVar.Common/Connections/JMS/CL_JMSSConnection/TransactionReportDestinationType</td>
<td>Specifies the type of the JMS destination to which the transaction reports are sent to the Central Logger by the gateway core engine. Possible values are queue or topic. Default value is queue. The Central logger always listens on a queue. If the value of destination type set to topic, the JMS administrator must configure a bridge between the topic and the queue.</td>
</tr>
</tbody>
</table>
### Core Engine Properties

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tibco.clientVar.ConfigApi.ConfigurationBackupProperty</td>
<td>Specifies the maximum number of latest revisions of the configuration projects to be kept.</td>
</tr>
</tbody>
</table>

### OAuth Server Related Settings

#### Properties For OAuth Data Space

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tibco.clientVar.oauth.dataspace.metaspace.name</td>
<td>Specifies the metaspace name used by the OAuth server. Default value is <code>ASG-OAuth-Tokens</code></td>
</tr>
<tr>
<td>tibco.clientVar.oauth.dataspace.local.discovery</td>
<td>Specifies the discovery URL for this OAuth instance of the metaspace discovers the current metaspace members. For example, tcp://localhost:6300/</td>
</tr>
<tr>
<td>tibco.clientVar.oauth.dataspace.local.listen</td>
<td>Specifies the listening URL for this OAuth instance of the metaspace. For example, tcp://localhost:6300/</td>
</tr>
<tr>
<td>tibco.clientVar.oauth.dataspace.remote.discovery</td>
<td>Specifies the remote discovery URL for this OAuth instance of the metaspace discovers the current metaspace members.</td>
</tr>
<tr>
<td>tibco.clientVar.oauth.dataspace.remote.listen</td>
<td>Specifies the remote listening URL for this OAuth instance of the metaspace.</td>
</tr>
<tr>
<td>tibco.clientVar.oauth.dataspace.load.batch.size</td>
<td>Specifies the maximum number of entries to return when query data such as access token.</td>
</tr>
</tbody>
</table>
Table 14  Core Engine Properties

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Properties For OAuth Persister</strong></td>
<td></td>
</tr>
<tr>
<td>tibco.clientVar.oauth.dataspac.persister.store</td>
<td></td>
</tr>
<tr>
<td>• Defines the type of persistence store. Possible values are:</td>
<td></td>
</tr>
<tr>
<td>- InMemory</td>
<td></td>
</tr>
<tr>
<td>- Database</td>
<td></td>
</tr>
<tr>
<td>If the Database is set, you must define the properties for database server connection.</td>
<td></td>
</tr>
<tr>
<td><strong>Properties For OAuth Server Persister Store of Database Type</strong></td>
<td></td>
</tr>
<tr>
<td>tibco.clientVar.oauth.dataspac.persister.jdbc.driver</td>
<td></td>
</tr>
<tr>
<td>Specifies the database jdbc driver when Database is used as OAuth persistence store.</td>
<td></td>
</tr>
<tr>
<td>tibco.clientVar.oauth.dataspac.persister.jdbc.url</td>
<td></td>
</tr>
<tr>
<td>Specifies the jdbc url for the database server when Database is used as OAuth persistence store.</td>
<td></td>
</tr>
<tr>
<td>tibco.clientVar.oauth.dataspac.persister.jdbc.username</td>
<td></td>
</tr>
<tr>
<td>Specifies the user to connect to the database server when Database is used as OAuth persistence store.</td>
<td></td>
</tr>
<tr>
<td>tibco.clientVar.oauth.dataspac.persister.jdbc.password</td>
<td></td>
</tr>
<tr>
<td>Specifies the password of the user to connect to the database server when Database is used as OAuth persistence store.</td>
<td></td>
</tr>
<tr>
<td><strong>Properties For OAuth Adapters</strong></td>
<td></td>
</tr>
<tr>
<td>tibco.clientVar.oauth.adapter.resource.home</td>
<td></td>
</tr>
<tr>
<td>Specifies the directory from where the custom adapters loads the resources such as properties file used by adapters. This directory location is relative to the ASG_HOME.</td>
<td></td>
</tr>
<tr>
<td>For example, if the value is specified as /examples/Adapters/resources, the custom adapter looks for the resources such as properties file in the directory ASG_HOME/examples/Adapters/resources.</td>
<td></td>
</tr>
</tbody>
</table>
Run Time Properties of Central Logger

The properties for the central logger are defined in the `asg_cl.properties` file located in `ASG_CONFIG_HOME` directory.

Following properties can be defined, if the default values do not serve your purpose:

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
</tr>
</thead>
</table>
| tibco.clientVar.CL/Logging/fileFilter | Lists the name of the facade operation as a pipe (`|`) separated string. The logs from these operations are logged to files instead of database. For example: 
  | ping|test |
Table 15  Central Logger Properties

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tibco.clientVar.CL/Logging/files/directory</td>
<td>Specifies the directory name for transactions log file used by the core engine to record the transactions.  Default is $ASG_HOME/bin/logs$</td>
</tr>
<tr>
<td>tibco.clientVar.CL/Logging/files/transactions</td>
<td>Specifies the name of the transactions log file used by the core engine to record the transactions. This is used only for the records which is filtered by the tibco.clientVar.CL/Logging/fileFilter property. Default is trans_log.txt</td>
</tr>
<tr>
<td>tibco.clientVar.CL/Logging/files/maxcount</td>
<td>Specifies the maximum number of log files to keep on roll over for the transactions log file.  Default is 3</td>
</tr>
<tr>
<td>tibco.clientVar.CL/Logging/files/maxsize</td>
<td>Specifies the maximum size of the log file, in bytes, at which to roll-over to the next log file.  Default is 1000</td>
</tr>
<tr>
<td>tibco.clientVar.ASG/Logging/tsFormat=yyyyMMdd HH:mm:ss.SSSS</td>
<td>Specifies the format of the log’s timestamp value.</td>
</tr>
<tr>
<td>tibco.clientVar.Common/Connections/RV/SubjectPrefix</td>
<td>Specifies the prefix for all Rendezvous subject names used between core engine and central logger.  Default is TIBCO.ASG.INTERNAL</td>
</tr>
</tbody>
</table>
Table 15  Central Logger Properties

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tibco.clientVar.ASG/CL/RV/RvService</td>
<td>Specifies the service parameter for Rendezvous used between core engine and central logger communication. Default is 7500</td>
</tr>
<tr>
<td>tibco.clientVar.ASG/CL/RV/RvDaemon</td>
<td>Specifies the daemon parameter for Rendezvous used between core engine and central logger communication. Default is tcp:7500</td>
</tr>
<tr>
<td>tibco.clientVar.ASG/CL/RV/RvNetwork</td>
<td>Specifies the network parameter for Rendezvous used between core engine and central logger communication.</td>
</tr>
</tbody>
</table>

**Database Connection Parameters**

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>be.dbconcepts.dburi</td>
<td>/CentralLogger/SharedResources/Logging Database.sharedjdbc</td>
</tr>
<tr>
<td>be.dbconcepts.connection.retry.count</td>
<td>Specifies the number of attempts to be made for the central logger to connect to the database server.</td>
</tr>
<tr>
<td>be.dbconcepts.connection.check.interval</td>
<td>Specifies the time interval (in seconds) for the central logger to connect to the database server. Default is 5</td>
</tr>
<tr>
<td>tibco.clientVar.CL/Database/Username</td>
<td>Specifies the username for the central logger to connect to the database server. Default is asguser</td>
</tr>
</tbody>
</table>
Table 15  Central Logger Properties

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tibco.clientVar.CL/Database/Password</td>
<td>Specifies the password for the central logger to connect to the database server.</td>
</tr>
<tr>
<td>tibco.clientVar.CL/Database/Driver</td>
<td>Specifies the database driver for the database server used by the central logger. For example: For oracle database, the value is defined as: oracle.jdbc.OracleDriver For MS SQL server, the value is defined as: com.microsoft.sqlserver.jdbc.SQLServerDriver</td>
</tr>
<tr>
<td>tibco.clientVar.CL/Database/Url</td>
<td>Specifies the database server connection URL for the central logger. For example: For oracle database, the value is defined as: jdbc:oracle:thin:@localhost:1521:asgstat For MS SQL server, the value is defined as: sqlserver://localhost:1433;databaseName=asgstat</td>
</tr>
<tr>
<td>tibco.clientVar.CL/Database/Schema</td>
<td>Specifies the database schema to be used by the central logger on the database server. For example: For oracle database, the value is defined as: asgstat For MS SQL server, the value is defined as: dbo</td>
</tr>
</tbody>
</table>

Enable JMS Transport for Central Logger Properties

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tibco.clientVar.ASG/Logging/transport</td>
<td>Specifies the transport to be used between the core engine and central logger component to log the transactions. Valid values are RV or JMS.</td>
</tr>
</tbody>
</table>
### Central Logger Properties

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
</tr>
</thead>
</table>
| `tibco.clientVar.Common/Connections/JMS/CL_JMSConnection/JMSProviderURL` | Specifies the URL to connect to the Enterprise Message Service (EMS) or a JMS server.  
   Example: tcp://localhost:7222                                     |
| `tibco.clientVar.Common/Connections/JMS/CL_JMSConnection/JNDIContextURL` | Specifies a JNDI connection URL to look up a JMS server.                      
   Example: tibjmsnaming://localhost:7222                             |
| `tibco.clientVar.Common/Connections/JMS/CL_JMSConnection/TopicConnectionFactoryName` | Specifies the name of `ConnectionFactory` object stored in JNDI. This object is used to create a topic connection with JMS server for the Central logger.  
   Default is `ConnectionFactory`                                       |
| `tibco.clientVar.Common/Connections/JMS/CL_JMSConnection/QueueConnectionFactoryName` | Specifies the name of `ConnectionFactory` object stored in JNDI. This object is used to create a queue connection with JMS server for the Central logger.  
   Default is `ConnectionFactory`                                        |
| `tibco.clientVar.Common/Connections/JMS/CL_JMSConnection/JNDIUsername` | Specifies the user name to use when logging into the JNDI server. If the JNDI provider does not require access control, this field can be empty.  
   Example, admin                                                        |
| `tibco.clientVar.Common/Connections/JMS/CL_JMSConnection/JNDIPassword` | Specifies the password for logging into the JNDI server. If the JNDI provider does not require access control, this field can be empty. |
| `tibco.clientVar.Common/Connections/JMS/CL_JMSConnection/JMSServerUsername` | Specifies the user name to use to authenticate to the JMS server.            |
### Table 15 Central Logger Properties

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tibco.clientVar.Common/Connections/JMS/CL_JMSConnection/JMSPassword</td>
<td>Specifies the password to use to authenticate to the JMS server.</td>
</tr>
<tr>
<td>tibco.clientVar.Common/Connections/JMS/CL_JMSConnection/TransactionReportDestinationName</td>
<td>Specifies the name of the JMS destination to which the transaction reports are sent to the Central Logger by the gateway core engine. For example, asg.cl.transaction.queue</td>
</tr>
<tr>
<td>tibco.clientVar.Common/Connections/JMS/CL_JMSConnection/TransactionReportDestinationType</td>
<td>Specifies the type of the JMS destination to which the transaction reports are sent to the Central Logger by the gateway core engine. Possible values are queue or topic. Default value is queue. The Central logger always listens on a queue. If the value of destination type set to topic, the JMS administrator must configure a bridge between the topic and the queue.</td>
</tr>
</tbody>
</table>
Chapter 4  Apache Module For TIBCO API Exchange Gateway

Topics

- Overview on page 94
- Installing Apache HTTP Server on page 95
- Configuring Apache HTTP Server Using HTTP Transport on page 96
- Secure Communications on page 99
- Two-way (Mutual) SSL Authentication on page 100
- Configuring SSL Communications on page 105
- Configuring One Way SSL Authentication on page 105
- Configuring Two-way (Mutual) SSL Authentication on page 106
- Configure Apache Server For Basic HTTP Authentication on page 114
- Configure Apache Module For RVRD Setup Through A Firewall (DMZ) on page 121
Overview

The Apache module terminates HTTP transports. It translates inbound HTTP(s) requests to Rendezvous messages, which it forwards to the facade component of the gateway operational layer. When service providers respond, the Apache module translates and forwards each reply to the appropriate requestor.

The gateway’s Apache module deploys and runs within an Apache HTTP server. The module uses the Rendezvous API to communicate with the gateway core engine.

The following components must be installed and operational for the Apache module to function:

- Apache HTTP server
- TIBCO Rendezvous
- API Exchange Gateway Apache Module
Installing Apache HTTP Server

To install the Apache HTTP server, see Installing Apache HTTP Server in the TIBCO API Exchange Gateway Installation guide.

Installing Apache HTTP Server With SSL

You can install the Apache HTTP server from the Apache website http://httpd.apache.org. Refer to the TIBCO API Exchange Gateway software readme for Apache server version requirements, and download the supported version. See Appendix section of TIBCO API Exchange Gateway Installation Guide for details.
Configuring Apache HTTP Server Using HTTP Transport

This section explains the configuration setup required to setup the Apache HTTP server using http transport.

1. Open the `APACHE_HOME/conf/httpd.conf` file for editing.

2. Add the following line in the file:
   
   ```
   Include ASG_HOME/modules/http_server/apache/mod_ASG.conf
   ```
   
   Replace the ASG_HOME to the directory location where TIBCO API Exchange Gateway product is installed.

3. In the same file, edit the value of listening port, if required:

   ```
   Listen 8080
   ```

   The default listening port is 80. You can change the value of the listening port, as required. The samples shipped with API Exchange Gateway product use the port value as 8080, so change the value of the listening port to run the samples.

4. Verify that you have setup the system environment variables on the machine where Apache HTTP server is installed. You should set the variables as below:

   **On Windows Platform**
   After the product is installed, verify that the `PATH` system variable includes `RV_HOME/bin`, where `RV_HOME` specifies the directory, where, TIBCO Rendezvous product is installed.

   **On Unix Platform**
   After the product is installed, ensure that the following environment variables are set correctly. Depending on the type of shell, you may have to use different commands to set these variables.

   - `RV_HOME`: Verify that this variable is set to the directory where TIBCO Rendezvous software is installed. If not, set it as follows:

     ```
     export RV_HOME=directory where the TIBCO Rendezvous product is installed.
     ```

   - `PATH`: Verify that this variable includes `RV_HOME/bin`. If not, set it as follows:

     ```
     export PATH=$RV_HOME/bin:$PATH
     ```
• **LD_LIBRARY_PATH**: Verify that this variable includes `$RV_HOME/lib`. If not, set it as follows:

```bash
export
LD_LIBRARY_PATH=$RV_HOME/lib:$RV_HOME/lib/64:$LD_LIBRARY_PATH
```
Running Apache HTTP Server

This section lists the basic commands to run Apache server. Refer to Apache HTTP server documentation for details.

**On Windows Platform**

To run the Apache HTTP server on Windows platform, perform the following steps:

1. Navigate to the directory:
   
   \( \text{APACHE\_HOME}/\text{bin} \)

2. Run the following command to run Apache HTTP server:

   \( \text{httpd.exe} \)

**On Unix Platform**

1. Navigate to the directory:

   \( \text{APACHE\_HOME}/\text{bin} \)

2. Run the following command to run Apache HTTP server:

   \( ./\text{apachectl start} \)
Secure Communications

TIBCO API Exchange Gateway supports transport for secure communications between the client requestor and Apache HTTP server. The transport uses Secure Sockets Layer (SSL) protocol for exchanging transactions in a secured way. SSL protocol uses signed digital certificates from a certificate authority (CA) for authentication.
Two-way (Mutual) SSL Authentication

When the client sends a request using transport, API Exchange Gateway supports the authentication of the clients based on digital certificates. This is known as mutual SSL authentication. Mutual SSL authentication is also referred to as client authentication as with client authentication, the client presents its certificate to the server after the server authenticates itself to the client.

API Exchange Gateway uses X.509 digital certificates for mutual SSL authentication, and also to authorize the client requests. In this case, the authorization of the request will be based on the trusted identity in the gateway processing pipeline. The trusted identity is represented by the digital certificate’s X.509 subject distinguished name or the certificate’s serial number.

API Exchange Gateway uses the Apache HTTP server to terminate the incoming HTTP and transports. The actual mutual SSL authentication is handled in the Apache module of the API Exchange Gateway. The Apache module authenticates each client request and extracts credentials from the X.509 certificate. The facade layer of the gateway uses those credentials to authorize the request before forwarding it to the gateway core engine.

Following setup is required for mutual SSL authentication:

Generate Keys and Certificates

See Generate Private Keys And Public Certificates with OpenSSL on page 547 section to generate the private keys and public certificates.

Configure SSL on Apache HTTP Server

Enabling SSL on Apache HTTP server provides a secure and encrypted connection to the client as Apache HTTP server authenticates itself to the client.

Enabling SSL communication requires the mod_ssl Apache module. The mod_ssl Apache module provides strong encryption using the secure sockets layer (SSL) and transport layer security (TLS) protocols for HTTP communication between a client and the Apache HTTP server. Using SSL/TLS, a private connection between the Apache HTTP server and the client is established. Data integrity is ensured and the client is able to authenticate the server. In this case, the Apache HTTP server sends its digital certificate to the client before any request is processed.
The `mod_ssl` Apache module on the server guarantees to the client that the server is an uniquely correct end point for the communication and the client uses the public key contained in the digital certificate to encrypt the communication between the client and the server. The `mod_ssl` Apache module does not implement the SSL/TSL protocols itself, but it acts as an interface between Apache and the OpenSSL library.

For configuration steps, see Configuring Mutual SSL on Apache HTTP Server on page 107.

### Configure Client Authentication With Digital Certificates On Apache HTTP Server

SSL communication lets the Apache HTTP server and clients communicate over an encrypted connection, which reduces the risk of exposing sensitive content in plain text. The secured communication using an encrypted connection ensures that the server always identifies itself to its clients, which guarantees that the server is the uniquely correct end point for the communication. However, if you want to authorize the service requests in TIBCO API Exchange Gateway, you will require the clients to authenticate themselves to the Apache HTTP server using its own client certificates.

The client authentication can be configured on the Apache HTTP server by setting the following Apache directives in the virtual host configuration for the SSL virtual server instance:

- **SSLVerifyClient**
  
  The `SSLVerifyClient` directive defines the verification type. Possible values are as follows:
  - `none`: Indicates that no client certificate is required at all.
  - `optional`: Indicates that client may present a valid certificate.
  - `require`: Indicates that the client has to present a valid certificate.

  The `require` value is used to ensure that the Apache HTTP server authenticates every client request before it forwards it to the TIBCO API Exchange Gateway.
  - `optional_no_ca`: Indicates that client may present a valid certificate but does not have to be successfully verified.

- **SSLVerifyDepth**

  The `SSLVerifyDepth` directive specifies the depth of the certificate issuer chains verification. If the server does not find a trusted CA within this depth, it declares the certificate invalid. The depth actually is the maximum number of intermediate certificate issuers, i.e. the maximum number of CA certificates
that will be followed while verifying the client certificate. For example, Depth 0 (zero) means all clients must present certificates that are self-signed and present in the server’s collection of trusted certificates. Depth 1 means that client certificates may be either self-signed (as above), or signed by a trusted CA. The default value is 1.

- **SSLCACertificatePath**

  The SSLCACertificatePath specifies the path to the directory containing the separate files for each certificate authority’s digital certificate. However, when you use this SSLCACertificatePath directive, the Apache HTTP server expects to have each file be named with the hash of the CA certificate that is in it, followed by a period and a sequence number that starts at 0 and gets incremented for each file. It does so for efficiency reasons as it would be inefficient, especially when you have a large number of CA certificates, to open and read every file in the directory every time it needs to find a specific certificate.

- **SSLCACertificateFile**

  The SSLCACertificateFile directive specifies the name and location of a single certificate file that contains all CA certificates.

Setting the SSLCACertificateFile directive is easy to configure. As the number of trusted certificate authorities grows large, it can be difficult or error prone to add, replace, or remove CA certificates in this file. When the number of trusted certificate authorities is large, using the SSLCACertificatePath directive is recommended.

See Configuring Client Authentication with Digital Certificates on Apache HTTP Server on page 109 for configuration steps.

### Configure Client Certificate Identification Details On Apache HTTP Server

After setting up client authentication configuration on the Apache HTTP server, configure the identity details of the authenticated client on the Apache HTTP server so that they can be forwarded as custom HTTP headers to the TIBCO API Exchange Gateway core engine. The gateway core engine matches the client identification details from the HTTP headers with the identification details configured on the gateway user interface.

By default, the Apache HTTP server does not forward the authenticated client identity to TIBCO API Exchange Gateway. Therefore, all requests that TIBCO API Exchange Gateway receives through this channel get identified as being sent by the anonymous user.
TIBCO API Exchange Gateway retrieves the client’s identity from two custom HTTP header fields `CAissuer` and `SerialNumber`. The `CAissuer` field contains the distinguished name of the certificate authority that issued the client certificate as a unique means of identification of that certificate authority. The `SerialNumber` HTTP header contains the unique identification of the client in the context of a TIBCO API Exchange Gateway partner. This could either be the client certificate’s serial number, or the certificate’s subject distinguished name.

When TIBCO API Exchange Gateway receives a request that includes these two HTTP header fields, it identifies the partner by matching the values in these two HTTP header fields with the `Partner CA Issuer` and `Partner Serial Number` fields in the partner tab configuration of the TIBCO API Exchange Gateway User interface. The `Partner CA Issuer` contains the identity realm and the `Partner Serial Number` represents the partner’s identity for that realm.

The serial number uniquely identifies a specific certificate that the partner uses to identify itself. If you use the serial number for the partner configuration, then the partner configuration needs to be updated to reflect a new serial number in case a partner’s certificate has expired.

When the partner renews its certificate after the certificate expiration, it’s not required to update the TIBCO API Exchange Gateway partner configuration in case you use the subject distinguished name.

Following configuration setup is required on the Apache HTTP server so that TIBCO API Exchange Gateway can identify a partner based on the `CAissuer` and `SerialNumber` HTTP header fields:

- **Configure SSL engine options**
  Configure SSL engine options to export the standard SSL/TLS related `SSL_*` environment variables. This makes the client certificate information available in the Apache server for further reference in request processing steps. This includes the issuer distinguished name, the certificate serial number and the subject distinguished name.

- **Enable `mod_headers` module**
  Enable `mod_headers` module to control and modify HTTP request and response headers.

- **Set RequestHeader directives**
  Set `RequestHeader` directives that add specific `CAissuer` and `SerialNumber` HTTP headers to the incoming request and populate it with the values retrieved from the SSL environment variables including issuer distinguished name, the certificate serial number or the subject distinguished name.
See Configure Client Certificate Identification Details On Apache HTTP Server on page 102 for configuration steps.

Register Partners On API Exchange Gateway User Interface

Register the partners in TIBCO API Exchange Gateway with digital certificate identification details. This means that you can add partners on the gateway user interface with the identity information that the client sends with its digital certificate.

See Registering Partners On API Exchange Gateway User Interface on page 112 for configuration steps.
Configuring SSL Communications

This section explains the configuration setup required for SSL communications between client requestor and Apache HTTP server.

Before you can setup the configuration for SSL, you should install Apache server with SSL enabled. See Configuring Two-way (Mutual) SSL Authentication on page 106.

Configuring One Way SSL Authentication

To enable one way SSL authentication, perform the following steps:

1. Ensure that mod_ssl module is available in your Apache HTTP server installation.

2. Enable the mod_ssl module. To enable the mod_ssl module, perform the following steps:
   a. Open the APACHE_HOME/conf/httpd.conf file for editing.
   b. Uncomment the following directive in the httpd.conf file, if commented. If this directive does not exist, add it in the file:
      
      ```
      LoadModule ssl_module APACHE_ROOT/modules/mod_ssl.so
      ```
      
      where APACHE_ROOT is the actual path of the Apache HTTP server installation which must be SSL enabled.
   c. Uncomment the following line in the file:
      ```
      #Include conf/extra/httpd-ssl.conf
      ```
      
   d. Save the changes in the file.

3. Open the APACHE_HOME/conf/extra/httpd-ssl.conf file for editing.
   a. Set the values as below for the specified directives, if not already set:
      ```
      SSLEngine on
      SSLCertificateFile "Name_of_Server_public_certificate"
      SSLCertificateKeyFile "Name_of_Server_private_key"
      SSLCACertificateFile Name_of_CA_Certificate
      SSLVerifyClient none
      ```
   b. Set the Listen directive if you want to change the default port value for SSL requests:
      ```
      Listen listening_port_value
      ```
5. Save the changes made to the APACHE_HOME/conf/extra/httpd-ssl.conf file.

6. Import the CA certificate as specified in the SSLCACertificateFile directive of the Apache Server configuration.

7. Verify that SSL configuration is working by performing the following steps:
   a. Open a web browser window.
   b. Enter the following URL to verify connection to Apache server.

```
://machine_name:listening_port_value
```

For example, `://<machine-name>:8443`

8. Verify that the connection to Apache server is successful

### Configuring Two-way (Mutual) SSL Authentication

To enable authentication and X.509-based authorization you must configure both the Apache HTTP server and the module for Apache server in API Exchange Gateway software.

Before you start configuring mutual authentication and authorization, see Pre Requisites For Mutual SSL Setup on page 106.

### Pre Requisites For Mutual SSL Setup

- Apache HTTP server with `mod_ssl` module. Refer to the TIBCO API Exchange Gateway software readme for Apache server version information. Verify that you have set it up as specified in Installing Apache HTTP Server on page 95.

- RSA private key in PEM format to be used by the Apache HTTP server.

- Digital certificate in PEM format that identifies the Apache HTTP server and includes the public key that corresponds to the Apache HTTP server’s private key.

- Digital certificate chain in PEM format for each certificate authority that will be trusted by the Apache HTTP server. This certificate chain is used to verify the digital certificates presented by the clients as part of the authentication step.

- The issuer distinguished name and the subject distinguished name (or optionally the certificate’s serial number) of each certificate that a client uses to identify himself to the Apache HTTP server.
Configuring Mutual SSL on Apache HTTP Server

To use the mod_ssl module with Apache HTTP server, you must ensure the following:

- OpenSSL is installed on the Apache server’s host computer.
- A RSA private key in PEM format used by Apache HTTP server.
- A digital certificate in PEM format that identifies the Apache HTTP server and includes the public key that corresponds to the Apache HTTP server’s private key. To ensure the integrity of the certificate, it must be signed by a party that every client trusts. For details, see Generate Private Keys And Public Certificates with OpenSSL on page 547.

To configure mutual SSL on Apache HTTP server, do the following steps:

1. Ensure that mod_ssl module is available in your Apache HTTP server installation.

2. Ensure that mod_ssl module is enabled on the Apache HTTP server installation. To enable the mod_ssl module, do the following steps:
   a. Open the APACHE_HOME/conf/httpd.conf file for editing.
   b. Uncomment the following directive in the httpd.conf file, if commented. If this directive does not exist, add it in the file:

   ```
   LoadModule ssl_module APACHE_ROOT/modules/mod_ssl.so
   ```
   where APACHE_ROOT is the actual path of the Apache HTTP server installation which must be SSL enabled.
   c. Uncomment the following line in the file:

   ```
   #Include conf/extra/httpd-ssl.conf
   ```
   d. Save the changes in the file.

3. Open the APACHE_HOME/conf/extra/httpd-ssl.conf file for editing.
   a. Set the values as below for the specified directives, if not already set:

   ```
   SSLEngine on
   SSLCertificateFile "Name_of_Server_public_certificate"
   SSLCertificateKeyFile "Name_of_Server_private_key"
   SSLCACertificateFile Name_of_CA_Certificate
   SSLVerifyClient require
   SSLVerifyDepth 1
   ```
   For example, the example values are set as below:

   ```
   SSLCertificateFile "C:/apache2/conf/server.crt"
   SSLCertificateKeyFile "C:/apache2/conf/server.key"
   ```
SSLCACertificateFile "/etc/ssl/certs/myrootca.crt"

- For details on each of the SSL specific properties, refer to Apache SSL documentation.
- The value of SSLVerifyDepth is set as 1 as you are doing only one level of authentication. You have configured only 1 CA which is the root CA.

b. Set the Listen directive if you want to change the default port value for SSL requests:

Listen listening_port_value

c. Ensure that the global SSL configuration directives are defined as below:

```
LoadModule ssl_module C:/apache2/modules/mod_ssl.so
SSLRandomSeed startup builtin
SSLRandomSeed connect builtin
AddType application/x-x509-ca-cert .crt
AddType application/x-pkcs7-crl .crl
SSLPassPhraseDialog builtin
SSLSessionCache "shmcb:c:/apache2/logs/ssl_scache(512000)"
SSLSessionCacheTimeout 300
SSLMutex default
```

d. Ensure that the SSL related directives are defined as below that should be set per virtual host instance basis:

```
SSLEngine on
SSLProtocol all -SSLv2
SSLCipherSuite HIGH:MEDIUM:!aNULL:!MD5
SSLCertificateFile "Name_of_Server_public_certificate"
```
SSLCertificateKeyFile "Name_of_Server_private_key"

Make sure that you have copied the private key and server's digital certificate in the proper directories as specified in the SSLCertificateFile and SSLCertificateKeyFile directives.

4. Save the changes made to the APACHE_HOME/conf/extra/httpd-ssl.conf file.

5. Import the CA certificate as specified in the SSLCACertificateFile directive of the Apache Server configuration.

6. Verify that SSL configuration is working by performing the following steps:
   a. Open a web browser window.
   b. Enter the following URL to verify connection to Apache server.
      
      ://machine_name:listening_port_value
      
      For example,://<machine-name>:8443
   c. Verify that the connection to Apache server is successful.

When you invoke a secure connection to Apache server for the first time using HTTP/S transport, the browser displays a message indicating that the server presented is an untrusted certificate. You should accept the certificate by clicking Yes and the following message should be displayed: "it works!".

**Configuring Client Authentication with Digital Certificates on Apache HTTP Server**

To configure the client authentication with digital certificates on Apache HTTP server, perform the following steps:

1. Open APACHE_HOME/conf/extra/httpd-ssl.conf file in a text editor.

2. Ensure that following SSL directives are defined, if not set before:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSLCACertificatePath</td>
<td>Location of the directory containing the separate files for each certificate authority's digital certificate</td>
</tr>
<tr>
<td></td>
<td>For example, /etc/apache2/ssl.crt</td>
</tr>
<tr>
<td></td>
<td>Set one of the SSLCACertificatePath or SSLCACertificateFile directive, not both.</td>
</tr>
</tbody>
</table>
3. Save the changes and close the file.
4. Restart the Apache HTTP server, if already running.
5. Test the configuration changes as below:
   a. Import a client certificate into the web browser:
      Import a PKCS12 archive file into the browser which contains the client’s
      X.509 certificate, corresponding private key and the public certificates of all
      the CAs in the chain of trust. The archive file must be trusted by one of the
      CAs as configured on the Apache HTTP server. Use an Internet browser as
      below:

      **Using Firefox**
      — Open the web browser.
      — Go to Tools > Options on the browser menu.
      — Select Advanced tab on the new window.
      — Select Security tab on the new dialog.
      — Click the View Certificates button.
      — Click Import and follow the wizard to import the file.
**Using Internet Explorer**

- Open the web browser.
- Go to **Tools > Internet Options** on the browser menu.
- Select **Content** tab on the new dialog window.
- Go to **Certificates** section and click **Certificates** tab.
- Click **Import** and follow the wizard to import the file.
  
  b. After you have imported the PKCS12 file, open a browser window.
  
  c. Enter the following URL to verify secure connection to Apache server.

  \[://machine_name:listening_port_value\]

  For example,://<machine-name>:8443
  
  d. Verify that the connection to Apache server is successful.

**Forwarding Client Certificate Identification Details On Apache HTTP Server to Gateway Engine**

To configure the setup so that the Apache HTTP server forwards the client identification details to the TIBCO API Exchange Gateway engine, do the following steps:


2. Add the following line to enable the **mod_headers** module:
   
   ```
   LoadModule headers_module APACHE_HOME/modules/mod_headers.so
   SSLOptions +StdEnvVars
   ```

3. Set **RequestHeader** directives as below:
   
   ```
   RequestHeader add X-SSL_PROTOCOL "{%SSL_PROTOCOL}s"
   RequestHeader add CAissuer "{%SSL_CLIENT_I_DN}e"
   RequestHeader add SerialNumber "{%SSL_CLIENT_S_DN}e"
   ```

4. Save the changes and close the file.

5. Restart the Apache HTTP server.

6. Start the gateway core engine, if not already running. See **Starting TIBCO API Exchange Gateway Engine** on page 56.

7. Test the configuration changes to see that only requests from clients that authenticate themselves with a client certificate are forwarded to TIBCO API Exchange Gateway core engine. Since no partners are configured yet on the TIBCO API Exchange Gateway User Interface with the credentials specified in the certificate, the incoming request fails the identification with this configuration.
8. To test this configuration setup, do the following steps:
   a. Enter the following URL to submit a ping operation request:
   
      \[://machine_name:listening_port_value/ping\]
   
   b. If you have configured everything on the Apache HTTP server but have not registered the partner in the API Exchange Gateway yet, you should receive the response from API Exchange Gateway on the web browser as below:

   
   <asg:ErrorMessage> Partner null not identified 
   </asg:ErrorMessage> </asg:Error>

Registering Partners On API Exchange Gateway User Interface

To register the partner with the identity information, use the following steps:

1. Start the gateway user interface, if not already started. See Starting GUI on page 166 for details.

2. Select your TIBCO API Exchange Gateway configuration.

3. Click the PARTNER tab.

4. Add a new partner. See Partners on page 205 for details.

5. Set the following fields for the new partner:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| Partner Serial Number | Specifies the client’s identity that the Apache HTTP server will forward in the SerialNumber HTTP header of requests that are submitted by this partner. This can either be the certificate’s serial number or the subject distinguished name as used for the digital certificate. For example, the value can be defined as below:  
Partner Serial Number: /C=US/ST=California/L=Palo Alto/O=TIBCO Software Inc./OU=ActiveMatrix Service Gateway/CN=ASG Demo Client01/emailAddress=asgclient01@tibasg.co.pd |
6. Click the Partner Operations tab.

7. Define a ping operation (internal_ping) for new partner. See Facade Access on page 206 for details.

8. Save the configuration.

9. Start the gateway core engine, if not running already. See Starting TIBCO API Exchange Gateway Engine on page 56

10. Test the configuration as below:

   a. Open an internet web browser window.

   b. Enter the following URL to submit a ping operation request:

      ```
      ://machine_name:listening_port_value/ping
      ```

   c. Verify that you receive **ASG is alive** response from TIBCO API Exchange Gateway on the web browser.

Partner Serial Number and Partner Issuer CA fields contain distinguished names as defined by the X.509 standard. The X.509 standard defines the fields, field names, and abbreviations used to refer to the fields.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Partner Issuer CA</td>
<td>Specifies the realm in which the client’s identity is valid. This will always be the issuer distinguished name as used from the digital certificate for that partner. For example, the value can be defined as below: Partner Issuer CA : /C=US/ST=California/O=TIBCO Software Inc./OU=ActiveMatrix Service Gateway/CN=TIBCO ASG Certificate Authority/emailAddress=<a href="mailto:admin@tibasg.co.pd">admin@tibasg.co.pd</a></td>
</tr>
</tbody>
</table>
Configure Apache Server For Basic HTTP Authentication

Basic HTTP authentication requires the client to provide user name and password when it sends the request to Apache HTTP server. The basic authentication typically is used over transport as it does not provide any protection of the submitted credentials from the client to the Apache HTTP server.

To use the basic authentication by the client, it's recommended that you should use one way SSL for secure communication between the Apache HTTP server and the requestor. You should configure the following for secured communication:

— Configure Apache HTTP server for one way SSL. See Configuring One Way SSL Authentication on page 105. If you use the HTTP transport, you do not have to setup the SSL configuration on Apache HTTP server.

— Configure Apache HTTP server for Basic Authentication. See Enable Basic Authentication on Apache HTTP Server on page 114 for configuration steps.

— Configure the gateway core engine for Basic Authentication. See Configuring TIBCO API Exchange Gateway for Basic Authentication on page 119.

Configure Apache HTTP server for Basic Authentication

Perform the following tasks to configure the Apache server for Basic authentication:

**Task A  Enable Basic Authentication on Apache HTTP Server**

To enable the basic authentication on Apache HTTP server, perform the following steps:
1. Open a terminal window.
2. Navigate to \$ASG_HOME/modules/http_server/apache directory.
3. Edit the mod_ASG.conf file.
4. Search the following section in the file:

```xml
<Location/>
  SetHandler asg_rv_inbound_handler
  AsgSubject _LOCAL.asg.north.request
  AsgTimeout 30
</Location/>
```

TIBCO API Exchange Gateway User's Guide
5. Insert the configuration for a new location as below above the old location configuration:

```
<Location /asg/ba>
AuthType Basic
AuthName "ASG"
# (Following line optional, file is default)
AuthBasicProvider file
AuthUserFile /home/asg/apache/htpasswd/htpasswords
Require validuser
SetHandler asg_rv_inbound_handler
AsgSubject _LOCAL.asg.north.request
AsgTimeout 30
</Location>
```

6. Verify that the location changes are as below:

```
<Location /asg/ba>
AuthType Basic
AuthName "ASG"
# (Following line optional, file is default)
AuthBasicProvider file
AuthUserFile /home/asg/apache/htpasswd/htpasswords
Require validuser
SetHandler asg_rv_inbound_handler
AsgSubject _LOCAL.asg.north.request
AsgTimeout 30
</Location>
```

```
<Location />
SetHandler asg_rv_inbound_handler
AsgSubject _LOCAL.asg.north.request
AsgTimeout 30
</Location>
```
This location change enforces the user access with basic authentication.

7. Save the `mod_ASG.conf` file.

**Task B  Create A Password File For Apache HTTP Server**

1. Open a terminal window.
2. Navigate to `APACHE_HOME`.
3. Create a `htpasswd` sub-directory to store the password file. Create a blank `htpasswords` file, if not already there in this directory.
4. Navigate to the `APACHE_HOME/bin` directory.
5. Create a partner identity using the Apache `htpasswd` utility for the user `asgpartner01` with password `asgpartner01`, shown as below:

   For example, on windows platform:
   ```
   htpasswd APACHE_HOME\htpasswd\htpasswords asgpartner01
   New Password:asgpartner01
   Enter New Password:asgpartner01
   ```

6. Create a second partner identity using the Apache `htpasswd` utility for the user `asgpartner02` with password `asgpartner02`, shown as below:

   For example, on windows platform:
   ```
   htpasswd APACHE_HOME\htpasswd\htpasswords asgpartner02
   New Password:asgpartner02
   Enter New Password:asgpartner02
   ```

**Task C  Reload the Configuration File For Apache HTTP Server**

You should restart the Apache HTTP server to reload the configuration file.

1. Open a terminal window.
2. Navigate to `APACHE_HOME/bin` directory.
3. Stop the apache server, if already running.
4. Run the following command to start apache server:
   ```
   httpd.exe
   ```

**On Unix Platform**

1. Open a terminal window.
2. Navigate to the directory:
   
   \( \text{APACHE_HOME/bin} \)

3. Run the following command to restart the apache server:
   
   .\apachectl restart

---

**Example Use Case**

**Configuring Client(Requestor) For Basic Authentication**

This section explains an example use case how to configure a client for basic authentication. This example shows the configuration setup for TIBCO Designer when TIBCO BusinessWorks is used as a client to send the request to API Exchange Gateway. You must customize the changes accordingly if you are using a different client to send requests to API Exchange Gateway.

Ensure that the URI used by the client to send the request (for example, BusinessWorks as a client) matches with the URI used by the receiving gateway (for example, API Exchange Gateway Facade). This example uses "\( /\text{asg/ba/} \)" as the URI.

---

**Task A  Configure the Endpoint URL For Transport**

4. In this example, the client side http URL contains "\( /\text{asg/ba/} \)" string in the endpoint URL to access the server running the gateway core engine.

   For example, Refer to BookQuery Project shipped with API Exchange Gateway software at the \( \text{ASG_HOME/examples/BookQuery/BookQuery} \) location as follows:

   a. Open the \( \text{ASG_HOME/examples/BookQuery/BookQuery} \) project using the TIBCO Designer.

   b. Navigate to BookQuery > Client Process

   c. Double Click QueryByTitleClient process to open.

   d. Click SOAPRequestReply activity to open it.

   e. Click Transport Details tab.

   f. To use the basic authentication, change the endpoint URL as below:

      — From: http://127.0.0.1:9696/ServerProcesses/GetBooksByTitleEndpoint

      — To:

      http://127.0.0.1:9696/asg/ba/ServerProcesses/GetBooksByTitleEndpoint

5. Save changes to the configuration.
Task B  Create an Identity Resource

1. You must create an identity as set on the Apache HTTP server configuration. See Create A Password File For Apache HTTP Server on page 116.

   For example, you can create an identity as follows for the BookQuery project:
   a. Select Client Process node.
   b. Go to Resources > Add Resources > General > Identity
   c. Input the values for following fields:
      — Name: A string value (For example, MyIdentity)
      — Type: Select Username/Password from the drop down list box.
      — Username: username (username must match with the user created at Apache server. For example, asgpartner01. See Create A Password File For Apache HTTP Server on page 116).
      — Password: password (password must match with the password created at Apache server. For example, asgpartner01. See Create A Password File For Apache HTTP Server on page 116).

2. Save changes to the configuration.

Task C  Configure Identity For Transport

You must set the identity (username and password) for the http transport as follows:

   a. Double Click QueryByTitleClient process to open.
   b. Click SOAPRequestReply activity to open it.
   c. Click Transport Details tab.
   d. Check the Use Basic Authentication checkbox.
   e. Set the Identity field as below:
      — Click the browse button next to Identity field.
      — Select the Identity resource created as explained Create an Identity Resource on page 118.
      — Click OK to select the identity resource.
   f. Click Apply to save the changes.
   g. Save changes to the configuration.
Configuring TIBCO API Exchange Gateway for Basic Authentication

Perform the following steps to configure the basic authentication for the API Exchange Gateway:

**Task A  Start API Exchange Gateway User Interface**

See Starting GUI on page 166 to launch the gateway user interface.

**Task B  Add New Facade Operation**

1. Create a new configuration or select an existing configuration, as applicable.
2. Click **ROUTING > Facade Operations** tab.
3. Add a new operation or select an existing operation. See Facade Operations on page 193.
4. The Operation URI field contains "/asg/ba/" string in the endpoint as below for the facade operation:
   
   For example, /asg/ba/ServerProcesses/GetBooksByTitleEndpoint
   
   Note that the client uses the "/asg/ba" in the transport URL to send the request to the gateway core engine.
5. Save changes to the configuration.

**Task C  Add a New Partner Group**

1. Create a new configuration or select an existing configuration, as applicable.
2. Add a new partner group. See Partner Groups on page 206.
3. Save changes to the configuration.

**Task D  Add a New Partner**

1. Create a new configuration or select an existing configuration, as applicable.
2. Add a new partner as setup for the client. See Partners on page 205
   - See Create an Identity Resource on page 118 for the username. The partner name must match with the username of the identity resource.
   
   For example, you should setup the partner name as below:
   
   — Partner Name: asgpartner01
3. Save changes to the configuration.
Task E  Add a New Partner Group

Create or add a new partner group for the partner. See Partner Groups on page 206.

Task F  Add a New Facade Access

You must add a new facade access to authorize the asgpartner01 partner to access the operation configured for basic authentication.

Follow the steps as below:

1. Create a new configuration or select an existing configuration, as applicable.
2. Add a new facade access to authorize the partner to access the operation. See Facade Access on page 206.
3. Save changes to the configuration.

Task G  Add routing

You must add the routing data for the partner and operation created in Add a New Partner on page 119 and Add New Facade Operation on page 119 sections so that the gateway core engine can route the facade request to the appropriate target operation. See Routing on page 203.
By default, the gateway core engine uses rvd daemon to communicate with the Apache module. The Apache module receives client requests directly from the Internet and performs SSL validation. By placing a firewall between the DMZ (De-Militarized Zone) and the rest of your system, you can protect against the threat of malicious communications and provide stronger security.

When the services are exposed to an un-secured network (such as the internet) it is usual to define different security zones with restricted connections allowed between them. Requests from the outside world are terminated behind a firewall in a demilitarized zone (DMZ). Applications running in the DMZ are not allowed to initiate connections into the more secured zones. In some cases defense-in-depth is applied and multiple DMZs are used.

TIBCO Rendezvous routing daemon can be configured to forward the Rendezvous messages from the DMZ network through the firewall to the internal network where the TIBCO API Exchange Gateway components are deployed.

This section explain the deployment topology illustrating the deployment of Apache HTTP Server separately in the DMZ and all other gateway components in a secure network. See Figure 5, Apache HTTP Server in DMZ and Other Components in Secure Network, page 122.

Rendezvous transport can be configured so that all connections between the gateway core engine (asg_core) and the Apache HTTP server are instantiated from the internal secure zone into the DMZ (i.e. the server running asg_core).

In this layout only, the authentication is carried out in the DMZ and the minimal possible gateway configuration needs to be available in the DMZ.

If you want to deploy TIBCO API Exchange Gateway in the DMZ setup between the firewalls, you will need to configure RVRD when the machines are in different subnets of the network.
This section explains the steps to setup the TIBCO API Exchange Gateway in a DMZ environment:

1. Install TIBCO Rendezvous on the Machine 1. Refer to the readme file located in `TIBCO_HOME` directory for the supported version of TIBCO Rendezvous.

2. Install Apache HTTP Server on the Machine 1. Refer to the readme file located in `TIBCO_HOME` directory for the supported version of Apache HTTP server.


4. Configure RVRD between Machine1 and Machine 2 (Machine 1 is outside the firewall and Machine 2 is inside the firewall) so that they can send and receive Rendezvous messages to each other. The subject used to configure RVRD should match with the value specified in the `AsgSubject` parameter defined in the `mod_ASG.conf` file located in the Apache Server Installation. See `TIBCO Rendezvous Administration` for detailed instructions to configure `rvrd` or `rvd`, as required.

5. Install the Apache module on Machine 1 as below:
   b. Browse to `ASG_HOME/modules/http_server/apache` directory.
   c. Copy the `mod_ASG.conf` file from Machine 2 and put it under Apache HTTP server installation directory on Machine 1.
6. On Machine 1 (where Apache HTTP server is installed), edit the `mod_ASG.conf` configuration file located in the Apache HTTP server installation to set the Rendezvous session connection parameters as described in the Rendezvous Session Connection Parameters For Apache Module on page 129.

7. On Machine 2, where TIBCO API Exchange Gateway software is installed, edit the `asg.properties` file, located under `ASG_CONFIG_HOME` to set the Rendezvous session connection parameters. See Rendezvous Session Parameters For Apache Module And Gateway Engine Communication on page 130.

8. If you want to change the default values for the Rendezvous session connection parameters for the gateway core engine and central logger component, set or edit the parameters in the `ASG_CONFIG_HOME/asg.properties` file. See Parameters for Rendezvous connection From Gateway Core Engine To Central Logger on page 131. Also, set or edit the Rendezvous session connection parameters in the `ASG_CONFIG_HOME/asg_cl.properties` file. See Parameters for Rendezvous connection For Central Logger on page 133.

9. To change the default values for the Rendezvous session connection parameters for the gateway core engine and the global throttle manager, set or edit the parameters in the `ASG_CONFIG_HOME/asg.properties` file as described in the table, Rendezvous Session Parameters For Core Engine And Global Throttle Manager Communication on page 134.

10. Save the changes to the file.

You can edit the parameters in the `ASG_CONFIG_HOME/asg.properties` file and `ASG_CONFIG_HOME/asg_cl.properties` file using the P icon on the gateway configuration user interface. See Run Time Properties on page 65 for details.
Chapter 5  Transport Communication

This chapter explains the configuration settings required for the RV and JMS transports supported by the API Exchange Gateway Engine at the facade and target side.

Topics

- Rendezvous Transport Communication, page 126
- Secure Deployments With TIBCO Rendezvous, page 135
- Configuration For Secure Rendezvous Daemon, page 137
- JMS Transport Communication, page 147
Rendezvous Transport Communication

Rendezvous is the internal transport used for communication between the various components of API Exchange Gateway Engine software.

Figure Rendezvous Transport Communication on page 127 shows the communication for the various components using Rendezvous transport.
Figure 6  Rendezvous Transport Communication
Configuration Setup

This section explains the configuration setup required for Rendezvous communication.

To ensure that Rendezvous is only used as an internal transport with API Exchange Gateway Engine, do the following actions:

- Kill any running Rendezvous daemons on the system.
- Explicitly start the Rendezvous daemons.

You can start the Rendezvous daemons using the Rendezvous session parameters as defined in the `ASG_CONFIG_HOME/asg.properties` file. By default, the Rendezvous connection parameters are defined as below:

- RvDaemon tcp:7500
- RvNetwork
- RvService 7500

If you want to change network, daemon or service parameters for the Rendezvous transport connection, edit the properties defined in the `ASG_CONFIG_HOME/asg.properties` and `ASG_CONFIG_HOME/asg_cl.properties` file. See Setting Rendezvous Transport Properties on page 128.

Setting Rendezvous Transport Properties

This section explains the properties required to be set for Rendezvous transport as a communication channel between the API Exchange Gateway Engine components.

Rendezvous Session Connection Parameters For Apache Module

This section lists the parameters required for Apache module to connect to the gateway core engine using Rendezvous daemon. The parameters are defined in the `mod_ASG.conf` file located under `ASG_HOME/modules/http_server/apache` directory. To set the parameters for Apache module, perform the following steps:

1. Navigate to the `ASG_HOME/modules/http_server/apache` directory.
2. Open the `mod_ASG.conf` file in a text editor.
3. Set the values for the following parameters:

Table 16 Rendezvous Session Connection Parameters For Apache Module

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AsgService</td>
<td>Specifies the Rendezvous daemon service parameter.</td>
</tr>
<tr>
<td></td>
<td>Default value is: 7500</td>
</tr>
<tr>
<td>AsgNetwork</td>
<td>Specifies the Rendezvous daemon network parameter.</td>
</tr>
<tr>
<td></td>
<td>Default value is: loopback 10</td>
</tr>
<tr>
<td>AsgDaemon</td>
<td>Specifies the Rendezvous daemon parameter. This parameter is set as the tcp port.</td>
</tr>
<tr>
<td></td>
<td>Default value is: tcp:7500</td>
</tr>
<tr>
<td>AsgSubject</td>
<td>• Specifies the subject to which the Rendezvous daemon sends the client request. Default value is: _LOCAL.asg.north.request You can change the default value, as required.</td>
</tr>
<tr>
<td></td>
<td>• This value should match with the value set in the tibco.clientVar.ASG/modRV/north_request parameter of the ASG_CONFIG_HOME/asg.properties file.</td>
</tr>
</tbody>
</table>

4. Save the changes to the file.
Rendezvous Session Connection Parameters For Gateway Core Engine

This section lists the properties required for the gateway core engine to listen to the messages from the Apache module using Rendezvous daemon. The properties are defined in the \texttt{ASG\_CONFIG\_HOME/asg.properties} file.

To set or edit the properties, do following:
1. Navigate to the \texttt{ASG\_CONFIG\_HOME} directory.
2. Open \texttt{asg.properties} file in a text editor.
3. Set the following properties for the Rendezvous communication between the apache module and the gateway core engine:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| \texttt{tibco.clientVar.ASG/modRV/facade\_request} | Specifies the subject name used by the Rendezvous daemon to listen to the requests from Apache module. The gateway core engine listens to the requests on the same subject.  
  
  - This property value must match with the subject value specified in the Apache module configuration file \texttt{(mod\_ASG.conf)} for Apache server.  
  
  Default value is: \texttt{\_LOCAL.asg.north.request} |
| \texttt{tibco.clientVar.ASG/modRV/RvDaemon} | Specifies the value of Rendezvous daemon for the gateway core engine to connect and listen for the requests from the Apache module.  
  
  Default value is: \texttt{tcp:7500} |
4. Save the changes to the file.

**Rendezvous Session Connection Parameters For Gateway Core Engine and Central Logger Communication**

This section lists the properties required for Rendezvous communication between the gateway core engine and the central logger. The properties are defined in the `ASG_CONFIG_HOME/asg.properties` and `ASG_CONFIG_HOME/asg_cl.properties` files.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tibco.clientVar.ASG/modRV/RvNetwork</td>
<td>Specifies the value of Rendezvous network for the gateway core engine to connect and listen for the requests from the Apache module. This property value must match with the network value specified in the Apache module configuration file for Apache server.</td>
</tr>
<tr>
<td>tibco.clientVar.ASG/modRV/RvService</td>
<td>Specifies the value of Rendezvous service for the gateway core engine to connect and listen for the requests from the Apache module. Default value is: 7500</td>
</tr>
</tbody>
</table>
Central Logger:

The gateway core engine uses the Rendezvous daemon to send messages to the central logger. Set the properties for the gateway core engine to communicate with the Rendezvous daemon for the central logger in `ASG_CONFIG_HOME/asg.properties` file as below:

Table 18  Rendezvous Session Parameters For Core Engine To Communicate With Central Logger

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tibco.clientVar.Common/Connections/RV/SubjectPrefix</td>
<td>Specifies the prefix for Rendezvous subject names used by the gateway core engine to communicate with the other components.</td>
</tr>
<tr>
<td></td>
<td>Default value is: TIBCO.ASG.INTERNAL</td>
</tr>
<tr>
<td>tibco.clientVar.ASG/CL/RV/RvDaemon</td>
<td>Specifies the value of Rendezvous daemon for the gateway core engine to send messages to the central logger.</td>
</tr>
<tr>
<td></td>
<td>Default value is: tcp:7500</td>
</tr>
<tr>
<td>tibco.clientVar.ASG/CL/RV/RvNetwork</td>
<td>Specifies the value of Rendezvous network for the gateway core engine to send messages to the central logger.</td>
</tr>
<tr>
<td>tibco.clientVar.ASG/CL/RV/RvService</td>
<td>Specifies the value of Rendezvous service for the gateway core engine to send messages to the central logger.</td>
</tr>
<tr>
<td></td>
<td>Default value is: 7500</td>
</tr>
</tbody>
</table>
Parameters for Rendezvous connection For Central Logger

The central logger uses the Rendezvous to listen to the messages from the gateway core engine. Set the properties for the central logger to communicate with the Rendezvous daemon in ASG_CONFIG_HOME/asg_cl.properties file as below:

Table 19  Rendezvous Session Connection Parameters For Central Logger

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tibco.clientVar.Common/Connections/RV/SubjectPrefix</td>
<td>Specifies the prefix for Rendezvous subject names used by the gateway engine to communicate with the central logger. Default value is: TIBCO.ASG.INTERNAL</td>
</tr>
<tr>
<td>tibco.clientVar.ASG/CL/RV/RvDaemon</td>
<td>Specifies the value of Rendezvous daemon for the central logger to listen to the messages from the gateway core engine. Default value is: tcp:7500</td>
</tr>
<tr>
<td>tibco.clientVar.ASG/CL/RV/RvNetwork</td>
<td>Specifies the value of Rendezvous network for the central logger to listen to the messages from the gateway core engine.</td>
</tr>
<tr>
<td>tibco.clientVar.ASG/CL/RV/RvService</td>
<td>Specifies the value of Rendezvous service for the central logger to listen to the messages from the gateway core engine. Default value is: 7500</td>
</tr>
</tbody>
</table>

For the communication between the gateway core engine and the central logger components, the Rendezvous session connection parameters in the ASG_CONFIG_HOME/asg.properties file and ASG_CONFIG_HOME/asg_cl.properties file must match.

Rendezvous Session Connection Parameters For Gateway Core Engine and Global Throttle
Manager Communication

This section lists the properties required for Rendezvous communication between the gateway core engine and the global throttle manager. The properties must be defined in the ASG_CONFIG_HOME/asg.properties file.

Table 20  Rendezvous Session Parameters For Core Engine And Global Throttle Manager Communication

| Parameter                                      | Description                                                                                                                                 |
|------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------|---|
| tibco.clientVar.Common/Connections/RV/SubjectPrefix | Specifies the prefix for Rendezvous subject names used by the gateway core engine to communicate with other components. |   |
|                                                | Default value is: TIBCO.ASG.INTERNAL                                                                                                                                                             |   |
| tibco.clientVar.ASG/GTM/RV/RvDaemon            | • Specifies the value of Rendezvous daemon for the gateway core engine to connect to the global throttle manager.                                                                           |   |
|                                                | Default value is: tcp:7500                                                                                                                                                                      |   |
| tibco.clientVar.ASG/GTM/RV/RvNetwork           | • Specifies the value of Rendezvous network for the gateway core engine to connect to the global throttle manager.                                                                           |   |
| tibco.clientVar.ASG/GTM/RV/RvService           | • Specifies the value of Rendezvous service for the gateway core engine to connect to the global throttle manager.                                                                           |   |
|                                                | Default value is: 7500                                                                                                                                                                         |   |
Secure Deployments With TIBCO Rendezvous

The Apache module of API Exchange Gateway Engine has been enhanced to support the secure communication using Rendezvous secure daemons, `rvsd` and `rvsrd`. For detailed information on secure daemons (`rvsd` and `rvsrd`), See Chapter 6, Secure Daemons in TIBCO Rendezvous Administration Guide.

For the DMZ (De-Militarized Zone) setup, the Apache server runs on the machine outside the firewall and the API Exchange Gateway core engine runs on a machine inside the firewall. You have the following options to run the Apache server:

- Option 1: Run the Apache server on the same machine where secure rvsrd (or rvsd) daemon runs.
- Option 2: Run the Apache server and secure rvsrd (or rvsd) daemon on the different machines. In this case, the Apache server acts a client to connect to rvsrd (or rvsd) daemons. If Apache server and rvsd daemon are running on the machines in the same subnet, no rvrd configuration is required to connect from Apache server to rvsd. However, if Apache server and rvsrd daemon are running on the machines in the different subnets, you must configure routing daemon between the two machines (one with Apache server machine running rvrd and the other machine running rvsrd).

The API Exchange Gateway core engine runs on a machine in a secure network inside the firewall which has rvrd running. You must configure rvsrd (running on a machine where Apache server runs in case of Option 1) and rvrd (running on a machine where the gateway core engine is running) as neighbors. See TIBCO Rendezvous Administration Guide for configuration setup details of Rendezvous daemons.

This chapter mainly explains the configuration details required for the Apache module to connect to rvsrd (or rvsd) daemons.
Figure 7 Secure Deployment with Rendezvous

Figure Secure Deployment with Rendezvous on page 136 illustrates an example deployment of Apache module and the gateway core engine in a DMZ setup where Apache module communicates with the secure Rendezvous daemons. In this deployment, Apache server and Rendezvous secure daemon (rvsrd) runs on the same machine, Machine 1. The deployment consists of following components:

Machine 1 components:

- Apache server. Refer to ASG_HOME/readme for supported version of Apache server.
- TIBCO Rendezvous secure daemon (rvsrd). Refer to ASG_HOME/readme for supported Rendezvous version.

Machine 2 components:

- API Exchange Gateway Engine software.
- TIBCO Rendezvous daemon (rvrd). Refer to ASG_HOME/readme for supported Rendezvous version.

If Machine 1 and Machine 2 are on same subnet, you can use rvsd on Machine 1 and rvd on the Machine2.
Configuration For Secure Rendezvous Daemon

This section explains the configuration steps required to setup the deployment for Apache server communicating with the secure Rendezvous daemon (rvsrd).

Configuration Tips

You must consider the following points when configuring Rendezvous daemons (rvsrd and rvrd), Apache module and API Exchange Gateway software.

- Ensure that the subject name configured for Local Area Network (LAN) during the rvsrd daemon configuration on Machine 1 must match with the subject name configured for LAN for rvrd daemon configuration on Machine 2. The authorized subject names used in the rvsrd configuration on Machine 1 must be same as subject name used for rvrd configuration on Machine 2.

- Make sure that the authorized subject names used in the configuration of rvsrd on Machine 1 must match with the subject name set using the AsgSubject parameter in the Apache module configuration file (mod_ASG.conf) on Machine 1.

- Ensure that the subject names configured for rvrd on Machine 2 matches with the subject name set using the tibco.clientVar.ASG/modRV/north_request property in the gateway core engine properties file. The properties file is located as ASG_CONFIG_HOME/asg.properties file.

- Make sure that the subject name configured using AsgSubject parameter in the Apache module configuration file (mod_ASG.conf) on Machine 1 match with the subject name set using the tibco.clientVar.ASG/modRV/north_request property in the gateway core engine properties file. The properties file is located as ASG_CONFIG_HOME/asg.properties file.

- The listen port used to start the rvsrd daemon on Machine 1 must be different than the listen port used to start the rvrd daemon on Machine 2.

  For example,
  
  — Start rvsrd daemon on Machine 1 as below:
  
  rvsrd -store rvsrd.store -http 3500 -listen 7500

  — Start rvrd daemon on Machine 2 as below:
  
  rvrd -store rvrd.store -http 3500 -listen 7502

- The network parameter configured in the Apache module configuration file (mod_ASG.conf) on Machine 1 must be same as the network property value set in the gateway core engine properties (asg.properties) file.
For example,

In the `mod_ASG.conf` file on Machine 1, configure the network daemon as below:

```
AsgNetwork ;239.1.1.11
```

In the gateway core engine properties (`asg.properties`) file on Machine 2, set the network property value as below:

```
tibco.clientVar.ASG/modRV/RvNetwork=;239.1.1.11
```

- The daemon certificate configured for the `rvsrd` setup must match with the certificate specified by `AsgSecureDaemonCert` parameter in the Apache module configuration file (`mod_ASG.conf`).
- The user certificate configured for a user during the `rvsrd` setup must match with the certificate specified by `AsgSecureDaemonKey` parameter in the Apache module configuration file (`mod_ASG.conf`).

**Setup and Configure Rendezvous daemons**

- Install TIBCO Rendezvous on the machines in the firewall security zone, Machine 1 and Machine 2 as shown in the Secure Deployment with Rendezvous on page 136 diagram.
- Configure secure Rendezvous daemon (`rvsrd` or `rvsd`) on the machine which is outside the firewall. This is shown as Machine 1 in the Secure Deployment with Rendezvous on page 136 diagram. See TIBCO Rendezvous Administration for detailed instructions to configure `rvsrd` or `rvsd`, as required.
- Configure the Rendezvous daemon (`rvrd` or `rvd`) on the machine which is inside the inner security zone. This is shown as Machine 2 in the Secure Deployment with Rendezvous on page 136 diagram. See TIBCO Rendezvous Administration for detailed instructions to configure `rvrd` or `rvd`, as required.
- For `rvsrd` or `rvsd` configuration, see Configuration Tips on page 137.

**Configuration Setup For Apache Module And API Exchange Gateway**

This section explains the configuration setup for Apache Server, Apache module and API Exchange Gateway software on the machines outside and inside the firewall.

**Install Apache Server**

- Install, configure and setup Apache server on Machine 1. Refer to API Exchange Gateway Engine Installation guide for details.
Install API Exchange Gateway Engine

- Install API Exchange Gateway Engine software on the machine within the firewall security zone (Machine 2).

Setup and Configure Apache Module

This section explains the configuration to setup the Apache module.

Setup Apache Module on Machine 1

To setup the Apache module on the machine where Apache server runs (Machine 1), do following:

1. Copy following files from the API Exchange Gateway Engine installation on Machine 2:
   - `ASG_HOME\modules\http_server\apache\mod_ASG.conf`
   - `ASG_HOME\modules\http_server\apache\mod_asg_rv_inbound.so`

2. Put the above files locally on Machine 1 where Apache server runs. You should put the files in a location accessed by Apache server.

3. Open the `APACHE_HOME/conf/httpd.conf` file for editing.

4. Add the following line in the file:
   ```
   Include <Full Path>/mod_ASG.conf
   ```

5. Save the changes.

Configure Apache Module on Machine 1:

You must configure the Apache module to connect to the secure Rendezvous daemon. To configure Apache module installed on the machine where Apache server is running (Machine 1), do following:

1. Open the `mod_ASG.conf` file for editing.

2. Set the parameters as described in the Apache Module Properties on page 139 table:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AsgService</td>
<td>Specifies the service parameter configured for <code>rvsrd</code> on Machine 1. For example, 1111. This parameter value must be configured different than the value specified for <code>rvrd</code> setup on Machine 2.</td>
</tr>
</tbody>
</table>
Table 21  Apache Module Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AsgNetwork</td>
<td>Specifies the network parameter set during the configuration of secure Rendezvous daemon (rvsrd). This parameter is a random multicast ip address used to broadcast messages to the machines in that multicast group. For example, ;239.1.1.11 • This network parameter value must match with the network value set during the configuration of rvsrd setup on Machine 2.</td>
</tr>
<tr>
<td>AsgDaemon</td>
<td>Specifies the daemon value set during the configuration of secure Rendezvous daemon (rvsrd). For example, ssl:ASGRVSecure:7500 You must specify ssl prefix before the machine name, else the connection will fail.</td>
</tr>
<tr>
<td>AsgSubject</td>
<td>Specifies the subject name used to send the message to secure Rendezvous daemon (rvsrd).</td>
</tr>
<tr>
<td>AsgSecureDaemon</td>
<td>A boolean property to enable or disable the secure Rendezvous daemon connection for the Apache module. The secure Rendezvous daemon can run on the same or different machine where Apache server is running. Possible values are On and Off. Set this value to On to enable the Apache module to connect to secure Rendezvous daemon (rvsrd).</td>
</tr>
<tr>
<td>AsgSecureDaemonCert</td>
<td>Specifies the path to the public certificate of secure Rendezvous daemon (rvsrd). This public certificate is configured during the rvsrd setup. Required. For example, C:\tibcoasg\tibrv\8.3\certs\cert2.pem</td>
</tr>
</tbody>
</table>
3. Save the changes to the file.

You can use the `asg-password-obfuscator` executable to obfuscate the password. The obfuscated password can be used in `AsgSecureDaemonPassword` parameter of `mod_ASG.conf` file of Apache module. See `asg-password-obfuscator Utility on page 142` for usage details.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>AsgSecureDaemonUsername</code></td>
<td>Specifies the username used in <code>rvsrd</code> configuration. Optional. If <code>AsgSecureDaemonUsername</code> is set, the Apache module uses username and password to connect to the <code>rvsrd</code> daemon. If <code>AsgSecureDaemonUsername</code> is not set, then <code>AsgSecureDaemonKey</code> parameter must be set. See <code>AsgSecureDaemonKey on page 141.</code></td>
</tr>
<tr>
<td><code>AsgSecureDaemonPassword</code></td>
<td>Specifies the password used by the client in <code>rvsrd</code> configuration. The password is required when connecting to the <code>rvsrd</code> daemon either using username or client certificate. You can specify an obfuscated password for this parameter. The obfuscated password is generated using the <code>asg-password-obfuscator</code> utility located in <code>ASG_HOME/bin</code> directory. Required.</td>
</tr>
<tr>
<td><code>AsgSecureDaemonKey</code></td>
<td>Specifies the path to the user certificate of secure Rendezvous daemon (<code>rvsrd</code>). This user certificate is configured for a user in the <code>rvsrd</code> setup. The certificate should be in text (PEM) format. Optional. The Apache module connects to the secure Rendezvous daemon (<code>rvsrd</code>) using the user certificate specified by this parameter. If this parameter is not set, the Apache module connects to the secure Rendezvous daemon (<code>rvsrd</code>) using the username and password specified by <code>AsgSecureDaemonUsername</code> and <code>AsgSecureDaemonPassword</code> parameters.</td>
</tr>
</tbody>
</table>
Sample Properties For Apache Module

Below is the list of properties with example values for the Apache module set in the mod_ASG.conf file. Refer to Apache Module Properties on page 139 for the properties description.

AsgService 1111
AsgNetwork ;239.1.1.11
AsgDaemon ssl:ASGRVSecure:7500
AsgSecureDaemon On
AsgSecureDaemonCert "C:\tibcoasg\tibrv\8.3\certs\cert2.pem"
AsgSecureDaemonUsername "user"
AsgSecureDaemonPassword "user"
AsgSecureDaemonKey "C:\tibcoasg\tibrv\8.3\certs\Usercert.pem"

asg-password-obfuscator Utility

API Exchange Gateway provides asg-password-obfuscator utility which allows you to generate an obfuscated password used by Apache module (C module) to communicate with Rendezvous daemon. The obfuscated password can be set for the AsgSecureDaemonPassword parameter in the Apache module configuration (mod_ASG.conf) file.

Usage

The usage of the utility is shown as below:

Usage : asg-password-obfuscator [-i | -hostname <hostname>] <password>
- i obfuscate with an internal key
 - hostname obfuscate using hostname

The asg-password-obfuscator utility has following options:

- The -i option generates the obfuscated password that can be used on any machine. This option is less secure as it is machine independent.
• The -hostname option generates the obfuscated password specifically for a hostname. This option is more secure as it restricts the encrypted password to be used only for that machine.

• Either -i or -hostname option generates the obfuscated password for Apache module to communicate with Rendezvous daemon. These options are used only to generate the encrypted password for configuration file of Apache module (mod_ASG.conf).

• If you do not use the -i or -hostname options for the asg-password-obfuscator utility, then the generated password can be used for the java based modules such as like WSS’s keystore.

Example Output

C:\tibcoasg\asg\1.2\bin>asg-password-obfuscator -i admin
Obfuscating password ...
Jul 15, 2013 1:52:54 PM com.tibco.security.providers.SecurityVendor_j2se <clinit>
INFO: Initializing JSSE’s crypto provider class
com.sun.net.ssl.internal.ssl.Provider
in default mode
Obfuscated password (in brackets): [#_R9gvPGRME0hRIveQJS9i9tAzshJUjfK]

C:\tibcoasg\asg\1.2\bin>asg-password-obfuscator -hostname secureHost
password
Obfuscating password ...
Jul 15, 2013 1:52:54 PM com.tibco.security.providers.SecurityVendor_j2se <clinit>
INFO: Initializing JSSE’s crypto provider class
com.sun.net.ssl.internal.ssl.Provider in default mode
Obfuscated password (in brackets): [#^R9gvPGRME0hRIveQJS9i9tAzshJUjfK]

Configuring The Gateway Engine Properties

You must set the following properties for the gateway core engine to receive the requests from the Apache module.

To set or edit the properties, follow the steps as below:
1. Navigate to the `ASG_CONFIG_HOME` directory.
2. Open `asg.properties` file in a text editor.
3. Set or edit the properties as described in **Gateway Engine Properties on page 144** table.

### Table 22  Gateway Engine Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>tibco.clientVar.ASG/modRV/facade_request</code></td>
<td>Specifies the subject name used by the Rendezvous daemon to listen the requests from Apache module. The gateway core engine listens to the requests on the same subject.</td>
</tr>
<tr>
<td></td>
<td>• This property value must match with the subject value specified in the Apache module configuration file (<code>mod_ASG.conf</code>) for Apache server.</td>
</tr>
<tr>
<td></td>
<td>• This property value must match with the subject value specified for the <code>rvrd</code> configuration.</td>
</tr>
<tr>
<td></td>
<td>Default value is: <code>MachineName.asg.north.request</code></td>
</tr>
</tbody>
</table>
4. Save the changes to the file.

### Table 22 Gateway Engine Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tibco.clientVar.ASG/modRV/RvDaemon</td>
<td>Specifies the value of Rendezvous daemon for the gateway core engine to connect and listen for the requests from the Apache module.</td>
</tr>
<tr>
<td></td>
<td>Default value is: 7500</td>
</tr>
<tr>
<td></td>
<td>- This property value should match with the listen port value given for the command to start the rvrd daemon on the machine where the gateway core engine runs.</td>
</tr>
<tr>
<td></td>
<td>For example,</td>
</tr>
<tr>
<td></td>
<td>Set this property value to 7502 for the following command used to start the rvrd daemon:</td>
</tr>
<tr>
<td></td>
<td>rvrd –store rvrd.store –http 3500 –listen 7502</td>
</tr>
<tr>
<td></td>
<td>- This property value must be different with the listen port value given for the command to start the rvsrd daemon on the machine where Apache server runs.</td>
</tr>
<tr>
<td>tibco.clientVar.ASG/modRV/RvNetwork</td>
<td>Specifies the value of Rendezvous network for the gateway core engine to connect and listen for the requests from the Apache module.</td>
</tr>
<tr>
<td></td>
<td>This property value must match with the network value specified in the Apache module configuration file for Apache server. See Setup and Configure Apache Module on page 139.</td>
</tr>
<tr>
<td></td>
<td>For example, the value can be specified as:</td>
</tr>
<tr>
<td></td>
<td>;239.1.1.11</td>
</tr>
<tr>
<td>tibco.clientVar.ASG/modRV/RvService</td>
<td>Specifies the value of Rendezvous service for the gateway core engine to connect and listen for the requests from the Apache module.</td>
</tr>
<tr>
<td></td>
<td>For example, 2222</td>
</tr>
</tbody>
</table>
Sample Properties For Gateway Engine

Below is the list of properties with example values for the gateway core engine set in the asg.properties file. Refer to Gateway Engine Properties on page 144 for the properties description.

```
tibco.clientVar.ASG/modRV/facade_request=ASG200-Test.asg.north.request
tibco.clientVar.ASG/modRV/RvDaemon=7502
tibco.clientVar.ASG/modRV/RvNetwork=:239.1.1.11
tibco.clientVar.ASG/modRV/RvService=2222
```
JMS Transport Communication

TIBCO API Exchange Gateway provides JMS communication for both facade and target sides.

When JMS transport is used, you can use it as SOAP/JMS or JMS/XML as ESB channels.

Legend

EMS = TIBCO Enterprise Message Service
ASG = TIBCO ActiveMatrix Service Gateway
SOAP JMS Transport

The SOAP JMS channel of the gateway core engine is configured to connect to a single JMS server on the facade side and a single JMS server on the target side. This enables the gateway engine to act as both SOAP JMS server and SOAP JMS client. You can use this transport at both facade and target sides.

When SOAP JMS is used for communication, the queue names for facade and target are specified as the global variables in

\[\text{ASG\_CONFIG\_HOME/\texttt{asg.properties}}\]

The queue names can be overwritten by editing the values of global variables. You can edit the global variables values as follows:

- Using the gateway user interface. See Set RunTime Properties on page 208 for details.
- Editing the \[\text{ASG\_CONFIG\_HOME/\texttt{asg.properties}}\] file using a text editor.

Gateway As SOAP JMS Server

API Exchange Gateway Engine acts as a SOAP JMS server at the facade side. To submit a request using the SOAP JMS transport at the facade side, the consumer or client must use a single incoming queue to put the request. By default, the queue name for incoming request is \[\text{asg.soap.in.request}\]. Similarly, you can configure another queue to store the response from the target operations at the facade side. By default, the queue name for storing responses at the facade side is \[\text{asg.soap.in.request.reply.0}\].

When a client sends a request using the SOAP JMS transport, it sets the \[\text{JMSReplyDestination}\] header field on the request message. The gateway core engine uses the destination name as specified in the \[\text{JMSReplyDestination}\] header field to send the response. If the \[\text{JMSReplyDestination}\] header field of request message is not set by the client, the gateway core engine uses the queue name to send the response as specified in the

\[\text{ASG\_CONFIG\_HOME/\texttt{asg.properties}}\] file.
The gateway core engine populates the JMSCorrelationId header field value of the response message with the value of JMSCorrelationId header which was received in the request from client. If the JMSCorrelationId header field value of the client request message is empty, the gateway core engine populates the JMSCorrelationId header field value of the response message with the value of JMSMessageId from the received message.

- You can define and configure one JMS server at the facade side to store the requests and responses at facade side.
- The request queue should exist on the JMS server at the facade side when the gateway core engine is started.
- The destination value as specified in the JMSReplyDestination header field must exist on the same JMS Server from where the request was received.
- The queue for storing the response at the facade side is used only if the JMSReplyDestination header value was not set in the request message from the client. This is used mostly for the asynchronous incoming client requests where it does not expect a response.

**Gateway As SOAP JMS Client**

API Exchange Gateway Engine acts as a SOAP JMS client at the target side.

**Request Destination At Target Side**

When the gateway core engine forwards the request to the target side, it uses the queue or topic name as configured in the Destination Name and Destination Type fields of Services tab configuration of the user interface to store the southbound request. The queue name for southbound requests can also be specified by the tibco.clientVar.ASG/Endpoint/SOAPJMS/DefaultTargetRequestQueue global variable in the ASG_CONFIG_HOME/asg.properties file. The queue name from this global variable is used only if the Destination Name field on the ROUTING > Target operations tab on the gateway configuration is empty.

The user interface gives an error if the value of Destination Name field of the ROUTING > Target operations tab on the gateway configuration is empty. You can set this value as empty in the TargetOperation.cfg directly located under ASG_CONFIG_HOME/config_name.

**Response Destination At Target Side:**
When SOAP JMS transport is used at the southbound (target) side, the gateway core engine uses the queue to store the southbound responses. This queue name is specified by the `tibco.clientVar.ASG/Facade/SOAPJMS/replyQueue` global variable in the `ASG_CONFIG_HOME/asg.properties` file. The gateway core engine sets the JMSReplyDestination as the value specified in the reply queue name to store the responses from the target operations.

When SOAP JMS transport is used at the target side, the default queue names for request and response are as follows:

Request queue: `asg.soap.forward`
Response queue: `asg.soap.forward.reply.0`

- You can define and configure one JMS server at the target side to store the requests and responses at target side.
- The response queue should exist on the JMS server at the target side when the gateway core engine is started. If the core engine do not find them on the JMS server, it throws an exception. When the core engine is started and SOAP JMS channel is enabled, the engine connects to the JMS server and starts listening to the configured queue during engine startup.

The gateway core engine uses a single JMS queue as the reply destination for all SOAPJMS requests sent by the engine. In case, if multiple instances of the gateway core engines (asg-core or asg-caching-core) are deployed, it is required that each instance has a unique setting for the `tibco.clientVar.ASG/Endpoint/SOAPJMS/TargetResponseQueue` global variable. This ensures that the responses are returned to the correct engine.

### Configuring SOAP JMS Transport

This section explains the configuration setup required for API Exchange Gateway Engine to use SOAP JMS transport.

#### Enable JMS Channels

By default, JMS channels are disabled. If they are enabled, the gateway engine tries to connect to JMS server on the startup. If you do not use JMS transport its better to keep them disabled so that the gateway engine does not attempt to connect to EMS server. If you use JMS transport, you will need to enable JMS channels so that the gateway engine can connect to the EMS server when started.

To enable the channels, perform the following steps:

1. Open the `ASG_HOME/bin/asg_core.cdd` file for editing in a text editor.
2. Search the following property:

<property-group comment="" name="Channel">

<property name="be.channel.deactivate" value="/DefaultImplementation/Channels/SouthboundEsb0Channel,/DefaultImplementation/Channels/SouthboundEsb1Channel,/DefaultImplementation/Channels/SouthboundEsb2Channel,/DefaultImplementation/Channels/North_ESBChannel,/DefaultImplementation/Channels/North_HTTPChannel,/ASG/Channels/SOAPJMSChannel_North,/ASG/Channels/SOAPJMSChannel_South"/>

3. To enable the SOAP JMS channels, rename the property name to `be.channel.deactivate.test` as below:

<property-group comment="" name="Channel">

<property name="be.channel.deactivate.test" value="/DefaultImplementation/Channels/SouthboundEsb0Channel,/DefaultImplementation/Channels/SouthboundEsb1Channel,/DefaultImplementation/Channels/SouthboundEsb2Channel,/DefaultImplementation/Channels/North_ESBChannel,/DefaultImplementation/Channels/North_HTTPChannel,/ASG/Channels/SOAPJMSChannel_North,/ASG/Channels/SOAPJMSChannel_South"/>

4. Save the file.

**Edit asg.properties File**

`ASG_CONFIG_HOME/asg.properties` file defines JMS server connections, queue names and the user details for SOAP JMS transport.

**JMS Server Connection Parameters**

JMS server connection parameters are defined by the following properties. Edit the property as per your JMS server connection details.

**JMS Server Connection Parameters at Facade Side**

Following properties define the JMS server connection parameters used at the facade (northbound) side. Edit the properties values as per your JMS server settings.

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>tibco.clientVar.Common/Connections/JMS/SOAPConnection_Facade/JMSProviderURL</td>
<td>tcp://localhost:7222</td>
</tr>
</tbody>
</table>
### JMS Server Connection Parameters at Target Side

Following properties define the JMS sever connection parameters used at the target (southbound) side. Edit the properties values as per your JMS Server settings:

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>tibco.clientVar.Common/Connections/JMS/SoapConnectionFactory_Facade/TopicConnectionFactoryName</td>
<td>TopicConnectionFactory</td>
</tr>
<tr>
<td>tibco.clientVar.Common/Connections/JMS/SoapConnectionFactory_Facade/QueueConnectionFactoryName</td>
<td>QueueConnectionFactory</td>
</tr>
<tr>
<td>tibco.clientVar.Common/Connections/JMS/SoapConnectionFactory_Facade/JNDIUsername</td>
<td>admin</td>
</tr>
<tr>
<td>tibco.clientVar.Common/Connections/JMS/SoapConnectionFactory_Facade/JNDIPassword</td>
<td></td>
</tr>
<tr>
<td>tibco.clientVar.Common/Connections/JMS/SoapConnectionFactory_Facade/JMSUsername</td>
<td>admin</td>
</tr>
<tr>
<td>tibco.clientVar.Common/Connections/JMS/SoapConnectionFactory_Facade/JMSPassword</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>tibco.clientVar.Common/Connections/JMS/SoapConnectionFactory_Target/JMSProviderURL</td>
<td>tcp://localhost:7222</td>
</tr>
<tr>
<td>tibco.clientVar.Common/Connections/JMS/SoapConnectionFactory_Target/JNDIContextURL</td>
<td>tibjmsnaming://localhost:7222</td>
</tr>
<tr>
<td>tibco.clientVar.Common/Connections/JMS/SoapConnectionFactory_Target/TopicConnectionFactoryName</td>
<td>TopicConnectionFactory</td>
</tr>
<tr>
<td>tibco.clientVar.Common/Connections/JMS/SoapConnectionFactory_Target/QueueConnectionFactoryName</td>
<td>QueueConnectionFactory</td>
</tr>
</tbody>
</table>
JMS Sever Queue Names

By default, queue names are defined by the global variables in the `ASG_CONFIG_HOME/asg.properties` file. You can override the default values, if needed.

Facade Queue Names For SOAPJMS Transport

The default queue names are defined by the following properties as below. Edit the default values, if needed.

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>tibco.clientVar.Common/Connections/JMS/SOAPConnection_Target/JNDIUsername</td>
<td>admin</td>
</tr>
<tr>
<td>tibco.clientVar.Common/Connections/JMS/SOAPConnection_Target/JNDIPassword</td>
<td></td>
</tr>
<tr>
<td>tibco.clientVar.Common/Connections/JMS/SOAPConnection_Target/JMSUsername</td>
<td>admin</td>
</tr>
<tr>
<td>tibco.clientVar.Common/Connections/JMS/SOAPConnection_Target/JMSPassword</td>
<td></td>
</tr>
</tbody>
</table>

Target Queue Names For SOAPJMS Transport

The queue names are defined by the following properties as below. Edit the default values, if needed:

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>tibco.clientVar.ASG/Facade/SOAPJMS/requestQueue</td>
<td>asg.soap.in.request</td>
</tr>
<tr>
<td>tibco.clientVar.ASG/Facade/SOAPJMS/replyQueue</td>
<td>asg.soap.in.request.reply.0</td>
</tr>
<tr>
<td>tibco.clientVar.ASG/Endpoint/SOAPJMS/DefaultTargetRequestQueue</td>
<td>asg.soap.forward</td>
</tr>
<tr>
<td>tibco.clientVar.ASG/Endpoint/SOAPJMS/TargetResponseQueue</td>
<td>asg.soap.forward.reply.0</td>
</tr>
</tbody>
</table>
Make sure that all queue names are created on the JMS server before the gateway core engine is started. See Create queues on EMS Server on page 154.

Create queues on EMS Server

By default, API Exchange Gateway Engine uses the queue names for SOAP JMS transport as defined in the `ASG_CONFIG_HOME/asg.properties` file. After the SOAP JMS transport is enabled, you should create the queues on the EMS server.

Create the following queues on the EMS server:

- `asg.soap.in.request`
- `asg.soap.in.request.reply.0`
- `asg.soap.forward`
- `asg.soap.forward.reply.0`

- The queue names can be overwritten by editing the values of global variables as defined in the `ASG_CONFIG_HOME/asg.properties` file. If you changed the default values as mentioned in Facade Queue Names For SOAPJMS Transport, page 153 and Target Queue Names For SOAPJMS Transport, page 153, you should create the queue names defined by the global variables.

- You can edit the global variables values as follows:
  - Using the gateway user interface. See Set RunTime Properties, page 208.
  - Editing the `ASG_CONFIG_HOME/asg.properties` file using a text editor.

Create Non Admin Users On JMS Server

Ask your JMS server administrator to create the non admin user. API Exchange Gateway Engine uses this user to connect to JMS server when JMS transport is used at facade and target side. The administrator should grant the following privileges to this user:

- Send
- Receive
- Browse

This user is specified by the global variable property in the `ASG_CONFIG_HOME/asg.properties` file. See JMS Server Connection Parameters, page 151.
JMS Jars In ClassPath

To include the JMS jar files in the classpath of the gateway core engine, do any of the following:

- **Copy JMS jars**
  
  Manually copy JMS jar files in the following location:
  
  \( \text{ASG\_HOME/\text{lib/\text{ext/tpcl}} } \)

- **Edit asg-engine.tra file to set the EMS\_HOME variable.**
  
  Set the \text{tibco.env.EMS\_HOME} property is defined in the \text{ASG\_HOME/bin/asg-engine.tra} file. This should be set to the TIBCO Enterprise Message Service installation home.
  
  For example,
  
  \text{tibco.env.EMS\_HOME=c:/tibco/ems/5.1}

GUI Configuration

This section describes the configuration required to process SOAP JMS requests using the gateway user interface.

**Configuring New Partner**

You should define a new partner to process the requests from this partner using SOAP JMS transport.

Follow the steps as below to define a new partner:

1. Start the gateway configuration interface. See Starting GUI on page 166.
2. Select an existing project configuration or add a new project configuration. See Manage A Gateway Project Configuration on page 180.
3. Define a new partner. See Partners on page 205 to add the partner data

Sample values are shown below as a reference for the required fields:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Sample Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Partner Name</td>
<td>anon_JMS</td>
</tr>
</tbody>
</table>

Partner name **anon_JMS** must be defined by the following property in the \text{ASG\_CONFIG\_HOME/asg.properties} file:

\text{tibco.clientVar.ASG/anonymous/PartnerName/Authenticated=anon\_JMS}
4. Save your changes.

**Configuring Operation**

You should define a new operation to process the request from the client using SOAP JMS transport. See Facade Operations on page 193.

Sample values are shown below as a reference for the required fields:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Sample Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operation Name</td>
<td>GetBooks1</td>
</tr>
<tr>
<td>SOAP Action</td>
<td>&quot;/QueryBooksByAuthor&quot;</td>
</tr>
<tr>
<td>Operation URI</td>
<td></td>
</tr>
<tr>
<td>Status Code on Error</td>
<td>500</td>
</tr>
</tbody>
</table>

The operation and SOAP Action must match with the client request.

**Configuring Partner Operation**

Define a partner operation for a partner. See Facade Access on page 206.

Sample values are shown below for the required fields:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Partner</td>
<td>anon_JMS</td>
</tr>
<tr>
<td>Partner Operation</td>
<td>GetBooks1</td>
</tr>
<tr>
<td>Partner Timeout</td>
<td>20000</td>
</tr>
<tr>
<td>Forward Mapping</td>
<td>Pass-Through</td>
</tr>
<tr>
<td>Reverse Mapping</td>
<td>Pass-Through</td>
</tr>
</tbody>
</table>
Configuring Service

You should define a new service configuration to forward the client requests to a backend service using the SOAP JMS transport. See Add A New Target Operation on page 195 to add a new service.

Sample values are shown below for the required fields:

Table 23 Parameter and Sample Values

<table>
<thead>
<tr>
<th>Operation Name</th>
<th>getByAuthorService</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>SOAP JMS</td>
</tr>
<tr>
<td>Timeout</td>
<td>30000</td>
</tr>
<tr>
<td>SOAP Action</td>
<td>&quot;/QueryBooksByAuthor&quot;</td>
</tr>
<tr>
<td>Destination Name</td>
<td>sample</td>
</tr>
<tr>
<td>Destination Type</td>
<td>Queue</td>
</tr>
</tbody>
</table>

5. Save the configuration changes.

ESB Transport/Channel

API Exchange Gateway Engine supports ESB transport to communicate with JMS servers both at facade and target side. ESB transport allows you to use JMS transport with XML messages and does not contain SOAP messages.

By default, you can define one ESB channel at facade side to process northbound requests. Similarly you can define three ESB channels at target side to process southbound requests. Each ESB channel is configured using the global variables specified in the ASG_CONFIG_HOME/asg.properties file.

You can add more ESB channels if required using the custom extension mechanism.

Enable ESB Channels

See Enable JMS Channels on page 150.

Edit asg.properties File

ASG_CONFIG_HOME/asg.properties file defines JMS server connections, queue names and the user details for ESB transport.
JMS Server Connection Parameters For ESB Channel

JMS server connection parameters are defined by the following properties. Edit the property as per your JMS server connection details.

JMS Server Connection Parameters at Facade Side

Following properties define the JMS sever connection parameters for ESB channel used at the facade (northbound) side. Edit the properties values as per your JMS server settings.

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>tibco.clientVar.Common/Connections/JMS/FacadeEsbConnection0/JMSPProviderURL</td>
<td>tcp://localhost:7222</td>
</tr>
<tr>
<td>tibco.clientVar.Common/Connections/JMS/FacadeEsbConnection0/JNDICContextURL</td>
<td>tibjmsnaming://localhost:7222</td>
</tr>
<tr>
<td>tibco.clientVar.Common/Connections/JMS/FacadeEsbConnection0/TopicConnectionFactoryName</td>
<td>TopicConnectionFactory</td>
</tr>
<tr>
<td>tibco.clientVar.Common/Connections/JMS/FacadeEsbConnection0/QueueConnectionFactoryName</td>
<td>QueueConnectionFactory</td>
</tr>
<tr>
<td>tibco.clientVar.Common/Connections/JMS/FacadeEsbConnection0/JNDIUsername</td>
<td>admin</td>
</tr>
<tr>
<td>tibco.clientVar.Common/Connections/JMS/FacadeEsbConnection0/JNDIPassword</td>
<td>admin</td>
</tr>
<tr>
<td>tibco.clientVar.Common/Connections/JMS/FacadeEsbConnection0/JMSSUsername</td>
<td>admin</td>
</tr>
<tr>
<td>tibco.clientVar.Common/Connections/JMS/FacadeEsbConnection0/JMSPassword</td>
<td>admin</td>
</tr>
</tbody>
</table>
JMS Server Connection Parameters at Target Side

Following properties define the JMS server connection parameters for ESB channels used at the target (southbound) side. By default, you can use three ESB channels at southbound side. Edit the properties values as per your JMS server settings.

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>tibco.clientVar.Common/Connections/JMS/TargetEsbConnection0/JMSProviderURL</td>
<td>tcp://localhost:7222</td>
</tr>
<tr>
<td>tibco.clientVar.Common/Connections/JMS/SouthboundEsbConnection0/JNDIContextURL</td>
<td>tibjmsnaming://localhost:7222</td>
</tr>
<tr>
<td>tibco.clientVar.Common/Connections/JMS/TargetEsbConnection0/TopicConnectionFactoryName</td>
<td>TopicConnectionFactory</td>
</tr>
<tr>
<td>tibco.clientVar.Common/Connections/JMS/TargetEsbConnection0/QueueConnectionFactoryName</td>
<td>QueueConnectionFactory</td>
</tr>
<tr>
<td>tibco.clientVar.Common/Connections/JMS/TargetEsbConnection0/JNDIUsername</td>
<td>admin</td>
</tr>
<tr>
<td>tibco.clientVar.Common/Connections/JMS/TargetEsbConnection0/JNDIPassword</td>
<td></td>
</tr>
<tr>
<td>tibco.clientVar.Common/Connections/JMS/TargetEsbConnection0/JMSUsername</td>
<td>admin</td>
</tr>
<tr>
<td>tibco.clientVar.Common/Connections/JMS/TargetEsbConnection0/JMSPassword</td>
<td></td>
</tr>
<tr>
<td>tibco.clientVar.Common/Connections/JMS/TargetEsbConnection1/JMSProviderURL</td>
<td>tcp://localhost:7222</td>
</tr>
<tr>
<td>tibco.clientVar.Common/Connections/JMS/TargetEsbConnection1/JNDIContextURL</td>
<td>tibjmsnaming://localhost:7222</td>
</tr>
<tr>
<td>tibco.clientVar.Common/Connections/JMS/TargetEsbConnection1/TopicConnectionFactoryName</td>
<td>TopicConnectionFactory</td>
</tr>
<tr>
<td>tibco.clientVar.Common/Connections/JMS/TargetEsbConnection1/QueueConnectionFactoryName</td>
<td>QueueConnectionFactory</td>
</tr>
<tr>
<td>tibco.clientVar.Common/Connections/JMS/TargetEsbConnection1/JNDIUsername</td>
<td>admin</td>
</tr>
</tbody>
</table>
Facade Queue Names For ESB Transport

The queue names for the ESB channel at facade side are defined by the following properties as below. Edit the default values, if needed.

### Property Name

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>tibco.clientVar.Common/Connections/JMS/TargetEsbConnection1/JNDIPassword</td>
<td></td>
</tr>
<tr>
<td>tibco.clientVar.Common/Connections/JMS/TargetEsbConnection1/JMSUsername</td>
<td>admin</td>
</tr>
<tr>
<td>tibco.clientVar.Common/Connections/JMS/TargetEsbConnection1/JMSPassword</td>
<td></td>
</tr>
<tr>
<td>tibco.clientVar.Common/Connections/JMS/TargetEsbConnection2/JMSProviderURL</td>
<td>tcp://localhost:7222</td>
</tr>
<tr>
<td>tibco.clientVar.Common/Connections/JMS/TargetEsbConnection2/JNDIContextURL</td>
<td>tibjmsnaming://localhost:7222</td>
</tr>
<tr>
<td>tibco.clientVar.Common/Connections/JMS/TargetEsbConnection2/TopicConnectionFactoryName</td>
<td>TopicConnectionFactory</td>
</tr>
<tr>
<td>tibco.clientVar.Common/Connections/JMS/TargetEsbConnection2/QueueConnectionFactoryName</td>
<td>QueueConnectionFactory</td>
</tr>
<tr>
<td>tibco.clientVar.Common/Connections/JMS/TargetEsbConnection2/JNDIUsername</td>
<td>admin</td>
</tr>
<tr>
<td>tibco.clientVar.Common/Connections/JMS/TargetEsbConnection2/JNDIPassword</td>
<td></td>
</tr>
<tr>
<td>tibco.clientVar.Common/Connections/JMS/TargetEsbConnection2/JMSUsername</td>
<td>admin</td>
</tr>
<tr>
<td>tibco.clientVar.Common/Connections/JMS/TargetEsbConnection2/JMSPassword</td>
<td></td>
</tr>
</tbody>
</table>

### Global Variable

<table>
<thead>
<tr>
<th>Global Variable</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>tibco.clientVar.ASG/Facade/ESB0/requestQueue</td>
<td>asg.in.request</td>
</tr>
<tr>
<td>tibco.clientVar.ASG/Facade/ESB0/replyQueue</td>
<td>asg.in.request.reply.0</td>
</tr>
</tbody>
</table>
Target Queue Names For ESB Transport

The queue names for the ESB channel at target side are defined by the following properties as below. Edit the default values, if needed:

<table>
<thead>
<tr>
<th>Global Variable</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>tibco.clientVar.ASG/Endpoint/ESB0/requestQueue</td>
<td>asg.out.request</td>
</tr>
<tr>
<td>tibco.clientVar.ASG/Endpoint/ESB0/replyQueue</td>
<td>asg.out.request.reply.0.0</td>
</tr>
<tr>
<td>tibco.clientVar.ASG/Endpoint/ESB1/requestQueue</td>
<td>asg.out.request</td>
</tr>
<tr>
<td>tibco.clientVar.ASG/Endpoint/ESB1/replyQueue</td>
<td>asg.out.request.reply.0.1</td>
</tr>
<tr>
<td>tibco.clientVar.ASG/Endpoint/ESB2/requestQueue</td>
<td>asg.out.request</td>
</tr>
<tr>
<td>tibco.clientVar.ASG/Endpoint/ESB2/replyQueue</td>
<td>asg.out.request.reply.0.2</td>
</tr>
</tbody>
</table>

Create queues on EMS Server

By default, API Exchange Gateway Engine uses the queue names for ESB transport as defined in the `ASG_CONFIG_HOME/asg.properties` file.

Create the following queues on the EMS server:

```
asg.out.request
asg.out.request.reply.0.0
asg.out.request.reply.0.1
asg.out.request.reply.0.2
```

- The queue names can be overwritten by editing the values of global variables as defined in the `ASG_CONFIG_HOME/asg.properties` file. If you changed the default values as mentioned in Facade Queue Names For ESB Transport, page 160 and Target Queue Names For ESB Transport, page 161, you should create the queue names defined by the global variables.

- You can edit the global variables values as follows:
  - Using the gateway user Interface. See Set RunTime Properties on page 208 for details.
  - Editing the `ASG_CONFIG_HOME/asg.properties` file using a text editor.
Create Non Admin Users On EMS Server

Ask your EMS server administrator to create the non admin user. API Exchange Gateway Engine uses this user to connect to EMS server when ESB transport channel is used at facade and target side. The administrator should grant the following privileges to this user:

- Send
- Receive
- Browse

This user is specified by the global variable properties in the `ASG_CONFIG_HOME/asg.properties` file. See JMS Server Connection Parameters For ESB Channel, page 158.
Chapter 6  Gateway Configuration User Interface

This chapter describes the graphical user interface that allows you to add a new gateway configuration data for partners, facade operations, target operations, routing, throttles, mapping, schemas and other configuration data required for various functionality of TIBCO API Exchange Gateway.

Topics

• Overview, page 164
• Starting GUI, page 166
• Authentication Process For Gateway User Interface, page 168
• Configuration Setup For Authentication Process, page 172
• Enabling Debug Logging For Gateway User Interface, page 178
• Manage A Gateway Project Configuration, page 180
• MAPPING, page 184
• SECURITY, page 187
• MONITORING, page 190
• ROUTING, page 193
• PARTNER, page 205
Overview

The gateway user interface allows you to manage a configuration for a gateway project. A gateway project configuration contains all the information related to the partners, partner groups, facade operations, target operations, mappings, schemas, throttles, routing which is required by the gateway core engine at runtime.

The details of the configuration data are saved in the files which are located in the following configuration folder:

\*ASG\*\_\*CONFIG\*\_\*HOME\*

For example, the configuration data for the default configuration is stored in the following location:

C:\tibco\ASGConfig\tibco\cfgmgmt\asg\default

A project configuration folder contains the following files:

- configuration (.cfg)
- properties
- XSLT
- XSD
- certificates and keys

When the API Exchange Gateway configuration server is started for the first time on a machine after the product installation, ASGConfig.war archive is extracted under ASG\_HOME/\*webapp\* directory. You may notice few errors if you kill the configuration server before the extraction of archive file completes.

When you login to the gateway user interface, it has following main sections on the home page:

- Global Properties
  
  Select Gateway Engine Properties from the drop down list box to set the properties for gateway core engine as below:

  ![Gateway Engine Properties](image)

  This section allows you to set the runtime properties for the gateway core engine. See Set RunTime Properties on page 208.

- Projects
This section allows you to manage a project configuration. See Manage A Gateway Project Configuration on page 180.

- Advanced Settings
  This section allows you to add the settings for an environment.
Starting GUI

This section explains the steps to start the configuration server in a test environment.

1. Open a window.
2. Navigate to \( \text{ASG_HOME}/\text{bin} \) directory.
3. Start the TIBCO API Exchange Gateway GUI as follows:
   
   On Windows platform, type the following command on a command prompt:
   
   \( \text{asg-configui.exe} \)
   
   On Unix platform, type the following command on a terminal window:
   
   \( \text)./asg-configui \)

4. Open a browser window and enter the following URL:
   
   \( \text{http://localhost:9200/ConfigUI} \)

5. Enter the login information as below:
   
   \( \text{username: asgadmin} \)
   
   \( \text{password: asgadmin} \)

6. Verify that the user logs on successfully on user interface.
7. Verify that the following default project configurations are displayed under Projects panel:
   
   — default
   
   — GetLocation
   
   — BookQuery
   
   — BookQueryWSS
   
   — Caching
   
   — BookQueryBE
   
   — BookQueryBEWSS
   
   — APIExchange
   
   — BookQuerySecurity
   
   — BookQueryWSS
Changing Login, Host and Port Information

The values for username, password, host and port are configured in the `ASG_HOME/bin/asg-configui.tra` file. By default, the values are shown as below:

- `tibco.env.ASG_HOST=localhost`
- `tibco.env.ASG_PORT=9200`
- `tibco.env.ASG_ADMIN_USERNAME=asgadmin`
- `tibco.env.ASG_ADMIN_PASSWORD=asgadmin`

To change the values for username, password, host and port, follow the steps:

1. Open a terminal window.
2. Navigate to `ASG_HOME/bin` directory.
4. Edit the `asg-configui.tra` file using a text editor to set the following properties, as required:
   - `tibco.env.ASG_HOST=hostname`
   - `tibco.env.ASG_PORT=port`
   - `tibco.env.ASG_ADMIN_USERNAME=username`
   - `tibco.env.ASG_ADMIN_PASSWORD=password`
5. Stop the GUI server (if running).
6. Restart the GUI server using `asg-configui` executable.

   - For example,
     - On Windows, run the `asg-configui.exe`.
     - On Unix, type the following command:
       ```bash
       ./asg-configui
       ```

For production systems, it is recommended to deploy the `ASGConfig.war` file on a secure application server running in the production environment. The war file is located under `ASG_HOME/webapp` directory.
Authentication Process For Gateway User Interface

API Exchange Gateway User Interface provides a login screen which allows you to enter username and password for authentication. API Exchange Gateway supports the authentication of the users on the LDAP server or in a file. This functionality provides you following benefits:

- Create multiple users to login to the graphical user interface.
- Define the timeout value for an active session of the user.
- Option to logout after the user logs to the graphical user interface.

By default, only one user is allowed to login to the graphical user interface. This user is specified with the `tibco.env.ASG_ADMIN_USERNAME` parameter in the `ASG_HOME/bin/asg-configui.tra` file.

Authentication Properties

The Authentication Properties on page 168 table explains the properties to enable the LDAP or FILE based user authentication for the API Exchange Gateway user interface.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>asg-configui.tra</td>
<td>File Properties</td>
</tr>
</tbody>
</table>

Table 24  Authentication Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>java.property.com.tibco.asg.designtime.configui.ASGAuthProcess</td>
<td>Specifies the type of the authentication process required to login to the user interface. Possible values are:</td>
</tr>
<tr>
<td></td>
<td>— LDAP</td>
</tr>
<tr>
<td></td>
<td>— FILE</td>
</tr>
<tr>
<td></td>
<td>• The default value of this property is blank. If you do not specify any value and keep it blank, the login user is authenticated with the values specified in the tibco.env.ASG_ADMIN_USERNAME and tibco.env.ASG_ADMIN_PASSWORD properties of asg-configui.tra file. See Default Authentication on page 177 for details.</td>
</tr>
<tr>
<td></td>
<td>• The value of this property is required if you want to authenticate the user on a LDAP server or in a FILE.</td>
</tr>
</tbody>
</table>
**Table 24 Authentication Properties**

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>java.property.com.tibco.asg.designtime.configui.ASGProp File</td>
<td>Specifies the path to the property file for the authentication process.</td>
</tr>
</tbody>
</table>

- Value of this property is required if the `java.property.com.tibco.asg.designtime.configui.ASGAuthProcess` property is set to FILE or LDAP.

- If the authentication process is done on the LDAP server, then this parameter specifies the path to a property file containing the complete details to connect to LDAP server.

  Example:
  
  ```text
  ASG_HOME/bin/ldapSearch.properties
  ```
  
  where ASG_HOME is set to the directory where API Exchange Gateway is installed.

- If the authentication process type is FILE, then this parameter specifies the path to a text file containing the credentials of the users.

  Example:
  
  ```text
  ASG_HOME/bin/pwd.txt
  ```
  
  where ASG_HOME is set to the directory where API Exchange Gateway is installed.

See Authentication Property Files on page 175 for details.
### Table 24 Authentication Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>java.property.com.tibco.asg.designtime.configui.ASGSessionTimeout</td>
<td>Specifies the timeout value (in minutes) for a session. Set as Integer value. The session does not time out if the value is set to a negative value. Required.</td>
</tr>
</tbody>
</table>

### web.xml file properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
</table>
**Configuration Setup For Authentication Process**

This section explains the configuration setup required for the authentication process when the user logs in to the API Exchange Gateway User Interface.

You can define the authentication process for a user on the LDAP server or in a file. The authentication process and session timeout values are defined in the following files:

- `ASG_HOME/asg-configui.tra`
- `ASG_HOME/webapp/ASGConfig/WEB-INF/web.xml`

**LDAP Server Authentication**

This section explains the configuration steps to authenticate a user on the LDAP server. LDAP Server Authentication can be defined either in `asg-configui.tra` file or `web.xml` file.

- To configure `asg-configui.tra` file, see Configuring `asg-configui.tra` File on page 172.
- To configure `web.xml` file, see Configuring `web.xml` File on page 173

**Configuring asg-configui.tra File**

To configure the authentication process type, authentication property file and session timeout in `asg-configui.tra` file, do the following:

1. Navigate to the `ASG_HOME` directory.
2. Open the `asg-configui.tra` file for editing.
3. Set the following properties. See Authentication Properties on page 168 table for the description of properties.

   ```java
   java.property.com.tibco.asg.designtime.configui.ASGAuthProcess=LDAP
   java.property.com.tibco.asg.designtime.configui.ASGPropFile=path of the property file for LDAP server details
   java.property.com.tibco.asg.designtime.configui.ASGSessionTimeout=Timeout value (An integer in minutes)
   ```
4. Save the changes to the file.
**Configuring web.xml File**

To configure the authentication process type, authentication property file and session timeout in the `web.xml` file, do following:

1. Navigate to the `ASG_HOME/webapp//ASGConfig/WEB-INF` directory.
2. Open the `web.xml` file for editing.
3. Set the following properties. See Authentication Properties on page 168 table for the description of properties.

   a. Set the authentication process type to LDAP, as below:

   ```xml
   <init-param>
   <param-name>asgAuthProcess</param-name>
   <param-value>LDAP</param-value>
   </init-param>
   ```

   b. Set the property file for the LDAP authentication, as below:

   ```xml
   <init-param>
   <param-name>asgAuthPropFile</param-name>
   <param-value>Path to the property file for LDAP Server property file</param-value>
   </init-param>
   ```

   c. Set the timeout value for the login session of the user on the user interface, as below:

   ```xml
   <session-config>
   <session-timeout>An integer value in minutes</session-timeout>
   </session-config>
   ```

4. Save the changes to the file.

**File Based Authentication**

This section explains the configuration steps to authenticate a user with the credentials stored in a file on the file system. File based authentication can be defined either in `asg-configui.tra` file or `web.xml` file.

- To configure `asg-configui.tra` file, see Configuring asg-configui.tra File on page 172.
- To configure `web.xml` file, see Configuring web.xml File on page 173

**Configuring asg-configui.tra File**

To configure the authentication process type as `FILE`, authentication property file and session timeout in `asg-configui.tra` file, do following:

1. Navigate to the `ASG_HOME` directory.
2. Open the `asg-configui.tra` file for editing.

3. Set the following properties. See Authentication Properties on page 168 table for the description of properties.

   ```java
   java.property.com.tibco.asg.designtime.configui.ASGAuthProcess=FILE
   java.property.com.tibco.asg.designtime.configui.ASGPropFile=Full path of the user credentials file
   java.property.com.tibco.asg.designtime.configui.ASGSessionTimeout=Timeout value (An integer in minutes)
   ```

4. Save the changes to the file.

**Configuring web.xml File**

To configure the authentication process type as FILE, authentication property file and session timeout in the `web.xml` file, do following:

1. Navigate to the `ASG_HOME/webapp//ASGConfig/WEB-INF` directory.
2. Open the `web.xml` file for editing.
3. Set the following properties. See Authentication Properties on page 168 table for the description of properties.

   a. Set the authentication process type to FILE, as below:
      ```xml
      <init-param>
        <param-name>asgAuthProcess</param-name>
        <param-value>FILE</param-value>
      </init-param>
      ```

   b. Set the property file for the FILE authentication, as below:
      ```xml
      <init-param>
        <param-name>asgAuthPropFile</param-name>
        <param-value>Path to the user credentials file</param-value>
      </init-param>
      ```

   c. Set the timeout value for the login session of the user on the user interface, as below:
      ```xml
      <session-config>
        <session-timeout>An integer value in minutes</session-timeout>
      </session-config>
      ```
4. Save the changes to the file.

- If the configuration of authentication type, authentication property file and session timeout parameters are done in both asg-configui.tra and web.xml files, asg-configui.tra has the precedence over web.xml file.
- If you want to use web.xml for configuring session timeout, you must remove ASGSessionTimeOut property from asg-configui.tra file.

**Authentication Property Files**

Based on the authentication type, you must define the property files. The property files are used in the configuration of the authentication process.

If you are using the LDAP server authentication, you should define a LDAP search property file. See **LDAP Server Property File For LDAP Server Authentication** on page 175.

If you are using the FILE based authentication, you must define a user credentials file. The user credentials file is a text file containing the users and passwords required to login the user interface. See **User Credentials File For File Based Authentication** on page 176.

**LDAP Server Property File For LDAP Server Authentication**

The LDAP server property file contains the LDAP search properties to connect to the LDAP server and authenticate user. You must define a property file for LDAP server authentication.

**Sample File**  
The sample file is located as ASG_HOME/bin/ldapSearch.properties.

**LDAP Server Properties Sample File: on page 176** shows the example properties to be defined in the property file for LDAP server authentication.
The user credentials file is a text file which contains the list of username and password for the users. Username and password are separated by comma and each pair must be present on a separate line. The usernames are used to login the API Exchange Gateway user interface for FILE based authentication.

The passwords in this file are encrypted using the `asg-password-obfuscator` utility. The `asg-password-obfuscator` utility is located under `ASG_HOME/bin`.

The sample file is located as `ASG_HOME/bin/pwd.txt`

Sample Property File For File Authentication on page 176 shows the example file for FILE authentication.
Default Authentication

By default, API Exchange Gateway does not authenticate the user from a file or LDAP server to login to the graphical user interface.

When the Authentication process property (`asgAuthProcess`) is blank, API Exchange Gateway user interface authenticates the credentials of the user with the values specified by the following parameters in the `ASG_HOME/bin/asg-configui.tra` file.

- `tibco.env.ASG_ADMIN_USERNAME`
- `tibco.env.ASG_ADMIN_PASSWORD`

If you do not want the user authentication on a LDAP server or in a file, you must do following:

1. Open the `ASG_HOME/bin/asg-configui.tra` file for editing.
2. Set the following property to blank as below:
   ```
   java.property.com.tibco.asg.designtime.configui.ASGAuthProcess=
   ```
   (You can also specify `asgAuthProcess` parameter as blank in `web.xml` file.)
3. You must set the following parameters to specify the username and password:
   ```
   tibco.env.ASG_ADMIN_USERNAME
   ```
   (Example, `asgadmin`)
   ```
   tibco.env.ASG_ADMIN_PASSWORD
   ```
   (Example, `asgadmin` or an encrypted value)
4. Save the changes to the file.

- The password value for the `tibco.env.ASG_ADMIN_PASSWORD` property can be in plaintext or TIBCO obfuscated form. The password can be obfuscated using `asg-password-obfuscator` utility in `ASG_HOME/bin` directory.
- When the Authentication process property (`asgAuthProcess`) is blank, API Exchange Gateway user interface allows only one user for the login as specified by `tibco.env.ASG_ADMIN_USERNAME` parameter. If you want multiple users to login to the API Exchange Gateway user interface, you must use LDAP or FILE based authentication.
Enabling Debug Logging For Gateway User Interface

This section explains the steps to enable the debug level logging for the gateway user interface.

To enable the logging, do the following steps:

Create Properties File

You must define a `log4j.properties` properties file as below:

1. Navigate to the `ASG_HOME\be\5.1\lib\ext\tpcl\apache` directory.
2. Create a file with name as `log4j.properties`.

Logging to stdout

To send the logging messages to the stdout, you must define the following properties in the `log4j.properties` file. To set the properties, do following:

1. Open the `log4j.properties` file in a text editor.
2. Add the following properties:
   - `log4j.rootLogger = DEBUG, Console`
   - `log4j.appender.Console.layout.conversionPattern=%m%n`
3. Save changes to the file.

Logging to a File

You can redirect the logging messages to a file instead of sending them to stdout. To do so, do following:

1. Edit the `log4j.properties` file in a text editor.
2. Add the following properties:
   - `log4j.appender.file=org.apache.log4j.RollingFileAppender`
   - `log4j.appender.file.File=test.log`
   - `log4j.appender.file.MaxFileSize=10MB`
   - `log4j.appender.file.MaxBackupIndex=2`
   - `log4j.appender.file.layout=org.apache.log4j.PatternLayout`
   - `log4j.appender.file.layout.ConversionPattern=%m%n`
3. Save changes to the file.
Use Properties File In The TRA File

You must use the `log4j.properties` in the TRA file of the gateway user interface after you have added the properties. To add the `log4j.properties` file in the TRA file, do the following steps:

1. Navigate to the `ASG_HOME/bin` directory.
2. Open the `asg-configui.tra` file for editing.
3. Edit the following property to append the `log4j.properties` as below:
   ```
   java.extended.properties=-server -Xmx256m -Dlog4j.debug
   -Dlog4j.configuration=log4j.properties
   ```
4. Save changes to the file.

You must restart the gateway configuration server to see the messages. See `Starting GUI on page 166`. 
Manage A Gateway Project Configuration

The gateway user interface allows you to create, modify, delete and duplicate a gateway project configuration. A gateway project configuration is a folder that contains all the information related to the partner data, partner operations, operations, services, mappings, routing etc.

All the configuration files related to the partner data, partner operations, operations, services, mappings, throttles etc for a gateway project configuration are saved in a folder under `ASG_CONFIG_HOME`. For example, when you create a new gateway project configuration with name as `ASG_Get_Start`, the configuration server creates a `ASG_Get_Start` folder under `ASG_CONFIG_HOME`. The `ASG_Get_Start` folder is created with the default configuration data files for partner data, partner operations, operations, services, mappings, routing etc.

To manage any project configuration, put the mouse over the configuration and then select any of the options listed in the table Manage Project Configuration on page 180.

<table>
<thead>
<tr>
<th>Action</th>
<th>Icon</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add a new project configuration</td>
<td><img src="add.png" alt="Add" /></td>
<td>Allows you to add a new configuration for the gateway project.</td>
</tr>
<tr>
<td>Duplicate an existing configuration</td>
<td><img src="duplicate.png" alt="Duplicate" /></td>
<td>Allows you to copy an existing project configuration to a new project configuration.</td>
</tr>
<tr>
<td>Rename a project configuration</td>
<td><img src="rename.png" alt="Rename" /></td>
<td>Allows you to rename the name of a project.</td>
</tr>
<tr>
<td>Validate an existing configuration</td>
<td><img src="validate.png" alt="Validate" /></td>
<td>Allows you to validate an existing configuration.</td>
</tr>
<tr>
<td>Publish a Project configuration</td>
<td><img src="publish.png" alt="Publish" /></td>
<td></td>
</tr>
<tr>
<td>Delete an existing configuration</td>
<td><img src="delete.png" alt="Delete" /></td>
<td>Allows you to delete an existing configuration.</td>
</tr>
<tr>
<td>Action</td>
<td>Icon</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>------</td>
<td>-------------</td>
</tr>
</tbody>
</table>


Validate Configuration

The Validate Configuration on the gateway user interface validates the data for a specific project configuration. To validate the data for a configuration, select the project configuration and click the Validate Configuration icon.

If any of the configuration is missing or not correct for a facade operation, target operation, mapping or routing, it reports the error for that tab.

To run the validation tool for a configuration on the command line, See Running asg-validate using asg-tools on page 59.
API Exchange Gateway user interface groups the configuration for any project into following categories:

- **MAPPING**

  MAPPING tab allows you to enter the configuration data required by the gateway core engine for mapping of the request and response documents. See **MAPPING on page 184** for details.

- **SECURITY**

  SECURITY tab allows you to configure the data related to the security of the facade and target operations such as configuring polices, keystores etc. See **SECURITY on page 187** for details.

- **MONITORING**

  MONITORING tab allows you to enter the data related to monitors to enforce throttle policy, KPI groups required for reporting purposes. See **MONITORING on page 190** for details.

- **ROUTING**

  ROUTING tab allows you to enter the configuration data for facade and target operations. You can configure the routing data required by the gateway core engine to route any facade request to a target operation. See **ROUTING on page 193** for details.

- **PARTNER**

  PARTNER tab allows you to enter the configuration data for the partner, partner groups and facade operation authorization data for the partners. See **PARTNER on page 205** for details.
MAPPING

The MAPPING tab of the gateway user interface provides the configuration parameters for following areas:

Mapping

Mapping tab allows you to register the transformation (XSLT) files with TIBCO API Exchange Gateway engine.

To add a new mapping, follow the steps:

1. Click on Mapping tab.
2. Click Add property icon to create a new mapping.
3. Enter the mapping configuration parameters as defined below:

Table 26 Mapping Configuration Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mapping Configuration</td>
<td></td>
</tr>
<tr>
<td>Type</td>
<td>Select the type of mapping from the drop down list box.</td>
</tr>
<tr>
<td></td>
<td>Possible values are:</td>
</tr>
<tr>
<td></td>
<td>• RV</td>
</tr>
<tr>
<td></td>
<td>• XSLT</td>
</tr>
<tr>
<td>RV Type</td>
<td></td>
</tr>
<tr>
<td>Subject</td>
<td>Specifies the RV subject to send the mapping request to.</td>
</tr>
<tr>
<td>Transformation Name</td>
<td>Specifies the name of transformation to perform on the message.</td>
</tr>
<tr>
<td>XSLT Type</td>
<td></td>
</tr>
<tr>
<td>New File</td>
<td>Location of the transformation file.</td>
</tr>
<tr>
<td>Existing Files</td>
<td>Existing transformation file.</td>
</tr>
</tbody>
</table>
The **Schemas** tab allows you to configure the list of XSD files. The XSD files are used to validate the northbound request and response documents.

To add a new schema, follow the steps as below:

1. Click on **Schemas** tab.
2. Click Add property icon to create a new schema.
3. Enter the details for schema files, as defined below:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Schema Key</td>
<td>Specifies the unique id of XSD schema file.</td>
</tr>
<tr>
<td>New XSD File</td>
<td>Specifies the location of the XSD file.</td>
</tr>
<tr>
<td>Existing XSD File</td>
<td>Allows you to select any existing XSD file.</td>
</tr>
</tbody>
</table>

**Error Maps**

The **ErrorMaps** tab allows you to define all the error messages supported by TIBCO API Exchange Gateway.

To add a new errormap, follow the steps:

1. Click on **ErrorMaps** tab.
2. Click Add property icon to create a new mapping.
3. Enter the details for Error Maps configuration, as defined below:
### ErrorMaps Configuration Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error Id</td>
<td>External Id of the ErrorMap concept to be created from this line in the configuration file. Must be globally unique.</td>
</tr>
<tr>
<td>Status</td>
<td>Specifies the status of the transaction reported by the gateway error message.</td>
</tr>
<tr>
<td>Component</td>
<td>Specifies the component of the gateway for this error message.</td>
</tr>
<tr>
<td>Error Description</td>
<td>Specifies the description of the error.</td>
</tr>
<tr>
<td>Category</td>
<td>Specifies the category of the error. Allowed values are:</td>
</tr>
<tr>
<td></td>
<td>— ServiceException(SVC:)</td>
</tr>
<tr>
<td></td>
<td>— PolicyException(POL:)</td>
</tr>
<tr>
<td>Fault Code</td>
<td>Specifies the SOAP fault code. Usually, the value is Client.</td>
</tr>
<tr>
<td>Fault String</td>
<td>Specifies the SOAP fault string. For example, SVC0002</td>
</tr>
<tr>
<td>Fault Actor</td>
<td>Specifies the SOAP Fault Actor.</td>
</tr>
<tr>
<td>Message Id</td>
<td>Specifies the message id or any client specific code. For example, SVC0001</td>
</tr>
<tr>
<td>Text</td>
<td>Specifies the error text with locations for variables to be embedded indicated by “%1”, “%2” etc.</td>
</tr>
<tr>
<td>Variables</td>
<td>A list of strings separated by “,” that the gateway client uses for token replacement in the text.</td>
</tr>
</tbody>
</table>
SECURITY

The SECURITY tab of the gateway user interface provides the configuration parameters for following areas:

WSS

The WSS tab on the gateway user interface allows you to register the WSS resources with the gateway.

To add a new WSS resource, follow the steps as below:

1. Click on WSS tab.
2. Click Add property icon to create a new WSS resource.
3. Enter the details for WSS resource parameters, as defined below:

Table 29  WSS Resource Configuration Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WSS Name</td>
<td>Unique name which identifies a WSS configuration.</td>
</tr>
<tr>
<td>Type</td>
<td>Type of the WSS configuration. Select the type from the drop down list box.</td>
</tr>
<tr>
<td></td>
<td>Possible values are:</td>
</tr>
<tr>
<td></td>
<td>WSS</td>
</tr>
<tr>
<td></td>
<td>Subject Identity</td>
</tr>
<tr>
<td></td>
<td>Trust Identity</td>
</tr>
<tr>
<td>New Property File</td>
<td>A new property file which defines the WSS resources configuration. See Define</td>
</tr>
<tr>
<td></td>
<td>the WSS Configuration Properties file, page 340.</td>
</tr>
<tr>
<td>Existing Property Files</td>
<td>Select an existing WSS resources configuration property files. The file must exist in ASG_CONFIG_HOME/ASG_Project_Configuration/wss directory.</td>
</tr>
</tbody>
</table>

KeyStores

The KeyStores tab on the gateway user interface allows you to upload the keystore configuration.

To add a new keystore file, follow the steps as below:

1. Click on KeyStores tab.
2. Click Add property icon to upload a new keystore file.
3. Enter the details for keystores file, as defined below:

### Table 30  KeyStore Configuration Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>New KeyStores File</td>
<td></td>
</tr>
<tr>
<td>New KeyStoresFiles</td>
<td></td>
</tr>
</tbody>
</table>

### Policy Mapping

The **Policy Mapping** tab on the gateway user interface allows you to upload a policy file.

To upload a new policy file, perform the following steps:
1. Click on **Policy Mapping** tab.
2. Click Add property icon to upload a new policy file.
3. Enter the details for policy file, as defined below:

### Table 31  Policy Mapping Configuration Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Policy Name</td>
<td>Specifies the name for the policy.</td>
</tr>
<tr>
<td>Intent(Type)</td>
<td>Set the type of the policy. For example, Authentication. See Security Policies on page 351 for details.</td>
</tr>
<tr>
<td>Qualifier (SubType)</td>
<td>Set the policy sub type. For example, UsernameToken See Security Policies on page 351 for details.</td>
</tr>
<tr>
<td>Existing Policy File</td>
<td>Specifies an existing policy definition file. The policy file must exist in the gateway ASG_CONFIG_HOME/configuration/policy folder. For example, for the default configuration, the policy file must exist in the ASG_CONFIG_HOME/default/policy folder.</td>
</tr>
</tbody>
</table>
**Policy Binding**

The **Policy Binding** tab on the gateway user interface allows you to associate a registered policy with one or more target operation endpoints.

To map a policy to a target operation, perform the following steps:

1. Click on **Policy Binding** tab.
2. Click Add property icon to map a policy file.
3. Enter the details for policy mapping configuration, as defined below:

*Table 32  Policy Mapping Configuration Parameters*

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Policy</td>
<td>Specifies a name for the policy. The policy name must be configured under <strong>Policy Mapping</strong> tab.</td>
</tr>
<tr>
<td>URI</td>
<td>Specifies the URI of the operation to which the policy is applied.</td>
</tr>
<tr>
<td>Transport</td>
<td>Specifies the transport of the request. For example, HTTP, HTTPS, JMS, or SOAP.</td>
</tr>
<tr>
<td>Facade Operation</td>
<td>Specifies the operation to which the policy is applied. The facade operation must be configured in the <strong>Facade Operations</strong> tab.</td>
</tr>
<tr>
<td>Binding</td>
<td>Specifies the binding component that the policy is applied to. This could be either a facade operation (service) or a target operation (reference).</td>
</tr>
<tr>
<td>Flow</td>
<td>Specifies the flow of the request. Possible values are <strong>in</strong> or <strong>out</strong>.</td>
</tr>
<tr>
<td>Target Operation</td>
<td>Specifies the target operation. The target operation must be configured in the <strong>Target Operations</strong> tab.</td>
</tr>
</tbody>
</table>
MONITORING

The MONITORING tab of the gateway user interface provides the configuration parameters for following areas:

Monitors

The Monitors tab allows you to define different types of throttles with different throttle metrics.

1. Click on Monitors tab.
2. Click Add property icon to create a new monitor.
3. Enter the parameters for monitors as explained below:

Table 33 Monitors Configuration Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Throttle Parameters</strong></td>
<td></td>
</tr>
<tr>
<td>Monitor Name</td>
<td>Logical name for the monitor. This is the value used in the configuration to identify the monitor.</td>
</tr>
<tr>
<td>Monitor Type</td>
<td>Type of the monitor. Possible values for monitors are as below:</td>
</tr>
<tr>
<td></td>
<td>— Rate</td>
</tr>
<tr>
<td></td>
<td>— Quota</td>
</tr>
<tr>
<td></td>
<td>— High Water Mark</td>
</tr>
<tr>
<td></td>
<td>— Error</td>
</tr>
<tr>
<td>Interval</td>
<td>The time interval during which the monitor is applied. It has different time units depending on the monitor type. For example,</td>
</tr>
<tr>
<td></td>
<td>• Rate and Error throttle types specifies the interval in seconds</td>
</tr>
<tr>
<td></td>
<td>• Quota throttle type specifies the interval in hours.</td>
</tr>
<tr>
<td></td>
<td>• There is no time interval for High Water Mark throttle type.</td>
</tr>
</tbody>
</table>
Max Limit

Number of requests allowed during the interval. This number should be a positive integer greater than zero.

Monitor Time Modifiers

Specifies the time modifiers to apply to the monitor. Multiple time modifiers are allowed.

This has following parameters:

- Max Count - Monitor limit to apply if time modifier is active (in range).
- Start Date - Specifies the start date when the time modifier can be applied. The format is yyyyMMdd. It can be combined with end date to specify a date range. Optional.
- End Date - Specifies the end date when the time modifier can be applied. The format is yyyyMMdd. Optional.
- Day of Week - Specifies a list of the days of the week when the time modifier can be applied. The order of the days is not important. Optional.
- Time Range - Specifies a list of time ranges when the modifier can be applied. The format is hh:mm:ss-hh:mm:ss. Multiple time ranges can be specified. Optional.

Monitor Count

- Specifies the type of the counter used to increment the count of the used monitor. Possible values are:
  - Request Count
  - Input Payload Size
  - Output Payload Size
  - Transaction Payload Size
- Monitor count is not applicable for Error and High Water Mark monitor types.

See Throttle Counter on page 304.
KPI Groups

KPI Groups tab allows you to configure the aggregation levels used in KPI generation by the Central Logger component. KPIs are maintained for every facade operation, target operation and partner as configured for a configuration.

To set the parameters for KPI generation, configure following:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>KPI Groups Configuration</td>
<td></td>
</tr>
<tr>
<td>KPI Group Name</td>
<td>External Id of the KPI_Group. Set a unique name for this field. For example, oneMinute Required.</td>
</tr>
<tr>
<td>Interval (ms)</td>
<td>Interval in milliseconds. The KPI data is written to the output file after this interval is expired. Required.</td>
</tr>
<tr>
<td>Prefix</td>
<td>Specifies the logical name of the KPI. This value is stored in the KPI_FREQUENCY column of the database table ASG_KPI of the central logger component.</td>
</tr>
</tbody>
</table>
The ROUTING tab of the gateway user interface provides the configuration parameters for following areas:

**Facade Operations**

A facade operation is any operation provided by the gateway. An operation request is defined as a single type of request sent to the gateway engine. The Facade Operations tab allows you to add the list of operations supported by the TIBCO API Exchange Gateway.

**Add a New Facade Operation**

To add a new facade operation, follow the steps:

1. Click on Facade Operations tab.
2. Click Add property icon to create a new facade operation.
3. Enter the details for operation, as defined below:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operation Name</td>
<td>Logical operation name.</td>
</tr>
<tr>
<td>SOAP Action</td>
<td>SOAP Action for this operation. This field is used to identify which operation an incoming request is applied to. Must be unique.</td>
</tr>
<tr>
<td>Operation URI</td>
<td>URI for this operation.</td>
</tr>
<tr>
<td>Operation Service Name</td>
<td>Logical service name (used for routing).</td>
</tr>
<tr>
<td>New ProcessBody Transform</td>
<td>New XSLT transformation sheet file containing the rules to parse and validate the message. Optional.</td>
</tr>
</tbody>
</table>
Table 34  Facade Operation Configuration Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ProcessBody Transform</td>
<td>Existing XSLT transformation sheet file containing the rules to parse and validate the message. Optional.</td>
</tr>
<tr>
<td>New FaultReport XSLT</td>
<td>New XSLT transformation used to produce fault message for provided fault data.</td>
</tr>
<tr>
<td>Existing FaultReport XSLT</td>
<td>Existing XSLT transformation used to produce fault message for provided fault data. Optional.</td>
</tr>
<tr>
<td>Request Transform</td>
<td>Reference to forward facade operation mapping. This mapping transforms from requestor API to canonical request format. If no mapping reference needed, then select the value as Pass-Through from the drop down list. Required.</td>
</tr>
<tr>
<td>Response Transform</td>
<td>Reference to reverse northbound mapping. This mapping transforms from canonical request format to requestor API. If no mapping reference needed, then select the value as Pass-Through from the drop down list. Required.</td>
</tr>
<tr>
<td>Operation Method</td>
<td>HTTP method used to separate REST requests which are made on same URI but with different operations. Optional.</td>
</tr>
<tr>
<td>Operation Features</td>
<td>List of keywords identifying features required by the operation. Supported features are: Validation – XSD validate northbound request and response.</td>
</tr>
<tr>
<td>Status Code on Error</td>
<td></td>
</tr>
<tr>
<td>Enable WSS</td>
<td></td>
</tr>
</tbody>
</table>
Delete an existing operation

To delete an existing operation, click the red cross icon located at the top left corner of Operation Name field.

Target Operations

A target operation is defined as a single type of request that the gateway instance sends to backend systems. The backed-API defines the request structure, and the expected reply structure returned.

The Target Operations tab allows you to configure a target operation details.

TIBCO API Exchange Gateway supports the following types of target operations:

- ESB
- HTTP
- SOAPJMS
- HTTP(s)

Add A New Target Operation

To add a new target operation, follow the steps:

1. Click on Target Operations tab.
2. Click Add property icon to create a new facade operation.
3. Enter the configuration parameters for target operation, as defined below:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Type:</strong> No Operation</td>
<td>Empty target operation. Does accept any messages. Used for operations that make use of the information retrieved during customer validation.</td>
</tr>
<tr>
<td>Operation Name</td>
<td>Name of the target operation.</td>
</tr>
<tr>
<td>Type</td>
<td>The type of the transport to use when accessing the target operation.</td>
</tr>
<tr>
<td></td>
<td>For example, No Operation.</td>
</tr>
</tbody>
</table>
### Table 35  Target Operation Configuration Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Target Operation Group</td>
<td>Name of the Target Operation group. See Target Operation Groups, page 202 to configure a target operation group.</td>
</tr>
<tr>
<td>Timeout</td>
<td>Timeout (in milliseconds) to use when accessing the target operation.</td>
</tr>
<tr>
<td>Request Transform</td>
<td>Mapping from request canonical form to target operation API. Mapping details are defined in MAPPING &gt; Mapping tab.</td>
</tr>
<tr>
<td>Response Transform</td>
<td>Mapping from backend service API to response canonical form. Mapping details are defined in MAPPING &gt; Mapping tab.</td>
</tr>
<tr>
<td>Monitor(s)</td>
<td>Throttle chain to be applied when invoking the backend service. You can add one or more throttle names. The details of the throttles are defined in the MONITORING &gt; Monitors tab.</td>
</tr>
</tbody>
</table>

### Type: ESB

When the JMS transport is used to invoke the target operation, configure the following parameters specific to JMS transport

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESB Channel</td>
<td>Number of predefined ESB channels. By default, three ESB channels are supported, this field can have the values as 0, 1, 2. You can add more ESB Channels by customizing ASG_DefaultImplementation project.</td>
</tr>
<tr>
<td>ESB Service</td>
<td>Name of the ESB service to call. This is an additional header value added to the outgoing JMS message if specified. The header name in the JMS message is mentioned as Service.</td>
</tr>
<tr>
<td>Service Instance</td>
<td>Identity of service instance to call. This is an additional header value added to the outgoing JMS message if specified. The header name in the JMS message is mentioned as ServiceInstance.</td>
</tr>
</tbody>
</table>
### Table 35  Target Operation Configuration Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESB Operation</td>
<td>Specifies the ESB called operation. This is an additional header value added to the outgoing JMS message if specified. The header name in the JMS message is mentioned as <code>Operation</code>. Note that this tuple determines SOAP Action used by ESB as: <code>&quot;/esb/service//operation&quot;</code></td>
</tr>
<tr>
<td>Destination Name</td>
<td>Name of the queue or topic for JMS channel used to override the default JMS destination. The default destination is specified by the global variables <code>tibco.clientVar.ASG/Endpoint/ESB0/requestQueue</code>, <code>tibco.clientVar.ASG/Endpoint/ESB1/requestQueue</code>, <code>tibco.clientVar.ASG/Endpoint/ESB2/requestQueue</code> in <code>ASG_CONFIG_HOME/asg.properties</code> file respectively. Optional.</td>
</tr>
<tr>
<td>Destination Type</td>
<td>Type of the destination for JMS channel. Default is queue. Optional.</td>
</tr>
<tr>
<td>Mode</td>
<td>Allows the user to select a mode for a backend JMS service. Valid values are <code>SYNC/ASYNC</code>. For sync mode, the gateway waits for the southbound response from the target operation after the southbound request is sent to the target operation. This is the default mode. For async mode, the gateway does not wait for the southbound response from the target operation after the southbound request is sent to target operation. A default northbound response payload is created after the async request is sent to the target operation.</td>
</tr>
</tbody>
</table>
Table 35  Target Operation Configuration Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Type: HTTP</strong></td>
<td></td>
</tr>
<tr>
<td>SOAP Action</td>
<td>The value of the SOAP Action as defined by the WSDL of the target operation API.</td>
</tr>
<tr>
<td>URI</td>
<td>The URI to use when invoking the target operation.</td>
</tr>
<tr>
<td>Host</td>
<td>The IP address or hostname of the target operation implementation when invoked over HTTP.</td>
</tr>
<tr>
<td>Port</td>
<td>The TCP port of the target operation implementation when invoked over HTTP.</td>
</tr>
<tr>
<td>Username</td>
<td>Username with BASIC authentication.</td>
</tr>
<tr>
<td>Password</td>
<td>Password with BASIC authentication.</td>
</tr>
</tbody>
</table>
**Headers To Forward**

This field allows the users to copy HTTP headers information from the northbound incoming request (operation) and forward it to the service call at the southbound side. This field can contain:

- a comma separated list of named HTTP header names.
- * as the wildcard symbol to forward all HTTP headers from the facade service request to the target reference request.
- * as the wildcard symbol in combination with a comma separated list of named HTTP header names prefixed with the - sign to drop these specific HTTP headers from the list of headers to forward.

For example,

- If the value of Headers To Forward field is specified as "+", then all the headers are copied.
- If the Headers To Forward field contains "*,-SoapAction" then any incoming SOAP Action header is removed from the incoming headers and the value set on the endpoint is ignored.

**Method**

Specifies the method to be used as a http method for sending a southbound request over http transport.

Following methods are available for http transport:

OPTIONS, GET, HEAD, POST, PUT,DELETE,TRACE,CONNECT,PATCH

Default: POST

**Retry Count**

Number of retries.

**Retry Interval**

Interval between http connection retries. A value of 0 indicates no retry.
Table 35  Target Operation Configuration Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Retry Timeout</td>
<td>Timeout value on each attempt of http connection. This value is specified in milliseconds. A value of 0 indicates no timeout.</td>
</tr>
</tbody>
</table>

Type: SOAPJMS

When the SOAPJMS transport is used to invoke the backend service, configure the following parameters specific to SOAPJMS transport

- **SOAP Action**: The value of the SOAP Action as defined by the WSDL of target operation.
- **JMS Priority**: The value of JMSPriority header to set in the outgoing JMS message.
- **JMS Expiration**: The value of JMSExpiration header to set in the outgoing JMS message.
- **Destination Name**: Name of the destination on JMS server.
- **Destination Type**: Type of the destination (TOPIC/QUEUE).
- **Target Service**: Name of the service to call.
- **Content Type**: The value of JMSType header to set in the outgoing JMS message.
- **Is Async**: 

Type: HTTP(s)

When the https transport is used to invoke the target operation, configure the following parameters specific to https transport

- **SOAP Action**: The value of the SOAP Action as defined by the WSDL of target operation.
- **URI**: The URI to use when invoking the target operation.
- **Host**: The IP address or hostname of the target operation implementation when invoked over https.
- **Port**: The TCP port of the target operation implementation when invoked over https.
### Target Operation Configuration Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Username</td>
<td>Username with BASIC authentication.</td>
</tr>
<tr>
<td>Password</td>
<td>Password with BASIC authentication.</td>
</tr>
<tr>
<td>Headers To Forward</td>
<td>This field allows the users to copy headers information from the northbound incoming request and forward it to the southbound side.</td>
</tr>
<tr>
<td></td>
<td>If the value of Headers To Forward is specified as &quot;*&quot;, then all the headers are copied.</td>
</tr>
<tr>
<td></td>
<td>If the Headers To Forward contains &quot;*,-SoapAction&quot; then any incoming SOAP Action header is removed from the incoming headers and the value set on the endpoint is ignored.</td>
</tr>
<tr>
<td>Method</td>
<td>Specifies the method to be used as a http method for sending a southbound request over https transport.</td>
</tr>
<tr>
<td></td>
<td>Following methods are available for https transport: OPTIONS, GET, HEAD, POST, PUT,DELETE,TRACE,CONNECT,PATCH.</td>
</tr>
<tr>
<td></td>
<td>Default: POST</td>
</tr>
<tr>
<td>Retry Count</td>
<td>Number of retries.</td>
</tr>
<tr>
<td>Retry Interval</td>
<td>Interval between http(s) connection retries. A value of 0 indicates no retry.</td>
</tr>
<tr>
<td>Retry Timeout</td>
<td>Timeout value on each attempt of http connection. This value is specified in milliseconds. A value of 0 indicates no timeout.</td>
</tr>
<tr>
<td>New Property File</td>
<td>Specifies the DSS properties file to use https transport for target operation. See Define DSS Properties For Services on page 344.</td>
</tr>
<tr>
<td>Existing Property Files</td>
<td>Specifies an existing DSS property file from the drop down list box if the file exists in the wss directory of the project configuration.</td>
</tr>
</tbody>
</table>
Target Operation Groups

The **Target Operation Groups** tab allows the user to define a target operation group with a load balancing policy type which refers to a routing algorithm. A service group can have multiple target operations so that they can participate in load balancing functionality. See Configure A Target Operation Group on page 278 for details.

To define a target operation group, configure the following fields:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Group Name</td>
<td>• User defined name of target operation group.</td>
</tr>
<tr>
<td></td>
<td>• Required.</td>
</tr>
<tr>
<td>Description</td>
<td>User defined description.</td>
</tr>
</tbody>
</table>

---

Table 35  **Target Operation Configuration Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Is Anonymous</td>
<td>This field is a boolean field and determines if the client authentication is</td>
</tr>
<tr>
<td></td>
<td>required or not. The client authentication, also known as mutual SSL</td>
</tr>
<tr>
<td></td>
<td>authentication is required if the &quot;Is Anonymous&quot; flag is</td>
</tr>
<tr>
<td></td>
<td>set to false. If the &quot;Is Anonymous&quot; flag is set to true, the service does</td>
</tr>
<tr>
<td></td>
<td>not require the authentication of client.</td>
</tr>
<tr>
<td></td>
<td>See Configure Secure Services With API Exchange Gateway on page 344 for</td>
</tr>
<tr>
<td></td>
<td>details.</td>
</tr>
</tbody>
</table>

---

Table 36  **Target Operation Groups Configuration Parameters**
Routing

The **Routing** tab allows you to configure the routing information for the gateway engine. The routing provides binding between a facade operation and a target operation.

To add a new routing configuration, do the following steps:

1. Click the **Routing** tab.
2. Enter the following parameters, as defined below

### Table 36 Target Operation Groups Configuration Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| Type           | Specifies the type of the target operation group. See Types of Target Operations Group on page 266 for details. Supported types of target operation group are as follows:  
  - LoadBalanced
  - RoundRobin
  - RoundRobinWithFaiOver
  - WeightedRoundRobin
  - WeightedRoundRobinWithFaiOver
  - StickyResourceAffinity |

| Target Operations | Specifies the list of configured target operations within a target operation group. The target operations must be configured in the ROUTING > Target Operations tab of the gateway user interface. You can add the target operations by clicking the "+" icon sign, then select a target operation from the drop down list box. |

---

**Routing**
Table 37  Routing Configuration Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operation Name</td>
<td>Name of the facade operation. This is defined in the Facade Operations tab.</td>
</tr>
<tr>
<td>Routing Type</td>
<td>Specifies the type of routing which indicates if the request is routed to a target operation or target operation group. Possible values are:</td>
</tr>
<tr>
<td></td>
<td>— Target Operation</td>
</tr>
<tr>
<td></td>
<td>— Target Operation Group</td>
</tr>
<tr>
<td>Routing Key</td>
<td>Evaluated routing key for the given operation. See Routing Key on page 257.</td>
</tr>
<tr>
<td><strong>Type: Target Operation</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td>When the routing type is specified as Target Operation, configure the following parameters specific to target operation:</td>
</tr>
<tr>
<td>Target Operation Version</td>
<td>Version number of the target operation.</td>
</tr>
<tr>
<td>Target Operation</td>
<td>Name of the target operation. The target operation must be defined in the Target Operation tab.</td>
</tr>
<tr>
<td><strong>Type: Target Operation Group</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td>When the routing type is specified as Target Operation Group, configure the following parameters specific to target operation group:</td>
</tr>
<tr>
<td>Target Operation Group</td>
<td>Name of the target operation group used for load balancing of requests for target operations. See Target Operation Groups, page 202 to configure a target operation group.</td>
</tr>
</tbody>
</table>
PARTNER

The PARTNER tab of the gateway user interface provides the configuration parameters for following areas:

Partners

The Partners tab allows you to configure and manage the information for a partner who is authorized to access the gateway and send an operation request.

To add the data for a new partner, follow the steps:

1. Click on Partner tab.
2. Click Add property icon to create a new partner.
3. Enter the details for partner data, as defined below:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Partner Name</td>
<td>Partner name. This field is Required.</td>
</tr>
<tr>
<td>Partner Email</td>
<td>Contact email for the partner.</td>
</tr>
<tr>
<td>Partner Phone</td>
<td>Contact phone for the partner.</td>
</tr>
<tr>
<td>Partner Group</td>
<td>Name of the group of which the partner is a member. Partner group name is</td>
</tr>
<tr>
<td></td>
<td>defined in the Partner Groups tab.</td>
</tr>
<tr>
<td>Partner Serial</td>
<td>Serial number of the Partner's SSL certificate.</td>
</tr>
<tr>
<td>Number</td>
<td></td>
</tr>
<tr>
<td>Partner Issuer CA</td>
<td>Issuing Certificate Authority of the Partner’s SSL certificate.</td>
</tr>
<tr>
<td>Enable Secondary</td>
<td>If checked, then this partner exists in the Secondary ACLs list.</td>
</tr>
<tr>
<td>ACL</td>
<td>If not checked, then this partner does not exist in the Secondary ACLs list.</td>
</tr>
<tr>
<td>Monitor(s)</td>
<td>Defines the throttle chain which is applied to the partner. You can define</td>
</tr>
<tr>
<td></td>
<td>a list of throttle names. The throttles are applied in the order given in</td>
</tr>
<tr>
<td></td>
<td>the chain list. The details of the throttles are defined in the Throttles</td>
</tr>
<tr>
<td></td>
<td>tab.</td>
</tr>
</tbody>
</table>
Partner Groups

The Partner Groups tab allows you to configure the information for a partner group and the throttle chain which is applied to any requests sent by a partner who belongs to this group.

Add a throttle for a partner group

To add a new partner group, follow the steps:

1. Click on Partner Groups tab.
2. Click Add property icon to create a new partner.
3. Enter the details for partner group and monitor, as defined below:

Table 39 Partner Group Configuration

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Group Name</td>
<td>Name of the Partner group. This field is Required.</td>
</tr>
<tr>
<td>Email</td>
<td>Contact email for the partner group.</td>
</tr>
<tr>
<td>Phone</td>
<td>Contact phone for the partner group.</td>
</tr>
<tr>
<td>Monitor(s)</td>
<td>Defines the throttle chain which is applied to the group. You can define a list of throttle names. The throttles are applied in the order given in the chain list. The details of the throttles are defined in the Throttles tab.</td>
</tr>
</tbody>
</table>

Facade Access

The Facade Access tab allows you to invoke a specific operation by a specific partner. This tab also defines any throttles that are applied for the operation from this partner.

Add A New Facade Access

To add a new partner to access a facade operation, follow the steps:

1. Click on Facade Access tab.
2. Click Add property icon to create a new facade operation for a partner.
3. Enter the details for partner data, as defined below:

Table 40  Facade Access Configuration

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Partner</td>
<td>Partner name. This is Required field.</td>
</tr>
<tr>
<td>Facade Operation</td>
<td>Name of the facade operation for which the partner is granted the access.</td>
</tr>
<tr>
<td></td>
<td>This operation is configured in Facade Operations tab.</td>
</tr>
<tr>
<td>Monitors</td>
<td>A list of the throttle names to be applied to any requests sent by this</td>
</tr>
<tr>
<td></td>
<td>partner which invoke the operation. The monitors will be applied in the</td>
</tr>
<tr>
<td></td>
<td>order given in the chain.</td>
</tr>
<tr>
<td>Partner Timeout</td>
<td>Timeout in milliseconds to any incoming request from the partner to the</td>
</tr>
<tr>
<td></td>
<td>selected operation.</td>
</tr>
<tr>
<td>Partner Secondary ACL Check</td>
<td>If checked, then this partner is verified in secondary ACL list when the</td>
</tr>
<tr>
<td></td>
<td>partner invokes the operation.</td>
</tr>
<tr>
<td>Request Transform</td>
<td>Partner-specific mapping reference to call on inbound message.</td>
</tr>
<tr>
<td>Response Transform</td>
<td>Partner-specific mapping reference to call on outbound message.</td>
</tr>
<tr>
<td>Allowed Requestor IDs</td>
<td>List of authorized partner’s references when partner is an aggregator.</td>
</tr>
<tr>
<td>API Key</td>
<td>Specifies the API Key used for partner identification when the request is</td>
</tr>
<tr>
<td></td>
<td>sent from the API Exchange Manager.</td>
</tr>
</tbody>
</table>
### Set RunTime Properties

You can set the run time properties of the gateway using the gateway user interface. The properties are defined on the home page of the gateway user interface by selecting the **Gateway Engine Properties** in the drop down list box next to icon.

![Gateway Engine Properties](image)

The gateway user interface allows you to set the run time properties in following sections:
- General
- Monitoring
- Database
- Transport
- Security

When you select the **Gateway Engine Properties** on the home page of gateway user interface, it displays the run time properties of the gateway core engine. These properties are saved in the `ASG_CONFIG_HOME/`asg.properties` and `ASG_CONFIG_HOME/asg_cl.properties` files respectively.

### General Properties

When you click the **General** under **Gateway Engine Properties** on the gateway user interface, it allows you to set the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Example Home</td>
<td>Specifies the home location for the examples directory shipped with API Exchange Gateway. For example, the home directory for the examples location: <code>ASG_HOME/examples</code></td>
</tr>
</tbody>
</table>
When you click the Monitoring under Gateway Engine Properties on the gateway user interface, it allows you to set the following properties:

Table 42 Monitoring Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Generic</td>
<td>A boolean field which is used to send a synchronous request from the client at the facade side using the HTTP transport.</td>
</tr>
<tr>
<td></td>
<td>This is a checkbox field. Check the box if you want to set this value as true. If the value is set to true, then the client sends the synchronous request for HTTP transport. If the value is set to false, then asynchronous request is sent for HTTP transport.</td>
</tr>
<tr>
<td></td>
<td>Default is false</td>
</tr>
</tbody>
</table>
Table 42 Monitoring Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Threads</td>
<td>Number of threads used for HTTP Client. This is relevant only if the value of the <code>useSynchHttpClient</code> is false (i.e when asynchronous request is sent for HTTP transport). Default is 10</td>
</tr>
<tr>
<td>ModRV North Request Subject Name</td>
<td>Specifies the Rendezvous subject name which is used by the gateway core engine to listen for requests from Apache module. Default is <code>_LOCAL.asg.north.request</code></td>
</tr>
<tr>
<td>RV Subject Prefix</td>
<td>Specifies the prefix for all Rendezvous subject names used between gateway core engine and central logger, gateway core engine and global throttle manager components. Default is <code>TIBCO.ASG.INTERNAL</code></td>
</tr>
<tr>
<td>Throttle Update Interval (sec)</td>
<td>Specifies the time interval (in seconds) for sending throttle updates to global throttle manager. Default is 10</td>
</tr>
<tr>
<td>Request Binary</td>
<td>Default value is <code>_LOCAL.asg.north.request_binary</code></td>
</tr>
<tr>
<td>RV Daemon</td>
<td>Specifies the value of Rendezvous daemon for the gateway core engine to listen for the requests from the Apache module. Default value is: tcp:7500</td>
</tr>
<tr>
<td>RV Network</td>
<td>Specifies the value of Rendezvous network for the gateway core engine listen for the requests from the Apache module.</td>
</tr>
<tr>
<td>RV Service</td>
<td>Specifies the value of Rendezvous service for the gateway core engine to connect and listen for the requests from the Apache module. Default value is: 7500</td>
</tr>
<tr>
<td>GTM RV Daemon</td>
<td>Specifies the value of Rendezvous daemon for the gateway core engine to connect to the global throttle manager. Default value is: tcp:7500</td>
</tr>
<tr>
<td>GTM RV Network</td>
<td>Specifies the value of Rendezvous network for the gateway core engine to connect to the global throttle manager.</td>
</tr>
</tbody>
</table>
### Table 42  Monitoring Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GTM RV Service</td>
<td>Specifies the value of Rendezvous service for the gateway core engine to connect to the global throttle manager. Default value is: 7500</td>
</tr>
</tbody>
</table>

#### Logging:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable Reporting to Central Logger</td>
<td>Specifies if the reporting to common logger is enabled or not. By default, the core engine does not record the transactions to common logger. Default is false. See Enable Reporting to Central Logger, page 438.</td>
</tr>
<tr>
<td>Logging Interval (ms)</td>
<td>Specifies the time interval (in milliseconds) between core engine and central logger to record transactions. Default is 30000</td>
</tr>
<tr>
<td>Minimum Log Level</td>
<td>This field specifies the logging level for the central logger component. Possible values are as below:</td>
</tr>
<tr>
<td></td>
<td>• DEBUG</td>
</tr>
<tr>
<td></td>
<td>• INFO</td>
</tr>
<tr>
<td></td>
<td>• WARN</td>
</tr>
<tr>
<td></td>
<td>• ERROR</td>
</tr>
<tr>
<td></td>
<td>• NO LOGGING</td>
</tr>
<tr>
<td></td>
<td>Default is DEBUG</td>
</tr>
<tr>
<td>Central Logger Log Level</td>
<td>Specifies if the detail level logging is enabled for the central logger component. Possible values are:</td>
</tr>
<tr>
<td></td>
<td>• Detail Logging ON</td>
</tr>
<tr>
<td></td>
<td>• Detail Logging OFF</td>
</tr>
<tr>
<td></td>
<td>If the detail logging is set to ON, all the details of transaction are logged. If the detail logging is set to OFF, the central logger records the high level transaction.</td>
</tr>
<tr>
<td></td>
<td>Default value is Detail Logging OFF</td>
</tr>
</tbody>
</table>
Transport Properties

When you click the **Transport** under **Gateway Engine Properties** on the gateway user interface, it allows you to set the following properties:

Table 43  Transport Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Facade</strong></td>
<td></td>
</tr>
</tbody>
</table>
| Port              | Specifies the port through which the gateway accepts HTTP requests from the client.  
                    Default value : 9222                                                      |
| **SSL**           |                                                                             |
| Port              | Specifies the port through which ASG accepts SSL enabled HTTP requests from client.  
                    Default value : 9233                                                      |
| Use               | This is a boolean field which indicates if ssl should be enabled for accepting HTTPS requests. If set to true, SSL is enabled to accept the requests using HTTPS transport. |
| Id                | Specifies an identity resource which is used by FacadeHTTPSSLLConnection HTTP shared resource to provide SSL properties. |
| Id URL            | Specifies the URL to the identity file if the Identity file type is of the type Identity File. |
| File Type Password| Specifies the password to the identity file if the Identity file type is of the type Identity File. |
| Certificate URL   | Specifies the URL to the certificate file if the Identity file type is of the type Certificate/Private Key. |
| Key URL           | Specifies the URL to the private key in certificate file if the Identity file type is of the type Certificate/Private Key. |
| Key Password      | Specifies the password for the private key if the identity file type is of the type Certificate/Private Key |
### Transport Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requires Client Authentication</td>
<td>Indicates a boolean flag to enable or disable mutual SSL authentication for https transport between the client and the gateway.</td>
</tr>
<tr>
<td><strong>OAuth HTTP</strong></td>
<td></td>
</tr>
<tr>
<td>Port</td>
<td>Specifies the non-SSL port number of the API Exchange GatewayOAuth Server.</td>
</tr>
<tr>
<td></td>
<td>Default value is 9322.</td>
</tr>
<tr>
<td><strong>JMS Facade ESB Connection</strong></td>
<td></td>
</tr>
<tr>
<td>JMS Provider URL</td>
<td>Specifies the connection URL for the EMS Server used for facade operation requests from ESB communication domain. ESB communication uses JMS transport with XML.</td>
</tr>
<tr>
<td></td>
<td>Default is tcp://localhost:7222</td>
</tr>
<tr>
<td>JNDI Context URL</td>
<td>Specifies the URL to the JNDI service provider used for facade operation requests with ESB communication domain.</td>
</tr>
<tr>
<td></td>
<td>Default is tibjmsnaming://localhost:7222</td>
</tr>
<tr>
<td>Topic Connection Factory Name</td>
<td>Specifies the name of TopicConnectionFactory object stored in JNDI. This object is used to create a topic connection with ESB services at the facade side.</td>
</tr>
<tr>
<td></td>
<td>Default is TopicConnectionFactory</td>
</tr>
<tr>
<td>Queue Connection Factory Name</td>
<td>Specifies the name of QueueConnectionFactory object stored in JNDI. This object is used to create a queue connection with ESB services at the facade side.</td>
</tr>
<tr>
<td></td>
<td>Default is QueueConnectionFactory</td>
</tr>
<tr>
<td>JNDI Username</td>
<td>Specifies the user name for logging into the JNDI server in the ESB communication domain at the facade side. If the JNDI provider does not require access control, this field can be empty.</td>
</tr>
<tr>
<td></td>
<td>Default is admin</td>
</tr>
</tbody>
</table>
Table 43  Transport Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>JNDI Password</td>
<td>Specifies the password for logging into the JNDI server in the ESB communication domain at the facade side. If the JNDI provider does not require access control, this field can be empty.</td>
</tr>
<tr>
<td>JMS Username</td>
<td>Specifies the user name for logging into the EMS server in the ESB communication domain at the facade side. Default is admin.</td>
</tr>
<tr>
<td>JMS Password</td>
<td>Specifies the password for logging into the EMS server in the ESB communication domain at the facade side.</td>
</tr>
<tr>
<td>Request Queue</td>
<td>Specifies the queue name for an ESB channel (one) communication for the target operation request. Default queue name is asg.out.request.</td>
</tr>
<tr>
<td>Reply Queue</td>
<td>Specifies the queue name for ESB channel (one) communication for the response from the target operation. Default queue name is asg.out.request.reply.0.0 - check</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>JMS Target ESB Connection Primary</th>
</tr>
</thead>
<tbody>
<tr>
<td>JMS Provider URL</td>
</tr>
<tr>
<td>JNDI Context URL</td>
</tr>
<tr>
<td>Topic Connection Factory Name</td>
</tr>
</tbody>
</table>
### Table 43  Transport Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Queue Connection Factory Name</td>
<td>Specifies the name of QueueConnectionFactory object stored in JNDI. This object is used to create a queue connection with ESB services at the facade side. Default is QueueConnectionFactory</td>
</tr>
<tr>
<td>JNDI Username</td>
<td>Specifies the user name for logging into the JNDI server in the ESB communication domain at the facade side. If the JNDI provider does not require access control, this field can be empty. Default is admin</td>
</tr>
<tr>
<td>JNDI Password</td>
<td>Specifies the password for logging into the JNDI server in the ESB communication domain at the facade side. If the JNDI provider does not require access control, this field can be empty.</td>
</tr>
<tr>
<td>JMS Username</td>
<td>Specifies the user name for logging into the EMS server in the ESB communication domain at the facade side.                                                                                  Default is admin</td>
</tr>
<tr>
<td>JMS Password</td>
<td>Specifies the password for logging into the EMS server in the ESB communication domain at the facade side.</td>
</tr>
<tr>
<td>Request Queue</td>
<td>Specifies the queue name for an ESB channel (one) communication for the target operation request. Default queue name is asg.out.request</td>
</tr>
<tr>
<td>Reply Queue</td>
<td>Specifies the queue name for ESB channel (one) communication for the response from the target operation. Default queue name is asg.out.request.reply.0.0 - check</td>
</tr>
</tbody>
</table>
Table 43  Transport Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>JMS Target ESB Connection Secondary</td>
<td></td>
</tr>
<tr>
<td>JMS Provider URL</td>
<td>Specifies the connection URL for the EMS Server used for facade operation requests from ESB communication domain. ESB communication uses JMS transport with XML. Default is tcp://localhost:7222</td>
</tr>
<tr>
<td>JNDI Context URL</td>
<td>Specifies the URL to the JNDI service provider used for facade operation requests with ESB communication domain. Default is tibjmsnaming://localhost:7222</td>
</tr>
<tr>
<td>Topic Connection Factory Name</td>
<td>Specifies the name of TopicConnectionFactory object stored in JNDI. This object is used to create a topic connection with ESB services at the facade side. Default is TopicConnectionFactory</td>
</tr>
<tr>
<td>Queue Connection Factory Name</td>
<td>Specifies the name of QueueConnectionFactory object stored in JNDI. This object is used to create a queue connection with ESB services at the facade side. Default is QueueConnectionFactory</td>
</tr>
<tr>
<td>JNDI Username</td>
<td>Specifies the user name for logging into the JNDI server in the ESB communication domain at the facade side. If the JNDI provider does not require access control, this field can be empty. Default is admin</td>
</tr>
<tr>
<td>JNDI Password</td>
<td>Specifies the password for logging into the JNDI server in the ESB communication domain at the facade side. If the JNDI provider does not require access control, this field can be empty.</td>
</tr>
<tr>
<td>JMS Username</td>
<td>Specifies the user name for logging into the EMS server in the ESB communication domain at the facade side. Default is admin</td>
</tr>
</tbody>
</table>

TIBCO API Exchange Gateway User's Guide
<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>JMS Password</td>
<td>Specifies the password for logging into the EMS server in the ESB communication domain at the facade side.</td>
</tr>
<tr>
<td>Request Queue</td>
<td>Specifies the queue name for an ESB channel (one) communication for the target operation request. Default queue name is asg.out.request</td>
</tr>
<tr>
<td>Reply Queue</td>
<td>Specifies the queue name for ESB channel (one) communication for the response from the target operation. Default queue name is asg.out.request.reply.0.0</td>
</tr>
</tbody>
</table>

### JMS Target ESB Connection Tertiary

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>JMS Provider URL</td>
<td>Specifies the connection URL for the EMS Server used for facade operation requests from ESB communication domain. ESB communication uses JMS transport with XML. Default is tcp://localhost:7222</td>
</tr>
<tr>
<td>JNDI Context URL</td>
<td>Specifies the URL to the JNDI service provider used for facade client requests with ESB communication domain. Default is tibjmsnaming://localhost:7222</td>
</tr>
<tr>
<td>Topic Connection Factory Name</td>
<td>Specifies the name of TopicConnectionFactory object stored in JNDI. This object is used to create a topic connection with ESB services at the facade side. Default is TopicConnectionFactory</td>
</tr>
<tr>
<td>Queue Connection Factory Name</td>
<td>Specifies the name of QueueConnectionFactory object stored in JNDI. This object is used to create a queue connection with ESB services at the facade side. Default is QueueConnectionFactory</td>
</tr>
</tbody>
</table>
JNDI Username Specifies the user name for logging into the JNDI server in the ESB communication domain at the facade side. If the JNDI provider does not require access control, this field can be empty.

Default is admin

JNDI Password Specifies the password for logging into the JNDI server in the ESB communication domain at the facade side. If the JNDI provider does not require access control, this field can be empty.

JMS Username Specifies the user name for logging into the EMS server in the ESB communication domain at the facade side.

Default is admin

JMS Password Specifies the password for logging into the EMS server in the ESB communication domain at the facade side.

Request Queue Specifies the queue name for an ESB channel (one) communication for the target operation request.

Default queue name is asg.out.request

Reply Queue Specifies the queue name for ESB channel (one) communication for the response from the target operation.

Default queue name is asg.out.request.reply.0.0

JMS SOAP Connection North

JMS Provider URL Specifies the connection URL for the EMS Server used for facade operation requests from ESB communication domain. ESB communication uses JMS transport with XML.

Default is tcp://localhost:7222

Table 43  Transport Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>JNDI Username</td>
<td>Specifies the user name for logging into the JNDI server in the ESB communication domain at the facade side. If the JNDI provider does not require access control, this field can be empty. Default is admin</td>
</tr>
<tr>
<td>JNDI Password</td>
<td>Specifies the password for logging into the JNDI server in the ESB communication domain at the facade side. If the JNDI provider does not require access control, this field can be empty.</td>
</tr>
<tr>
<td>JMS Username</td>
<td>Specifies the user name for logging into the EMS server in the ESB communication domain at the facade side. Default is admin</td>
</tr>
<tr>
<td>JMS Password</td>
<td>Specifies the password for logging into the EMS server in the ESB communication domain at the facade side.</td>
</tr>
<tr>
<td>Request Queue</td>
<td>Specifies the queue name for an ESB channel (one) communication for the target operation request. Default queue name is asg.out.request</td>
</tr>
<tr>
<td>Reply Queue</td>
<td>Specifies the queue name for ESB channel (one) communication for the response from the target operation. Default queue name is asg.out.request.reply.0.0</td>
</tr>
<tr>
<td>JMS Provider URL</td>
<td>Specifies the connection URL for the EMS Server used for facade operation requests from ESB communication domain. ESB communication uses JMS transport with XML. Default is tcp://localhost:7222</td>
</tr>
</tbody>
</table>
### Table 43  Transport Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>JNDI Context URL</td>
<td>Specifies the URL to the JNDI service provider used for facade client requests with ESB communication domain.</td>
</tr>
<tr>
<td></td>
<td>Default is <code>tibjmsnaming://localhost:7222</code></td>
</tr>
<tr>
<td>Topic Connection Factory Name</td>
<td>Specifies the name of TopicConnectionFactory object stored in JNDI. This object is used to create a topic connection with ESB services at the facade side.</td>
</tr>
<tr>
<td></td>
<td>Default is <code>TopicConnectionFactory</code></td>
</tr>
<tr>
<td>Queue Connection Factory Name</td>
<td>Specifies the name of QueueConnectionFactory object stored in JNDI. This object is used to create a queue connection with ESB services at the facade side.</td>
</tr>
<tr>
<td></td>
<td>Default is <code>QueueConnectionFactory</code></td>
</tr>
<tr>
<td>JNDI Username</td>
<td>Specifies the user name for logging into the JNDI server in the ESB communication domain at the facade side. If the JNDI provider does not require access control, this field can be empty.</td>
</tr>
<tr>
<td></td>
<td>Default is <code>admin</code></td>
</tr>
<tr>
<td>JNDI Password</td>
<td>Specifies the password for logging into the JNDI server in the ESB communication domain at the facade side. If the JNDI provider does not require access control, this field can be empty.</td>
</tr>
<tr>
<td>JMS Username</td>
<td>Specifies the user name for logging into the EMS server in the ESB communication domain at the facade side.</td>
</tr>
<tr>
<td></td>
<td>Default is <code>admin</code></td>
</tr>
<tr>
<td>JMS Password</td>
<td>Specifies the password for logging into the EMS server in the ESB communication domain at the facade side.</td>
</tr>
<tr>
<td>Request Queue</td>
<td>Specifies the queue name for an ESB channel (one) communication for the target operation request.</td>
</tr>
<tr>
<td></td>
<td>Default queue name is <code>asg.out.request</code></td>
</tr>
</tbody>
</table>
Table 43  Transport Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reply Queue</td>
<td>Specifies the queue name for ESB channel (one) communication for the response from the target operation. Default queue name is asg.out.request.reply.0.0</td>
</tr>
<tr>
<td>JMS SOAP Connection South</td>
<td></td>
</tr>
<tr>
<td>JMS Provider URL</td>
<td>Specifies the connection URL for the EMS Server used for facade operation requests from ESB communication domain. ESB communication uses JMS transport with XML. Default is tcp://localhost:7222</td>
</tr>
<tr>
<td>JNDI Context URL</td>
<td>Specifies the URL to the JNDI service provider used for facade client requests with ESB communication domain. Default is tibjmsnaming://localhost:7222</td>
</tr>
<tr>
<td>Topic Connection Factory Name</td>
<td>Specifies the name of TopicConnectionFactory object stored in JNDI. This object is used to create a topic connection with ESB services at the facade side. Default is TopicConnectionFactory</td>
</tr>
<tr>
<td>Queue Connection Factory Name</td>
<td>Specifies the name of QueueConnectionFactory object stored in JNDI. This object is used to create a queue connection with ESB services at the facade side. Default is QueueConnectionFactory</td>
</tr>
<tr>
<td>JNDI Username</td>
<td>Specifies the user name for logging into the JNDI server in the ESB communication domain at the facade side. If the JNDI provider does not require access control, this field can be empty. Default is admin</td>
</tr>
<tr>
<td>JNDI Password</td>
<td>Specifies the password for logging into the JNDI server in the ESB communication domain at the facade side. If the JNDI provider does not require access control, this field can be empty.</td>
</tr>
</tbody>
</table>
When you click the Database under Global Properties on the user interface, it allows you to set the following properties:

**Table 43 Transport Properties**

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>JMS Username</td>
<td>Specifies the user name for logging into the EMS server in the ESB communication domain at the facade side. Default is admin</td>
</tr>
<tr>
<td>JMS Password</td>
<td>Specifies the password for logging into the EMS server in the ESB communication domain at the facade side.</td>
</tr>
<tr>
<td>Request Queue</td>
<td>Specifies the queue name for an ESB channel (one) communication for the target operation request. Default queue name is asg.out.request</td>
</tr>
<tr>
<td>Reply Queue</td>
<td>Specifies the queue name for ESB channel (one) communication for the response from the target response. Default queue name is asg.out.request.reply.0.0</td>
</tr>
</tbody>
</table>

**Database Properties**

When you click the Database under Global Properties on the user interface, it allows you to set the following properties:

**Table 44 Database Properties**

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connection Check Interval</td>
<td>Specifies the time interval (in seconds) for the central logger to connect to the database server.</td>
</tr>
<tr>
<td>Connection Retry Count</td>
<td>Specifies the number of attempts to be made for the central logger to connect to the database server if the database server goes down.</td>
</tr>
<tr>
<td>DB Driver</td>
<td>Specifies the database driver for the database server used by the central logger.</td>
</tr>
</tbody>
</table>
Table 44  Database Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DB URL</td>
<td>Specifies the database server connection URL for the central logger.</td>
</tr>
<tr>
<td></td>
<td>For example, for the MS SQL server, the following value is used:</td>
</tr>
<tr>
<td></td>
<td>jdbc:sqlserver://10.107.174.56:1433;databaseName=asgstat</td>
</tr>
<tr>
<td>DB Username</td>
<td>Specifies the username for the central logger to connect to the database server.</td>
</tr>
<tr>
<td></td>
<td>Default is asguser</td>
</tr>
<tr>
<td>DB Password</td>
<td>Specifies the password for the central logger to connect to the database server.</td>
</tr>
<tr>
<td></td>
<td>Default is asguser</td>
</tr>
<tr>
<td>DB Schema</td>
<td></td>
</tr>
</tbody>
</table>

Security Properties

When you click the Security under Global Properties on the user interface, it allows you to set the following properties:

Table 45  Security Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Common</td>
<td></td>
</tr>
<tr>
<td>Anonymous Partner Name</td>
<td>Specifies a default partner name for the unauthenticated requests.</td>
</tr>
</tbody>
</table>

**OAuth - Defines all the OAuth server related properties.**

See OAuth Server Properties on page 416 for details.
This chapter discusses the full cycle of transaction pipeline processing and discusses the mapping and transformation capabilities of TIBCO API Exchange Gateway software.

Topics

- Transaction Pipeline processing, page 224
- Parsing Step, page 228
- Mappings and Transformations, page 240
- Transformations (XSLT Mapping), page 245
- Mapping Schemas, page 251
Transaction Pipeline processing

TIBCO API Exchange Gateway uses a staged event-driven architecture. TIBCO API Exchange Gateway supports the processing of the transactions into a set of stages for high performance.

The transaction pipeline processing consists of following:

- Request pipeline processing
- Response pipeline processing

Request pipeline processing

The request processing cycle indicates the normal life cycle of the incoming request message and consists of the following phases. This life cycle assumes that there are no errors in any of the processing stage.

1. Apache Level Authentication

   When the partner sends a request using http transport, the first level authentication happens at the Apache server depending upon the first part of uri.

   The authentication type is configured in the apache server configuration file as `asg_mod.conf`. See Configure Apache Server For Basic HTTP Authentication, page 114 for details.

   Following types of authentication are supported:

   - No Authentication - This indicates that the request does not have any user credentials. In this case, the request is processed as an anonymous user.
   - Basic Authentication - This indicates that the request has the user credentials. In this case, the user is authenticated.
   - Digest Authentication
   - Mutual (SSL) Authentication.

   For example, No Authentication is defined as below:

   ```
   <Location / >
   SetHandler asg_rv_inbound_handler
   AsgSubject _LOCAL.asg.north.request
   AsgTimeout 30
   </Location>
   ```
After the request is processed by the Apache server for authentication, the request is passed to the gateway core engine over Rendezvous transport.

2. Operation Identification

When the gateway engine receives the request from the Apache server, it identifies the operation as below:

— For SOAP requests, the operation is identified from the SOAP Action header, and/or URI as defined in the HTTP header.

— For HTTP/XML requests, the operation is identified from the combination of method and URI.

— For HTTP/S REST requests, the operation is identified from the combination of method, URI and the value of some named HTTP header.

The operation details are configured in the Facade Operations tab of the gateway user interface.

3. Partner Identification

After the operation is identified, the partner is identified from the request context message. See Partner Identification on page 348.

4. Request Parsing (Optional)

After the operation and partner are identified from the incoming request, the next step is parse the request. The parse step is an optional preprocessing of the facade request. See Parsing Step, page 228 for details on parsing step of request processing.

5. Authorization

The gateway engine checks if the identified partner is authorized to access the requested operation by finding the associated configuration in Facade Access tab under PARTNER tab of the gateway user interface.

6. Request message validation

If the flag for request validation is enabled, the request message is validated for syntax against XSD for incoming northbound request message.

7. Facade Throttling:

Facade throttling allows you to enforce the commercial throttles for Service level agreements for a partner request. Facade throttling is done on the partner, partner group, partner operations.

After the request reaches this stage in the processing pipeline, a check is made on how often can this partner invoke the operation.

8. Forward northbound Mapping:
After the request passes the facade throttle check, the request is processed by the northbound mapper for any transformations required from operation request message to canonical request message. Whether the mapping is required for this request operation or not, is configured using the Request Transform field on the ROUTING > Facade Operations tab. The transformation details are defined in MAPPING > Mappings tab of the gateway configuration user interface.

By default, If no mappings are defined, then the request message is just copied as the output request message at this stage.

9. Routing:

Based on the operation name and routing key defined in ROUTING > Routing tab of the gateway user interface, the gateway engine determines the target operation endpoint for the incoming request. See Routing, page 255 for details.

10. Service Throttling:

Once the name of the southbound service endpoint is derived for the invoked operation, service throttling policy is applied. Service throttles are technical throttles implemented to protect the over use of the service endpoints.

11. Forward southbound mapping:

If the service throttle is not violated, the request message is processed by the southbound mapper for any transformations required from the canonical request message to the service request message. Whether the mapping is required for this request operation or not, is configured using the Request Transform field on the ROUTING > Target Operations tab. The transformation details are defined in MAPPING > Mappings tab of the gateway configuration user interface.

By default, If no mappings are defined, then the request message is just copied as the output request message at this stage.

12. Invoke southbound service:

This is the final stage where the gateway engine invokes the southbound service for the requested operation.

Response pipeline processing

The response processing cycle indicates the life cycle of the response message and consists of the following phases:

1. Reverse southbound mapping:
After the response is received from the southbound service, the response document is processed for any transformations required from the service response message to canonical response message. Whether the mapping is required for this request operation or not, is configured using the Response Transform Mapping field on the ROUTING > Target Operations tab. The transformation details are defined in Mappings tab of the configuration GUI.

By default, if no mappings are defined, then the response message is just copied as the output response message at this stage.

2. Reverse northbound mapping:

   This stage allows any transformations required from canonical response message to northbound response message. Whether the mapping is required for this request operation or not, is configured using the Reverse Mapping field on the Operations tab. The transformation details are defined in Mappings tab of the gateway user interface.

   This mapping can be used for censor response policy which allows the users to hide certain fields from the response message so that they are not exposed to the requestor.

3. Termination:

   The response message is finally sent back to the original requestor in this stage.

**Generate Transaction**

After the request and response messages are processed, the gateway engine generates the events to audit log the transaction details. The Central logger component receives the events and logs the transaction details in a database or file, as configured.
**Parsing Step**

The parse step is an optional preprocessing of the facade request used both for content validation and for identifying and normalizing control data. Using the parse step you can enhance the downstream processing by setting or overwriting a predefined parameters that are derived either from the request message content or the request transport context. The parsing of the request is done using an XSLT based transformation file configured in the New ProcessBody Transform or ProcessBody Transform field on the ROUTING > FacadeOperations tab of the gateway configuration user interface.

The transformation (XSLT) file defined in the parsing step is used for the following purposes:

- Setting a partner identity for the requester that is derived from the payload content or transport context of the request.
- Setting a routing key that is used to identify the route for the request to the appropriate target operation endpoint.
- Enrich the request’s transaction audit trail logging information.
- Validate the request content with the ability to set an error code and error message when a request doesn’t pass the content validation rules.
- Setting a metric increment for content based throttles.
- Set sticky key for load-balancing with StickyResourceAffinity routing algorithm type.

**Set The Partner Identity For Request**

TIBCO API Exchange Gateway supports multiple identity types for identification of the requester. These identities use different protocol standards to support either transport level identification or request based identification. In addition to the standards based protocols, TIBCO API Exchange Gateway also provides a way to derive custom identities from either the payload message of request or the transport context of request.

The XSLT transformation file in the parse step can be used to populate the partner identity for the request. If the partner identity is set by the parse XSLT file, it will override any other standards based identity associated with the request.
In the situation when a standards based identity is associated with the request which gets overridden by the parse XSLT, the original partner identity is stored as a row in the ASG_TRANSACTION_KEYS table that is associated with the audit trail log record of the request in the ASG_TRANSACTIONS table. The override log records or such credential mapping records will be stored with a value of incomingPartner for the key_type column.

**Set The Routing Key For Request**

For a single facade operation, multiple routes can be configured. Each of the routing configurations map a facade operation to a different target operation or target group. When multiple routes are configured for a facade operation, then the request processing requires to derive a routing key for that request to determine which of the configured routes should process the request. When the routing key should be dynamically derived from the request payload message or the request transport context, the parse step XSLT can be used to set the routing key for the request. See Routing, page 255 Chapter.

**Enrich The Audit Trail Log For Request**

You can extract data from the request payload message or the request transport context and use it to enrich the audit trail logging data that is stored in the Central Logger database. The parse step XSLT transformation supports the following elements in its output document to support this function:

- **Reference Id**

  referenceId is a specific element that is used to store an external transaction id or reference key for the request. Any value mapped into the referenceId element of the output document generated by the parse step XSLT transformation will be stored in the TRN_TRANSACTION_ID column of the ASG_TRANSACTIONS table in the Central Logger database.

- **Key**

  key is a repeatable element in the output document of the parse step XSLT transformation that can be used in more generic fashion. It is typically used to store business keys extracted from the received request. Each populated key element is stored as a separate row in the ASG_TRANSACTION_KEYS table and includes a foreign key to the associated row in the main ASG_TRANSACTIONS table of the Central Logger table. The value of the key element is stored in the KEY_VALUE column.

  The key element provides following attributes:
  - @type
  - @log
@type attribute

When you set a @type attribute for the key element, the @type attribute will be stored in the KEY_TYPE column in the ASG_TRANSACTION_KEYS table.

The use of key element and @type attribute for the key element allows users to search for specific audit trial logging records in the Central Logger database via the business keys from the request message. This also provides the flexibility to enrich the standard audit trail logging data with any key/value pair based contextual information without the need for customization of the TIBCO API Exchange Gateway itself.

@log attribute

If you do not want to store the parameters to be stored in the Central Logger database, you can use the @log attribute of the key element. The @log attribute is a boolean attribute and its value determines whether this key should be stored in the Central Logger database or not. When there is no @log attribute set for key elements, a default value of true will be set for the key elements and the key elements will be stored as rows in the ASG_TRANSACTION_KEYS table of the Central Logger database. If you do not want to store the key elements in the database, set the @log attribute to false.

You can use the key elements populated by the parse step XSLT transformation step to pass parameters to custom policies that can be developed leveraging the custom extension mechanism. The SampleStage custom extension in the ASGDefaultImplementation project uses the key elements.

Validate The Request Content

You can use XPath expressions in the parse step XSLT transformation. This allows you to apply content validation logic to the request payload message. When the requests fails a validation check, an appropriate error code and associated error message can be set in the parse step XSLT transformation.

If the parse step XSLT transformation returns an error code the transaction is terminated and an error response is generated and returned to the client. This functionality allows users to create parse XSLT to return the error codes and descriptions that are expected by the API of which it is part. Then if an error XSLT is configured for the operation it will be called so any API-specific error document can be created.

Set Metric Increment For Content Based Throttles

TIBCO API Exchange Gateway supports the notion of content based throttles, by extending a throttle configuration with a monitor that allows you to define a content based metric increment counter.
The content based throttles allows you to define custom throttle policies that enforce API usage controls on business metrics such as for example order amounts. When a throttle configuration has been extended with a monitor definition, the parse step XSLT transformation is used at run-time to map the business metric containing element from the request payload message to a throttle monitor specific element in the output document generated by the parse step XSLT transformation. In order to support the content based throttle extension mechanism for multiple throttle definitions on a request's throttle chain the monitor element is defined as a repeatable element in the output document of the parse step XSLT transformation. The monitor element itself is a node element that contains two elements which need to be defined when setting content based throttle metric increment. The first element of the monitor node element is the metric name element that must contain the throttle name to which the content based metric counter increment must be applied. The second element of the monitor node element is the metric increment that contains the actual counter increment that has been derived from the request message payload and needs to be applied to the throttle.

For more information on how to configure and use content based throttles, See Content Based Throttles, page 310 section.

Set Sticky Key For Load-Balancing With StickyResourceAffinity

TIBCO API Exchange Gateway supports multiple load-balancing policy types for target operations that are grouped within an target group. When a load-balancing policy with StickyResourceAffinity is configured for a target group, then a parse step XSLT transformation is used to derive the sticky key from the request payload message or request transport context. The output document as generated by the parse step XSLT transformation contains a stickyRoutingKey element that need to be set to associate a sticky routing key to the request for further processing.

For more information on how to configure and use load-balancing policies for target groups with StickyResourceAffinity, See StickyResourceAffinity Target Operation Group Configuration, page 284.
**Parsing XSLT Documents**

The XLST transformation file used in the parse step of request processing has a pre-defined input document as well as a pre-defined output document.

**Input Document**

The input document used by the XLST transformation file of parse step has the same structure as the input document that is used by other mapping and transformation steps in the TIBCO API Exchange Gateway. The structure of this generic transformation and mapping input document is described in section Mapping Schemas, page 251.

**Output Document**

The output document is unique for the XLST transformation file of the parse step as it supports specific behaviour of the TIBCO API Exchange Gateway.

Table Elements And Attributes of Parsing Output Document, page 232 describes the elements and attributes of output document used for parsing step. See Parsing Output Document Schema, page 236:

<table>
<thead>
<tr>
<th>XML Node</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>/output</td>
<td>Sequence of Complex Type</td>
<td>Yes</td>
<td>Root element for the document.</td>
</tr>
<tr>
<td>/output/requester</td>
<td>String</td>
<td>No</td>
<td>This value is stored in the RequestorId attribute of the API Exchange Gateway transaction object. This is currently not used.</td>
</tr>
<tr>
<td>/output/serviceInterface/Version</td>
<td>String</td>
<td>No</td>
<td>This value is stored in the ServiceInterfaceVersion attribute of the API Exchange Gateway transaction object. This is currently not used.</td>
</tr>
</tbody>
</table>
Table 46  Elements And Attributes of Parsing Output Document

<table>
<thead>
<tr>
<th>XML Node</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>/output/referenceId</td>
<td>String</td>
<td>No</td>
<td>The value stored in this field will be stored by the Central Logger in the TRN_TRANSACTION_ID column of the main ASG_TRANSACTIONS table. It may be used to store an external transaction id or reference key that is extracted from the payload message of the received request.</td>
</tr>
<tr>
<td>/output/serviceId</td>
<td>String</td>
<td>No</td>
<td>This is currently not used.</td>
</tr>
<tr>
<td>/output/timestamp</td>
<td>String</td>
<td>No</td>
<td>This is currently not used.</td>
</tr>
<tr>
<td>/output/correlationId</td>
<td>String</td>
<td>No</td>
<td>This is currently not used.</td>
</tr>
<tr>
<td>/output/identityId</td>
<td>String</td>
<td>No</td>
<td>This is currently not used.</td>
</tr>
<tr>
<td>/output/opCoId</td>
<td>String</td>
<td>No</td>
<td>This value is stored in the OpCo attribute of the API Exchange Gateway transaction object. It is currently not used.</td>
</tr>
<tr>
<td>/output/partnerId</td>
<td>String</td>
<td>No</td>
<td>This element is used to perform content based identity mapping for the requester. It must contain the partner name as configured in the gateway user interface.</td>
</tr>
<tr>
<td>/output/routingKey</td>
<td>String</td>
<td>No</td>
<td>If routingKey element is present, it's value will be used to set the routing key for the request to determine which target operation or target group the request should be routed to.</td>
</tr>
<tr>
<td>/output/stickyRoutingKey</td>
<td>String</td>
<td>No</td>
<td>This element is used to set the sticky key for a request for a load-balancing policy with StickyResourceAffinity target operation type.</td>
</tr>
</tbody>
</table>
Table 46  Elements And Attributes of Parsing Output Document

<table>
<thead>
<tr>
<th>XML Node</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>/output/monitor</td>
<td>Sequence of Complex Type</td>
<td>No</td>
<td>This is a node element for defining content based throttle monitor configuration.</td>
</tr>
<tr>
<td>/output/monitor/metricName</td>
<td>String</td>
<td>Yes</td>
<td>Specifies a throttle name for which a content based counter increment needs to be set.</td>
</tr>
<tr>
<td>/output/monitor/metricIncrement</td>
<td>Integer</td>
<td>Yes</td>
<td>Specifies content based counter increment value that will be applied to the throttle as defined in the metricName element of the monitor.</td>
</tr>
<tr>
<td>/output/key</td>
<td>String</td>
<td>No</td>
<td>This is a repeating element that may be used to store business keys extracted from the received request. The keys can be accessed by custom extensions and are also logged to the transaction database by the CentralLogger component.</td>
</tr>
<tr>
<td>/output/key@type</td>
<td>String</td>
<td>No</td>
<td>Specifies the key used to store the contextual information in the Central Logger database. It’s value will be stored in the KEY_TYPE column of the ASG_TRANSACTION_KEYS table.</td>
</tr>
</tbody>
</table>
### Table 46: Elements And Attributes of Parsing Output Document

<table>
<thead>
<tr>
<th>XML Node</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
</table>
| /output/key@log   | Boolean            | No       | • This is a boolean field used to suppress the logging of a key in the Central Logger database. This attribute is useful when key is used to pass parameters to custom policies that are not required to be logged.  
                      • Default value is True, including when you omit the attribute and do not set it.  
                      • This is applicable when you set the value to False. |
| /output/tns:context | Complex Type       | No       | This is currently not used.                                                                                                                                                                             |
| /output/errorCode | String             | Yes      | • If the parse step returns an errorCode then the further processing of the request is terminated and the response is generated by the fault mapper. If this value is registered in the Error Maps of API Exchange Gateway and a FaultReport XSLT is defined on the Facade operation then the mapping will succeed. Otherwise a default value of 1001 will be used.  
                      • Use of the errorCode and errorMessage elements are mutually exclusive with the other elements in the output document of the parse step XSLT transformation. |
| /output/errorMessage | String             | Yes      | Specifies the detailed error description which indicates that the validation check has been violated.                                                                                                                                                       |
Parsing Output Document Schema

Below is the schema for the output document used for parsing step:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:tns="http://www.tibco.com/schemas/asg/context"
elementFormDefault="qualified" attributeFormDefault="unqualified">
  <xs:annotation>
    <xs:documentation>This schema describes the structure of the XML document that should be returned from the ActiveMatrix Service Gateway's "parse" step</xs:documentation>
  </xs:annotation>
  <xs:import namespace="http://www.tibco.com/schemas/asg/context"
schemaLocation="context.xsd"/>
  <xs:element name="output">
    <xs:annotation>
      <xs:documentation>If the root document contains a ctx:context element (from namespace http://www.tibco.com/schemas/asg/context) it will be merged into the ASG Transaction's context document. This allows the parse step to pass arbitrary content to one of the other mappers</xs:documentation>
    </xs:annotation>
    <xs:complexType>
      <xs:choice>
        <xs:sequence minOccurs="0" maxOccurs="unbounded">
          <xs:choice minOccurs="0">
            <xs:element ref="requester" minOccurs="0"/>
            <xs:element ref="serviceInterfaceVersion" minOccurs="0"/>
            <xs:element ref="referenceId" minOccurs="0"/>
            <xs:element ref="serviceId" minOccurs="0"/>
            <xs:element ref="timestamp" minOccurs="0"/>
            <xs:element ref="correlationId" minOccurs="0"/>
            <xs:element ref="identityId" minOccurs="0"/>
            <xs:element ref="opCoId" minOccurs="0"/>
            <xs:element ref="partnerId" minOccurs="0"/>
            <xs:element ref="routingKey" minOccurs="0"/>
            <xs:element ref="stickyRoutingKey" minOccurs="0"/>
            <xs:element ref="monitor" minOccurs="0" maxOccurs="unbounded"/>
            <xs:element ref="key" minOccurs="0" maxOccurs="unbounded"/>
            <xs:element ref="tns:context"/>
          </xs:choice>
        </xs:sequence>
      </xs:choice>
    </xs:complexType>
  </xs:element>
  <xs:element ref="errorCode"/>
  <xs:element ref="errorMessage"/>
</xs:choice>
</xs:sequence>
</xs:choice>
```
</xs:complexType>
</xs:element>
<xs:element name="requester" type="xs:string">
<xs:annotation>
<xs:documentation>This value is stored in the RequestorId attribute of the API Exchange Gateway transaction object. This is currently not used.</xs:documentation>
</xs:annotation>
</xs:element>
<xs:element name="serviceInterfaceVersion" type="xs:string">
<xs:annotation>
<xs:documentation>This value is stored in the Service-InterfaceVersion attribute of the API Exchange Gateway transaction object. This is currently not used.</xs:documentation>
</xs:annotation>
</xs:element>
<xs:element name="referenceId" type="xs:string">
<xs:annotation>
<xs:documentation>The value stored in this field will be stored by the Central Logger in the TRN_TRANSACTION_ID column of the main ASG_TRANSACTIONS table. It may be used to store an external transaction id or reference key that is extracted from the payload message of the received request.</xs:documentation>
</xs:annotation>
</xs:element>
<xs:element name="serviceId" type="xs:string">
<xs:annotation>
<xs:documentation>Currently not used</xs:documentation>
</xs:annotation>
</xs:element>
<xs:element name="timestamp" type="xs:string">
<xs:annotation>
<xs:documentation>Currently not used</xs:documentation>
</xs:annotation>
</xs:element>
<xs:element name="correlationId" type="xs:string">
<xs:annotation>
<xs:documentation>Currently not used</xs:documentation>
</xs:annotation>
</xs:element>
<xs:element name="identityId" type="xs:string">
<xs:annotation>
<xs:documentation>Currently not used</xs:documentation>
</xs:annotation>
</xs:element>
<xs:element name="opCoId" type="xs:string">
<xs:annotation>
<xs:documentation>This value is stored in the OpCo attribute of the API Exchange Gateway transaction object. This is currently not used.</xs:documentation>
</xs:annotation>
</xs:element>
<xs:element name="partnerId" type="xs:string">
<xs:annotation>
This element is used to perform content based identity mapping for the requester. It must contain the partner name as configured in the Configuration UI. The original partner id will be stored as an address element with type "incomingPartner" so that the data will be logged to the central logger.

This element is used to set the sticky key for a request for which a load-balancing policy with StickyResourceAffinity.

Node element for defining content based throttle monitor configuration.

Throttle name for which a content based counter increment needs to be set.

Content based counter increment value that will be applied to the throttle as defined in the metricName element.

This element may be used to store business keys extracted from the received request. These can be accessed by custom extensions and are also logged to the transaction DB by the CentralLogger.
<xs:element name="errorCode" type="xs:string">
  <xs:annotation>
    <xs:documentation>If the parse step returns an errorCode then the further processing of the request is terminated and the response is generated by the fault mapper. If this value is registered in ASG's "Error Maps" and a "FaultReport XSLT" is defined on the Operation then the mapping will succeed. Otherwise a default value of 1001 will be used.</xs:documentation>
  </xs:annotation>
</xs:element>

<xs:element name="errorMessage" type="xs:string">
  <xs:annotation>
    <xs:documentation>The detailed error description</xs:documentation>
  </xs:annotation>
</xs:element>
Mappings and Transformations

TIBCO API Exchange Gateway provide message transformations using mappings. This allows the users to map the gateway endpoint operation request and response messages to the gateway reference operation request and response messages with another format.

The mapping capabilities of TIBCO API Exchange Gateway allows you to decouple the northbound operations interface and southbound services interface by providing a mutual common canonical message format between the operations interface and services interface.

TIBCO API Exchange Gateway supports the mappings at various levels and reduces the number of point-to-point mappings between gateway endpoint operations (exposed by the gateway) and gateway reference operations (service clients of the gateway that invoke internal service operations).

TIBCO API Exchange Gateway provides the transformation of request and response messages at following four points:

- Facade request handler to Router boundary:
  After the request has been received by Facade request handler and before it is been passed to the Router.

- Router to Service endpoint handler boundary:
  After the request has been routed but before it is passed to the service endpoint handler.

- Service Endpoint handler to Router boundary:
  After the response has been received from the service endpoint handler and before it is passed to the Router.

- Router to Facade request handler
  After the response has been routed from the router to the facade request handler and before the response is sent back to the original requestor.

The mapping transformations allows you to:

- Access the multiple versions and formats of service APIs
- Add semantic content validation rules to the incoming requests
- Access the fields from the request context and payload
- Change the routing of request and response messages based on the error handling in case of message validation failures
- Protect the service end-points using service validation policies
TIBCO API Exchange Gateway supports the transformations of both request and response messages using:

- Forward mapping - The transformations are done from request canonical form to backend service API as per defined mapping.
- Reverse mapping - The transformations are done from backend service API to response canonical form as per defined mapping.

Mappings allows you to have six different versions of documents as below:
- Northbound request message
- Canonical request message
- Southbound request message
- Southbound response message
- Canonical response message
- Northbound response message

TIBCO API Exchange Gateway supports mapping at both northbound and southbound sides.

**Northbound Mapping**
TIBCO API Exchange Gateway allows you to map:
- Map northbound request message to canonical request
- Map canonical response to northbound response message
- Map errors to northbound fault response

Northbound mapping configuration is defined in ROUTING > Facade Operations and PARTNER > Partner Operations tab of the GUI.

**Southbound Mapping**
TIBCO API Exchange Gateway allows you to map:
- Map canonical request to southbound request message
- Map southbound response message to canonical response

Southbound mapping configuration is defined in ROUTING > Target Operations tab of the GUI.

**Mapping Types**
Following types of mapping are supported:
Identity mapping

Identity mapping is a single mapping doing no transformation, and just copies input message to output message recursively. This is called **Pass-Through** Mapping.

Rendezvous mapping

Rendezvous mapping allows you to access an external mapping handler as a service using Rendezvous. Rendezvous starter process that calls service endpoint handler and listens to the matched requested subject must exist.

Rendezvous mapping provides external processes to execute mappings between corresponding but not compatible APIs. This provides a set of services called with Rendezvous messages that transform given input message(s) to another message(s). The gateway core engine calls the mappers services whenever it transforms messages between northbound, canonical and southbound form.

XSLT mapping

XSLT mapping describes embedded mapping transformation. The transformation of the request and response documents are done according to the XSLT files defined in the mappings configuration. This can be defined for both forward and reverse mappings. See Transformations (XSLT Mapping), page 245 for details.

Mapping Configuration

The gateway user interface allows you to add the mapping configuration for northbound and southbound request messages (gateway operation endpoint) as well as southbound response and northbound response messages (gateway reference endpoint).

This section explains the steps to configure mappings for gateway endpoint operation and gateway service endpoint.

Identity mapping type configuration

By default, mapping id ID is available for forward and reverse mappings to operation endpoint and reference endpoint. ID mapping id refers the Identity mapping. See Identity mapping, page 242.

Rendezvous mapping type configuration

1. Start the GUI, if not already started. See Starting GUI, page 166 for details.
2. Click the **Mapping** tab
   Enter the following parameters, as defined below:

**Table 47  RV Mapping Type Configuration**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mapping Name</td>
<td>Name of mapping</td>
</tr>
<tr>
<td>Type</td>
<td>RV</td>
</tr>
<tr>
<td>Subject</td>
<td>RV subject to send the mapping request to</td>
</tr>
<tr>
<td>Transformation</td>
<td>Transformation to perform on the message. If it is default, the transformation will be determined by mpSubject. Other values force specific transformation.</td>
</tr>
</tbody>
</table>

**XSLT mapping type configuration**

1. Start the GUI, if not already started. See *Starting GUI, page 166* for details.
2. Click the **Mappings** tab.
3. Enter the following parameters, as defined below:

**Table 48  XSLT Mapping Type Configuration**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mapping Name</td>
<td>Name of mapping</td>
</tr>
<tr>
<td>Type</td>
<td>XSLT (Select from the drop down list box)</td>
</tr>
<tr>
<td>New File</td>
<td>File name containing the transformations (xsl file)</td>
</tr>
<tr>
<td>Existing Files</td>
<td>File name containing the transformations (xsl file)</td>
</tr>
</tbody>
</table>
| Response Type   | Output Type from the transformation. It can have following values:  

- Payload: Output document from the transformation contains just the payload.  
- Full: Output document from the transformation contains both the request context and the payload.  
  <Details> (Payload | Full) |
Assign to the gateway operation endpoint

After the mappings are registered, they are assigned to the gateway operation endpoint for forward and reverse mappings to be performed for request operations. They are defined in the Facade Operations and Facade Access tab of the GUI. See Facade Operations, page 193 and Facade Access, page 206.

Assign to the gateway reference endpoint

After the mappings are registered, they are assigned to the gateway operation endpoint for forward and reverse mappings to be performed for request operations. They are defined in the Target Operations tab of the GUI. See Add A New Target Operation, page 195.
Transformations (XSLT Mapping)

To define XSLT mapping for facade or target request and response messages, files are defined in the **Mapping** tab of the configuration UI. See XSLT mapping type configuration, page 243.

For the XSLT mapping, the `responseType` indicates the output of the transformations, which can be as follows:

**Payload**

If the `responseType` is specified as `payload`, the output document of transformation contains just the payload and is used as the request payload for the next stage in the request pipeline processing.

The output document of the mapping transformation either becomes a canonical request or a southbound request, just depending upon where you are in the request processing pipeline.

For example, the output document from the northbound forward mapping is a payload XML document. This becomes the canonical request and is populated as the input for the southbound forward mapping.

**Full**

If the `responseType` is specified as `full`, it allows you to access the additional fields from the transaction object. The main field is request context field which contains the HTTP headers information, URI, user name, and other authentication fields.

This provides the mapping and transformations of full transaction (request) object. The transaction context contains the header information and the payload. Once the transformation is done:

- the value of the transaction context is replaced by the output document from the transformation.
- the payload is extracted from the transaction context and is used for the next stage in the request pipeline processing.

The transformation with `responseType` as `full` allows you to:

- map and update the document with additional fields available in the request context (HTTP headers, JMS headers etc).
- Set error code for content validation.
- create documents for enumeration orchestration.
Following functionality is implemented using the full responseType of mapping transformation capability:

Set error codes for content validation

With the mapping transformation with responseType as full, it allows you to set the error code when the validation of the contents fails. The output of the transformation is a document which contains the updated transaction object with error code.

For example, in case of extra validation of requests like semantic validation using XPath, it sets the error code if it fails. With full mapping type, the error code is updated in the transaction object. The request message processing follows the error handler path in the request processing pipeline, then sent back to the original requestor with the error message. See Implement Request Validation, page 246.

Validation

This functionality allows you to do the schema validation against XSD for an incoming request message. Request validation is performed during the request parsing step and after the Authorization step in the request processing pipeline.

Enabling Validation

To enable the request validation, perform the following:

- Set Operation Features for the operation
  
  In the Operations tab of configuration GUI, set the Operation Features parameter as below:

  Operation Features Validation

- Configure Schema

  In the Schemas tab of configuration GUI, set the XSD file reference as below:

  New XSD File Select the location of the XSD file

Implement Request Validation

Validation of requests is supported for the following stages:

- Request Parsing

Request message can be validated at the request parsing stage. Following elements exist in the schema for the output message to be used by the Parse XSLT:

<errorCode>
<errorMessage>

If errorCode is set, then the request is handled by the error processing path of the processing pipeline. An error XSLT, as defined, for the operation generates an error response message by executing the FaultReportXSLT that is defined for the operation. The request is rejected with the error message sent back as the response to the request.

- Mapping and Transformation Stage

Extended request or response validation can be implemented at any of the four mapping steps. If the message does not pass the validation rules, an error message is constructed and returned in the message.

To set an Error Code in the transformation step, do the following things:

- Include the following name spaces in the XSLT
  
  `xmlns:map="http://www.tibco.com/asg/mapping"
  xmlns:err="http://www.tibco.com/schemas/asg/error"

- With these namespaces, use the following elements in the output message of the mapper:

  `<map:mapping-result>`
  `<map:error>`

- Within "http://www.tibco.com/schemas/asg/error" namespace, use the following error elements to set error code and message details:

  `<errorCode>`
  `<errorMessage>`
  `<errorBody>`
  `<errorDetails>`
  `<nestedError>`
  `</errorDetails>`

With the mapping registered as Full mode, if an error element within the registered namespaces is found in the output message, the error attributes for the request object are set with the corresponding values in the elements from the `/schemas/asg/error` schema from the output message.

This instructs the gateway engine to follow the error handling path of the processing pipeline. An error XSLT, as defined, for the operation generates an error message. The request is rejected and the error message is sent back as the request response.

In Parsing step, you can only set `errorCode` and `errorMessage`. 
In any of the four mapping steps, you can return four error elements: **errorCode**, **errorMessage**, and additional elements as **errorBody** and **errorDetails**. In case, if **errorCode** and **errorMessage** are only set, then the request processing is handled in same way as in the parse step. The additional elements, **errorBody** and **errorDetails** elements allows you to construct the actual error response message for the request, which can override the creation of error message by the FaultReportXSLT. This helps to speed up processing as no second XSLT action have to be executed.

**Map the protocol headers in request context**

This functionality allows the users to do the transformations of the protocol headers with in the request and response messages which includes the transport related information.

For any mapper using the responseType as **full** allows you to access the request context field. The request context field contains the transport level information and is available for any transformations. This allows you to:

- Map the transport header properties and pass it to the next stage in request processing pipeline
- Set the transport header properties. For example, when the JMS transport is used as the channel for incoming request, the JMS priority property can be set based on the XPATH value received in a request.

**Enumeration Orchestration**

Enumeration Orchestration is implemented using the mapping capability of TIBCO API Exchange Gateway.

With parallel orchestration, a single inbound request is split into a set of multiple outbound sub-requests. Each sub-request may be routed differently to various service endpoints. After processing and receiving the responses for each sub-requests, all responses are recombined into a single response message for the original inbound request.

You can define the transformations for the northbound request forward mapping in such a way that the output document contains the sequence that defines the enumeration orchestration. This output document from this transformation serves as an input for the southbound forward request mapping. This contains the `<asg_map:repeat>` tag for enumeration and instructs the gateway engine to split the requests.
Enumeration orchestration is done at the southbound forward mapping meaning that one payload is split into multiple sub-requests independently. It also allows you to override the routing key. For each sub-request, the routing key can be defined so that each sub-request is forwarded to a separate service endpoint. This allows you to modify the routing of sub-requests to a different service at the southbound forward mapping boundary.

Following shows the example snippet of forward southbound mapping transformation for enumeration orchestration:

```xml
<asg_map:repeat>
  <asg_map:payload-xml>
    <asg_map:mapping-result>
      <asg_map:routingKey>getbookbyAuthor_AAA.HTTP</asg_map:routingKey>
      <asg_map:payload>
        <SOAP-ENV:Envelope>
          <SOAP-ENV:Body>
            <...
            <...
          </SOAP-ENV:Body>
        </SOAP-ENV:Envelope>
      </asg_map:payload>
    </asg_map:mapping-result>
  </asg_map:payload-xml>
  <asg_map:payload-xml>
    <asg_map:mapping-result>
      <asg_map:routingKey>getbookbyTitle_BBB.JMS</asg_map:routingKey>
      <asg_map:payload>
        <SOAP-ENV:Envelope>
          <SOAP-ENV:Body>
            <...
            <...
          </SOAP-ENV:Body>
        </SOAP-ENV:Envelope>
      </asg_map:payload>
    </asg_map:mapping-result>
  </asg_map:payload-xml>
</asg_map:repeat>
```
For example, GetLocation is a service which finds the location for a single service. This service can be exposed as a service to find location for a group of services. This means that the gateway has to map one request to multiple requests per service. For this case, the payload in the operation request contains a sequence of multiple requests which has to be spilt and sent individually to each service endpoint.

**Re-combination of response documents**

Transformations are defined at the southbound reverse response mapper boundary to re-join the multiple response documents for each sub-request into a single response document.
Mapping Schemas

Mapping Container

The input document to the XSLT file for XSLT transformations is described by the following schema(XSD):

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"

elementFormDefault="unqualified"

attributeFormDefault="unqualified">
<xs:element name="transformation" type="transformationType"/>
<xs:complexType name="transformationType">
<xs:sequence>
<xs:element name="nbRequest" type="stageType" minOccurs="0"/>
<xs:element name="cnRequest" type="stageType" minOccurs="0"/>
<xs:element name="sbRequest" type="stageType" minOccurs="0" maxOccurs="unbounded"/>
<xs:element name="sbResponse" type="stageType" minOccurs="0" maxOccurs="unbounded"/>
<xs:element name="cnResponse" type="stageType" minOccurs="0"/>
<xs:element name="nbResponse" type="stageType" minOccurs="0"/>
<xs:element name="context" type="stageType" minOccurs="0"/>
</xs:sequence>
</xs:complexType>
<xs:complexType name="stageType">
<xs:attribute name="href" type="xs:string"/>
</xs:complexType>
</xs:schema>
```

The request payloads and the request context is passed to the XSLT as a map, keyed using the values of the “href” attribute for each element. In order to access the actual payload, it is necessary to load it using the document() function.

The schemas contains the following elements:

- **nbRequest** – This element represents the northbound request payload.
- **cnRequest** – This element represents the request payload after the northbound forward mapping has been applied.
- **sbRequest** – This one element is present for each of the array of request payloads after the southbound forward mapping has been applied
- **sbResponse** – This one element for each request sent to the southbound service endpoint. It contains either the received response or an error document.
- cnResponse – This element represents the response payload generated by the southbound reverse mapping.
- nbResponse – This element represents the response payload after the northbound reverse mapping has been applied.
- context – This element represents additional context document related to the request. It contains the headers from request and response and transport related information.

At any stage during request processing, only the versions of the request payload created up to that point are available. Though the mapping container has the corresponding element present, an attempt to load the payload via document() causes the XSLT processor to throw an error.

For example to map a value when the received request is SOAP use the following XSLT snippet:

```xml
<xsl:variable name="nbRequest">
  <xsl:value-of select="/transformation/nbRequest/@href"/>
</xsl:variable>
<xsl:choose
  xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
><xsl:when
  test="count(document($nbRequest)/soap:Envelope/soap:Body)=1">
  This is a SOAP request
</xsl:when>
<xsl:otherwise>This is not a SOAP request</xsl:otherwise>
</xsl:choose>
```

**Context Document**

The contents of the context document are described by the following XSD:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:c="http://www.tibco.com/schemas/asg/context"
  targetNamespace="http://www.tibco.com/schemas/asg/context"
  elementFormDefault="qualified"
  attributeFormDefault="unqualified">
  <xs:element name="context">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="c:entry" minOccurs="0"
          maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```
If the incoming request is received from Apache HTTP server, the context document contains an entry with key set to `asg:httpRequest`, which conforms to the following XSD:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns="http://www.tibco.com/asg/protocols/http"
  targetNamespace="http://www.tibco.com/asg/protocols/http"
  elementFormDefault="qualified"
  attributeFormDefault="unqualified">
  <xs:element name="request">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="server-ip" type="xs:string" minOccurs="0"/>
        <xs:element name="server-port" type="xs:string" minOccurs="0"/>
        <xs:element name="client-ip" type="xs:string" minOccurs="0"/>
        <xs:element name="client-port" type="xs:string" minOccurs="0"/>
        <xs:element name="scheme" type="xs:string" minOccurs="0"/>
        <xs:element name="method" type="xs:string" minOccurs="0"/>
        <xs:element name="request-uri" type="xs:string" minOccurs="0"/>
        <xs:element name="protocol-version" type="xs:string" minOccurs="0"/>
        <xs:element name="header" minOccurs="0" maxOccurs="unbounded">
          <xs:complexType>
            <xs:simpleContent>
              <xs:extension base="xs:string">
                <xs:attribute name="name" type="xs:string"/>
              </xs:extension>
            </xs:simpleContent>
          </xs:complexType>
        </xs:element>
        <xs:element name="body" type="xs:string" minOccurs="0"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```
Chapter 8  Routing

This chapter explains the routing functionality of TIBCO API Exchange Gateway software.

Topics

- Routing Overview, page 256
- Routing Key, page 257
- How To Derive And Configure Routing Key, page 258
- Target Operation Group, page 266
- Routing Algorithms For Target Operation Group, page 268
- Target Operation Group Configuration, page 278
- HealthCheck For Reference, page 289
Routing Overview

The routing capabilities of TIBCO API Exchange Gateway determines the target operation to process a facade request. API Exchange Gateway uses the routing key to route a facade request to either one of the following:

- **Target Operation**
  
  A target operation is an external operation which is called by the gateway to process a facade request.

- **Target Operation Group**
  
  A target operation group is used to group multiple target operations, which helps the gateway core engine to balance the load of processing requests across the target operations. See Target Operation Group on page 266.

When the gateway core engine routes the facade request to a target operation group, an appropriate target operation within the target operation group is selected based on the type of the target operation group. See Types of Target Operations Group on page 266 for details.

Routing key is used to determine the target operation or a target operation group to process an incoming facade request. See Routing Key on page 257 for details.

API Exchange Gateway provides following options to derive a routing key:

**Content Based Routing**

When the routing key is generated from the content (data) of the request message to route the client request to a target operation or target operation group, this routing is defined as content based routing.

**Context Based Routing**

When the routing key is derived from the HTTP or JMS header fields header fields in the request context message to route the client request to a target operation or target operation group, this routing is defined as context based routing.
Routing Key

A routing key is used by the gateway core engine to select a target operation or target operation group for an incoming facade request. Thus, the routing key is the key factor to route a facade request for processing. Based on the routing key, a facade request can be routed to any target operation or target operation group.

Typically, a routing key is extracted from the parsing of the facade request message using the transformation (XSLT) files. You can conditionally evaluate the facade request message and parse the data content as well as the context of the request message in the parsing step of the request processing pipeline to derive a routing key. If no routing key is derived from the parsing of the request message, default routing key is used.

The routing key can have following values:

- **default**: If the `ProcessBody` transform field for an facade operation does not generate a routing key, the routing key is `default`.

- **routingKeyValue**: `routingKeyValue` indicates a specific value of the routing key which is obtained either as one of the following ways:
  
  — A value populated from the transformation (XSLT) file using the `routingKey` element tag. This XSLT file is uploaded in the `ProcessBody` transform field for a facade operation configuration.
  
  — Any hard coded value.

A default routing key is used when a routing key is not configured by the user.

If you want to use a custom (non-default) routing key to route a facade request to an appropriate target operation or target operation group, you must do following primary actions:

- Derive A Routing Key
- Configure A Routing Key

To derive a routing key, you must define a XSLT file with `routingKey` element tag and upload this XSLT file in the `ProcessBody Transform` for a facade operation configuration. In such a case, the incoming facade operation request is parsed as per the defined XSLT file and a routing key is returned. After the routing key is populated from the transformation, the gateway core engine checks the routing key configuration to determine the target operation or target operation group for a facade operation request. The routing key configuration contains the routing key, the facade operation name and the target operation name or target operation group name and is configured in the `Routing` tab of the gateway user interface. See How To Derive And Configure Routing Key on page 258 for details.
How To Derive And Configure Routing Key

Follow the steps as below to derive and configure a routing key:

Task H Define a Transformation File

You should define a transformation (XSLT) file. The transformation file contains the routingKey element tag to populate the routing key.

See following references:

- See Routing Schema document on page 258 as a reference to the XSLT file schema for routing key.
- See Example XSLT File To Derive Routing Key on page 259.

Routing Schema document

The following elements are available as input for the transformations and can be used to derive the routing key:

- Facade operation request content (as defined by nbRequest element).
- Facade operation request context (as defined by context element).

In the request processing pipeline of a facade operation request, the input document to the XSLT file for parsing the facade request is defined by the following schema:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<x:schema xmlns:x="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="unqualified"
  attributeFormDefault="unqualified">
  <x:element name="transformation" type="transformationType"/>
  <x:complexType name="transformationType">
    <x:sequence>
      <x:element name="nbRequest" type="stageType" minOccurs="0"/>
      <x:element name="cnRequest" type="stageType" minOccurs="0"/>
      <x:element name="sbRequest" type="stageType" minOccurs="0" maxOccurs="unbounded"/>
      <x:element name="sbResponse" type="stageType" minOccurs="0" maxOccurs="unbounded"/>
      <x:element name="cnResponse" type="stageType" minOccurs="0"/>
      <x:element name="nbResponse" type="stageType" minOccurs="0"/>
      <x:element name="context" type="stageType" minOccurs="0"/>
    </x:sequence>
  </x:complexType>
</x:schema>
```
Example XSLT File To Derive Routing Key

Refer to the following transformation (XSLT) file shipped with GetLocation example:

- \texttt{ASG\_CONFIG\_HOME/GetLocation/xslt/operations/parse\_getLocation.xml}

- The example illustrates that the routing key is populated based on the value of \texttt{opCoId}, which is derived as a substring of the address element in the request message.

```xml
<routingKey>
    <xsl:choose>
        <xsl:when test="$opCoId != ''" select="$opCoId"/>
        <xsl:otherwise>undefined</xsl:otherwise>
    </xsl:choose>
</routingKey>
```

Task I  Go To ROUTING Tab

1. Start the GUI server, if not already running. See Starting GUI, page 166.
2. Login to the gateway user interface with your credentials.
3. Add a new project or select an existing project under Projects.
4. Click on ROUTING tab on right hand side.

Task J  Upload the Transformation (XSLT) File

To upload the XSLT transformation file for an facade operation request, perform the following steps:

1. Ensure that you are on the Routing tab of the gateway user interface. See Go To ROUTING Tab on page 259.
2. Click on Facade Operations tab on the top menu.
3. Add a new facade operation or choose an existing facade operation.
4. Upload the XSLT transformation file as below:
   a. Click **Choose File** button in the **New ProcessBody Transform** field to select a XSLT transformation file.
   b. Click **Open** button to upload the XSLT file.

   - You must define the XSLT file to upload in the **New ProcessBody Transform** field. See Define a Transformation File on page 258.
   - If the XSLT file is located in the \(\text{ASG\_CONFIG\_HOME/ASG\_ConfigName/xslt/operations} \) directory, then you can choose the XSLT file in the **Existing ProcessBody Transform** field.
   - If **New ProcessBody Transform** or **ProcessBody Transform** field of the facade operation configuration contains no XSLT file or has a XSLT file but no **routingKey** element tag, the gateway core engine uses the Routing configuration with default routing key.

5. Save changes to the configuration.

### Task K  Routing Configuration

You can configure the routing key to map a facade request to a target operation or a target operation group using the **ROUTING > Routing** tab of the gateway user interface as follows:

- See Routing Configuration For A Target Operation on page 260
- See Routing Configuration For A Target Operation Group on page 261

### Routing Configuration For A Target Operation

To configure the routing key for a target operation, perform the following steps:

1. Ensure that you are on the **ROUTING** tab of the gateway user interface. See Go To ROUTING Tab on page 259.
2. Click **Routing** tab on the top menu.
3. Enter the following parameters, as defined below

Table 49  Routing Configuration For A Target Operation

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operation Name</td>
<td>• Specifies the name of the facade operation. The operation name must be defined in the Facade Operations tab of the user interface.</td>
</tr>
<tr>
<td></td>
<td>• This field is a drop down list box, so you can select an pre-defined operation.</td>
</tr>
<tr>
<td></td>
<td>• This is a Required field.</td>
</tr>
<tr>
<td>Routing Type</td>
<td>• Determines whether the facade request is routed to a target operation or a target operation group containing target operations.</td>
</tr>
<tr>
<td></td>
<td>• Possible values are Target Operation or Target Operation Group.</td>
</tr>
<tr>
<td></td>
<td>• You must select Target Operation to route the facade request to a target operation.</td>
</tr>
<tr>
<td></td>
<td>• This is a Required field.</td>
</tr>
<tr>
<td>Routing Key</td>
<td>• Specifies the evaluated routing key for the given operation. A routing key must be defined. See How To Derive And Configure Routing Key on page 258 for details.</td>
</tr>
<tr>
<td></td>
<td>• Default value is default.</td>
</tr>
<tr>
<td></td>
<td>• This is a Required field.</td>
</tr>
<tr>
<td>Target Operation Version</td>
<td>Specifies the version number of the target operation.</td>
</tr>
<tr>
<td>Target Operation Name</td>
<td>• Specifies the name of the target operation. The target operation name must be defined in the Target Operations tab of the gateway user interface.</td>
</tr>
<tr>
<td></td>
<td>• This field is a drop down list box, so you can select an pre-defined target operation.</td>
</tr>
<tr>
<td></td>
<td>• This is a Required field.</td>
</tr>
</tbody>
</table>

4. Save changes to your configuration.

Routing Configuration For A Target Operation Group

To configure the routing key for a target operation group, perform the following steps:
1. Ensure that you are on the ROUTING tab of the gateway user interface. See Go To ROUTING Tab on page 259.
2. Click Routing tab on the top menu.
3. Enter the following parameters, as defined below

Table 50  Routing Configuration For A Target Operation Group

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operation Name</td>
<td>• Specifies the name of the facade operation. The operation name must be defined in the Facade Operations tab of the user interface.</td>
</tr>
<tr>
<td></td>
<td>• This field is a drop down list box, so you can select an pre-defined operation.</td>
</tr>
<tr>
<td></td>
<td>• This is a Required field.</td>
</tr>
<tr>
<td>Routing Type</td>
<td>• Determines whether the facade operation request is routed to a target operation or a target operation group containing target operations.</td>
</tr>
<tr>
<td></td>
<td>• Possible values are Target Operation or Target Operation Group.</td>
</tr>
<tr>
<td></td>
<td>• You must select Target Operation Group to route the operation request to a target operation group. The target operation group is a group of target operations.</td>
</tr>
<tr>
<td></td>
<td>• This is a Required field.</td>
</tr>
<tr>
<td>Routing Key</td>
<td>• Specifies the evaluated routing key for the given operation. A routing key must be defined. See How To Derive And Configure Routing Key on page 258 for details.</td>
</tr>
<tr>
<td></td>
<td>• Default value is default.</td>
</tr>
<tr>
<td></td>
<td>• This is a Required field.</td>
</tr>
<tr>
<td>Target Operation Group</td>
<td>• Specifies the name of the target operation group for load balancing functionality.</td>
</tr>
<tr>
<td></td>
<td>• This field is a drop down list box, so you can select an pre-defined target operation group. The target operation group must be defined in the Target Operation Groups tab of the gateway user interface. See Target Operation Group on page 266 for more details.</td>
</tr>
<tr>
<td></td>
<td>• This is a Required field.</td>
</tr>
</tbody>
</table>

4. Save changes to your configuration.
Routing Use Case

This section describes the GetLocation sample shipped with the API Exchange Gateway software to illustrate the configuration steps required for routing. Refer to the GetLocation example in ASG_HOME/examples directory.

Sample Name  GetLocation

Description  GetLocation example illustrates how to use a routing key to route the facade request to a different target operation or target operation group based on the input phone number specified in the address element of the request. The routing key is derived from the telephone number specified in the address element. You can route the client request to a different target operation or target operation group based on the telephone number in the request. The target operation or target operation group must be configured in the gateway user interface.

For example,

- If the value of the address element is specified as "tel:+498948956000", the opCoId is derived using the substring function (substring($address,6,2)) from the address element as 49. The routing key is populated based on the opCoId as 49. See Configuration on page 263 for routing configuration.

Sample Location  ASG_CONFIG_HOME/GetLocation

Configuration  Perform the following steps for routing configuration:

Task A  Define A Transformation File

Refer to the sample file located as Sample XSLT File, page 263 to define a transformation file. This XSLT file shows that the routing key is derived based on the value of address element of the request message. The address element contains a telephone number. You can edit the file, as needed.

Sample XSLT File Location  ASG_CONFIG_HOME/GetLocation/xslt/operations/parse_getLocation.xsl

Sample XSLT File

<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="2.0"
Task B  Upload Transformation File

To upload the `parse_getLocation.xsl` transformation file, do following:

1. Start the GUI server, if not already running. See Starting GUI, page 166.
2. Login to the gateway user interface with your credentials.
3. Select GetLocation under Projects.
4. Click on **ROUTING** tab on right hand side.
5. Click on **Facade Operations** tab.
6. Expand the `getLocationBW` operation.
7. Make sure that `operations/parse_getLocation.xsl` file is populated in the **ProcessBody Transform** field. If it is not selected, select it from the drop down list box.
8. Save any changes to the configuration.

**Task C  Routing Configuration**

You must configure a routing key for `getLocationBW` operation to route the request to the `http.getLocation` target operation as below:

1. On ROUTING tab, click **Routing** tab on the top menu.
2. Click **Add Property** to add a new routing configuration.
3. Enter the fields as below:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operation Name</td>
<td>Select <code>getLocationBW</code> from the drop down list box.</td>
</tr>
<tr>
<td>Routing Type</td>
<td>Select <strong>Target Operation</strong> from the drop down list box.</td>
</tr>
<tr>
<td>Routing Key</td>
<td>Enter the Routing Key as 49.</td>
</tr>
<tr>
<td>Target Operation</td>
<td>Select the target operation as <code>http.getLocation</code> from the drop down list box. The <code>http.getLocation</code> target operation is defined in the <strong>Target Operations</strong> tab.</td>
</tr>
</tbody>
</table>

4. Save any changes to the configuration.

Routing configuration for GetLocation example demonstrates that for the `getLocationBW` operation, if the routing key is populated as 49 from the data content of the incoming request message, the request is routed to the `http.getLocation` target operation. Similarly, you can define additional routing configuration to route the request to a different target operation for a different routing key.
**Target Operation Group**

API Exchange Gateway provides the load balancing functionality to distribute the client requests across the target operations grouped within a target operation group. The target operations can be grouped together to achieve maximum productivity and distribute the load of requests from clients. API Exchange Gateway supports the failover for a target operation group which allows the request to be processed by an alternate target operation within the target operation group if the primary target operation is not running or available.

**Overview**

This section gives an overview of the target operation groups followed by the details on the supported routing algorithms for a target operation group. The routing algorithm type determines the routing of the request to an appropriate target operation grouped within a target operation group.

API Exchange Gateway allows you to group the target operations in a target operation group. You can add multiple number of target operations in a target operation group. You can also create multiple target operation groups as per your needs. When any client request is routed to a target operation group containing multiple target operations, the gateway core engine forwards the request to a target operation based on the type of the target operation group. The type of the target operation group indicates a routing algorithm to be used to select a target operation for processing a client request.

See [Types of Target Operations Group on page 266](#) for supported types of target operation groups.

**Types of Target Operations Group**

API Exchange Gateway supports following types of target operation group to provide the load balancing of the requests processing within target operations in a group.

- **LoadBalanced**: When the type of target operation group is defined as **LoadBalanced**, the gateway engine picks up a random target operation from the list of target operations within this target operation group to process the facade request from the client.

  See [LoadBalanced on page 268](#) for details.
• **RoundRobin**: When the type of target operation group is defined as RoundRobin, the load balancing of target operations within the target operation group can be classified as following sub-types:
  
  — RoundRobin
  
  — RoundRobin with Failover

  See *RoundRobin* on page 268 for details.

• **Weighted RoundRobin**: When the type of target operation group is defined as Weighted RoundRobin, that means that you can assign a weight value to each target operation in the target operation group. With Weighted RoundRobin routing type, the load balancing of target operations within the target operation group can be classified as following sub-types:

  — Weighted RoundRobin
  
  — Weighted RoundRobin with Failover

  See *Weighted RoundRobin* on page 271 for details.

• **Sticky Resource Affinity**: When the type of target operation group is defined as Sticky Resource Affinity, the routing of the request can be done based on various parameters such as IP address of client machine, XPath, machine domain name.

  See *Sticky Resource Affinity* on page 275 for details.
Routing Algorithms For Target Operation Group

This section explains the various types of supported routing algorithms for a target operation group. A target operation group type is defined based on the routing algorithm.

Example

Let us consider a target operation group SG1 containing three target operations as A, B, C. The gateway core engine routes the incoming facade operation requests (for example, Request 1...n) to SG1 target operation group based on the routing key. The gateway core engine selects the target operation within SG1 target operation group based on the type of the target operation group which are defined as below:

LoadBalanced

If the target operation group type for the SG1 target operation group is defined as LoadBalanced, the gateway core engine uses the LoadBalanced routing algorithm. For LoadBalanced routing algorithm type, the gateway core engine picks up a random target operation which is a part of this target operation group. The API Exchange Gateway core engine distributes the requests evenly over a large number of requests.

When API Exchange Gateway core engine routes the client requests (Request 1...n) to SG1 target operation group with LoadBalanced target operation group type, the target operation is selected as follows to process the client requests:

- The first facade request Request 1 is forwarded to target operation B.
- The second facade request Request 2 is forwarded to target operation C.
- The third facade request Request 3 is forwarded to target operation A.
- The fourth request Request 4 is forwarded to target operation A.
- The subsequent facade requests are forwarded to target operations in a random fashion.

RoundRobin

If the target operation group type for the SG1 target operation group is defined as RoundRobin, the gateway core engine uses the RoundRobin routing algorithm. For RoundRobin routing algorithm type, the gateway core engine selects the target operation in a roundrobin fashion. This means that the gateway core engine rotates through the list of target operations one at a time to process the client requests.
Figure **RoundRobin Routing** on page 270 illustrates the SG1 target operation group with RoundRobin type.

When API Exchange Gateway core engine routes the client requests to SG1 target operation group, the target operation from the SG1 target operation group is selected as follows to process the client request:

- The first facade request Request 1 is forwarded to target operation A.
- The second facade request Request 2 is forwarded to target operation B.
- The third facade request Request 3 is forwarded to target operation C.
- The fourth request Request 4 is forwarded to target operation A. The subsequent facade requests are forwarded to target operation B, then target operation C and so on.

**Scheduling Pattern**

The gateway core engine uses the scheduling pattern as follows for RoundRobin algorithm type:

(Target Operation A, Target Operation B, Target Operation C, Target Operation A, Target Operation B, Target Operation C ,Target Operation A......)

If the selected target operation is not running or times out to process the facade request, then the gateway core engine processes the request as follows:

- Retries to route the request to the same target operation as per the retry mechanism.
- The gateway core engine does not route the facade request to an alternate target operation in the target operation group.
- Returns the fault message back to the client for the facade request.
Figure 10  RoundRobin Routing
Weighted RoundRobin

For the Weighted RoundRobin routing algorithm, a weight value is assigned to each target operation. The weight value for a target operation specifies a priority for each target operation in a target operation group. You can assign a weight value to each target operation in a target operation group using TIBCO API Exchange Gateway user interface.

When you assign the weight to a target operation, the weight value indicates the capacity of that target operation in comparison to other target operations within the target operation group.

You can assign same weight values to multiple target operations. When the target operations in a target operation group have the same weight values, then a target operation is selected in a roundrobin way to process the facade request.

When a client request is forwarded to a target operation group with type as Weighted RoundRobin, the API Exchange Gateway engine selects the target operation within the target operations group based on the weight values assigned to each target operation.

Figure Weighted RoundRobin on page 273 shows the SG1 target operation group containing three target operations (target operation A, target operation B, target operation C) with Weighted RoundRobin type. Target operation A is assigned weight value as 5, Target operation B is assigned weight value as 2, Target operation C is assigned weight value as 3.

When API Exchange Gateway core engine routes any facade request to SG1 target operation group, the gateway core engine processes the request in the following way:

- The first facade request Request 1 is forwarded to target operation A as that has the maximum weight as 5.
- The second facade request Request 2 is forwarded to target operation A.
- The third facade request Request 3 is forwarded to target operation A.
- The fourth facade request Request 4 is forwarded to target operation C.
- The fifth facade request Request 5 is forwarded to target operation A.
- The sixth facade request Request 6 is forwarded to target operation B.

Scheduling Pattern

The gateway core engine uses the scheduling pattern as follows for Weighted RoundRobin algorithm type:
(Target Operation A, Target Operation A, Target Operation A, Target Operation C, Target Operation A, Target Operation B, Target Operation C and so on.)
If the selected target operation is not running or times out to process the facade request, then the gateway core engine process the request as follows:

- Retries to forward the request to the same target operation as per the retry mechanism.
- The gateway core engine does not route the facade request to an alternate target operation in the target operations group.
- Returns the fault message back to the client for the facade request.

When the target operations in a target operation group have same weight values, the gateway core engine selects the target operations in a round robin fashion.
Figure 11  Weighted RoundRobin
RoundRobin With Failover

For a target operation group of RoundRobin With Failover type, the gateway core engine selects the target operation as per the RoundRobin algorithm. See RoundRobin on page 268 for details. After the target operation is selected, the gateway core engine checks the health of the target operation using the healthcheck reference functionality of the API Exchange Gateway. See HealthCheck For Reference on page 289. The API Exchange Gateway core engine processes the client request as follows:

- API Exchange Gateway core engine selects the target operation in a RoundRobin fashion.
- If the selected target operation within the target operation group is running and available, the facade request is forwarded to this target operation for processing.
- If the target operation is not running or times out, an alternate target operation is selected from the group of target operations using roundrobin algorithm. This process continues until a target operation is found within the target operations group which is available to process the request. The facade request is routed, then, to the available target operation for processing.

Weighted RoundRobin With Failover

For a target operation group of Weighted RoundRobin With Failover type, the gateway core engine selects the target operation as per the assigned weight to the target operations. See Weighted RoundRobin on page 271 for details. After the target operation is selected, the gateway core engine checks the health of the target operation using the healthcheck functionality. See HealthCheck For Reference on page 289 for details. If the target operation is available, the facade request is forwarded to this target operation. If the target operation is not available, the gateway core engine finds an alternate target operation based on the weight assigned to the remaining target operations in the target operations group. This process continues until a target operation is found within the target operations group which is running. The facade request is routed to the available target operation for processing.
Sticky Resource Affinity

When a target operation group type is configured as Sticky Resource Affinity, then the gateway core engine distributes the client requests between the target operations in the target operations group based on information such as the sticky key. When the gateway core engine routes a client request to a target operation group of Sticky Resource Affinity type, the target operation is selected based on the value of a sticky key.

- If the sticky key to the target operation map already exists, the client request is routed to the same target operation.
- If the sticky key to the target operation map does not exist, a new map is created using the sticky key and the next available target operation in the target operations group. The gateway core engine balances the load of the target operations in the target operation group when it selects the available target operation from the group.

You must define a sticky key to route the facade operation request to a specific target operation. See Defining And Configuring Sticky Key on page 276 for details.

For the SG1 target operation group with Sticky Resource Affinity type, consider the following configuration:

- Sticky Routing Key SK1 is mapped to target operation A. Target operation A has processed 10 requests.
- Sticky Routing Key SK2 is mapped to target operation B. Target operation B has processed 7 requests.
- Sticky Routing Key SK3 is mapped to target operation C. Target operation C has processed 4 requests.

For this scenario, the gateway core engine processes the client requests as below:

- Any incoming client request with SK1 sticky routing key is forwarded to target operation A.
- Any incoming client request with SK2 sticky routing key is forwarded to target operation B.
- Any incoming client request with SK3 sticky routing key is forwarded to target operation C.
- A client request with SK4 sticky routing key is forwarded to target operation C. For this client request, the gateway core engine does not associate the SK4 sticky routing key with the request, chooses target operation A as next target operation in round robin way. Since target operation A has processed 10 requests, so the gateway core engine forwards this request to target operation C to balance the load between A, B and C target operations. Any further client
requests with SK4 sticky routing key is forwarded to target operation C for processing.

Figure 12  StickyResourceAffinity Routing Algorithm

Defining And Configuring Sticky Key
To use the Sticky Resource Affinity routing algorithm, you must define a sticky routing key. The sticky routing key is derived in a XSLT transformation file as part of the parsing step in the request processing pipeline of a facade operation. You can upload the XSLT file for a facade operation configuration in the gateway user interface. The gateway core engine retrieves the sticky key from the transformation. See StickyResourceAffinity Target Operation Group Configuration on page 284.
Target Operation Group Configuration

This section explains the configuration setup for target operation group to select a routing algorithm. The routing algorithm determines the target operation within the target operation group to process the facade request received by this target operation group.

Configure A Target Operation Group

To configure a target operation group, perform the following steps:

1. Ensure that you are on the Routing tab of the gateway user interface. See Go To ROUTING Tab on page 259.

2. Click on Target Operation Groups tab on the top menu.

3. Enter the value for following fields:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Group Name</td>
<td>Specifies the name of the target operation group. This can be any name defined by a user.</td>
</tr>
<tr>
<td></td>
<td>This is a Required field.</td>
</tr>
<tr>
<td>Description</td>
<td>A Short user description of the target operation group.</td>
</tr>
<tr>
<td>Type</td>
<td>Specifies the type of target operation group. See Routing Algorithms For Target Operation Group, page 268 for details.</td>
</tr>
<tr>
<td></td>
<td>Select one of the possible values from the drop down list box:</td>
</tr>
<tr>
<td></td>
<td>• LoadBalanced</td>
</tr>
<tr>
<td></td>
<td>• RoundRobin. See Configure A RoundRobin Target Operation Group on page 279.</td>
</tr>
<tr>
<td></td>
<td>• WeightedRoundRobin</td>
</tr>
<tr>
<td></td>
<td>• RoundRobinWithFailover</td>
</tr>
<tr>
<td></td>
<td>• WeightedRoundRobinWithFailover</td>
</tr>
<tr>
<td></td>
<td>• StickyResourceAffinity</td>
</tr>
</tbody>
</table>
4. Save changes to the configuration.

If you configure a target operation group of type RoundRobinWithFailover or WeightedRoundRobinWithFailover, you must define the health check configuration for each target operation in the group. See HealthCheck Configuration For Target Operation, page 292 for configuration details.

Configure A RoundRobin Target Operation Group

To configure a target operation group of RoundRobin type, you must select the type of the target operation group as RoundRobin. Do the following steps:

1. Click on Target Operation Groups tab.
2. Enter the value for following fields:

   3. Save changes to the configuration.

   4. Figure RoundRobin Target Operation Group Configuration on page 280 shows the RoundRobin Target Operation Group configuration.
Configure A WeightedRoundRobin Target Operation Group

To configure a target operation group of WeightedRoundRobin type, you must select the type of the target operation group as WeightedRoundRobin. Do the following steps:

1. Click on Target Operation Groups tab.
2. Enter the value for following fields:

Table 54  WeightedRoundRobin Target Operation Group Configuration

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Group Name</td>
<td>• Specifies the name of the target operation group. This can be any name</td>
</tr>
<tr>
<td></td>
<td>defined by a user.</td>
</tr>
<tr>
<td></td>
<td>• This is a Required field.</td>
</tr>
<tr>
<td>Description</td>
<td>Short user description of the target operation group.</td>
</tr>
</tbody>
</table>
3. Save changes to the configuration.

4. Figure WeightedRoundRobin Target Operation Group Configuration on page 281 shows the RoundRobin Target Operation Group configuration.

**Figure 14  WeightedRoundRobin Target Operation Group Configuration**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Select <strong>WeightedRoundRobin</strong> from the drop down list box. See</td>
</tr>
<tr>
<td></td>
<td>WeightedRoundRobin Target Operation Group Configuration on page 281.</td>
</tr>
<tr>
<td>Target Operations</td>
<td>Specifies the list of target operations in the group. You can add multiple target operations using the + icon to a target operations group. The target operations must be configured under the Target Operations group.</td>
</tr>
</tbody>
</table>

**Configure A RoundRobinWithFailOver Target Operation Group**

To configure a target operation group of RoundRobinWithFailover type, you must select the type of the target operation group as RoundRobinWithFailover. Do the following steps:

To configure a target operation group of RoundRobinWithFailover type, you must select the type of the target operation group as RoundRobinWithFailover. Do the following steps:

1. Click on **Target Operation Groups** tab.
2. Enter the value for following fields:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| Group Name      | Specifies the name of the target operation group. This can be any name defined by a user.  
                    | • This is a Required field.                                                 |
| Description     | Short user description of the target operation group.                       |
| Type            | Select RoundRobinWithFailover from the drop down list box. See RoundRobinWithFailOver Target Operation Group on page 283. |
| Target Operations | Specifies the list of target operations in the group. You can add multiple target operations using the + icon to a target operations group. The target operations must be configured under the Target Operations group. |

3. Save changes to the configuration.

4. Figure RoundRobinWithFailOver Target Operation Group on page 283 shows the RoundRobinWithFailover Target Operation Group configuration:
Configure A WeightedRoundRobinWithFailOver Target Operation Group

To configure a target operation group of WeightedRoundRobinWithFailover type, you must select the type of the target operation group as WeightedRoundRobinWithFailover. Do the following steps:

1. Click on Target Operation Groups tab.
2. Enter the value for following fields:

Table 56 WeightedRoundRobinWithFailOver Target Operation Group Configuration

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| Group Name      | • Specifies the name of the target operation group. This can be any name defined by a user.  
|                 | • This is a Required field.                                                |
| Description     | Short user description of the target operation group.                       |
3. Save changes to the configuration.

4. Figure WeightedRoundRobinWithFailOver Target Operation Group on page 284 shows the RoundRobinWithFailover Target Operation Group configuration:

**Figure 16** WeightedRoundRobinWithFailOver Target Operation Group

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Select WeightedRoundRobinWithFailover from the drop down list box. See WeightedRoundRobinWithFailOver Target Operation Group on page 284.</td>
</tr>
<tr>
<td>Target Operations</td>
<td>Specifies the list of target operations in the group. You can add multiple target operations using the + icon to a target operations group. The target operations must be configured under the <strong>Target Operations</strong> group.</td>
</tr>
</tbody>
</table>

---

**StickyResourceAffinity Target Operation Group Configuration**

This section explains the steps required to configure the target operation group of **StickyResourceAffinity** type:
Task A Define Sticky Routing Key

The sticky routing key can be defined based on many parameters such as IP address, machine domain name etc. The StickyKey is populated from the <stickyRoutingKey> element tag in the transformation (XSLT) file.

Refer to Example XSLT File For StickyRoutingKey on page 285 to define a XSLT file for sticky routing key.

Example XSLT File For StickyRoutingKey

The following example illustrates a XSLT file to derive a StickyRouting Key. Refer to the <stickyRoutingKey> element tag in the file as defined below:

```xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="2.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:c="http://www.tibco.com/schemas/asg/context"
    xmlns:h="http://www.tibco.com/asg/protocols/http"
    xmlns:f="http://www.tibco.com/asg/functions/form"
    xmlns:form="http://www.tibco.com/asg/content-types/form"
    xmlns:book="http://www.example.com/xsd/books"
    xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
    <xsl:output
        omit-xml-declaration="yes"
        indent="yes" />

    <xsl:variable name="contextHref">
        <xsl:value-of select="/transformation/context/@href"/>
    </xsl:variable>

    <xsl:variable name="httpRequest">
        <xsl:copy-of
            select="document($contextHref)/c:context/c:entry[@key='asg:httpRequest']/h:request/*"/>
    </xsl:variable>

    <xsl:variable name="parsedQueryString">
        <xsl:value-of select="$httpRequest/h:client-ip"/>
    </xsl:variable>

    <xsl:variable name="nbRequestHref">
        <xsl:value-of select="/transformation/nbRequest/@href"/>
    </xsl:variable>

    <xsl:variable name="nbRequest">
        <xsl:copy-of
            select="document($nbRequestHref)/soapenv:Envelope/soapenv:Body/*"/>
    </xsl:variable>
</xsl:stylesheet>
```
<xsl:template match="/">
  <output>
    <xsl:variable name="username">
      <xsl:value-of select="$nbRequest/book:Author"/>
    </xsl:variable>
    <username><xsl:value-of select="$username"/></username>
    <parsedQueryString><xsl:value-of select="$parsedQueryString"/></parsedQueryString>
    <routingKey>
      <xsl:choose>
        <xsl:when test="$username = 'Vivek Ranadive'">Vivek Ranadive</xsl:when>
        <xsl:when test="$parsedQueryString != ''">default</xsl:when>
        <xsl:otherwise>default</xsl:otherwise>
      </xsl:choose>
    </routingKey>
    <stickyRoutingKey>
      <xsl:choose>
        <xsl:when test="$parsedQueryString != ''">default</xsl:when>
        <xsl:otherwise>default</xsl:otherwise>
      </xsl:choose>
    </stickyRoutingKey>
  </output>
</xsl:template>
</xsl:stylesheet>

**Task B  Upload Sticky Routing Key File**

To upload the XSLT file containing the <stickyRoutingKey> element, perform the following steps:

1. Start the GUI server, if not already running.
2. Login to the gateway user interface with your credentials.
3. Add a new project or select an existing project under **Projects**.
4. Click on **ROUTING** tab on the right hand side.
5. Click on **Facade Operations** tab on the top menu.
6. Add a new operation or choose an existing operation.
7. Upload the XSLT file as below:
   a. Click Choose File button in the New ProcessBody Transform field to select the XSLT transformation file containing the stickyRoutingKey.
   b. Click Open on the new dialog to upload the file.
   c. If the XSLT file is located in the \(\text{ASG\_CONFIG\_HOME/ASGConfigName/xslt/operations}\) directory, then you can choose the XSLT file in the ProcessBody Transform field.

8. Save changes to the configuration.

**Task C Configure StickyResourceAffinity Type Target Operation Group**

To configure a target operation group of StickyResourceAffinity type, you must select the type of the target operation group as StickyResourceAffinity. Do the following steps:

1. Click on Target Operation Groups tab.
2. Enter the value for following fields:

   3. Save changes to the configuration.
   4. Figure StickyResourceAffinity Target Operation Group Configuration on page 287 shows the StickyResourceAffinity Target Operation Group configuration:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Group Name</td>
<td>Specifies the name of the target operation group.</td>
</tr>
<tr>
<td></td>
<td>This is a Required field.</td>
</tr>
<tr>
<td>Description</td>
<td>Short user description of the target operation group.</td>
</tr>
<tr>
<td>Type</td>
<td>Select StickyResourceAffinity from the drop down list box. See StickyResourceAffinity Target Operation Group Configuration on page 287 for details.</td>
</tr>
<tr>
<td>Target Operations</td>
<td>Specifies the list of target operations in the group. You can add multiple target operations using the + icon to a target operations group. The target operations must be configured under the Target Operations group.</td>
</tr>
</tbody>
</table>

3. Save changes to the configuration.

4. Figure StickyResourceAffinity Target Operation Group Configuration on page 287 shows the StickyResourceAffinity Target Operation Group configuration:
<table>
<thead>
<tr>
<th>Facade Operations</th>
<th>Target Operations</th>
<th>Target Operation Groups</th>
<th>Routing</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SG1</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Group Name**: SG1
- **Description**: ServiceGroup1
- **Type**: StickyResourceAffinity
- **Target Operations**:
  - http.GetBooksByAuthor
HealthCheck For Reference

This section explains the healthcheck for reference functionality provided by API Exchange Gateway software.

Overview

API Exchange Gateway provides Health Check For Reference functionality to monitor the availability of the target operations in a target operations group.

The Load Balancing router component of the API Exchange Gateway uses the Health Check For Reference functionality when it routes an operation request to a target operation group of type as below:

- RoundRobinWithFailover
- WeightedRoundRobinWithFailover

When an facade operation request is routed to a target operation group of RoundRobinWithFailover or WeightedRoundRobinWithFailover type, the load balancer router of the gateway uses healthcheck to check the health of the target operations configured in the target operations group. If the selected target operation is not available or running, the gateway forwards the request to an alternate target operation which is running.

The Healthcheck allows the router component of the gateway to intelligently pick the available target operation to increase the throughput and reduce the latency of the request processing at runtime.

The healthcheck functionality is available to the gateway load balancer router component only to monitor the health of target operations in a target operations group with failover routing algorithms (RoundRobinWithFailover or WeightedRoundRobinWithFailover).

HealthCheck Modes For A Target Operation

Health Check for Reference provides the following modes for any target operation in a target operations group:

- Timer

For each target operation in the target operation group, you can configure a timer based healthcheck. Healthcheck for reference checks the status of the target operation each time the timer expires. You must configure a health check method for the timer mode. See HealthCheck Methods For Timer Mode on page 290 for details.
Reset

You can configure reset mode healthcheck for each target operation in the target operation group. When Reset mode is used, the healthcheck for reference resets the health status of a target operation after the specified interval (in milliseconds). See Reset Interval on page 290.

Reset Interval

The healthcheck module resets the health status of a target operation after the time interval as specified by the ResetInterval configuration parameter. This parameter is used when the Reset mode for healthcheck configuration is selected for a target operation in the target operation group.

HealthCheck Methods For Timer Mode

You must choose a method for timer mode of healthcheck. HealthCheck For reference uses the specified method to check the health status of the target operation. Following types of healthcheck methods are supported:

- HTTP

  API Exchange Gateway core engine forms a http based URL from the fields configured for the http based target operation such as service-type, URI, host, port. Once the URL is formed, API Exchange Gateway core engine invokes a http connection to ping the target operation. Based on the response code returned from the http connection, healthcheck determines the health of the target operation. For example, if the returned response code indicates a success, then the healthcheck returns that the target operation is up and running. If the returned response code indicates a failure, the healthcheck returns that the target operation is not running. The response code returned is checked to determine the health of the target operation.

- API Exchange Gateway core engine forms a based URL from the fields configured for the based target operation such as service-type, URI, host, port and the ssl properties set in dss.properties. Once the URL is formed, API Exchange Gateway core engine invokes a URL connection to ping the target operation. Based on the response code returned from the http connection, healthcheck determines the health of the target operation. For example, if the returned response code indicates a success, then the healthcheck returns that the target operation is up and running. If the returned response code indicates a failure, the healthcheck returns that the target operation is not running.

For the based target operation, you must configure the ssl properties for mutual SSL authentication. If the based target operation is enabled for one way authentication, ssl properties configuration is not required.
• HealthCheckURL

HealthCheckURL method allow you to specify the URL to invoke a http connection. API Exchange Gateway core engine attempts to invoke the http connection with the configured URL for this method. API Exchange Gateway core engine uses the returned response code to determine the health of the target operation. For example, if the returned response code is a success, the healthcheck returns that the target operation is up and running. If the returned response code is a failure, the healthcheck returns that the target operation is not running.

• TCPEcho

TCPEcho method expects that the TCP server is running at the specified host and port configured for this method. TCPEcho method sends a ping request to the running TCP server. The healthcheck uses the returned echo response to determine if the target operation is running or not.

You must configure following parameters for the TCPEcho method:

— TCP Host
— TCP Port

• ContentVerification

ContentVerification method uses the configured Health Check URL and sends a http GET request to the URL. The healthcheck module examines the returned response content and searches for the configured Keyword in the response content. If the Keyword is found in the response content returned from the http GET request, the healthcheck returns that the target operation is up and running. If the Keyword is not found in the response content returned from the http GET request, the healthcheck returns that the target operation is not running.

You must configure following parameters for the TCPEcho method:

— Health Check URL
— Keyword

• SampleRequest

SampleRequest method reads the data from a file as specified by the Content File parameter and sends the data content of this file in the HTTP POST request to the URL specified by the Health Check URL parameter to check the health of the target operation. The healthcheck module examines the returned response code to determine if the target operation is running or not.
You must configure following parameters for the TCPEcho method:

- Health Check URL
- Content File

### HealthCheck Configuration For Target Operation

This section explains the configuration setup to use the healthcheck functionality for the target operations in a target operation group. You must configure the health check parameters for each target operation in a target operation group for `RoundRobinWithFailover` or `WeightedRoundRobinWithFailover` group type.

### Configuration for Reset Mode Of HealthCheck

When you enable the Reset mode of healthcheck for each target operation in a target operation group of `RoundRobinWithFailover` or `WeightedRoundRobinWithFailover`, you must configure the `ResetInterval` parameter.

To configure the `ResetInterval` parameter for a target operation, perform the following steps:

1. Start the GUI server, if not already running.
2. Login to the user interface with your credentials.
3. Add a new project or select an existing project under **Projects**, as applicable.
4. Click on **Target Operation Groups** tab.
5. Select the target operation.
6. Enter the `ResetInterval` parameter. See *Reset Interval* on page 290.

### Configuration for Timer Based HealthCheck

When you enable the timer based health check for each target operation in a target operation group of `RoundRobinWithFailover` or `WeightedRoundRobinWithFailover`, you must configure the healthcheck parameters.

To configure the healthcheck parameters for a target operation, perform the following steps:

1. Start the GUI server, if not already running.
2. Login to the gateway user interface with your credentials.
3. Add a new project or select an existing project under Projects, as applicable.
4. Click on Target Operation Groups tab.
5. Select a Target Operation of RoundRobinWithFailover or WeightedRoundRobinWithFailover type.
6. Expand the Health Check node to enter the healthcheck configuration parameters, as described below:

Table 58  HealthCheck Configuration Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HealthCheckMode</td>
<td>• Specifies the mode for the healthcheck functionality.</td>
</tr>
<tr>
<td></td>
<td>• Possible values are:</td>
</tr>
<tr>
<td></td>
<td>— Timer</td>
</tr>
<tr>
<td></td>
<td>— Reset</td>
</tr>
<tr>
<td></td>
<td>• You must select the Timer from the drop down list box.</td>
</tr>
<tr>
<td>HealthCheckMethod</td>
<td>• Specifies the method to be used by healthcheck to check the availability of the target operation. Possible values are:</td>
</tr>
<tr>
<td></td>
<td>— HTTP</td>
</tr>
<tr>
<td></td>
<td>— HealthCheckURL</td>
</tr>
<tr>
<td></td>
<td>— TCPEcho</td>
</tr>
<tr>
<td></td>
<td>— ContentVerification</td>
</tr>
<tr>
<td></td>
<td>— SampleRequest</td>
</tr>
<tr>
<td></td>
<td>• This is a Required field.</td>
</tr>
<tr>
<td>CheckInterval</td>
<td>Specifies a time interval (in milliseconds) which is used by healthcheck as an expiration time to check the health status of the target operation. For example, if this value is specified as 10000, then the healthcheck checks the health status of the target operation after every 10 seconds whether the target operation is running or not. The health status of the target operation is used by routing functionality to determine whether to route a request to this target operation or not.</td>
</tr>
</tbody>
</table>

7. Save changes to your configuration.
Configuration for HTTP HealthCheck Method

If you want to use the healthcheck method as HTTP, do following:

- Configure the parameters as explained in HealthCheck Configuration Parameters on page 293 table.
- Ensure that HTTP is selected in the HealthCheckMethod field from the drop down list box.

Configuration for HealthCheck Method

If you want to use the healthcheck method as, do following:

- Configure the parameters as explained in HealthCheck Configuration Parameters on page 293 table.
- Ensure that is selected in the HealthCheckMethod field from the drop down list box.

Configuration for HealthCheckURL HealthCheck Method

If you want to use the healthcheck method as HealthCheckURL, do following:

- Configure the parameters as explained in HealthCheck Configuration Parameters on page 293 table.
- Ensure that HealthCheckURL is selected in the HealthCheckMethod field from the drop down list box.
- Enter the following parameter:
  - HealthCheckURL: Specifies the URL to invoke a http connection. See HealthCheckURL on page 291.

Configuration for TCPEcho HealthCheck Method

If you want to use the healthcheck method as TCPEcho, do following:

- Configure the parameters as explained in HealthCheck Configuration Parameters on page 293 table.
- Ensure that TCPEcho is selected in the HealthCheckMethod field from the drop down list box.
Enter the parameters as described below:

Table 59  TCPEcho HealthCheckMethod Configuration Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCPEcho</td>
<td>Specifies the host on which the TCP server is running.</td>
</tr>
<tr>
<td>TCPPort</td>
<td>Specifies the port on which the TCP server is running.</td>
</tr>
</tbody>
</table>

**Configuration for ContentVerification HealthCheck Method**

If you want to use the healthcheck method as ContentVerification, do following:

- Configure the parameters as explained in HealthCheck Configuration Parameters on page 293 table.
- Ensure that ContentVerification is selected in the HealthCheckMethod field from the drop down list box.
- Enter the parameters as described below:

Table 60  ContentVerification HealthCheckMethod Configuration Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Health Check URL</td>
<td>Specifies the URL to which the healthcheck module sends the http GET request.</td>
</tr>
<tr>
<td>Keyword</td>
<td>Specifies the keyword to be searched in the content returned in response to the http GET request.</td>
</tr>
</tbody>
</table>

**Configuration for SampleRequest HealthCheck Method**

If you want to use the healthcheck method as SampleRequest, do following:

- Configure the parameters as explained in HealthCheck Configuration Parameters on page 293 table.
- Ensure that SampleRequest is selected in the HealthCheckMethod field from the drop down list box.
- Enter the parameters as described below:

Table 61  SampleRequest HealthCheckMethod Configuration Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Health Check URL</td>
<td>Specifies the URL to which the healthcheck module sends the http POST request.</td>
</tr>
</tbody>
</table>
Table 61  SampleRequest HealthCheckMethod Configuration Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Content File</td>
<td>Specifies a file from where the healthcheck module reads the data contents and sends it in http POST request.</td>
</tr>
<tr>
<td>Existing Content File</td>
<td>Specifies an existing content file from the healthcheck directory under ASG_CONFIG_HOME. The healthcheck directory is created when the first file is uploaded. The healthcheck module reads the data contents from this content file and sends it in http POST request.</td>
</tr>
</tbody>
</table>
Chapter 9  Throttles

This chapter explains the throttle functionality of TIBCO API Exchange Gateway product.

Topics

- Overview, page 298
- Throttle Types, page 299
- Throttle Chaining, page 305
- Configuring Throttles, page 306
- Content Based Throttles, page 310
- Payload Size Throttles, page 316
- Payload Size Throttles, page 316
Overview

TIBCO API Exchange Gateway supports throttles to control the flow of requests from the client to the target operations. API Exchange Gateway uses the throttles for following purposes:

- Protect the target operations from over-usage.
- Maintain the limits of the load of requests on a target operation.
- Protect the target operation to be accessed by unauthorized partners repeatedly.
- Enforce service level agreements at a partner level.

Throttles allow you to define the maximum number of requests that are handled by a target operation in a defined time interval. You must define the maximum count and the time interval for a throttle.

API Exchange Gateway uses a throttle policy to determine if a client request is passed on to a target operation or it should be rejected.

Throttles define a condition for a type and metric (entity). API Exchange Gateway checks the condition for an incoming request before processing the request. For example, you can define a condition to allow only 5 client requests within 10 seconds to the target operation for a partner request.

You can define a throttle policy using Monitors. After defining a throttle policy, it can be applied to a metric such as partner, partner group, partner operation and target operation.

Facade Throttles

Facade throttles are designed for the partners, partner groups and partner+operation. API Exchange Gateway core engine applies the facade throttle after it identifies the operation.

The supported throttles types can be applied at the following level:

- Partner
  
  After defining a throttle policy, you can apply it at a partner level. The gateway core engine checks the throttle condition on every request sent by the partner.

- Partner Group
After defining a throttle policy, you can apply it at a partner group level. The gateway core engine checks the throttle condition on every request sent by the partners in the group.

- Partner + Operation
  After defining a throttle policy, you can apply the throttle by the combination of the partner and the operation. The gateway core engine checks the throttle condition on every request sent by the partner for a specific operation.

### Service Throttles

Service throttles are designed for a target operation. API Exchange Gateway core engine applies the service throttle after it identifies the target operation.

The supported throttles types can be applied at the following level:

- Target Operation
  After defining a throttle policy, you can apply it for a target operation. The gateway core engine checks the throttle condition on every request sent to the target operation.

For both facade and service throttles, if an error occurs before the throttle has been identified, then the throttle is not applied.

### Throttle Types

API Exchange Gateway supports following types of throttles:

#### Rate

Rate throttle is a throttle that allows the requests to pass-through until a limit is reached for a time interval.

The rate throttle count is increased based on following:

- Request Count
- Payload Size. See Payload Size Throttles, page 316.
- Monitor metric retrieved from the context of the input request message. See Content Based Throttles, page 310.

The rate throttle is a technical throttle which can be applied to any metric. For example,

- You can define a rate throttle to apply at a partner level. This allows you to limit the number of requests from a partner.
You can define a rate throttle to apply at a target operation level. This allows you to protect the target operations from over usage.

You may define the rate throttles for any metric if you want to measure the number of requests within a small time interval. After each request is processed, the current throttle count is usually incremented by 1 provided the throttle counter is set to Request Count. See Throttle Counter on page 304 for details.

Rate throttle is applied for shorter intervals and is reset once the interval is reached.

For a rate throttle, you must define throttle interval (in seconds) and throttle max limit. The throttle max limit is reset as specified by Throttle UpdateInterval. See Throttle UpdateInterval on page 304.

Example

Example

For example, set the following throttle configuration on the API Exchange Gateway user interface if you allow only 5 maximum requests to be processed by a target operation within 10 seconds. This throttle configuration allows 5 requests every 10 seconds and rejects other requests within 10 seconds.

Interval : 10
Max Limit: 5
Monitor Type: Rate Throttle
Monitor Counter: Request Count

Quota

Quota throttle is quite similar to the Rate throttle, but it uses a much larger count over much longer intervals (such as hours). The quota throttles are increased on the request. For a quota throttle, you must define throttle interval (in hour) and throttle max limit provided the throttle counter is set to Request Count.

Quota throttles are commercial throttles designed to prevent commercial over-use of target operations, for example wholesale usage.

Example

Example

Set the following throttle configuration for a target operation on the TIBCO API Exchange Gateway user interface if you allow only 10 maximum requests to be processed by a target operation within 1 hour. This throttle configuration allows 10 requests to be processed within 1 hour and rejects other requests within 1 hour.

Interval : 1
Max Limit: 10
Type: Quota Throttle
Monitor Counter: Request Count
High Water Mark

High Water Mark throttle is similar to the Rate throttle, but this throttle also decrements the count after the passed-on requests are completed and the response is ready to return to the requestor.

The High Water Mark throttle increments the throttle count on the request and decreases the throttle count once the response for that request is sent. You must define a throttle max count for a high water mark throttle. High Water Mark throttle allows you to process specific number of requests in parallel by a target service.

Example

High water mark throttle can be defined when the requests are sent in parallel by different users to a target operation. The target operation is slow in responding to such requests sent to it in parallel.

For example, if you want to send 5 requests to a target operation A by 5 users concurrently, you must define the throttle configuration for a target operation on the TIBCO API Exchange Gateway user interface as below. TIBCO API Exchange Gateway allows only 5 parallel requests to be served by a target operation A at a time with this throttle configuration.

Max Limit: 5
Type: High Water Mark Throttle

Error

Error Throttles acts as a Rate Throttle in logic, but error throttle counts the number of error responses as opposed to number of requests. The throttle count of an error throttle is increased on the error responses.

For an error throttle, you must define throttle interval (in seconds) and throttle max count. The throttle max count is reset as specified by throttle updateInterval. See Throttle UpdateInterval on page 304.

Example

You can define an error throttle for a target operation when you want to block the requests after certain number of error responses are returned from this target operation.

For example, set the following throttle configuration on the API Exchange Gateway user interface if you allow only 10 error responses from a target operation within 10 seconds. If 10 requests are sent by the client within 10 seconds to a target operation and all the requests are processed with errors. An error
response for each request is sent back to the client and the throttle count is incremented after each request is processed. API Exchange Gateway core engine rejects the 11th request onwards. After 10 seconds, the throttle max limit is reset to 0 so that the client can send the requests to the target operation again.

Interval : 10  
Max Limit: 10  
Throttle Type: Error Throttle

Following diagram shows how the throttles are implemented for a throttle type:

Figure 18  Throttle Types
Monitor Time Modifiers

API Exchange Gateway allows you to design the throttles that work on the time modifiers. You can modify the throttle limits depending on the time of the day by applying the time modifiers. Throttles time modifier specify the time ranges when the time modifier can be active. You can specify the multiple days and time ranges. For example, a throttle can relax or tighten its limit based on the time of day, such as between 9am and 11am.

The time modifiers of a throttle specify when any throttle can be overridden with the new values such as date & time range, days of week and throttle max count. You can specify multiple time modifiers for a single throttle of each type. The time modifier limit applies if the current time meets all the rules specified in the time modifier configuration.

Configuring Time Modifier Throttles

To configure a time modifier throttle, you must edit a throttle configuration and add the day, date and time modifiers. The parameters for throttle time modifiers are explained in table Throttle Time Modifiers on page 303.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Max Count</td>
<td>• Specifies to a maxcount that is active during this time.</td>
</tr>
<tr>
<td></td>
<td>• Refers to a throttle limit to apply if throttle is in range.</td>
</tr>
<tr>
<td></td>
<td>• Required.</td>
</tr>
<tr>
<td>Start Date</td>
<td>• Specifies the start date when the time modifier can be active.</td>
</tr>
<tr>
<td></td>
<td>• Optional.</td>
</tr>
<tr>
<td>End Date</td>
<td>• Specifies the start date when the time modifier can be active.</td>
</tr>
<tr>
<td></td>
<td>• Optional.</td>
</tr>
<tr>
<td>Day of Week</td>
<td>• Specifies the days of the week when the time modifier can be active.</td>
</tr>
<tr>
<td></td>
<td>• Optional.</td>
</tr>
<tr>
<td>Time Range</td>
<td>• Specifies the time ranges when the time modifier can be active. Multiple ranges can be supplied.</td>
</tr>
<tr>
<td></td>
<td>• Optional.</td>
</tr>
</tbody>
</table>
**Throttle Counter**

The throttle counter provides the type of the counter used to increment the throttle count. Possible values for throttle counter are:

- **Request Count**
  
The throttle count is incremented by 1 after each request is processed. You must define Request Count as the throttle type for Rate, Quota, High Water Mark, Error and Content based throttles.

- **Input Payload Size**
  
The throttle count is incremented by the size of request payload message (in bytes) after each request is processed. Input Payload Size throttle counter is used for Payload size throttle type. See [Payload Size Throttles](#) on page 316.

- **Output Payload Size**
  
The throttle count is incremented by the size of response payload message (in bytes) after each request is processed. Output Payload Size throttle counter is used for Payload size throttle type. See [Payload Size Throttles](#) on page 316.

- **Transaction Payload Size**
  
The throttle count is incremented by the size of both request and response payload messages (in bytes) after each request is processed. Transaction Payload Size throttle counter is used for Payload size throttle type. See [Payload Size Throttles](#) on page 316.

**Throttle UpdateInterval**

Throttle UpdateInterval is the time interval (in seconds) which is used by TIBCO API Exchange Gateway core engine to calculate the throttle count allowed until the next UpdateInterval period is lapsed.

The default value for Throttle UpdateInterval is 10 seconds. This means that the gateway core engine refreshes the throttle count allowed for the next 10 seconds using the Interval and Max Limit configured for the throttle Instance in the gateway user interface. For example, if the Interval is 60 seconds and Max Limit is 6, at most 1 request allowed in every 10 seconds.

Throttle UpdateInterval is defined by the following property in the `ASG_CONFIG_HOME/asg.properties` file.

```
tibco.clientVar.ASG/Throttle/UpdateIntervalSec=10
```
Throttle Chaining

API Exchange Gateway allows you to assemble throttles into a throttle chain. You can define multiple throttles for a partner request and service endpoint for each enforcement point.

By defining multiple throttles you can pass a partner request through a chain of throttles to meet the complex throttling needs. For example, following throttle chains can be configured:

- Rate & High-Water-Mark: This throttle chain addresses interface load.
- Quota: This throttle chain addresses interface agreements.

The configuration of the throttles determines whether the request is passed on to be processed by the gateway core engine or whether it is applied to any other throttle. When the request is passed on to another throttles, its called throttle chaining.
Configuring Throttles

API Exchange Gateway user interface allows you to configure all types of throttles.

This section explains how to create a throttle policy definition and assign a throttle policy to a partner, partner group, partner operation, or target operation.

Configuration Parameters for Throttles

Table Throttle Configuration Parameters on page 306 describes the parameters required to configure the throttles.

Table 63 Throttle Configuration Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| Monitor Name   | • Specifies the name of the throttle. The name of the throttle is required to configure the throttle for a service, partner, partner group or partner operation.  
• Required.                       |
| Monitor Type   | • Specifies the type of throttle. Select one of the supported values from the drop down list box:  
  — Rate  
  — Quota  
  — HWM  
  — Error  
• Required.                       |
| Interval       | Specifies a time period for which the maximum count of throttles are allowed. |
| Max Limit      | Specifies the maximum number of throttle count allowed during throttle interval period. |
Create a throttle policy definition

This section explains how to configure a quota monitor T1 with a limit of 5 requests within 24 hour for service throttle metric.

**Prerequisites**

This example assumes that you have completed Configuring an endpoint operation for API Exchange Gateway, page 42.

**Modify the existing Configuration to add throttles**

This section explains to open the existing configuration.

1. **Start the GUI, if not already started.** See Starting GUI, page 166 for details.
2. **Expand Projects node, click the ASG_Get_Start configuration.**

**Define A Quota Throttle**

Add a new throttle as follows:

1. **Click on MONITORING tab.**
2. Click on Monitors tab.
3. Click Add property icon to create a new monitor.
4. Enter the throttles details, as defined below:

Table 64  Quota Throttle Configuration

<table>
<thead>
<tr>
<th>Monitor Name</th>
<th>T1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Monitor Type</td>
<td>Quota</td>
</tr>
<tr>
<td>Interval</td>
<td>24</td>
</tr>
<tr>
<td>Max Limit</td>
<td>5</td>
</tr>
</tbody>
</table>

**Assign a throttle policy to target operation**

To assign a Quota throttle to the target operation, follow the steps:
1. Click on ROUTING tab on the left hand side.
2. Click on Target Operations tab on the top menu.
3. Edit the http.GetBooksByAuthor target operation configuration.
4. Enter the details, as defined below

Table 65  Edit Target Operation Configuration

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operation Name</td>
<td>http.GetBooksByAuthor</td>
</tr>
<tr>
<td>Type</td>
<td>HTTP (select from the drop down list box)</td>
</tr>
<tr>
<td>Monitor(s)</td>
<td>T1 (select from the drop down list box)</td>
</tr>
</tbody>
</table>

**Test the Target Operation**

1. Launch TIBCO Designer and open the project: 
   
   `ASG_HOME/examples/BookQuery/BookQuery`

2. Run the following server processes:
   
   — BooksInterface-service1
3. Run the following client process:
   — QueryByAuthorClient
4. Verify that the process runs successfully without any errors.
5. Send few more requests.
6. Notice that after 5 successful requests, the error appears for Quota Throttle violation.
Content Based Throttles

TIBCO API Exchange Gateway allows you to create custom throttles based on the content information in the facade request.

For any throttle configuration specified in the TIBCO API Exchange Gateway user interface, you can extend a throttle configuration and create a monitor to enforce a custom throttle policy based on the data of the message. You can create a monitor to extract a value based on the data in the request message. This value is used as an increment counter for the throttle count.

You can create custom throttle monitor by defining a metric based on the request content of the message. The metric is defined by following parameters -

- metricName - Specifies the name of the throttle to be used from the gateway user interface configuration.
- metricIncrement - Specifies the increment value for the throttle count. The metric increment value for the throttle (monitor) is defined typically as a XPath formula based on the data content of the request message.

The monitor for a throttle is defined in a transformation (XSLT) file. The XSLT file is configured as a parse XSLT file for an operation. When a request goes through the parsing step of the request operation, the gateway core engine reads all the data from the XSLT file and creates the throttle monitors.

For example, you can define a throttle for a bookorder service which allows you to order the books from a store so that the service is not overloaded with the client requests. A throttle T1_BookOrder is defined in such a way that the throttle T1_BookOrder allows a user to order only 20 books in 10 seconds. In such a case, a throttle T1_BookOrder is configured in the TIBCO API Exchange Gateway user interface as below:

<table>
<thead>
<tr>
<th>Throttle Name</th>
<th>T1_BookOrder</th>
</tr>
</thead>
<tbody>
<tr>
<td>Throttle Type</td>
<td>Rate</td>
</tr>
<tr>
<td>Throttle Interval</td>
<td>10 seconds</td>
</tr>
<tr>
<td>Throttle Max Count</td>
<td>20</td>
</tr>
</tbody>
</table>

For this scenario, the TIBCO API Exchange Gateway core engine allows a maximum of 20 requests within 10 seconds. You can extend the throttle policy by creating a monitor to allow specific number of book orders in a single request.
For example, you can create a custom throttle monitor for this throttle T1_BookOrder to increment the throttle count based on the number of book orders in the request. This allows you to customize the throttle count by providing the increment counter defined as metricIncrement for the monitor. The metricIncrement for this monitor can be populated using a XPath formula based on the total number of book orders in the request message. In this case, you can define the metricIncrement to be count of book orders in the request i.e if the count of book orders in a request is 5, then it increments the throttle count as 5 allowing only 4 similar requests within 10 seconds.

Content based throttles are not applicable for Error and High Water Mark throttle types.

Configuring Content Based Throttles

To configure the content based throttles, you need a XSLT file which typically contains the XPath formula. The XSLT file defines the metric definition to create the custom monitor. API Exchange Gateway core engine evaluates and parses the XSLT file in the parsing step for an operation in the request processing pipeline and creates the monitor based on the metric definition.

To configure the content based throttle monitors, perform the following steps:

**Configure throttle**

You must define the throttle in the TIBCO API Exchange Gateway user interface to create a custom throttle monitor. To configure a throttle, do following:

1. Start the GUI server, if not already running.
2. Login to the user interface with your credentials.
3. Add a new project or select an existing project under **Projects**.
4. Click on **MONITORING** tab.
5. Click on **Monitors** tab.
6. Define a throttle T1_BookOrder.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Monitor Name</td>
<td>• Specifies the name of the throttle. For example, T1_BookOrder.</td>
</tr>
<tr>
<td></td>
<td>• Required.</td>
</tr>
</tbody>
</table>
7. Save changes to your configuration.

**Define XSLT File**

You must define an XSLT transformation file to configure a throttle monitor.

For example, the monitor for a throttle T1 is defined in the XSLT file using a `<monitor>` tag as below:

```xml
<output>
  <xsl:variable name="childnodes">
    <xsl:value-of
    select="$nbRequest/loc:NewOrderReq/count(loc:OrderDtl)"/>
  </xsl:variable>
  <monitor>
    <metricName>T1</metricName>
    <metricIncrement>
      <xsl:value-of
      select="$childnodes"/>
    </metricIncrement>
  </monitor>
</output>
```
In the above example, the `metricIncrement` is assigned based on the count of the `OrderDtl` element in the payload of an incoming request. If you have an input payload with `n` number of `OrderDtl` elements, you can parse the number of `OrderDtl` elements and assign that number to `metricIncrement`. The `metricIncrement` is applied to the throttle configuration.

- The throttle monitor defined in the XSLT file must be configured in the TIBCO API Exchange Gateway user interface under **Monitors** tab.
- You can create multiple monitors in a XSLT file. Each monitor is defined using a separate `<monitor>` tag.
- If the XPath formula used in the `metricIncrement` field returns invalid value, the content throttle increments the throttle count value by 1, which is default.

### Example XML (Payload) Files

This section shows example XML files which shows payloads with 2 Orderdetails and 4 Orderdetails. Refer to **Example XSLT File on page 314** for the XSLT file for these payloads.

- Payload with 2 Orderdetails

```xml
<?xml version = "1.0" encoding = "UTF-8"?>
  <CustID>11111</CustID>
  <EmailID>user01@edusvr</EmailID>
  <OrderDtl>
    <ProductID>1001</ProductID>
    <ProductDesc>Executive Black Leather Chair</ProductDesc>
    <Qty>1</Qty>
    <UnitPrice>159.99</UnitPrice>
  </OrderDtl>
  <OrderDtl>
    <ProductID>3002</ProductID>
    <ProductDesc>Medium point, black ink pens, 50-pk</ProductDesc>
    <Qty>5</Qty>
    <UnitPrice>17.99</UnitPrice>
  </OrderDtl>
  <ShippingDtl>
    <Address>3303 Hillview</Address>
    <City>Palo Alto</City>
    <State>CA</State>
    <Zipcode>94303</Zipcode>
    <Country>USA</Country>
  </ShippingDtl>
</NewOrderReq>
```

- Payload with 4 Orderdetails
<?xml version = "1.0" encoding = "UTF-8"?>
  <CustID>11111</CustID>
  <EmailID>user01@edusvr</EmailID>
  <OrderDtl>
    <ProductID>1001</ProductID>
    <ProductDesc>Executive Black Leather Chair</ProductDesc>
    <Qty>1</Qty>
    <UnitPrice>159.99</UnitPrice>
  </OrderDtl>
  <OrderDtl>
    <ProductID>3002</ProductID>
    <ProductDesc>Medium point, black ink pens,50-pk</ProductDesc>
    <Qty>5</Qty>
    <UnitPrice>17.99</UnitPrice>
  </OrderDtl>
  <OrderDtl>
    <ProductID>3002</ProductID>
    <ProductDesc>Medium point, black ink pens,50-pk</ProductDesc>
    <Qty>5</Qty>
    <UnitPrice>17.99</UnitPrice>
  </OrderDtl>
  <OrderDtl>
    <ProductID>3002</ProductID>
    <ProductDesc>Medium point, black ink pens,50-pk</ProductDesc>
    <Qty>5</Qty>
    <UnitPrice>17.99</UnitPrice>
  </OrderDtl>
  <OrderDtl>
    <ProductID>3002</ProductID>
    <ProductDesc>Medium point, black ink pens,50-pk</ProductDesc>
    <Qty>5</Qty>
    <UnitPrice>17.99</UnitPrice>
  </OrderDtl>
  <ShippingDtl>
    <Address>3303 Hillview</Address>
    <City>Palo Alto</City>
    <State>CA</State>
    <Zipcode>94303</Zipcode>
    <Country>USA</Country>
  </ShippingDtl>
</NewOrderReq>

**Example XSLT File**

This section shows an example XSLT file which you can use as a reference to create your XSLT file. This XSLT file illustrates how you can use the number of OrderDtl element in a payload to assign a
Upload XSLT File

After you define the XSLT file for a throttle monitor, you must upload the file in the New ProcessBody XSLT field on Operations tab in the API Exchange Gateway user interface.

To upload the XSLT file, perform the following steps:

1. Start the GUI server, if not already running.
2. Login to the user interface with your credentials.
3. Add a new configuration or select an existing configuration.
4. Click on **ROUTING** tab on the right hand side.
5. Click on **Facade Operations** tab on the top menu.
6. Add a new operation or choose an existing operation.
7. Upload the XSLT transformation file to create a monitor as below:
   a. Click **Choose File** button in the New ProcessBody Transform field to select monitor XSLT transformation file.
   b. Click **Open** button to upload the XSLT file.
8. Save changes to your configuration.
Payload Size Throttles

TIBCO API Exchange Gateway allows you to create throttles based on the size of a payload in the message. For payload size throttles, the throttle count for a throttle is incremented as per the size of the payload in the message.

For the rate and quota throttle types, you can define the throttles based on the payload size as follows:

- payload size in the request message.
- payload size in the response message.
- payload size in both request and response message.

For payload size throttles, you must define a throttle counter to specify the message type for which the payload size is calculated. For example, if the throttle counter is defined as Input Payload Size, then the throttle count is incremented by the payload size of the request message. The throttle count is incremented by the payload size of the response message if the throttle counter is defined as Output Payload Size.

- The throttle Max Count for a payload size throttle should be specified in bytes.
- The payload size value in a message is used as an increment counter for the throttle count. The throttle counter field determines if the payload size is calculated for request message, or response message, or both request and response message.
- The payload size throttles are not supported for Error and High Water Mark (HWM) throttle types.

Payload Size Throttle Types

Based on the throttle counter value, payload size throttles can be of following sub-types:

- Input Payload Size
  
  Specifies that the throttle count for a throttle is incremented by the size of the input message (i.e request message).

- Output Payload Size
  
  Specifies that the throttle count for a throttle is incremented by the size of the output message (i.e response message).

- Transaction Payload Size
Specifies that the throttle count for a throttle is incremented by the sum of size of input and output message (i.e request and response payload).

For example, you can define a payload size throttle T2 as below:

```
Throttle Name: T2
Throttle Type : Rate
Throttle Interval : 10 seconds
Throttle Max Count: 60000
Throttle Counter: Input Payload Size
```

For this throttle T2 configuration, the gateway core engine allows 10000 bytes of request payload size in 10 seconds. For instance, if the first incoming request has a payload size of 5000 bytes, the gateway core engine performs following actions:

- Checks the current throttle count, which is 0 bytes as this is the first request.
- Compares if current throttle count (0 bytes) is less than or equal to the throttle max count (10000 bytes).
- Increments the current throttle count value by 5000 bytes, so the value of throttle count becomes 5000 bytes.
- Checks if current throttle count (5000 bytes) is less than or equal to the throttle max count (10000 bytes).
- Forwards the request to backend service for processing as the throttle count is less than max count.

If the second request received after 3 seconds by the gateway has a payload size of 5000 bytes, the gateway core engine performs following actions:

- Checks the current throttle count, which is 5000 bytes.
- Compares if current throttle count (5000 bytes) is less than or equal to the throttle max count (10000 bytes).
- Increments the current throttle count value by 5000 bytes, so the value of throttle count becomes 10000 bytes.
- Checks if current throttle count (10000 bytes) is less than or equal to the throttle max count (10000 bytes).
- Forwards the request to backend service for processing as the throttle count(10000) is less than max count (10000).

If the third request received after 2 seconds by the gateway has a payload size of 1000 bytes, the gateway core engine performs following actions:

- Checks the current throttle count, which is 10000 bytes.
Configuring Payload Size Throttles

The payload size throttles are configured using the TIBCO API Exchange Gateway user interface. You can configure the throttles for the payload size of request, or response, or both request and response messages.

To configure the payload size throttles, do the following:

1. Start the GUI server, if not already running.
2. Login to the user interface with your credentials.
3. Add a new configuration or select an existing configuration.
4. Click on Monitors tab on the right hand side.
5. Click on Monitoring tab.
6. Specify the following parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Monitor Name</td>
<td>Specifies the name of the throttle. For example, T2</td>
</tr>
<tr>
<td></td>
<td>Required.</td>
</tr>
<tr>
<td>Monitor Type</td>
<td>Specifies the type of throttle. Select one of the possible values from the drop down list box for payload size throttle:</td>
</tr>
<tr>
<td></td>
<td>Rate</td>
</tr>
<tr>
<td></td>
<td>Quota</td>
</tr>
<tr>
<td></td>
<td>Required.</td>
</tr>
<tr>
<td></td>
<td>Example, Rate</td>
</tr>
<tr>
<td>Interval</td>
<td>Specifies a time period for which the maximum count of throttles are allowed.</td>
</tr>
</tbody>
</table>
Configure Payload Size Throttle For Request Message

To configure the payload size throttle for a request message, configure the Throttle Counter as Input Payload Size for a throttle configuration.

Configure Payload Size Throttle For Response Message

To configure the payload size throttle for a response message, configure the Throttle Counter as Output Payload Size for a throttle configuration.

Configure Payload Size Throttle For Request And Response Message

To configure the payload size throttle for both request and response message, configure the Throttle Counter as Transaction Payload Size for a throttle configuration.

If the XSLT is configured to define content based throttle using metricName and metricIncrement for a throttle which also has payload based throttle configured, TIBCO API Exchange Gateway core engine applies the content based throttle policy during the runtime.
Traffic Shaping

Shaping is a new throttle violation policy. API Exchange Gateway is capable to shape the requests traffic upon reaching the throttle metric. By default, API Exchange Gateway blocks the throttled request and sends an error response.

When the Shaping feature is enabled for a facade operation request and the request violates rate throttle metric, then the transaction is placed on an internal queue and the transaction state is set to held. When the throttle metric has been reset by the global throttle manager, all the held transactions are released up to the limit for the throttle and the gateway engine continues to process these transactions. This means that all throttle chains etc are re-evaluated when the transaction is released.

For performance reasons, after the transactions are processed from the Shaping queue, the gateway engine does not remove these transactions from the queue. The gateway engine marks these transactions as processed. The gateway uses a separate timer to compact the queue to truncate the queue. See QueueCompactionInterval on page 321.

If the transactions in this queue time out, the gateway engine processes the request with normal error handling.

- Shaping is only valid for rate throttles.
- The Shaping feature does not support custom monitors.
- When the Shaping feature is enabled for a rate throttle and if the rate throttle violated is applied at the end of a throttle chain, the earlier throttles in the chain are not decremented.

Configuration

To enable the Shaping for a facade operation, do following:

1. Start the API Exchange Gateway user interface.
2. Create a new configuration project or select an existing project.
3. Click ROUTING > Facade Operations tab.
4. Create a new facade operation or select an existing facade operation.
5. Enter Shaping in Operation Features field.
6. Save the configuration changes.
QueueCompactInterval  Specifies a time interval (in milliseconds) used for cleaning up the shaping queue. The time interval can be set by the following property in ASG_CONFIG_HOME/asg.properties file.
   ASG/Throttle/Shaping/QueueCompactionInterval

The default value is 30000(ms).
Chapter 10  Authentication and Authorization

This chapter explains the user authentication and authorization functionality supported by TIBCO API Exchange Gateway.

Topics

- User Authentication Overview, page 324
- WS Security Services Authentication, page 326
- Configuring Web Services Security Authentication, page 340
- Partner Authorization Overview, page 348
User Authentication Overview

TIBCO API Exchange Gateways supports authentication at two levels:

— Transport Layer

This layer supports the authentication mechanism supported by the transport.

— Gateway Layer

This layer supports Web Services Security Authentication.

Transport and Protocol Level Authentication

When the partner sends a request, the first level authentication happens at the transport layer. TIBCO API Exchange Gateways supports authentication for following transports:

— HTTP(s)

TIBCO API Exchange Gateway uses the Apache HTTP Server.

— JMS

TIBCO API Exchange Gateway uses the TIBCO Enterprise Message Service.

Authentication at Apache HTTP server

Following types of authentication are supported at this layer:

— No Authentication - This indicates that the request does not have any user credentials. In this case, the request is processed as an anonymous user.

— Basic Authentication - This indicates that the request has the user credentials. In this case, the user is authenticated.

— Mutual Authentication using SSL certificates

Refer to the Apache HTTP server documentation to configure the authentication type for Apache server.

Authentication at TIBCO Enterprise Message Service

TIBCO API Exchange Gateway provides the authentication mechanism supported by the TIBCO Enterprise Message Service for the JMS transport.
For each of the authentication mechanism, the request is created with header fields for HTTP transport and JMS Application Header fields for JMS transport which are part of the request context message. This request context message becomes part of the RV message using the protocol termination functionality when forwarded to the gateway.
WS Security Services Authentication

TIBCO API Exchange Gateway supports the web services security (WSS) authentication services for the northbound messages.

TIBCO API Exchange Gateway supports following security token profiles:

- User name
  TIBCO API Exchange Gateway provides the user authentication for the northbound requests with the LDAP system.

- SAML 1.1 and SAML 2.0
  TIBCO API Exchange Gateway provides SAML based sign-in authentication of the northbound requests.

- X.509
  TIBCO API Exchange Gateway uses X.509 protocol to process the requests and confirm that integrity and confidentiality is maintained.

TIBCO API Exchange Gateway provides the processing of northbound messages as follows:

- Northbound Request Messages
  The gateway engine can verify the signature of the sender of the request using trust store as well as can decrypt it.

- Northbound Response Messages
  The gateway engine can sign the response message using private key to maintain integrity, as well as can encrypt it using the trust store and public certificate of the receiver of the response.

TIBCO API Exchange Gateway ensures availability, integrity and confidentiality by implementing the following protocols:

- SAML 1.1 and SAML 2.0 Authentication.
- X.509 based signature verification and public key infrastructure for non-repudiation.
- Signs the response using private keys issued by CA.
- Decrypts the request with private keys issued by CA. TIBCO API Exchange Gateways supports variety of encryption algorithms and modes.
- TIBCO API Exchange Gateway can encrypt the response document with the consumer’s public certificates.
Security Service Providers

This section describes the following types of security service providers:

- **Authentication Service Provider**
  
  Authenticated Service provider ensures that access is restricted to authenticated user. To access the web services managed by the API Exchange Gateway, the user must include an appropriate token in the SOAP header to authenticate.

  The supported authentication token are:

  - Web Services Security usernameToken
  - Web Services Security X.509 Certificate
  - Web Services Security SAML token profile

- **Identity Service Providers**

  Identity service providers makes use of public and private credentials for common trust and identity operations such as token signing, data encryption and creation of SSL connections. The main types of identity service providers are Trust Identity Provider and Subject Identity Provider.

Web Services Security (WSS) Properties

This section explains the Web Services Security (WSS) properties for TIBCO API Exchange Gateway.

Types of Security Service Providers

Table Types of Service Providers on page 327 lists the types of service providers used by WSS configuration.

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LDAP</td>
<td>LDAP authentication service provider (LDAP ASP) provides the ability to authenticate a username and password against an LDAP server.</td>
</tr>
<tr>
<td>Trust Identity</td>
<td>The Trust Identity Provider is used for retrieving certificates required for performing trust operations from a credential store. For example, use Trust identity provider (TIP) for verifying signature or encryption and SSL client authentication.</td>
</tr>
</tbody>
</table>
Table 67  Types of Service Providers

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Subject Identity</td>
<td>The Subject Identity Provider is used for retrieving and using private credentials obtained from a credential store. For example, use Subject identity provider (SIP) for signing or decryption.</td>
</tr>
<tr>
<td>WSS</td>
<td>WS security authentication provider is used as a combination of LDAP, Trust Identity Provider (TIP), and Subject Identity Provider (SIP). WSS service provider is a combination of LDAP authentication, Trust Identity and Subject Identity Providers. Depending on the usage of service provider, WSS can be configured to include one or more types of service providers that it is used for. Trust Identity Provider (TIP) and Subject Identity Provider (SIP) depends on Keystore Credential Provider (KCP), so TIP and SIP always include an associated KCP.</td>
</tr>
</tbody>
</table>

Configuring LDAP Authentication Service Provider (LDAP ASP)

**Description**  The LDAP authentication service provider is used to authenticate the user name and password against the LDAP server. The user name is specified as the usernameToken in the incoming request from the client.

**Use Case**  Verifying usernameToken in the incoming request.
Example Properties

See below:

```java
# LDAP
com.tibco.trinity.runtime.core.provider.authn.ldap.serverURL=ldap://trinity.rs.tibco.com:1389
com.tibco.trinity.runtime.core.provider.authn.ldap.userDNWithoutEncorePolicy=com.sun.jndi.ldap.LdapCtxFactory
com.tibco.trinity.runtime.core.provider.authn.ldap.userDNTemplate=uid(0),ou=people,dc=policy,dc=tibco,dc=com
com.tibco.trinity.runtime.core.provider.authn.ldap.userAttributesExtra=email,givenName
com.tibco.trinity.runtime.core.provider.authn.ldap.userSearchScopeSubtree=true
com.tibco.trinity.runtime.core.provider.authn.ldap.groupSearchExpression=uniqueMember=10
com.tibco.trinity.runtime.core.provider.authn.ldap.groupSearchBaseDN=ou=groups,dc=policy,dc=tibco,dc=com
com.tibco.trinity.runtime.core.provider.authn.ldap.groupAttributeUsersName=cn
com.tibco.trinity.runtime.core.provider.authn.ldap.groupIndication=groupHasUsers
com.tibco.trinity.runtime.core.provider.authn.ldap.followReferrals=true
com.tibco.trinity.runtime.core.provider.authn.ldap.connectionPools=5
com.tibco.trinity.runtime.core.provider.authn.ldap.searchTimeOut=1
com.tibco.trinity.runtime.core.provider.authn.ldap.enableSAML20Assertion=true
com.tibco.trinity.runtime.core.provider.authn.ldap.enableSAML11Assertion=false
com.tibco.trinity.runtime.core.provider.authn.ldap.samlValiditySeconds=300
```

Properties Table Properties for LDAP Authentication Service Provider on page 329 describes the properties for LDAP Authentication Service Provider.

### Table 68 Properties for LDAP Authentication Service Provider

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>com.tibco.asg.intent.usernameToken</td>
<td>Boolean intent property indicates if the LDAP authentication method can be enforced on the request message or not. Possible values are true or false. If the value of this property set to true, the request message must contain a valid username token.</td>
</tr>
<tr>
<td>com.tibco.trinity.runtime.core.provider.authn.ldap.initialCtxFactory</td>
<td>Specifies the name of the JNDI Factory to use. Default value is com.sun.jndi.ldap.LdapCfxFactory (Sun's LdapCtxFactory). Optional.</td>
</tr>
</tbody>
</table>
com.tibco.trinity.runtime.core.provider.authn.ldap.serverURL

Specifies the URL to connect to the LDAP directory server. The LDAP URL is defined as: "ldap://hostname:port". The LDAP SSL URL is defined as: "ldaps://hostname:port"

Required.

com.tibco.trinity.runtime.core.provider.authn.ldap.searchTimeOut

The time (in milliseconds) to wait for a response from the LDAP directory server. A value of 0 causes it to wait indefinitely. If a negative number is specified, it uses the provider's default setting.

Optional.

com.tibco.trinity.runtime.core.provider.authn.ldap.userAttributeUsersName

The name of the attribute in the user object that represents the user's name. The value depends on what LDAP server is used. If you are use ActiveDirectory LDAP Server, set this value as "CN". If SunOne or OpenLDAP LDAP Server is used, set this value as "uid".

Optional.

com.tibco.trinity.runtime.core.provider.authn.ldap.userAttributesExtra

 Specifies the optional list of user attributes to retrieve from the LDAP directory during authentication. Separation characters for the list of user attributes are comma, any ASCII whitespace or semicolon.

For example, mail givenname

Optional.

com.tibco.trinity.runtime.core.provider.authn.ldap.userSearchBaseDN

Specifies the base distinguished name (DN) where the searches for the users begin. You must supply the base DN that narrows the search to the smallest set of objects that includes all valid users. This is relevant only when used with administrator's credentials in search mode.

For example, ou=people,ou=na,dc=example,dc=org

Required in admin (search) mode.
**Table 68  Properties for LDAP Authentication Service Provider**

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>com.tibco.trinity.runtime.core.provider.authn.ldap.userSearchExpression</td>
<td>Specifies the expression to be used for searching in admin mode against potential user objects. For example, search expression is specified as: ((&amp;(uid={0})(objectClass=person))). In this string, the variable {0} represents the name of the user. The code substitutes the user name for this variable, and passes the resulting boolean expression to the LDAP server. The LDAP server matches that search expression against user objects to find a match. The search result must contain exactly one match. This property is relevant only when credentialProvider property is set and the binding is done as administrator; otherwise userDNTemplate is used. Required in admin (search) mode.</td>
</tr>
<tr>
<td>com.tibco.trinity.runtime.core.provider.authn.ldap.userDNTemplate</td>
<td>Specifies a template to be used when formatting user's DN before binding. It is used as an alternative to admin (search) mode. For example, uid={0},ou=employee,ou=tsi,o=tibco Required for bind mode (not in admin (search) mode).</td>
</tr>
<tr>
<td>com.tibco.trinity.runtime.core.provider.authn.ldap.userAttributeGroupsName</td>
<td>If you specified &quot;LDAP user indicates groups&quot; (as either userHasGroups or userDNHasGroups) then you must supply the name of the attribute in each user object that lists the groups to which the user belongs. Otherwise, this parameter is not relevant. Mandatory when relevant.</td>
</tr>
<tr>
<td>com.tibco.trinity.runtime.core.provider.authn.ldap.userAttributesExtraList</td>
<td>Same as userAttributesExtra property but this is specified in list form. Optional.</td>
</tr>
<tr>
<td>com.tibco.trinity.runtime.core.provider.authn.ldap.userSearchScopeSubtree</td>
<td>A boolean property which determines if the entire sub-tree is searched or not. If true value is specified, the entire sub-tree starting at the base DN is searched. Otherwise, the nodes one level below the base DN is searched. Optional.</td>
</tr>
</tbody>
</table>
### Table 68  Properties for LDAP Authentication Service Provider

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>com.tibco.trinity.runtime.core.provider.authn.ldap.groupSearchBaseDN</td>
<td>Specifies the base distinguished name (DN) where the searches for the groups begin. You must supply the base DN that narrows the search to the smallest set of objects that includes all valid groups. For example, <code>ou=groups,ou=na,dc=example,dc=org</code> Default value is empty. Optional.</td>
</tr>
<tr>
<td>com.tibco.trinity.runtime.core.provider.authn.ldap.enableNestedGroupSearch</td>
<td>Indicates the flag to determine if nested groups should be searched for. If the value is not set to <code>true</code>, the groups are only returned in which the user is the direct member. Default value is <code>false</code>. Optional.</td>
</tr>
<tr>
<td>com.tibco.trinity.runtime.core.provider.authn.ldap.groupSearchExpression</td>
<td>Specifies the expression to be used for searching against potential groups. For example, search expression is specified as: <code>(uid={0})(objectClass=person)</code>. In this string, the variable <code>{0}</code> represents the name of the user though. The code substitutes the user name for this variable, and passes the resulting boolean expression to the LDAP server. The LDAP server matches that search expression against groups to find all groups containing the username. The values might be different for different LDAP server. For example, its defined as <code>uniquemember={0}</code> for SunOne, <code>cn={0}</code> for OpenLDAP, <code>member={0}</code> for Active Directory. Required.</td>
</tr>
<tr>
<td>com.tibco.trinity.runtime.core.provider.authn.ldap.groupSearchScopeSubtree</td>
<td>A boolean property which determines if the entire sub-tree is searched or not. If <code>true</code> value is specified, the entire sub-tree starting at the base DN for groups is searched. Otherwise, the nodes one level below the base DN is searched. Optional.</td>
</tr>
</tbody>
</table>
Specifies how the group memberships for users are found.

Default value is `noGroupInfo`.

Optional.

Possible values are as below:
- `userHasGroups`
- `userDNHasGroups`
- `groupHasUsers`
- `noGroupInfo`

- If the value has `userHasGroups`, you must specify the attribute name which points the groups the user belongs to in the `userAttributeGroupsName` property.

- If the value has `userDNHasGroups`, the `userAttributeGroupsName` property has the attribute name which hold the DNs of groups to which the user belongs. You must specify `groupAttributeGroupsName` property to get a specific part of the DN name.

- If the value has `groupHasUsers`, each group object includes a list of users that belong to the group.

- If the value has `noGroupInfo`, group memberships aren't handled.

Depending on `groupIndication's` value:

- `groupHasUsers`: group attribute holding the group’s name. Example value for OpenLDAP: `cn`, for Active Directory: `sAMAccountName`. Mandatory.

- `userHasGroups`: group's name part holding group’s name. If not specified the group’s whole DN will be used. Example `cn` otherwise ignored.

Specifies the name of the attribute in each group object denoting subgroups.

For example, the value is defined as `uniqueMember` for OpenLDAP server, `member` for ActiveDirectory LDAP server.

Optional.
Table 68  Properties for LDAP Authentication Service Provider

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>com.tibco.trinity.runtime.core.provider.authn.ldap.groupAttributeUsersName</td>
<td>Specifies the attribute name if the groupIndication property has groupHasUsers value. It specifies the name of the attribute in each group object denoting its users. For example, the value is uniqueMember for OpenLDAP, member for ActiveDirectory Server. Required if the groupIndication property has groupHasUsers value.</td>
</tr>
<tr>
<td>followReferrals</td>
<td>Determines if the client follow referrals returned by the LDAP server. Default value is false. Optional.</td>
</tr>
<tr>
<td>com.tibco.trinity.runtime.core.provider.identity.trust.trustStoreServiceProvider</td>
<td>Specifies the Identity trust provider configuration to provide SSL support for LDAP</td>
</tr>
<tr>
<td>com.tibco.trinity.runtime.core.provider.credential.keystore.truststore.keyStoreLocation</td>
<td>Specifies the location of the keystore for the credentials.</td>
</tr>
<tr>
<td>com.tibco.trinity.runtime.core.provider.credential.keystore.truststore.keyStorePassword</td>
<td>Specifies the location of the keystore for the credentials.</td>
</tr>
<tr>
<td>com.tibco.trinity.runtime.core.provider.credential.keystore.truststore.keyStoreRefreshInterval</td>
<td>Specifies the refresh interval (milliseconds).</td>
</tr>
<tr>
<td>com.tibco.trinity.runtime.core.provider.credential.keystore.truststore.keyStoreType</td>
<td>Specifies the keystore type. Supported formats are JKS,PKCS12.</td>
</tr>
</tbody>
</table>

Sample File  The properties and example configuration for LDAP authentication service providers are found in the following sample files:
• \texttt{ASG\_CONFIG\_HOME}/default/wss/req_usernametoken\_ldapbind.properties

This file lists the properties with the example configuration for the LDAP server in bind mode.

• \texttt{ASG\_CONFIG\_HOME}/default/wss/req_usernametoken\_ldapsearch.properties

This file lists the properties with the example configuration for the LDAP server in search mode.

• \texttt{ASG\_CONFIG\_HOME}/default/wss/req_usernametoken\_ldapbindssl.properties

This file lists the properties with the example configuration for the LDAP server in SSL mode.

### Configuring Trust Identity Provider

**Description**

The Trust Identity Provider is used to retrieve public certificates from a credential store required to perform trust operations. You must store the public certificate and provide its location. The certificates are used by the gateway core engine to verify the signatures when the payload in the incoming request is signed. The gateway core engine uses the public certificate to encrypt the response payload before it sends the response back to the client.

**Use Case**

• Verify signatures for the signed request payload.
• Encrypt the response payload.

**Example Properties**

See below:

```ini
# TIP
com.tibco.trinity.runtime.core.provider.identity.trust.trustStoreServiceProvider=class:com.tibco.trinity.runtime.core.provider.credential.KeyStore
com.tibco.trinity.runtime.core.provider.credential.keyStore=\$HOME/tibco/trinity/keystore/keystore.jks
com.tibco.trinity.runtime.core.provider.credential.keyStoreKey=\$HOME/tibco/trinity/keystore/keystore.jks
com.tibco.trinity.runtime.core.provider.credential.keyStorePassword=\$HOME/tibco/trinity/keystore/keystore.jks
```

```ini
# JKS for JFIP
com.tibco.trinity.runtime.core.provider.credential.keyStore=\$HOME/tibco/trinity/keystore/keystore.jks
com.tibco.trinity.runtime.core.provider.credential.keyStoreKey=\$HOME/tibco/trinity/keystore/keystore.jks
com.tibco.trinity.runtime.core.provider.credential.keyStorePassword=\$HOME/tibco/trinity/keystore/keystore.jks
```
Table Properties for Trust Identify Provider (TIP) on page 336 describes the properties for Trust Identify Provider.

**Table 69 Properties for Trust Identify Provider (TIP)**

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>com.tibco.asg.intent.signature</td>
<td>Boolean intent property indicates if the incoming request message is signed or not. If signed, then the signatures are verified using the trust identity provider properties (public credentials). Possible values are true or false. If the value of this property set to true, the request message must have valid signatures.</td>
</tr>
<tr>
<td>com.tibco.trinity.runtime.core.provider.identity.trust.trustStoreServiceProvider</td>
<td>Specifies the name of the credential service provider containing the credentials for establishing trust.</td>
</tr>
<tr>
<td>com.tibco.trinity.runtime.core.provider.credential.keystore.truststore.keyStoreType</td>
<td>Specifies the keystore type. Supported formats are JKS,PKCS12.</td>
</tr>
<tr>
<td>com.tibco.trinity.runtime.core.provider.credential.keystore.truststore.keyStoreLocation</td>
<td>Specifies the location(s) of the keystore.</td>
</tr>
<tr>
<td>com.tibco.trinity.runtime.core.provider.credential.keystore.truststore.keyStorePassword</td>
<td>Specifies the password to unlock the keystore.</td>
</tr>
<tr>
<td>com.tibco.trinity.runtime.core.provider.credential.keystore.truststore.keyStoreRefreshInterval</td>
<td>Specifies the refresh interval (milliseconds).</td>
</tr>
</tbody>
</table>

Sample File
- See `ASG_CONFIG_HOME/default/wss/req_verifysig.properties` file for the properties and example configuration for verifying the signature in the request message.
- See `ASG_CONFIG_HOME/default/wss/resp_encrypt.properties` file for the properties and example configuration for encrypting the response message.
Configuring Subject Identity Provider

**Description**
The Subject Identity Provider is used to retrieve private keys (credentials) from a credential store. You must store the private keys and provide its location. The private keys are used by the gateway core engine to decrypt the message when the payload in the incoming request is encrypted. The gateway core engine uses the private keys to sign the response message before sending it back to the client.

**Use Case**
- Decrypt the request payload.
- Sign the response payload.

**Example**
See below:

### Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>com.tibco.asg.intent.decrypt</td>
<td>Boolean intent property indicates if the incoming request message is encrypted or not. If encrypted, then the request message payload is decrypted using the subject identity provider properties (private credentials). Possible values are <code>true</code> or <code>false</code>. If the value of this property set to <code>true</code>, the request message must be encrypted.</td>
</tr>
<tr>
<td>com.tibco.trinity.runtime.core.provider.identity.subject.identityStoreServiceProvider</td>
<td>Specifies the name of the credential service provider containing the private credentials for establishing the subject's identity.</td>
</tr>
</tbody>
</table>
Table 70  Properties for Subject Identity Provider (SIP)

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>com.tibco.trinity.runtime.core.provider.identity.subject.keyAlias</td>
<td>Specifies an alias name for the key corresponding to the private credentials in the credential store for establishing the subject's identity.</td>
</tr>
<tr>
<td>com.tibco.trinity.runtime.core.provider.identity.subject.keyPassword</td>
<td>Specifies the protection parameter of the private credentials in the credential store for establishing the subject's identity.</td>
</tr>
<tr>
<td>com.tibco.trinity.runtime.core.provider.credential.keystore.keyStoreType</td>
<td>Specifies the keystore type of the private credentials.</td>
</tr>
<tr>
<td>com.tibco.trinity.runtime.core.provider.credential.keystore.keyStoreLocation</td>
<td>Specifies the location(s) of the keystore of the private credentials.</td>
</tr>
<tr>
<td>com.tibco.trinity.runtime.core.provider.credential.keystore.keyStorePassword</td>
<td>Specifies the password to unlock the keystore.</td>
</tr>
<tr>
<td>com.tibco.trinity.runtime.core.provider.credential.keystore.keyStoreRefreshInterval</td>
<td>Specifies the refresh interval in milliseconds.</td>
</tr>
</tbody>
</table>

Sample File
- See `ASG_CONFIG_HOME/default/wss/req_decrypt.properties` file for the properties and example configuration for decrypting a request message.
- See `ASG_CONFIG_HOME/default/wss/resp_sign.properties.properties` file for the properties and example configuration for encrypting a request message.

Configuring WSS Service Provider

Description
You can combine the properties of LDAP, Subject Identity Provider (SIP) and Trust Identity Provider(TIP) to obtain more than one functionality. For example, you can verify the signatures in an incoming payload, when signed and also decrypt the request payload, when encrypted.

Use Case
- Verify signatures in the request payload and decrypt the request payload.
- Sign and Encrypt the response payload.
Example Properties

See below:

```java
#Example configuration where incoming message is encrypted and signed
com.tibco.agg.intent.signature=true
com.tibco.agg.intent.decrypt=true

#SIP Namespace. Used for token signing service or decrypting the message before authentication.
com.tibco.trinity.runtime.core.provider.identity.subject.keyAlias=self-signed-Server-key-cert-pkcs1-trusted-by-client
com.tibco.trinity.runtime.core.provider.identity.subject.keyPassword=#{fGqMyE7ToEs8y1QEt7aykDYhfWg9m3jXMVasJwsSNAC4=
com.tibco.trinity.runtime.core.provider.identity.subject.enableCredentialStoreAccess=true
com.tibco.trinity.runtime.core.provider.identity.subject.enableTrustStoreAccess=true

#KCP for SIP namespace. Used for token signing service or decrypting the message before authentication.
com.tibco.trinity.runtime.core.provider.credential.keystore.keyStoreLocation=keystores/ServerSignatureKS.jks
com.tibco.trinity.runtime.core.provider.credential.keystore.keyStorePassword=#{fGqMyE7ToEs8y1QEt7aykDYhfWg9m3jXMVasJwsSNAC4=
com.tibco.trinity.runtime.core.provider.credential.keystore.keyStoreType=JKS
com.tibco.trinity.runtime.core.provider.credential.keystore.keyStoreRefreshInterval=60000

#KCP for TIP Namespace. Used for verifying the signature.
com.tibco.trinity.runtime.core.provider.credential.keystore.truststore.keyStoreLocation=keystores/server-CAs.jks
com.tibco.trinity.runtime.core.provider.credential.keystore.truststore.keyStorePassword=#{fGqMyE7ToEs8y1QEt7aykDYhfWg9m3jXMVasJwsSNAC4=
```

Properties

The properties for WSS Service Provider are defined as a combination of LDAP authentication, Subject Identity and Trust Identity provider.

See the following properties to define the WSS Service Provider properties:

- Configuring LDAP Authentication Service Provider (LDAP ASP) on page 328
- Configuring Trust Identity Provider on page 335
- Configuring Subject Identity Provider on page 337

Sample File

- See `ASG_CONFIG_HOME/default/wss/req_decrypt_verifysig.properties` file for the properties and example configuration for decrypting and verifying signatures for the request message.

- See `ASG_CONFIG_HOME/default/wss/resp_sign_encrypt.properties` file for the properties and example configuration for signing and encrypting the response message.

Limitations

- WSS authentication does not check if the request contains SAML 1.1 or SAML 2.0 version.
WSS authentication validates the request only if the SAML assertion is valid but does not enforce a specific SAML version or issuer on the request.

Configuring Web Services Security Authentication

This section explains the procedure to configure the web services security for the gateway engine.

Define the WSS Configuration Properties file

This section explains how to define the properties files required for the WSS shared resources configuration.

Sample Files

TIBCO API Exchange Gateway provides the sample configuration file for the shared resources for each of the security type profile. It is recommended to use the sample files as templates and edit the properties as per your requirement. The sample files are located in the ASG_CONFIG_HOME/asg/default/wss directory.

The property files are defined depending on the type of WSS configuration selected. Following section explains the WSS type and a sample property file which can be used for that type:

User name token

TIBCO API Exchange Gateway authenticates the user with the LDAP system and requires to create the configuration file for LDAP configuration as follows:

LDAP configuration for bind mode

This configuration type provides the authentication based on the user name token with a LDAP system for bind mode.

The sample file req_usernametoken_ldapbind.properties for LDAP shared resource configuration is located in the following directory:

ASG_CONFIG_HOME/asg/default/wss

You can use this file as a template and edit the LDAP server properties as per your environment.

LDAP configuration in bind mode with SSL Enabled

This configuration type provides the authentication based on the user name token with a LDAP system with SSL enabled in bind mode.

The sample file req_usernametoken_ldapbindssl.properties for LDAP shared resource configuration is located in the following directory:

ASG_CONFIG_HOME/asg/default/wss
You can use this file as a template and edit the LDAP server with SSL properties as per your environment.

**LDAP configuration for search mode**

This configuration type provides the authentication based on the user name token with a LDAP system for search mode.

The sample file `req_usernametoken_ldapsearch.properties` for LDAP shared resource configuration is located in the following directory:

`ASG_CONFIG_HOME/asg/default/wss`

You can use this file as a template and edit the LDAP server properties for search mode as per your environment.

- **Subject Identity**

  The configured keystore along with a valid key from keystore can be used to provide an identity of the interested subject. Identity provider takes as an input the password of the Key alias, and it is used to access the private key of that particular alias. This is used for signing.

  TIBCO API Exchange Gateway requires certain properties to be defined for this type. These properties are defined in a file, which, can be imported in the configuration GUI. See Define the WSS Configuration Properties file, page 340

  This configuration type provides the properties for the keystore configuration (private key) to sign the message or decrypt the message.

  The sample file `resp_sign.properties` describes the keystore properties required to sign the message. This file is located in the following directory:

  `ASG_CONFIG_HOME/asg/default/wss`

  You can use this file as a template and edit the keystore configuration as per your environment.

- **Trust Identity**

  The trust store consumes a keystore provider and it is used for accessing public keys of the keys for signature verification or for encryption.

  TIBCO API Exchange Gateway requires certain properties to be defined for this type. These properties are defined in a file, which, can be imported in the configuration GUI. See Define the WSS Configuration Properties file, page 340

  This configuration type provides the properties for the keystore configuration to verify the signatures or encrypt the message.

  The sample file `resp_encrypt.properties` describes the certificate keystore properties required to encrypt the message. This file is located in the following directory: `ASG_CONFIG_HOME/asg/default/wss`
You can use this file as a template and edit the keystore configuration as per your environment.

**Register WSS resources with TIBCO API Exchange Gateway**

The **WSS** tab on the configuration allows you to register the WSS resources with TIBCO API Exchange Gateway.

To setup the WSS configuration, perform the following steps:

1. Start the GUI, if not already started.
2. Click **WSS** tab.
3. Enter the details for WSS, as defined below:

<table>
<thead>
<tr>
<th>Table 71 WSS Configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Parameter</strong></td>
</tr>
<tr>
<td>WSS Name</td>
</tr>
<tr>
<td>Type</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>New Property File</td>
</tr>
<tr>
<td>Existing Property Files</td>
</tr>
</tbody>
</table>

**Define the WSS security operations**

This section explains the steps to define a WSS enabled security operation. An operation is WSS enabled using the **Operations** tab of the GUI.

To setup the WSS configuration, perform the following steps:

1. On the configuration GUI, Click **ROUTING** tab.
2. Click on the **Facade Operations** tab.
3. Add a new operation. Enter the details of the Operation. See **Facade Operations** on page 193
4. Check the **Enable WSS** checkbox.
5. Enter the details for WSS enabled operation as defined below:

Table 72  WSS Enabled Operation Configuration

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WSS Request</td>
<td>This is the name of the WSS configuration from WSS tab. The property file from this configuration is used for northbound request processing.</td>
</tr>
<tr>
<td>WSS Response</td>
<td>This is the name of the WSS configuration from WSS tab. The property file from this configuration is used for northbound response processing.</td>
</tr>
<tr>
<td>Encrypt Response</td>
<td>This checkbox flag indicates whether to encrypt the response message.</td>
</tr>
<tr>
<td>Sign Response</td>
<td>This checkbox flag indicates whether to sign the response message.</td>
</tr>
<tr>
<td>Encryption</td>
<td>This list box allows to select the algorithm to use for data encryption. Supported values are:</td>
</tr>
<tr>
<td>Algorithm</td>
<td>TRIPLE_DES, AES_128, AES_256, AES_192</td>
</tr>
<tr>
<td>Key Algorithm</td>
<td>This list box allows to select the algorithm to use for key encryption. Supported values are:</td>
</tr>
<tr>
<td></td>
<td>RSA15, RSAOEP, AES128, AES192, AES256, TRIPLEDES</td>
</tr>
<tr>
<td>Signing</td>
<td>This list box allows to select the algorithm to use for signing. Supported values are:</td>
</tr>
<tr>
<td>Algorithm</td>
<td>HMAC_MD5, DSA_SHA1, HMAC_SHA1, RSA_SHA1, RSA_MD5, RSA_RIPEMD160, RSA_SHA256, RSA_SHA384, RSA_SHA512, HMAC_RIPEMD160, HMAC_SHA256, HMAC_SHA384, HMAC_SHA512</td>
</tr>
<tr>
<td>Key Type</td>
<td>Supported values are:</td>
</tr>
<tr>
<td></td>
<td>BST_DIRECT_REFERENCE, ISSUER_SERIAL, X509_KEY_IDENTIFIER, SKI_KEY_IDENTIFIER, EMBEDDED_KEYNAME, EMBED_SECURITY_TOKEN_REF, UT_SIGNING, THUMBPRINT_IDENTIFIER</td>
</tr>
<tr>
<td>Keystore Alias</td>
<td>A keystore file alias as configured in the Keystores tab of the UI.</td>
</tr>
</tbody>
</table>
Configure Secure Services With API Exchange Gateway

API Exchange Gateway provides the transport for the secure communication with the web services at the backend. You can access the services with or without client authentication.

The back end services may or may not require X.509 client authentication. The "Is Anonymous" flag for a service determines if the client authentication is required or not. The client authentication, also known as mutual SSL authentication is required if the "Is Anonymous" flag is set to false. If the "Is Anonymous" flag is set to true, the service does not require the authentication of client.

Define DSS Properties For Services

To use the services, you must define the DSS properties in a file. The DSS properties file is used during the configuration of the service using API Exchange Gateway User Interface. See Configuring Services on page 346.

This section explains the properties required to use the backend services using the transport.

Properties For SSL Authentication (isAnonymous = true)

Trust Identity Provider (TIP) properties are used if the "Is Anonymous" flag is set to true for any service. API Exchange Gateways supports the one way SSL authentication i.e, when the service is accessed by the API Exchange Gateway core engine and the service does not require the authentication of the client.

Use Case

Use service when no authentication of the client required (one way SSL).

Example Properties

<table>
<thead>
<tr>
<th>Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>com.tibco.trinity.runtime.core.provider.credential keystore.truststore.keyStoreLocation=E:/Keystores/trust_bw.jks</td>
</tr>
<tr>
<td>com.tibco.trinity.runtime.core.provider.credential keystore.truststore.keyStorePassword=password</td>
</tr>
<tr>
<td>com.tibco.trinity.runtime.core.provider.credential keystore.truststore.keyStoreRefreshInterval=60000</td>
</tr>
<tr>
<td>com.tibco.trinity.runtime.core.provider.credential keystore.truststore.keyStoreType=JKS</td>
</tr>
</tbody>
</table>
Table SSL Authentication Properties for Service on page 345 explains the properties for SSL authentication (one way SSL authentication) for the service.

### Properties For Mutual SSL Authentication (isAnonymous = false)

Subject Identity Provider (SIP) properties are used if the "Is Anonymous" flag is set to false for any service. API Exchange Gateway supports the mutual SSL authentication to access the service.

#### Use Case

Using service when client authentication (mutual SSL authentication) required.

#### Example

See below:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>com.tibco.trinity.runtime.core.provider.credential.keystore.truststore.keyStoreLocation</td>
<td>Specifies the location(s) of the keystore.</td>
</tr>
<tr>
<td>com.tibco.trinity.runtime.core.provider.credential.keystore.truststore.keyStorePassword</td>
<td>Specifies the password to unlock the keystore.</td>
</tr>
<tr>
<td>com.tibco.trinity.runtime.core.provider.credential.keystore.truststore.keyStoreRefreshInterval</td>
<td>Specifies the refresh interval (milliseconds).</td>
</tr>
<tr>
<td>com.tibco.trinity.runtime.core.provider.credential.keystore.truststore.keyStoreType</td>
<td>Specifies the keystore type. Supported formats are JKS, PKCS12.</td>
</tr>
</tbody>
</table>

Properties

```java
com.tibco.trinity.runtime.core.provider.identity.subject.keyAlias=my_key
com.tibco.trinity.runtime.core.provider.identity.subject.keyPassword=password
com.tibco.trinity.runtime.core.provider.identity.subject.enableCredentialStoreAccess=true
com.tibco.trinity.runtime.core.provider.identity.subject.enableTrustStoreAccess=true
com.tibco.trinity.runtime.core.provider.credential.keystore.keyStoreLocation=keystores/default_keystore.jks
com.tibco.trinity.runtime.core.provider.credential.keystore.keyStorePassword=password
com.tibco.trinity.runtime.core.provider.credential.keystore.keyStoreRefreshInterval=60000
```
### Properties

Table Mutual Authentication SSL Properties For service on page 346 explains the properties for mutual SSL authentication (client authentication) for a service.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>com.tibco.trinity.runtime.core.provider.identity.subject.keyAlias</td>
<td>Specifies an alias name for the key corresponding to the private credentials in the credential store for establishing the subject's identity.</td>
</tr>
<tr>
<td>com.tibco.trinity.runtime.core.provider.identity.subject.keyPassword</td>
<td>Specifies the protection parameter of the private credentials in the credential store for establishing the subject's identity.</td>
</tr>
<tr>
<td>com.tibco.trinity.runtime.core.provider.credential.keystore.keyStoreType</td>
<td>Specifies the keystore type of the private credentials.</td>
</tr>
<tr>
<td>com.tibco.trinity.runtime.core.provider.credential.keystore.keyStoreLocation</td>
<td>Specifies the location(s) of the keystore of the private credentials.</td>
</tr>
<tr>
<td>com.tibco.trinity.runtime.core.provider.credential.keystore.keyStorePassword</td>
<td>Specifies the password to unlock the keystore.</td>
</tr>
<tr>
<td>com.tibco.trinity.runtime.core.provider.credential.keystore.keyStoreRefreshInterval</td>
<td>Specifies the refresh interval in milliseconds.</td>
</tr>
</tbody>
</table>

### Configuring Services

You can create and configure a service (target operation) using API Exchange Gateway User Interface.

To configure a target operation, you must provide a DSS properties file. See Define DSS Properties For Services, page 344.

To configure a service, do following:

1. Start the API Exchange Gateway User Interface.
2. Login to the user interface using your credentials.
3. Create a new project configuration.
4. Select and click the newly created configuration.
5. Select ROUTING from the top menu bar.
7. Create a new target operation using the "+" icon at the top menu bar.
8. Make sure to select from the drop down list box for the Type field.
9. Set the following fields as below:
   a. Is Anonymous - Set this field to false for mutual SSL authentication. If you do not require client authentication, set this field to true.
   b. New Property File: Select the DSS property file you created to define the properties as explained in Define DSS Properties For Services, page 344.
   c. Existing Property Files: You can select an existing DSS property file from the drop down list box if the file exists in the wss directory of the configuration.
10. Save the changes to the services configuration.
Partner Authorization Overview

TIBCO API Exchange Gateway supports the authorization based on following actions:

**Operation Identification**

TIBCO API Exchange Gateway identifies the operation based as follows:

- For SOAP requests, the operation is identified from the SOAP Action header, and/or URI as defined in the HTTP header.
- For HTTP/XML requests, the operation is identified from the combination of method and URI.
- For HTTP/S REST requests, the operation is identified from the combination of method, URI and the value of some named HTTP header.

**Partner Identification**

TIBCO API Exchange Gateway uses the Partner Serial number and Partner Issuer CA from the header fields of the incoming request to uniquely identify the partner. The gateway maps the authenticated users from the transport headers to validate the identified partner in the gateway configuration repository.

The Partner Serial Number and Partner Issuer CA are configured on the PARTNER > Partners tab of the gateway user interface for a project configuration.

For example, for HTTP/(s) transport, the partner is identified as follows:

- Anonymous user
  
  If no user is specified in the incoming request, the gateway engine considers this request as a request from anonymous user which is not authenticated. The gateway engine looks for the partner name defined by the tibco.clientVar.ASG/anonymous/PartnerName/Authenticated property in ASG_CONFIG_HOME/asg.properties file. The gateway engine matches the
value of this property with the value defined by Partner Name field under Partners tab on the gateway user interface. If both the values match, the gateway engine further processes the request.

For example, the property is defined as below in ASG_CONFIG_HOME/asg.properties file:

tibco.clientVar.ASG/anonymous/PartnerName/Authenticated=anon_partner

To process any unauthenticated requests where no user is specified in the request, you must configure a partner as anon_partner under PARTNER > Partners tab on the gateway user interface.

If there is a mismatch, then the gateway engine rejects the partner with Authorization error.

By default, the gateway provides an anon partner to handle the requests from unauthenticated users.

- Mutual SSL Authentication

If the gateway engine receives the request using mutual SSL authentication mechanism, the partner is identified by the certificate issuer and serial number from the certificate retrieved from the SSL headers.

The gateway engine retrieves the user name and issuer CA from the request headers. The gateway engine matches the user name and issuer CA as specified in the request header with the Partner Serial Number and Partner Issuer CA fields under Partners tab on the gateway user interface.

If there is a mismatch, then the gateway engine rejects the partner with Authorization error.

**Partner Authorization**

After the operation and partner is identified, TIBCO API Exchange Gateway validates that the identified partner is authorized to invoke the operation. You can setup the configuration details under PARTNER > Facade Access tab of the gateway user interface where you specify the operation which the identified partner is allowed to access.
Chapter 11  Security Policies

This chapter explains the security policies supported by TIBCO API Exchange Gateway software.

Topics

- Overview on page 352
- Types of Security Policies on page 357
- Manage Policies on page 361
- Policy Use Cases on page 368
- Types of Security Shared Resources on page 387
- Shared Resources Properties on page 388
- Shared Resources Properties Sample Files on page 398
Overview

API Exchange Gateway allows you to secure a facade operation or a target operation using various types of security policies. This allows you to apply the policy to the incoming messages received from the service consumers and also apply the policy to the outgoing messages forwarded to the service providers. You can apply the policies at the endpoints of facade or target operations. See Types of Security Policies, page 357 for the details on the supported policies.

Figure Security Enforcement EndPoints on page 353 illustrates the security policy enforcement points in the standard request processing pipeline.
Figure 19  Security Enforcement EndPoints
Security Concepts

This section explains few terms required to understand how the policies can secure an incoming request or outgoing request.

Authentication

Authentication is a process of identifying the credential of the party who sent the request. API Exchange Gateway supports the following types of authentication:

- Basic
  
  In the basic authentication, the credential used for authentication is obtained from the HTTP authorization header in the form of username and password. The username and password are authenticated against an LDAP authentication provider.

- UsernameToken
  
  In UsernameToken authentication, the credential used for authentication is the usernameToken obtained from the security header of the SOAP message. The username and password from the usernameToken are authenticated against an LDAP authentication provider.

- Security Assertion Markup Language (SAML)
  
  In SAML authentication, the credential used for authentication is the SAML assertion derived from the security header of the SOAP message. The SAML assertion is authenticated using an identity service provider.

- X509
  
  In X509 authentication, the credential used for authentication is the X509 certificate in the SAML assertion from the security header of the SOAP message. To use the X509 authentication, the SOAP message must be sent using X509 token profile. The SAML assertion is authenticated using an identity service provider.

Authorization

Authorization is a process of authorizing the party who has been authenticated to access some resources and allowing the party to proceed with the incoming request. API Exchange Gateway supports the authorization of a request on the basis of roles. When a request is authenticated, a SAML assertion is generated that may contains the roles as attributes of the SAML assertions. The roles in the SAML assertion may be originated as follows:

- From the groups defined in the LDAP which is applicable for basic or usernameToken authentication.

- From the authenticated SAML assertion which is applicable for SAML or X509 authentication.
Confidentiality

Confidentiality is a process to ensure that the data is accessible to the intended party only. To achieve this goal, the data is encrypted by the sender using a public certificate. The receiver decrypts the data using a private key before using the data.

Integrity

Integrity is a process to ensure that the data has not been tampered with. To achieve this goal, the data is signed by the party who sent the request and includes the signature along with a digital certificate in the request. The receiver can verify signature using the certificate to determine the integrity of the data received.

Credential Mapping

Credential Mapping is a process of propagating an identity to the outgoing request. The gateway propagates the credentials using usernameToken or SAML assertion.

Policy

A policy specifies how the gateway enforces the security constraints applied to facade or target operations. Each policy has an assertion to perform an intended security constraint such as authentication, authorization, confidentiality, integrity, or credential mapping. You must refer to following topics for details to understand the policy:

See Policy Use Cases on page 368 for details on assertions for specific policy.

To enforce or process a policy at runtime, the gateway requires following external resources:

- Authentication service providers
- Identity service providers
- Trust service providers

Shared Resource

Any provider such as Authentication service provider, Identity service provider or Trust service provider may be used by more than one policy. This means that these providers are shared among the policies as a collection of shared resources. A policy usually refers to a service provider as a resource instance. A policy views a specific service provider as a resource instance which is configured as a shared resource.

A shared resource is a configured resource that may be used by one or more policy. For example, If you configure a resource instance named LdapAspRI, the same resource can be used for LDAP authentication as well as WSS authentication.

See Define Shared Resource Properties File on page 361 for the list of shared resource that are applicable to specific type of policy.
## Policy Types and Subtypes

The following table lists the policy types and subtypes supported by API Exchange Gateway.

**Table 75  Policy Types and SubTypes**

<table>
<thead>
<tr>
<th>Policy Type</th>
<th>Policy Subtype</th>
<th>Endpoints</th>
</tr>
</thead>
<tbody>
<tr>
<td>Authentication</td>
<td>— Basic</td>
<td>• Facade Operation</td>
</tr>
<tr>
<td></td>
<td>— UsernameToken</td>
<td></td>
</tr>
<tr>
<td></td>
<td>— SAML</td>
<td></td>
</tr>
<tr>
<td></td>
<td>— OAuth</td>
<td></td>
</tr>
<tr>
<td>Authorization</td>
<td>— Role</td>
<td>• Facade Operation</td>
</tr>
<tr>
<td>Integrity</td>
<td>— Sign</td>
<td>• Facade Operation</td>
</tr>
<tr>
<td></td>
<td>— Verify Signature</td>
<td>• Target Operation</td>
</tr>
<tr>
<td>Confidentiality</td>
<td>— Encrypt</td>
<td>• Facade Operation</td>
</tr>
<tr>
<td></td>
<td>— Decrypt</td>
<td>• Target Operation</td>
</tr>
<tr>
<td>Credential Mapping</td>
<td>— Basic</td>
<td>• Target Operation</td>
</tr>
<tr>
<td></td>
<td>— UsernameToken</td>
<td></td>
</tr>
<tr>
<td></td>
<td>— SAML</td>
<td></td>
</tr>
<tr>
<td></td>
<td>— OAuth</td>
<td></td>
</tr>
</tbody>
</table>
Types of Security Policies

This section explains the following types of security policies supported by API Exchange Gateway.

Authentication

An authentication policy determines how to authenticate the users. An authentication policy requires that the incoming request must provide the identities of the sender so that the gateway will authenticate those identities before processing the request.

You can define an authentication policy for a client to require that target services must authenticate the client’s identity before processing a request. A client authentication policy is usually applied at target services.

API Exchange Gateway supports following types of authentication policies:

Basic

When the client sends the username and password in the HTTP basic authentication header of the request message, you can enforce a basic authentication policy to authenticate the client’s identity. The basic authentication policy authenticates the username and password in the client request against LDAP Authentication service provider and generates SAML 2.0 assertion which is forwarded to the API Exchange Gateway.

UsernameToken

The usernameToken authentication policy authenticates the username and password specified with the usernameToken in the client request message using a specified LDAP shared resource.

SAML

API Exchange Gateway provides SAML authentication policy which allows you to authenticate the credentials in the SAML assertion from the security header of the SOAP message. The SAML assertion is authenticated using an identity service provider shared resource.
**X509**

API Exchange Gateway provides the X509 security policy so that the target operations with SOAP bindings can authenticate the consumer’s identity using the consumer’s X509 signature. The consumer’s identity is authenticated using an identity service provider shared resource.

See following policies:

- `BasicAuthentication.policy on page 369`
- `AuthenticationByUsernameToken.policy on page 370`
- `AuthenticationBySaml.policy on page 370`

**OAuth**

API Exchange Gateway provides the authentication by OAuth policy. The authentication by OAuth policy ensures that any access to a target operation with this policy enforced must be authenticated by an OAuth authorization server. The authorization server used is specified in the policy along with the client ID and client secret registered with an OAuth authorization server.

See `AuthenticationbyOAuth Policy on page 372`.

**Authorization**

API Exchange Gateway supports following authorization policies:

**Role**

Authorization by role policy of API Exchange Gateway provides a way to authorize the user based on the role.

See `Authorization By Role Policy on page 374`.

**Confidentiality**

API Exchange Gateway enforces the confidentiality of the data in the requests and responses as follows:

- Decrypts the encrypted data in the facade request.
- Encrypts the data in the target request to forward to any external target operation.
- Encrypts the data in the target response.

See following policies:
• Encryption.policy on page 381
• Decryption.policy on page 382

**Integrity**

API Exchange Gateway ensures the integrity of inbound and outbound requests by virtue of Integrity policy in following ways:

• Verify the signatures of the users of the incoming facade request.
• Sign the request to forward to any external target operation.
• Sign the facade response.

See following policies:

• Sign on page 377
• Verify Signature on page 378

**CredentialMapping**

API Exchange Gateway can map the credentials of the subject from the authenticated principal in the form of SAML assertion, or can map the username and password in the security header or the HTTP Authorization header by virtue of Credential mapping policies.

API Exchange Gateway supports following policies for credential mapping:

• Basic
• usernameToken
• SAML
• OAuth

See following policies:

• CredentialMappingByUsernameToken Policy on page 383
• CredentialMappingBySAML Policy on page 384
• Credential Mapping by OAuth Policy on page 385
Credentiaal Mapping By OAuth

API Exchange Gateway supports the credential mapping by OAuth policy. The policy generates the access token using the credentials configured in the policy. The credential mapping uses the OAuth password credential or client credential authorization flow to obtain the access token required to access the protected target operation, therefore, the previous authentication or authorization is not needed.
Manage Policies

This section explains the configuration setup required to manage the policies by API Exchange Gateway. The gateway allows you to configure various types of policies to support authentication, authorization, integrity, confidentiality and credential mapping.

API Exchange Gateway provides sample template policy files for all types of supported policies at the following location:

`ASG_CONFIG_HOME/default/policy`.

Table Sample Template Policies on page 365 lists the sample template policy files for each supported policy.

To manage policies in API Exchange Gateway product, you must do the following configuration setup:

- Define the shared resources. See Configure Shared Resource on page 361.
- Create policies for the intended usage. See Create Policy on page 363.
- Register the policies to the system. See Register Policy on page 366.
- Apply the policies to target operation. See Apply Policy on page 367.

You must perform the following steps to manage any policy:

Configure Shared Resource

You may configure an appropriate shared resource before you can create a policy. The table Types of Security Shared Resources, page 387 explains the types of shared resources supported by API Exchange Gateway product.

To configure a shared resource, perform the following steps:

Define Shared Resource Properties File

This section explains how to define the properties files required for the shared resource configuration.

TIBCO API Exchange Gateway provides the sample configuration file for the shared resources for each of the security type profile. It is recommended to use the sample files as templates and edit the properties as per your requirement. See Shared Resources Properties on page 388 for details of properties for each supported shared resource.
Sample Files

- The property files for various supported shared resources are located under:
  
  \texttt{ASG\_CONFIG\_HOME/default/resource}

- See Shared Resources Properties Sample Files on page 398 for sample file for each shared resource.

Register Shared Resource with API Exchange Gateway

The Shared Resources tab on the configuration allows you to register the shared resources with TIBCO API Exchange Gateway.

To configure a shared resource, perform the following steps:

1. Start the API Exchange Gateway user interface server, if not already started. See Starting GUI on page 166.
2. Create a new project or select an existing project under Projects.
3. Click SECURITY tab on the right hand side.
4. Click Shared Resources tab on top menu.
5. Enter the details for shared resource, as defined below:

\begin{table}
\centering
\caption{Shared Resource Name Configuration}
\begin{tabular}{|l|l|}
\hline
\textbf{Parameter} & \textbf{Description} \\
\hline
New Resources File & A new property file which defines the Shared Resources configuration. See Shared Resources Properties on page 388 for details \\
\hline
Existing Resources Files & Select a configuration file of Shared Resource configuration. See Shared Resources Properties on page 388 for details \\
\hline
\end{tabular}
\end{table}

- Save changes to the project.

- The shared resource name is defined in the properties file by the following property:

  \begin{verbatim}
  com.tibco.governance.sharedresource.name
  \end{verbatim}

  For example, the shared resource name is defined in the properties file as below:

  \begin{verbatim}
  com.tibco.governance.sharedresource.name=LdapAsp
  \end{verbatim}

- The shared resource name from the properties file must match with the shared resource name in the policy file defined by ResourceInstance property.

  \begin{verbatim}
  <tpa:SharedResourceLoginModule ResourceInstance="LdapAsp"/>
  \end{verbatim}
Create Policy

To create a policy, perform the following steps:

Define Shared Resource For a Policy

Before you create a policy, make sure that you have created the appropriate shared resource properties file for that policy. See Configure Shared Resource on page 361 for details.

You must define the correct shared resource for a specific policy. For example, you must define LDAP shared resource for username token authentication policy.

Following table lists the shared resource required for a specific policy.

Table 77 Policy And Shared Resource Property File

<table>
<thead>
<tr>
<th>Policy</th>
<th>Shared Resource</th>
<th>Shared Resource Property File (Resource Files)</th>
</tr>
</thead>
<tbody>
<tr>
<td>• UsernameToken authentication</td>
<td>LDAP Shared Resource</td>
<td>Properties for LDAP Authentication Shared Resource on page 388</td>
</tr>
<tr>
<td>• Basic authentication</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SAML Authentication</td>
<td>WSS Shared Resource</td>
<td>WssAsp.properties on page 401</td>
</tr>
<tr>
<td>Sign</td>
<td>Subject Shared Resource</td>
<td>Properties for Subject Identify Provider (SIP) on page 396</td>
</tr>
<tr>
<td>Decryption</td>
<td>Subject Shared Resource</td>
<td>Properties for Subject Identify Provider (SIP) on page 396</td>
</tr>
<tr>
<td>Verify Signature</td>
<td>Trust shared resource</td>
<td>Properties for Trust Identify Provider (TIP) on page 395</td>
</tr>
<tr>
<td>Encryption</td>
<td>Trust shared resource</td>
<td>Properties for Trust Identify Provider (TIP) on page 395</td>
</tr>
</tbody>
</table>
**Create Policy File**

You can create any supported policy file in the following ways:

a. Copy a sample template file from the following location:

   \texttt{ASG\_CONFIG\_HOME/default/policy}

b. Edit the parameters in the file as required. For example, you must change the \texttt{ResourceInstance} parameter to match with the shared resource name defined in the properties file.

   \texttt{ResourceInstance="LdapAsp"}.

   - Policy must be a well-formed WS policy.
   - All the resource instances in the policy must be defined in the shared resource properties file.

---

**Table 77  Policy And Shared Resource Property File**

<table>
<thead>
<tr>
<th>Policy</th>
<th>Shared Resource</th>
<th>Shared Resource Property File (Resource Files)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Credential Mapping</td>
<td>• UsernameToken - Password identity provider</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Keystore - Password identity provider</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• SAML - Subject Identity Provider</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• UsernameToken</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• SAML</td>
<td></td>
</tr>
</tbody>
</table>
Table 78  Sample Template Policies

<table>
<thead>
<tr>
<th>Policy Type</th>
<th>Template File</th>
</tr>
</thead>
<tbody>
<tr>
<td>Authentication</td>
<td>Basic Authentication Policy</td>
</tr>
<tr>
<td></td>
<td>BasicAuthentication.policy on page 369</td>
</tr>
<tr>
<td></td>
<td>Username token Authentication Policy</td>
</tr>
<tr>
<td></td>
<td>AuthenticationByUsernameToken.policy on page 370</td>
</tr>
<tr>
<td></td>
<td>SAML Authentication Policy</td>
</tr>
<tr>
<td></td>
<td>AuthenticationBySaml.policy on page 370</td>
</tr>
<tr>
<td></td>
<td>OAuth Authentication Policy</td>
</tr>
<tr>
<td></td>
<td>AuthenticationbyOAuth Policy</td>
</tr>
<tr>
<td></td>
<td>Authorization By Role Policy</td>
</tr>
<tr>
<td></td>
<td>Authorization By Role Policy on page 374</td>
</tr>
<tr>
<td>Confidentiality</td>
<td>Decryption policy</td>
</tr>
<tr>
<td></td>
<td>Decryption.policy on page 382</td>
</tr>
<tr>
<td></td>
<td>Encryption policy</td>
</tr>
<tr>
<td></td>
<td>Encryption.policy on page 381</td>
</tr>
<tr>
<td>Integrity</td>
<td>Sign policy</td>
</tr>
<tr>
<td></td>
<td>Sign.policy on page 377</td>
</tr>
<tr>
<td></td>
<td>VerifySignature policy</td>
</tr>
<tr>
<td></td>
<td>VerifySignature.policy on page 378</td>
</tr>
<tr>
<td>CredentialMapping</td>
<td>UsernameToken Credential Mapping</td>
</tr>
<tr>
<td></td>
<td>UsernameToken Credential Mapping on page 383</td>
</tr>
<tr>
<td></td>
<td>SAML Credential Mapping</td>
</tr>
<tr>
<td></td>
<td>SAML Credential Mapping on page 384</td>
</tr>
<tr>
<td></td>
<td>OAuth Credential Mapping</td>
</tr>
<tr>
<td></td>
<td>CredentialMappingByOAuth.policy on page 385</td>
</tr>
</tbody>
</table>
Register Policy

You can register a policy on the API Exchange Gateway user interface by uploading a policy file and set the name for a policy.

To register a policy, perform the following steps:

1. Start the gateway user interface, if not running.
2. Create a new project or select an existing project under Projects.
3. Click on SECURITY tab on the right hand side.
4. Click on Policy Mapping tab on the top menu.
5. Click Add Property icon to add a new policy mapping.
6. Enter the following parameters for the policy:

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Policy Name</td>
<td>Specifies the name for the policy. Required.</td>
</tr>
<tr>
<td>Type</td>
<td>Set the type of the policy. For example, Authentication.</td>
</tr>
<tr>
<td></td>
<td>See Policy Types and SubTypes on page 356.</td>
</tr>
<tr>
<td></td>
<td>See Types of Security Policies on page 357 for details.</td>
</tr>
<tr>
<td>SubType</td>
<td>Set the policy sub type. For example, UsernameToken</td>
</tr>
<tr>
<td></td>
<td>See Policy Types and SubTypes on page 356.</td>
</tr>
<tr>
<td></td>
<td>See Types of Security Policies on page 357 for details.</td>
</tr>
<tr>
<td></td>
<td>See Create Policy on page 363 to create the policy definition file.</td>
</tr>
<tr>
<td>Existing Policy File</td>
<td>Specifies an existing policy definition file. The policy file must exist in</td>
</tr>
<tr>
<td></td>
<td>the gateway ASG_CONFIG_HOME/configuration/policy folder. For example, for the</td>
</tr>
<tr>
<td></td>
<td>default configuration, the policy file must exist in the</td>
</tr>
<tr>
<td></td>
<td>ASG_CONFIG_HOME/default/policy folder.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>7. Save changes to the project or configuration.</td>
</tr>
</tbody>
</table>
Apply Policy

This section explains the steps to apply any registered policy to a target operation or reference endpoints. Policy Binding allows you to associate the policy with one or more target operation endpoints.

You can apply a policy on the API Exchange Gateway user interface by associating an existing policy to a target operation or reference endpoint as below:

To apply a policy, perform the following steps:

1. Start the API Exchange Gateway user interface, if not running.
2. Create a new configuration or select an existing configuration.
3. Click on SECURITY tab on the right hand side.
4. Click on Policy Binding tab on the top menu.
5. Click Add Property icon to add a new policy binding.
6. Enter the following parameters for the policy:

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Policy</td>
<td>Specifies a name for the policy. The policy name must be configured under Policy Mapping tab.</td>
</tr>
<tr>
<td>URI</td>
<td>Specifies the URI of the operation to which the policy is applied.</td>
</tr>
<tr>
<td>Transport</td>
<td>Specifies the transport of the request. For example, HTTP, JMS, or SOAP.</td>
</tr>
<tr>
<td>Facade Operation</td>
<td>Specifies the operation to which the policy is applied. The facade operation must be configured in the Facade Operations tab.</td>
</tr>
<tr>
<td>Binding</td>
<td>Specifies the binding component that the policy is applied to. This could be either a facade operation (service) or a target operation (reference).</td>
</tr>
<tr>
<td>Flow</td>
<td>Specifies the flow of the request. Possible values are \texttt{in} or \texttt{out}.</td>
</tr>
<tr>
<td>Target Operation</td>
<td>Specifies the target operation. The target operation must be configured in the Target Operations tab.</td>
</tr>
</tbody>
</table>

7. Save changes to the project or configuration.
Policy Use Cases

This section describes the use cases for supported policies.

Authentication Policies

Figure Authentication Policies on page 368 illustrates how to apply the authentication policies for an incoming request.

Figure 20  Authentication Policies

To configure the authentication policies, do the following tasks:

Task D Configure Shared Resource

See Configure Shared Resource on page 361 for details.

Task E Create Policy

See Create Policy on page 363 for details.
Task F  Register Policy

See Register Policy on page 366 for detailed steps. You must choose the correct type and subtype to register the policy as shown in the following table:

<table>
<thead>
<tr>
<th>Policy</th>
<th>Type</th>
<th>SubType</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic Authentication</td>
<td>Authentication</td>
<td>Basic</td>
</tr>
<tr>
<td>Username Token</td>
<td>Authentication</td>
<td>UsernameToken</td>
</tr>
<tr>
<td>SAML Authentication</td>
<td>Authentication</td>
<td>SAML</td>
</tr>
<tr>
<td>Authentication by OAuth</td>
<td>Authentication</td>
<td>OAuth</td>
</tr>
</tbody>
</table>

Task G  Apply Policy

See Apply Policy on page 367 for details.

API Exchange Gateway supports following authentication policies:

Basic
  - The client sends the request with user name/password in HTTP basic authentication header.
  - Basic Authentication policy authenticates the request against LDAP.

Example Policy  BasicAuthentication.policy

```xml
<wsp:Policy
xmlns:wssp="http://docs.oasis-open.org/ws-sx/ws-securypolicy/200702">
  <wsp:All>
    <tpa:AuthenticationByJaas>
      <wssp:SupportingTokens>
        <tpa:Any>
          <wssp:HttpBasicAuthentication/>
        </tpa:Any>
      </wssp:SupportingTokens>
    </tpa:AuthenticationByJaas>
  </wsp:All>
</wsp:Policy>
```

TIBCO API Exchange Gateway User's Guide
UsernameToken

- The client sends the request containing usernameToken in WS-Security header of the SOAP message.
- UsernameToken authentication policy authenticates the request against LDAP.

Example Policy  
**AuthenticationByUsernameToken.policy**

```xml
xmlns:wssp="http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702"
  <wsp:All>
    <wsp:ExactlyOne>
      <tpa:AuthenticationByJaas>
        <wssp:SupportingTokens>
          <tpa:ExactlyOne>
            <wssp:UsernameToken />
          </tpa:ExactlyOne>
        </wssp:SupportingTokens>
        <tpa:SharedResourceLoginModule ResourceInstance="LdapAsp" />
      </tpa:AuthenticationByJaas>
    </wsp:ExactlyOne>
  </wsp:All>
</wsp:Policy>
```

SAML

The client sends the request containing a SAML assertion in WS-Security header of the SOAP message.

Example Policy  
**AuthenticationBySaml.policy**

```xml
xmlns:wssp="http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702">
  <wsp:All>
    <wsp:ExactlyOne>
      <tpa:AuthenticationByJaas>
        <wssp:SupportingTokens>
          <tpa:ExactlyOne>
            <wssp:UsernameToken />
          </tpa:ExactlyOne>
        </wssp:SupportingTokens>
        <tpa:SharedResourceLoginModule ResourceInstance="LdapAsp" />
      </tpa:AuthenticationByJaas>
    </wsp:ExactlyOne>
  </wsp:All>
</wsp:Policy>
```
OAuth

When a request is received from the user, the gateway redirects the user to OAuth Authorization server to login and grant access to the protected facade operation.
When the OAuth authentication server sends the authorization code back to the policy callback endpoint after the user’s successful login and grant access to the facade operation, the gateway exchanges the authorization code for an access token from the authorization server. Since the protected facade operation does not access user’s resources, therefore, the access token is only used for authentication purposes.

**Example Policy**

**Authentication by OAuth Policy**

```xml
<?xml version="1.0" encoding="UTF-8"?>
    <ns:OAuthToken>
      <ns:ClientID>644768512218118</ns:ClientID>
      <ns:ClientSecret>3e66e0dacf8addae262832d8f83fb53</ns:ClientSecret>
      <ns:CallbackURI>http://localhost:4080/asg/oauth/client/callback</ns:CallbackURI>
      <ns:Provider>Facebook</ns:Provider>
    </ns:OAuthToken>
  </ns:AuthenticationByJaas>
</wsp:Policy>
```

**Authorization Policies**

Figure **Authorization Policies on page 373** illustrates how to apply an authorization policy based on the role or how you can apply an OAuth policy.
Configuration

To configure a authorization policies, do following tasks:

**Task A Configure Shared Resource.**

See Configure Shared Resource on page 361 for details.

**Task B Create Policy**

See Create Policy on page 363 for details.

**Task C Register Policy**

See Register Policy on page 366 for detailed steps. You must choose the correct type and subtype to register the policy as shown in the following table:

<table>
<thead>
<tr>
<th>Policy</th>
<th>Type</th>
<th>SubType</th>
</tr>
</thead>
<tbody>
<tr>
<td>Authorization By Role</td>
<td>Authorization</td>
<td>Role</td>
</tr>
</tbody>
</table>

**Task D Apply Policy**

See Apply Policy on page 367 for details.

API Exchange Gateway supports the following authorization policies:
Role

When an authorization policy by role is applied, an authenticated user with a specific role defined by the policy will be authorized to access all the functions of the target operation.

- Consumer sends a request with username and password in HTTP header or as a UsernameToken in WS-Security header of the SOAP message.
- Basic or UsernameToken authentication policy authenticates the request against LDAP and retrieves LDAP attributes/roles for the user.
- After authentication policy, the gateway invokes an authorization policy. Using SOAP operation information from SOAP request and user role information retrieved from LDAP during authentication, the authorization policy determines if the user that is sending the request is authorized to invoke the SOAP operation.

Example Policy  Authorization By Role Policy

```xml
<?xml version="1.0" encoding="UTF-8"?>
<wsp:Policy
xmlns:wssp="http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702"
xmlns:xacml="urn:oasis:names:tc:xacml:2.0:policy:schema:os"
xmlns:sp="http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702">
  <wsp:All>
    <wsp:Policy>
      <wsp:All>
        <tpa:Authorization>
          <tpa:ByRole>
            <tpa:Default>
              <xacml:Rule Effect="Deny" RuleId="">
                <xacml:Condition>
                  <xacml:Apply FunctionId="always-true">
                </xacml:Apply>
              </xacml:Condition>
            </tpa:Default>
            <tpa:Operation>
              <xacml:Rule Effect="Permit" RuleId="">
                <xacml:Target>
                  <xacml:Actions>
                    <xacml:Action>
                      <xacml:ActionMatch MatchId="http://tempuri.org">
                        <!--
                        <xacml:AttributeValue
                          DataType="xsd:string">GetBooks</xacml:AttributeValue>

                        
                        
                        -->
                      </xacml:ActionMatch>
                    </xacml:Action>
                  </xacml:Actions>
                </xacml:Target>
              </xacml:Rule>
            </tpa:Operation>
          </tpa:ByRole>
        </tpa:Authorization>
      </wsp:All>
    </wsp:Policy>
  </wsp:All>
</wsp:Policy>
```
Integrity Policies

Figure Sign Policy on page 375 shows how to apply a sign policy.

Figure 22  Sign Policy
Figure VerifySignature Policy on page 376 shows how to apply a verify signature policy:

Figure 23 VerifySignature Policy

Configuration

To configure the integrity policies, do the following tasks:

Task A Configure Shared Resource

See Configure Shared Resource on page 361 for details.
Task B  Create Policy

See Create Policy on page 363 for details.

Task C  Register Policy

See Register Policy on page 366 for detailed steps. You must choose the correct type and subtype to register the policy as shown in the following table:

<table>
<thead>
<tr>
<th>Policy</th>
<th>Type</th>
<th>SubType</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sign</td>
<td>Integrity</td>
<td>Sign</td>
</tr>
<tr>
<td>Verify Signature</td>
<td>Integrity</td>
<td>Verify</td>
</tr>
</tbody>
</table>

Task D  Apply Policy

See Apply Policy on page 367 for details.

API Exchange Gateway supports following types of integrity policies:

Sign

- Sign policy can be used to sign the outgoing SOAP message. Before a request is forwarded, the signature policy is applied.
- The message is signed using the shared resource specified in the policy.

Example Policy  Sign.policy

```xml
<wsp:Policy
<wsp:All>
<wsp:Policy
xmlns:ns1="http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702">
<ns:Signature
ResourceInstance="SubjectIsp">
<ns1:SignedParts >SubjectIsp</ns1:SignedParts>
<ns1:Body />
<ns1:Header />
</ns1:SignedParts>

<!-- The keyAlias should be replaced with a valid one from the keystore from the SubjectIsp -->
<ns:keyAlias>john_key</ns:keyAlias>
```
<!-- Change the AlgorithmSuite to use a different encryption algorithm -->
<ns1:AlgorithmSuite>
  <ns1:Basic128 />
</ns1:AlgorithmSuite>
</ns:Signature>
</wsp:Policy>
</wsp:All>
</wsp:Policy>

Verify Signature

When a signed request is received by the gateway, the verify signature policy is applied.

- The signature in the message is verified using the shared resource specified in the policy.
- The policy verifies that there is a signature in the message and it has been verified.

Example Policy VerifySignature.policy

<wsp:Policy

xmlns:wssp="http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702">
  <wsp:All>
    <wsp:Policy>
      <tpa:WssProcessor ResourceInstance="WssAsp"/>
    </wsp:Policy>
  </wsp:Policy>
  <wsp:Policy>
    <tpa:VerifyAuthentication>
      <wssp:SignedSupportingTokens>
        <wssp:SamlToken />
      </wssp:SignedSupportingTokens>
    </tpa:VerifyAuthentication>
  </wsp:Policy>
  <wsp:Policy>
    <tpa:VerifySignature>
      <wssp:SignedParts>
        <wssp:Header
          Namespace="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd"/>
        <wssp:Body />
      </wssp:SignedParts>
    </tpa:VerifySignature>
  </wsp:Policy>
</wsp:Policy>
Confidentiality Policies

Figure Encryption Policy on page 379 shows how to apply an encrypt policy.

Figure 24  Encryption Policy

Figure Decrypt Policy on page 380 shows how to apply the decrypt policy.
Configuration

To configure the confidentiality policies, do the following tasks:

**Task A Configure Shared Resource**
See Configure Shared Resource on page 361 for details.

**Task B Create Policy**
See Create Policy on page 363 for details.

**Task C Register Policy**
See Register Policy on page 366 for detailed steps. You must choose the correct type and subtype to register the policy as shown in the following table:

<table>
<thead>
<tr>
<th>Policy</th>
<th>Type</th>
<th>SubType</th>
</tr>
</thead>
<tbody>
<tr>
<td>Encrypt</td>
<td>Confidentiality</td>
<td>Encrypt</td>
</tr>
<tr>
<td>Decrypt</td>
<td>Confidentiality</td>
<td>Decrypt</td>
</tr>
</tbody>
</table>
Task D  Apply Policy

See Apply Policy on page 367 for details.

API Exchange Gateway supports following types of confidentiality policies:

Encryption

- Before a request is forwarded the Encryption policy is applied.
- The message is encrypted using the shared resource specified in the policy.

Example Policy Encryption.policy

```xml
<wsp:Policy
<wsp:All>  
<wsp:Policy
xmlns:ns1="http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702">  
<ns:Encryption
ResourceInstance="TrustIsp">  
<ns1:EncryptedParts>  
<ns1:Body />  
<ns1:Header />  
</ns1:EncryptedParts>

<!-- The keyAlias should be replaced with a valid one from the
keystore from the TipIsp -->  
<ns:keyAlias>john_key</ns:keyAlias>

<!-- Change the AlgorithmSuite to use a different encryption
algorithm -->  
<ns1:AlgorithmSuite>  
<ns1:Basic128 />  
</ns1:AlgorithmSuite>  
</ns:Encryption>  
</wsp:Policy>  
</wsp:All>  
</wsp:Policy>
```

Decryption

- When a request is received, the decryption policy is applied.
- The message is decrypted using the shared resource specified in the policy. The policy verifies that the message is decrypted.
### Example Policy: Decryption.policy

```xml
<wsp:Policy
xmlns:wssp="http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702">
  <wsp:All>
    <wsp:Policy>
      <tpa:WssProcessor ResourceInstance="WssAsp"/>
    </wsp:Policy>
    <wsp:Policy>
      <tpa:VerifyAuthentication>
        <wssp:SignedSupportingTokens>
          <wssp:SamlToken />
        </wssp:SignedSupportingTokens>
      </tpa:VerifyAuthentication>
    </wsp:Policy>
    <wsp:Policy>
      <tpa:VerifyDecryption/>
    </wsp:Policy>
  </wsp:All>
</wsp:Policy>
```

### Credential Mapping Policies

Figure CredentialMapping Policies on page 383 illustrates how you can apply a credential mapping policy.
API Exchange Gateway supports following types of credential mapping policies:

**UsernameToken Credential Mapping**

For UsernameToken Credential Mapping:

- There is no authentication needed.
- The client sends request to the facade operation.
- When the service invoked by the client request calls the target operation, UsernameToken credential mapping policy is applied. A UsernameToken is then added to the outgoing request using the credentials extracted from the policy or the shared resource specified in the policy.

**Example Policy**

```
<wsp:Policy
xmlns:wssp="http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702"
>
<wsp:All>
<wsp:Policy
<ns:CredentialMapping
```
SAML Credential Mapping

For SAML Credential Mapping:

- The client sends request as a UsernameToken in WS-Security header of the SOAP message.
- UsernameToken authentication policy authenticates the request against LDAP and retrieves LDAP attributes/roles for the user.
- When the service invoked by the client request calls any external service and forwards the outgoing request, SAML credential mapping policy is applied. The SAML assertion generated from the previous UsernameToken authentication is added to the outgoing request.

Example Policy  CredentialMappingBySAML Policy

```xml
<wsp:Policy
<wsp:All>
```

<wsp:Policy
.ns:CredentialMapping ResourceInstance="SubjectIsp"

<ns:Saml>
<ns:WSS>
<ns:IssuerName>urn:kimyou.tibco.com</ns:IssuerName>
<ns:ValidPeriod>300</ns:ValidPeriod>
</ns:WSS>
</ns:Saml>
<wssp:SignedSupportingTokens
xmlns:wssp="http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702">
<wssp:SamlToken>
<wssp:IssuerName>urn:www.example.com</wssp:IssuerName>
<wssp:WssSamlV20Token11 />
</wssp:SamlToken>
</wssp:SignedSupportingTokens>
</ns:CredentialMapping>
</wsp:Policy>
</wsp:All>
</wsp:Policy>

**Credential Mapping by OAuth Policy**

For Credential Mapping by OAuth:

- The client sends request to the facade operation.
- When the service invoked by the client request calls the target operation, OAuth credential mapping policy is applied. An access token is obtained using either client credential or owner credential from the token endpoint of the Authorization server. It then propagate the access token to the target operation by adding the access token to the query string when calling the target operation.

**Example Policy**  **CredentialMappingByOAuth.policy**
<wsp:Policy
<wsp:CredentialMapping
<ns:OAuth>
<ns:Provider>Facebook</ns:Provider>
<ns:ClientID>374219866025950</ns:ClientID>
<ns:ClientSecret>69d82cd674baf69ab73472b8f7f98995</ns:ClientSecret>
<ns:CallbackURI>://sample.acme.com/ProtectedAPI</ns:CallbackURI>
<ns:Profile>User-Agent</ns:Profile>
<ns:Grant>Password</ns:Grant>
<ns:Username>username</ns:Username>
<ns:Password>PASSWORD</ns:Password>
<ns:Parameter>Header</ns:Parameter>
</ns:OAuth>
</ns:CredentialMapping>
</wsp:Policy>
Types of Security Shared Resources

Table Types of Security Shared Resources on page 387 lists the types of shared resources types used by different policies. You must use an appropriate shared resource properties file to create the policy. See Policy And Shared Resource Property File on page 363.

For example, to create authentication policy to authenticate an username against LDAP server, you must register the LDAP shared resource property file.

Table 80  Types of Security Shared Resources

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LDAP</td>
<td>LDAP authentication shared resource provides the ability to authenticate a username and password against an LDAP server.</td>
</tr>
</tbody>
</table>
| Trust Identity      | The Trust Identity Provider is used for retrieving certificates required for performing trust operations from a credential store.  
For example, use Trust identity provider (TIP) for verifying signature or encryption and SSL client authentication. |
| Subject Identity    | The Subject Identity Provider is used for retrieving and using private credentials obtained from a credential store.  
For example, use Subject identity provider (SIP) for signing or decryption. |
| WSS                 | WS security authentication provider is used as a combination of LDAP, Trust Identity Provider(TIP), and Subject Identity Provider(SIP). |

- WSS shared resource is a combination of LDAP authentication, Trust Identity and Subject Identity Providers. Depending on the usage of shared resource, WSS can be configured to include one or more types of shared resource that it is used for.
- Trust Identity Provider (TIP) and Subject Identity Provider (SIP) depends on Keystore Credential Provider (KCP), so TIP and SIP always include an associated KCP.
Shared Resources Properties

This section explains the properties of supported shared resources.

Configuring LDAP Authentication Shared Resource

- **Description**: The LDAP authentication shared resource is used to authenticate the user name and password against the LDAP server. The user name is specified as the usernameToken in the incoming request from the client.

- **Use Case**: Verifying usernameToken in the incoming request.

- **Properties**: Table Properties for LDAP Authentication Shared Resource on page 388 describes the properties for LDAP Authentication Shared Resource.

### Table 81 Properties for LDAP Authentication Shared Resource

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>com.tibco.asg.intent.usernameToken</td>
<td>Boolean intent property indicates if the LDAP authentication method can be enforced on the request message or not. Possible values are <code>true</code> or <code>false</code>. If the value of this property set to <code>true</code>, the request message must contain a valid username token.</td>
</tr>
<tr>
<td>com.tibco.trinity.runtime.core.provider.authn.ldap.initialCtxFactory</td>
<td>Specifies the name of the JNDI Factory to use. Default value is <code>com.sun.jndi.ldap.LdapCtxFactory</code> (Sun’s LdapCtxFactory). Optional.</td>
</tr>
<tr>
<td>com.tibco.trinity.runtime.core.provider.authn.ldap.serverURL</td>
<td>Specifies the URL to connect to the LDAP directory server. The LDAP URL is defined as: <code>ldap://hostname:port</code>. The LDAP SSL URL is defined as: <code>ldaps://hostname:port</code>. Required.</td>
</tr>
</tbody>
</table>
Table 81 Properties for LDAP Authentication Shared Resource

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>com.tibco.trinity.runtime.core.provider.authn.ldap.searchTimeOut</td>
<td>The time (in milliseconds) to wait for a response from the LDAP directory server. A value of 0 causes it to wait indefinitely. If a negative number is specified, it uses the provider's default setting.</td>
</tr>
<tr>
<td></td>
<td>Optional.</td>
</tr>
<tr>
<td>com.tibco.trinity.runtime.core.provider.authn.ldap.userAttributeUsersName</td>
<td>The name of the attribute in the user object that represents the user's name. The value depends on what LDAP server is used. If you are use ActiveDirectory LDAP Server, set this value as CN. If SunOne or OpenLDAP LDAP Server is used, set this value as uid.</td>
</tr>
<tr>
<td></td>
<td>Optional.</td>
</tr>
<tr>
<td>com.tibco.trinity.runtime.core.provider.authn.ldap.userAttributesExtra</td>
<td>Specifies the optional list of user attributes to retrieve from the LDAP directory during authentication. Separation characters for the list of user attributes are comma, any ASCII whitespace or semicolon.</td>
</tr>
<tr>
<td></td>
<td>For example, mail givenname</td>
</tr>
<tr>
<td></td>
<td>Optional.</td>
</tr>
<tr>
<td>com.tibco.trinity.runtime.core.provider.authn.ldap.userSearchBaseDN</td>
<td>Specifies the base distinguished name (DN) where the searches for the users begin. You must supply the base DN that narrows the search to the smallest set of objects that includes all valid users. This is relevant only when used with administrator's credentials in search mode.</td>
</tr>
<tr>
<td></td>
<td>For example, ou=people, ou=na, dc=example, dc=org</td>
</tr>
<tr>
<td></td>
<td>Required in admin (search) mode.</td>
</tr>
</tbody>
</table>
### Properties for LDAP Authentication Shared Resource

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>com.tibco.trinity.runtime.core.provider.authn.ldap.searchTimeOut</td>
<td>The time (in milliseconds) to wait for a response from the LDAP directory server. A value of 0 causes it to wait indefinitely. If a negative number is specified, it uses the provider's default setting. Optional.</td>
</tr>
<tr>
<td>com.tibco.trinity.runtime.core.provider.authn.ldap.userAttributeUsersName</td>
<td>The name of the attribute in the user object that represents the user's name. The value depends on what LDAP server is used. If you are use ActiveDirectory LDAP Server, set this value as CN. If SunOne or OpenLDAP LDAP Server is used, set this value as uid. Optional.</td>
</tr>
<tr>
<td>com.tibco.trinity.runtime.core.provider.authn.ldap.userAttributesExtra</td>
<td>Specifies the optional list of user attributes to retrieve from the LDAP directory during authentication. Separation characters for the list of user attributes are comma, any ASCII whitespace or semicolon. For example, mail givenname Optional.</td>
</tr>
<tr>
<td>com.tibco.trinity.runtime.core.provider.authn.ldap.userSearchBaseDN</td>
<td>Specifies the base distinguished name (DN) where the searches for the users begin. You must supply the base DN that narrows the search to the smallest set of objects that includes all valid users. This is relevant only when used with administrator's credentials in search mode. For example, ou=people,ou=na,dc=example,dc=org Required in admin (search) mode.</td>
</tr>
</tbody>
</table>
Table 81  Properties for LDAP Authentication Shared Resource

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>com.tibco.trinity.runtime.core.provider.authn.ldap.userSearchExpression</td>
<td>Specifies the expression to be used for searching in admin mode against potential user objects. For example, search expression is specified as: $(uid={0})(objectClass=person)$. In this string, the variable {0} represents the name of the user. The code substitutes the user name for this variable, and passes the resulting boolean expression to the LDAP server. The LDAP server matches that search expression against user objects to find a match. The search result must contain exactly one match. This property is relevant only when credentialProvider property is set and the binding is done as administrator; otherwise userDNTemplate is used. Required in admin (search) mode.</td>
</tr>
<tr>
<td>com.tibco.trinity.runtime.core.provider.authn.ldap.userDNTemplate</td>
<td>Specifies a template to be used when formatting user’s DN before binding. It is used as an alternative to admin (search) mode. For example, uid={0}, ou=employee, ou=tsi, o=tibco Required for bind mode (not in admin (search) mode).</td>
</tr>
<tr>
<td>com.tibco.trinity.runtime.core.provider.authn.ldap.userAttributeGroupsName</td>
<td>If you specified &quot;LDAP user indicates groups&quot; (as either userHasGroups or userDNHasGroups) then you must supply the name of the attribute in each user object that lists the groups to which the user belongs. Otherwise, this parameter is not relevant. Mandatory when relevant.</td>
</tr>
<tr>
<td>com.tibco.trinity.runtime.core.provider.authn.ldap.userAttributesExtraList</td>
<td>Same as userAttributesExtra property but this is specified in list form. Optional.</td>
</tr>
<tr>
<td>com.tibco.trinity.runtime.core.provider.authn.ldap.userSearchScopeSubtree</td>
<td>A boolean property which determines if the entire sub-tree is searched or not. If true value is specified, the entire sub-tree starting at the base DN is searched. Otherwise, the nodes one level below the base DN is searched. Optional.</td>
</tr>
</tbody>
</table>
### Properties for LDAP Authentication Shared Resource

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
</table>
| com.tibco.trinity.runtime.core.provider.authn.ldap.groupSearchBaseDN     | Specifies the base distinguished name (DN) where the searches for the groups begin. You must supply the base DN that narrows the search to the smallest set of objects that includes all valid groups. For example, ou=groups,ou=na,dc=example,dc=org  
Default value is empty.                                                                 Optional.                                                                                           |
| com.tibco.trinity.runtime.core.provider.authn.ldap.enableNestedGroupSearch | Indicates the flag to determine if nested groups should be searched for. If the value is not set to true, the groups are only returned in which the user is the direct member.  
Default value is false.                                                                 Optional.                                                                                           |
| com.tibco.trinity.runtime.core.provider.authn.ldap.groupSearchExpression  | Specifies the expression to be used for searching against potential groups. For example, search expression is specified as: (&(uid={0})(objectClass=person)).  
In this string, the variable {0} represents the name of the user though. The code substitutes the user name for this variable, and passes the resulting boolean expression to the LDAP server. The LDAP server matches that search expression against groups to find all groups containing the username.  
The values might be different for different LDAP server.  
For example, its defined as uniquemember={0} for SunOne, cn={0} for OpenLDAP, member={0} for Active Directory.  
Required.                                                                                             |
| com.tibco.trinity.runtime.core.provider.authn.ldap.groupSearchScopeSubtree | A boolean property which determines if the entire sub-tree is searched or not. If true value is specified, the entire sub-tree starting at the base DN for groups is searched. Otherwise, the nodes one level below the base DN is searched.  
Optional.                                                                                           |
### Table 81 Properties for LDAP Authentication Shared Resource

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
</table>
| com.tibco.trinity.runtime.core.provider.authn.ldap.groupIndication | Specifies how the group memberships for users are found. Default value is noGroupInfo. Optional. Possible values are as below:  
- userHasGroups  
- userDNHasGroups  
- groupHasUsers  
- noGroupInfo  
  - If the value has userHasGroups, you must specify the attribute name which points the groups the user belongs to in the userAttributeGroupsName property.  
  - If the value has userDNHasGroups, the userAttributeGroupsName property has the attribute name which hold the DNs of groups to which the user belongs. You must specify groupAttributeGroupsName property to get a specific part of the DN name.  
  - If the value has groupHasUsers, each group object includes a list of users that belong to the group.  
  - If the value has noGroupInfo, group memberships aren't handled. |
| com.tibco.trinity.runtime.core.provider.authn.ldap.groupAttributeGroupsName | Depending on groupIndication’s value:  
  - groupHasUsers: group attribute holding the group’s name. Example value for OpenLDAP: cn, for Active Directory: sAMAccountName. Mandatory  
  - userHasGroups: group's name part holding group's name. If not specified the group’s whole DN will be used. Example cn  
    otherwise ignored |
| com.tibco.trinity.runtime.core.provider.authn.ldap.groupAttributeSubgroupsName | Specifies the name of the attribute in each group object denoting subgroups. For example, the value is defined as uniqueMember for OpenLDAP server, member for ActiveDirectory LDAP server. Optional |
Table 81  Properties for LDAP Authentication Shared Resource

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>com.tibco.trinity.runtime.core.provider.authn.ldap.groupAttributeUsersName</td>
<td>Specifies the attribute name if the groupIndication property has groupHasUsers value. It specifies the name of the attribute in each group object denoting its users. For example, the value is uniqueMember for OpenLDAP, member for ActiveDirectory Server. Required if the groupIndication property has groupHasUsers value.</td>
</tr>
<tr>
<td>followReferrals</td>
<td>Determines if the client follow referrals returned by the LDAP server. Default value is false. Optional.</td>
</tr>
<tr>
<td>com.tibco.trinity.runtime.core.provider.identity.trust.trustStoreServiceProvider</td>
<td>Specifies the Identity trust provider configuration to provide SSL support for LDAP</td>
</tr>
<tr>
<td>com.tibco.trinity.runtime.core.provider.credential.keystore.truststore.keyStoreLocation</td>
<td>Specifies the location of the keystore for the credentials.</td>
</tr>
<tr>
<td>com.tibco.trinity.runtime.core.provider.credential.keystore.truststore.keyStorePassword</td>
<td>Specifies the location of the keystore for the credentials.</td>
</tr>
<tr>
<td>com.tibco.trinity.runtime.core.provider.credential.keystore.truststore.keyStoreRefreshInterval</td>
<td>Specifies the refresh interval (milliseconds).</td>
</tr>
<tr>
<td>com.tibco.trinity.runtime.core.provider.credential.keystore.truststore.keyStoreType</td>
<td>Specifies the keystore type. Supported formats are JKS,PKCS12.</td>
</tr>
</tbody>
</table>

Sample File  The properties and example configuration for LDAP authentication shared resource is provided in the following sample file:
Configuring Trust Identity Provider

**Description**
The Trust Identity Provider is used to retrieve public certificates from a credential store required to perform trust operations. You must store the public certificate and provide its location. The certificates are used by the gateway core engine to verify the signatures when the payload in the incoming request is signed. The gateway core engine uses the public certificate to encrypt the response payload before it sends the response back to the client.

**Use Case**
- Verify signatures for the signed request payload.
- Encrypt the request payload to forward to the external target operation.
- Encrypt the response payload.

**Properties**
Table Properties for Trust Identity Provider (TIP) on page 395 describes the properties for Trust Identity Provider.

*Table 82 Properties for Trust Identity Provider (TIP)*

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>com.tibco.asg.intent.signature</code></td>
<td>Boolean intent property indicates if the incoming request message is signed or not. If signed, then the signatures are verified using the trust identity provider properties (public credentials). Possible values are true or false. If the value of this property set to true, the request message must have valid signatures.</td>
</tr>
<tr>
<td><code>com.tibco.trinity.runtime.core.provider.identity.trust.trustStoreServiceProvider</code></td>
<td>Specifies the name of the credential service provider containing the credentials for establishing trust.</td>
</tr>
<tr>
<td><code>com.tibco.trinity.runtime.core.provider.credential.keystore.truststore.keyStoreType</code></td>
<td>Specifies the keystore type. Supported formats are JKS,PKCS12.</td>
</tr>
<tr>
<td><code>com.tibco.trinity.runtime.core.provider.credential.keystore.truststore.keyStoreLocation</code></td>
<td>Specifies the location(s) of the keystore.</td>
</tr>
</tbody>
</table>

- See ASG_CONFIG_HOME/default/resource/LdapAsp.properties, as below:

LdapAsp.properties on page 398
Table 82 Properties for Trust Identify Provider (TIP)

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>com.tibco.trinity.runtime.core.provider.credential.keystore.truststore.keyStorePassword</td>
<td>Specifies the password to unlock the keystore.</td>
</tr>
<tr>
<td>com.tibco.trinity.runtime.core.provider.credential.keystore.truststore.keyStoreRefreshInterval</td>
<td>Specifies the refresh interval (milliseconds).</td>
</tr>
</tbody>
</table>

Sample File

- See ASG_CONFIG_HOME/default/resource/TrustIsp.properties, as below:
  TrustIsp.properties on page 400

Configuring Subject Identity Provider

Description

The Subject Identity Provider is used to retrieve private keys (credentials) from a credential store. You must store the private keys and provide its location. The private keys are used by the gateway core engine to decrypt the message when the payload in the incoming request is encrypted. The gateway uses the private keys to sign the response message before sending it back to the client.

Use Case

- Decrypt the request payload.
- Sign the request message to forward to any external target operation.
- Sign the response payload.

Properties

Table Properties for Subject Identify Provider (SIP) on page 396 describes the properties for Subject Identify Provider.

Table 83 Properties for Subject Identify Provider (SIP)

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>com.tibco.asg.intent.decrypt</td>
<td>Boolean intent property indicates if the incoming request message is encrypted or not. If encrypted, then the request message payload is decrypted using the subject identity provider properties (private credentials). Possible values are true or false. If the value of this property set to true, the request message must be encrypted.</td>
</tr>
</tbody>
</table>
### Table 83  Properties for Subject Identify Provider (SIP)

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>com.tibco.trinity.runtime.core.provider.identity.subject.identityStoreServiceProvider</td>
<td>Specifies the name of the credential service provider containing the private credentials for establishing the subject's identity.</td>
</tr>
<tr>
<td>com.tibco.trinity.runtime.core.provider.identity.subject.keyAlias</td>
<td>Specifies an alias name for the key corresponding to the private credentials in the credential store for establishing the subject's identity.</td>
</tr>
<tr>
<td>com.tibco.trinity.runtime.core.provider.identity.subject.keyPassword</td>
<td>Specifies the protection parameter of the private credentials in the credential store for establishing the subject's identity.</td>
</tr>
<tr>
<td>com.tibco.trinity.runtime.core.provider.credential.keystore.keyStoreType</td>
<td>Specifies the keystore type of the private credentials.</td>
</tr>
<tr>
<td>com.tibco.trinity.runtime.core.provider.credential.keystore.keyStoreLocation</td>
<td>Specifies the location(s) of the keystore of the private credentials.</td>
</tr>
<tr>
<td>com.tibco.trinity.runtime.core.provider.credential.keystore.keyStorePassword</td>
<td>Specifies the password to unlock the keystore.</td>
</tr>
<tr>
<td>com.tibco.trinity.runtime.core.provider.credential.keystore.keyStoreRefreshInterval</td>
<td>Specifies the refresh interval in milliseconds.</td>
</tr>
</tbody>
</table>

**Sample File**  
- See ASG_CONFIG_HOME/default/resource/SubjectIsp.properties, as below:
  
  ```ini
  SubjectIsp.properties on page 399
  ```
Shared Resources Properties Sample Files

This section lists the sample files for different types of security shared resources.

LdapAsp.properties

com.tibco.trinity.runtime.core.provider.lookup=com.tibco.trinity.runtime.core.provider.authn.ldap
com.tibco.governance.sharedresource.name=LdapAsp
com.tibco.governance.sharedresource.type=LdapConfiguration

#Example configuration where incoming message must contain a valid username token.
com.tibco.asg.intent.usernameToken=true

#WSS Authn Namespace
com.tibco.trinity.runtime.core.provider.authn.wss.usernameTokenValidationService=class:com.tibco.trinity.runtime.core.provider.authn.ldap
com.tibco.trinity.runtime.core.provider.authn.wss.samlValiditySeconds=60
com.tibco.trinity.runtime.core.provider.authn.wss.enableSAML11Assertion=false

#LDAP namespace. Used to verify user name token. This configuration is for search mode.
com.tibco.trinity.runtime.core.provider.authn.ldap.securityAuthentication=simple
com.tibco.trinity.runtime.core.provider.authn.ldap.initialCtxFactory=com.sun.jndi.ldap.LdapCtxFactory
com.tibco.trinity.runtime.core.provider.authn.ldap.userDNTemplate=uid={0},ou=people,dc=policy,dc=tibco,dc=com
com.tibco.trinity.runtime.core.provider.authn.ldap.userAttributeUsersName=uid
com.tibco.trinity.runtime.core.provider.authn.ldap.userAttributesExtra=mail,givenname
com.tibco.trinity.runtime.core.provider.authn.ldap.userSearchScopeSubtree=true
com.tibco.trinity.runtime.core.provider.authn.ldap.keyPassword=#{fGgMyESTOe58y1Qe77yskDyhfWq9mjkMVssHsNAC4=
com.tibco.trinity.runtime.core.provider.authn.ldap.keyAlias=uid=Manager,ou=people,dc=example,dc=org

# Group configuration
# For LDAP that uses group to find list of users that belong to the group.
com.tibco.trinity.runtime.core.provider.authn.ldap.groupIndication=groupHasUsers
# For LDAP that uses user to find list of groups to which the user belongs.
#com.tibco.trinity.runtime.core.provider.authn.ldap.groupIndication=userHasGroups
# For LDAP user's DN as group, use
#com.tibco.trinity.runtime.core.provider.authn.ldap.groupIndication=userDNHasGroups

com.tibco.trinity.runtime.core.provider.authn.ldap.groupSearchExpression=uniquemember={0}
com.tibco.trinity.runtime.core.provider.authn.ldap.groupSearchBaseDN=ou=groups,dc=policy,dc=tibco,dc=com
com.tibco.trinity.runtime.core.provider.authn.ldap.groupSearchScopeSubtree=true
com.tibco.trinity.runtime.core.provider.authn.ldap.groupAttributeGroupsName=cn
com.tibco.trinity.runtime.core.provider.authn.ldap.groupAttributeUsersName=cn
com.tibco.trinity.runtime.core.provider.authn.ldap.groupIndication=groupHasUsers
com.tibco.trinity.runtime.core.provider.authn.ldap.enableNestedGroupSearch=true

# Credential provider configuration to provide details for ldap admin user/admin password.
com.tibco.trinity.runtime.core.provider.authn.ldap.credentialProvider=class:com.tibco.trinity.runtime.core.provider.credential.password
com.tibco.trinity.runtime.core.provider.credential.password.usernameToken=uid=Manager\,ou=people\,dc=example\,dc=org,#!fGqMyEST0e58y1QEt7sykDYhfWq9mjKMVsJwsSHnAC4=
com.tibco.trinity.runtime.core.provider.credential.password.protectionParameter=password

SubjectIsp.properties

com.tibco.trinity.runtime.core.provider.lookup=com.tibco.trinity.runtime.core.provider.identity.subject
com.tibco.governance.sharedresource.name=SubjectIsp
com.tibco.governance.sharedresource.type=SubjectConfiguration

#SIP For Decryption
com.tibco.trinity.runtime.core.provider.identity.subject.identityStoreServiceProvider=class:com.tibco.trinity.runtime.core.provider.credential.keystore
com.tibco.trinity.runtime.core.provider.identity.subject.trustStoreServiceProvider=class:com.tibco.trinity.runtime.core.provider.credential.keystore
com.tibco.trinity.runtime.core.provider.identity.subject.keyAlias= john_key
com.tibco.trinity.runtime.core.provider.identity.subject.keyPassword=password
com.tibco.trinity.runtime.core.provider.identity.subject.enableCredentialStoreAccess=true
com.tibco.trinity.runtime.core.provider.identity.subject.enableTrustStoreAccess=true

#KCP for SIP
com.tibco.trinity.runtime.core.provider.credential.keystore.keyStoreLocation=keystore/default_keystore.jks
com.tibco.trinity.runtime.core.provider.credential.keystore.keyStorePassword=password
com.tibco.trinity.runtime.core.provider.credential.keystore.keyStoreType=JKS
com.tibco.trinity.runtime.core.provider.credential.keystore.keyStoreRefreshInterval=60000

TrustIsp.properties

com.tibco.trinity.runtime.core.provider.lookup=com.tibco.trinity.runtime.core.provider.identity.trust
com.tibco.governance.sharedresource.name=TrustIsp
com.tibco.governance.sharedresource.type=TrustConfiguration

#TIP for Signature verification
com.tibco.trinity.runtime.core.provider.identity.trust.trustStoreServiceProvider=class:com.tibco.trinity.runtime.core.provider.credential.keystore
com.tibco.trinity.runtime.core.provider.identity.trust.trustStoreEnableTrustStoreAccess=true

#KCP for TIP Namespace. Used for verifying the signature.
com.tibco.trinity.runtime.core.provider.credential.keystore.trustsStore.keyStoreType=JKS
com.tibco.trinity.runtime.core.provider.credential.keystore.trustsStore.keyStoreRefreshInterval=60000
com.tibco.trinity.runtime.core.provider.credential.keystore.trustsStore.keyStoreLocation=keystore/john_keystore.jks
com.tibco.trinity.runtime.core.provider.credential.keystore.trustsStore.keyStorePassword=password

IdentityIsp.properties

com.tibco.trinity.runtime.core.provider.lookup=com.tibco.trinity.runtime.core.provider.identity.subject
com.tibco.governance.sharedresource.name=IdentityIsp
com.tibco.governance.sharedresource.type=SubjectConfiguration
com.tibco.trinity.runtime.core.provider.identity.subject.identityStoreServiceProvider=class:com.tibco.trinity.runtime.core.provider.credential.keystore
com.tibco.trinity.runtime.core.provider.identity.subject.trustStoreServiceProvider=class:com.tibco.trinity.runtime.core.provider.credential.keystore
com.tibco.trinity.runtime.core.provider.identity.subject.keyAlias=john_key
com.tibco.trinity.runtime.core.provider.identity.subject.keyPassword=password
com.tibco.trinity.runtime.core.provider.identity.subject.enableCredentialStoreAccess=true
com.tibco.trinity.runtime.core.provider.identity.subject.enableTrustStoreAccess=true
com.tibco.trinity.runtime.core.provider.credential.keystore.keyStoreLocation=keystore/PAJohnIdentity.jceks
com.tibco.trinity.runtime.core.provider.credential.keystore.keyStorePassword=password
com.tibco.trinity.runtime.core.provider.credential.keystore.keyStoreType=jceks
com.tibco.trinity.runtime.core.provider.credential.keystore.keyStoreRefreshInterval=60000

WssAsp.properties

com.tibco.trinity.runtime.core.provider.lookup=com.tibco.trinity.runtime.core.provider.authn.wss
com.tibco.governance.sharedresource.name=WssAsp
com.tibco.governance.sharedresource.type=WSSConfiguration
com.tibco.trinity.runtime.core.provider.authn.wss.enableSAML11Assertion=true
com.tibco.trinity.runtime.core.provider.authn.wss.signatureValidationService=class:com.tibco.trinity.runtime.core.provider.identity.subject
com.tibco.trinity.runtime.core.provider.authn.wss.usernameTokenValidationService=class:com.tibco.trinity.runtime.core.provider.authn.ldap
com.tibco.trinity.runtime.core.provider.authn.ldap.enableSAML11Assertion=true
com.tibco.trinity.runtime.core.provider.authn.saml.enableSAML11Assertion=true
com.tibco.trinity.runtime.core.provider.authn.signature.enableSAML11Assertion=true

#SIP For Decryption
com.tibco.trinity.runtime.core.provider.identity.subject.identityStoreServiceProvider=class:com.tibco.trinity.runtime.core.provider.credential.keystore
com.tibco.trinity.runtime.core.provider.identity.subject.trustStoreServiceProvider=class:com.tibco.trinity.runtime.core.provider.credential.keystore
com.tibco.trinity.runtime.core.provider.identity.subject.keyAlias=john_key
com.tibco.trinity.runtime.core.provider.identity.subject.keyPassword=password
com.tibco.trinity.runtime.core.provider.identity.subject.enableCredentialStoreAccess=true
com.tibco.trinity.runtime.core.provider.identity.subject.enableTrustStoreAccess=true

#KCP for SIP
com.tibco.trinity.runtime.core.provider.credential.keystore.keyStoreLocation=keystore/default_keystore.jks
com.tibco.trinity.runtime.core.provider.credential.keystore.keyStorePassword=password
com.tibco.trinity.runtime.core.provider.credential.keystore.keyStoreType=JKS
com.tibco.trinity.runtime.core.provider.credential.keystore.keyStoreRefreshInterval=60000

#TIP for Signature verification
com.tibco.trinity.runtime.core.provider.identity.trust.trustStoreServiceProvider=class:com.tibco.trinity.runtime.core.provider.credential.keystore
com.tibco.trinity.runtime.core.provider.identity.trust.enableTrustStoreAccess=true

#KCP for TIP Namespace. Used for verifying the signature.
com.tibco.trinity.runtime.core.provider.credential.keystore.trusts
tore.keyStoreType=JKS
com.tibco.trinity.runtime.core.provider.credential.keystore.trusts
tore.keyStoreRefreshInterval=60000
com.tibco.trinity.runtime.core.provider.credential.keystore.trusts
tore.keyStoreLocation=keystore/john_keystore.jks
com.tibco.trinity.runtime.core.provider.credential.keystore.trusts
tore.keyStorePassword=password
#com.tibco.trinity.runtime.core.provider.credential.keystore.trust
store.keyStoreProvider=

#LDAP namespace. Used to verify user name token.
com.tibco.trinity.runtime.core.provider.authn.ldap.securityAuthentication=simple
com.tibco.trinity.runtime.core.provider.authn.ldap.initialCtxFactory=com.sun.jndi.ldap.LdapCtxFactory
com.tibco.trinity.runtime.core.provider.authn.ldap.userDNTemplate=uid={0},ou=people,dc=policy,dc=tibco,dc=com
com.tibco.trinity.runtime.core.provider.authn.ldap.userAttributeUserNames=uid
com.tibco.trinity.runtime.core.provider.authn.ldap.userAttributesExtra=mail,givenname
com.tibco.trinity.runtime.core.provider.authn.ldap.userSearchScopeSubtree=true
com.tibco.trinity.runtime.core.provider.authn.ldap.groupSearchExpression=uniqueMember={0}
com.tibco.trinity.runtime.core.provider.authn.ldap.groupSearchBaseDN=ou=groups,dc=policy,dc=tibco,dc=com
com.tibco.trinity.runtime.core.provider.authn.ldap.groupSearchScopeSubtree=true
com.tibco.trinity.runtime.core.provider.authn.ldap.groupAttributeGroupsName=cn
com.tibco.trinity.runtime.core.provider.authn.ldap.groupAttributeValueUsersName=cn
com.tibco.trinity.runtime.core.provider.authn.ldap.groupIndication=groupHasUsers
com.tibco.trinity.runtime.core.provider.authn.ldap.followReferrals=true
com.tibco.trinity.runtime.core.provider.authn.ldap.connectionPools=5
com.tibco.trinity.runtime.core.provider.authn.ldap.searchTimeOut=-1
com.tibco.trinity.runtime.core.provider.authn.ldap.enableSAML20Assertion=true
com.tibco.trinity.runtime.core.provider.authn.ldap.enableSAML11Assertion=false
com.tibco.trinity.runtime.core.provider.authn.ldap.samlValiditySeconds=300
Chapter 12  OAuth Server

This chapter describes the support of OAuth server provided by TIBCO(R) API Exchange Gateway.

Topics

- Overview on page 406
- OAuth Server Components Interactions on page 409
- Use Cases on page 414
- Configuration Setup of OAuth 2.0 Server on page 416
- Owner Service Provider Interface on page 423
- Client Service Provider Interface on page 426
- Owner Service Provider Interface on page 423
- How to Deploy Service Provider Interface on page 432
Overview

TIBCO API Exchange Gateway provides the support for both OAuth server and client. See Security Policies on page 351 for the OAuth client policies. This chapter mainly discusses the functionality of OAuth server support.

The OAuth 2.0 specification enables a third party application to access private data to which a user has granted permission. It is a standard protocol that allows a user to share data from the site that stores his data to the application using or accessing the data without exposing his credentials to that third-party application.

Based on HTTP protocol, OAuth provides the third party application access to protected resource owned by a resource owner. The access to the resource is based on scopes and a granting interaction defined by the OAuth specification among the resource server, application and resource owner. The interaction pattern keeps the resource owner and third party application’s credentials from being revealed to each other while allowing the third party to access resource on behalf of the resource owner.

The OAuth specification is found at http://www.oauth.net.

Concepts

The following concepts are used in OAuth protocol specification:

- Resource Owner
  Resource owner is an entity capable of granting access to a protected resource. When the resource owner is a person, it is referred to as an end-user.

- Client
  A client is an application making protected resource requests on behalf of the resource owner and with its authorization.

- Authorization Server
  Authorization Server is the server issuing access tokens to the client after successfully authenticating the resource owner and obtaining authorization.

- Resource Server
  Resource server is the server hosting the protected resources, capable of accepting and responding to protected resource requests using access tokens.

- Authorization Code
The authorization code is obtained from an authorization server when the resource owner grants access to the owner's resource.

- **Access Token**
  
  Access tokens are credentials used to access protected resources.

- **Authorization Endpoint**
  
  The authorization endpoint is the endpoint on the authorization server where the client requests for authorization. The request will be redirected to allow the resource owner to log in and grant authorization to the client.

- **Token Endpoint**
  
  The token request endpoint is the endpoint on the authorization server where the client application exchanges the authorization code for an access token.

- **Redirect Endpoint**
  
  The redirect endpoint is the endpoint in the client application where the authorization server redirects to after the resource owner grants authorization to the client application. The client will receive an authorization code which can be used to exchange for an access token.

**Example Scenario**

For example, a client application can access an API from the gateway as long as the owner of the API allows the application to do so.

Figure Figure 27, OAuth Server Overview, on page 408 illustrates this example scenario.

The OAuth process flow for the above example is explained as follows:

- A user goes to a web site hosting an application that will display some data from APIs on the gateway.

- The user first goes to the web site of the application. Before the application access the APIs on the gateway, the application requests an access token from an Authorization server.

- In the process of obtaining the access token, the user is requested to login and grant the application to the data that the application wish to access.

- After the user logged in and granted access, the application receives an authentication code, which the application to exchange for an access token.

- The application uses the access token it received to access the APIs on the gateway.
Benefits Of Using API Exchange Gateway OAuth Server

API Exchange Gateway OAuth server allows to decouple OAuth server interaction from the target service implementation. By plugging in user's OAuth adapter, the user is able to delegate OAuth interaction with the client application, access token management, access token validation to API Exchange Gateway OAuth server. This allows exposing the existing backend service implementation as OAuth protected resource to the client applications seamlessly. Keeping the backend service implementation as agnostic as possible to the OAuth concept can increase the portability and maintainability of the backend service implementation.
OAuth Server Components Interactions

Following diagram depicts the logical view of interaction between various components that participate in an OAuth protected data request.
The interactions between components of OAuth server are explained below:
Request Data from API Service

When a resource is protected by OAuth, a resource owner may allow a relying party such as an API explorer or a client application to get the resource on behalf of the resource owner himself. In a typical OAuth scenario, the resource owner sends the request from a user agent, such as browser, to the relying party.

Requesting Access Token

When the relying party, such as a client application, that is serving the request, receives the request from the resource owner, it will first request an access token from the authorization server. It may use one of the following authorization flows to obtain the access token:

- Authorization code flow
- Password flow
- Client credential flow

For details of the call for various flows, Refer to the RFC 6749, "The OAuth 2.0 Authorization Framework".

Sending Request with Access Token

When the relying party has an access token, it can forward the request to get the resource by passing the access token in the request.

Login and Grant Access

In authorization code flow scenario, to authenticate the resource owner, Authorization Server delegates the login to an Identity Service Provider. After a successful login, the Identity Service Provider returns the owner credentials to the Authorization server. The Authorization Server then make another request to the Authentication server to enable the logged in owner grant access to the client. When access is granted to the client, the Authorization Server generates an authorization code and returns the code to the relying party.

Authenticate Owner & Client and Retrieve Attributes

To authenticate the resource owner and client, the Identity Service Provider uses an authentication adapter to perform the authentication. Successful authentication of an owner or client will return the credentials or attributes of the owner or client. The adapter may authentication owner using LDAP, Database, or an SSO provider such as Ping Federate. The adapter most likely authenticated the client via the API Exchange Gateway Portal engine as the Portal engine manages the clients.
Assigning Scope to Resource

In order to perform resource authorization at runtime, the Resource Server relies on resource manager to provide information needed to perform the authorization. The management of scope and its associates with the owner's resource is outside the scope of this document as it depends on how the owner's resources and scopes are manages.

Validate Access Token

Before the resource server, API Exchange Gateway forwards the request from relying party to API provider. It will validate the access token from the request to ensure that the access token is valid and the request is authorized based on the information obtained from the access token. The Authorization server maintains the list of tokens that it has issued. The tokens can be retrieved using REST API.

Forward Request with Access Token

If owner, client, and scopes of the access are valid for the target operation, the resource server will forward the access token or optionally convert the access token to a SAML assertion and forward the assertion in the HTTP header.

Manage Access Token

The Authorization server maintains the list of tokens that has been issued by it. The tokens can be retrieved using REST API.

1. Retrieving tokens

   Following are REST APIs for retrieving the tokens:
   — Retrieving all tokens

   ```
   GET /oauth2/token/owners HTTP/1.1
   Authorization: Basic czZCaGRSa3F0MzpnWDFmQmF0M2JW
   Host: server.example.com
   ```

   — Retrieving tokens for specific owner

   ```
   GET /oauth2/token/owners/{ownerID} HTTP/1.1
   Authorization: Basic czZCaGRSa3F0MzpnWDFmQmF0M2JW
   Host: server.example.com
   ```

Output Result

The result of the API is a list of tokens in JSON format.
For example, the response of the API call is as follows:

```
[
{
...
}
{
  "id": "f0c997d31db2f8945ef63115d04783",
  "owner": "uid=john,ou=owner,dc=tibco,dc=com",
  "client": "playground",
  "scopes": "public",
  "accessToken": "3f8f4bfe51b7a99455e3a6bfccaf12b5",
  "refreshToken": "e118715f2783d1cfe11e17e5c4b93318",
  "createdOn": 1381246937747,
  "expiresOn": 1381250541022,
  "callback": "https://appHost:8080/site/client/callback",
  "contact": "John Doe",
  "email": "john.doe@tibco.com",
  "stringId": "f0c997d31db2f8945ef63115d04783"
}
{
...
}
]
```

2. Revoking tokens

The access token can be revoked using the following API:

```
DELETE /access_token/{tokenID} HTTP/1.1
Host: server.example.com
Authorization: Basic czZCaGRSa3F0MzpnWDFmQmF0M2JW
Content-Type: application/x-www-form-urlencoded
```
Use Cases

This section describes few use cases based on different OAuth flow:

Authentication Code Flow Use Case

In the authentication code flow, the owner of the data is the user who is using the application. The OAuth server authenticates the user and requests the user to grant the client access to his data. The client application does not have access to the user’s credential and the scope of the data access is known to the user. The OAuth provider also authenticates the client using the client ID and secret before an access token is given to the client.

For details of authentication code flow, See Section 4.1 of RFC 6749, "Authorization Code Grant".

Following is an example scenario of this use case:

- A user uses an application developed by a third party developer that will access photos that he has uploaded to an OAuth server.
- The third party developer registered the application with the OAuth server.
- The user uses the application to view the photo he has uploaded.
- The application requests an access token using authentication flow before retrieving the photo.

Password Credential Flow Use Case

In the password credential flow, the owner of the data is the user who is using the application. The difference between the password credential flow use case and authorization code flow use case is that the application has access to the user's credential. This use case usually apply to application design for mobile device where user credential is stored on the mobile device.

For details of password credential flow, See Section 4.3 of RFC 6749, "Resource Owner Password Credentials Grant”.

The following is an example scenario of this use case:

- A user uses an application on his mobile device that will access photos that he has uploaded to an OAuth server.
- The user uses the application to view the photo he has uploaded.
- The application requests an access token using password flow before retrieving the photo.
Client Credential Flow Use Case

In the client credential flow, the owner of the data that client access belongs to the client. The client does not use or need the credential of the user who use the application. The client uses its own credential to get the data from the resource server using its own credential. The OAuth server authenticates the client using the client ID and client secret of the client.

For details of client credential flow, See Section 4.4 of RFC 6749, "Client Credentials Grant".

The following is an example scenario of this use case:

- A user uses an application that provide some data that the client has access to.
- The application requests an access token using client credential flow before retrieving the data.
**Configuration Setup of OAuth 2.0 Server**

This section explains the configuration setup required to use API Exchange Gateway OAuth server. To enable the OAuth server, you must set the OAuth server related properties using the API Exchange Gateway user interface.

Do the following steps to set the properties:

1. Navigate to ASG_HOME/bin.
2. Type `asg-configui` command to start the gateway user interface.
3. Enter your credentials to login to the user interface.
4. On the Home page of the user interface, in the upper panel, select the Gateway Engine Properties from the drop down list box.
5. Click the arrow on the left hand side to expand the Gateway Engine Properties node.
6. Click the Security bullet.
7. Click the Security tab.
8. Expand the OAuth node to see the OAuth server properties.
9. Set the following properties:

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Properties For OAuth Data Space</strong></td>
<td></td>
</tr>
<tr>
<td>MetaSpace Name</td>
<td>Specifies the metaspace name used by the OAuth server.</td>
</tr>
<tr>
<td></td>
<td>Default value is ASG-OAuth-Tokens</td>
</tr>
<tr>
<td>Local Discovery URL</td>
<td>Specifies the discovery URL for this OAuth instance of the metaspace discovers the current metaspace members.</td>
</tr>
<tr>
<td></td>
<td>For example, tcp://localhost:6300/</td>
</tr>
</tbody>
</table>
Table 84  OAuth Server Properties

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Local Listen URL</td>
<td>Specifies the listening URL for this OAuth instance of the metaspace.</td>
</tr>
<tr>
<td></td>
<td>For example, tcp://localhost:6300/</td>
</tr>
<tr>
<td>Remote Discovery URL</td>
<td>Specifies the remote discovery URL for this OAuth instance of the metaspace discovers the current metaspace members.</td>
</tr>
<tr>
<td>Remote Listen URL</td>
<td>Specifies the remote listening URL for this OAuth instance of the metaspace.</td>
</tr>
<tr>
<td>Batch Size</td>
<td>Specifies the maximum number of entries to return when query data such as access token.</td>
</tr>
</tbody>
</table>

Properties For OAuth Persister

Persister Store

- Defines the type of persistence store. Possible values are:
  - InMemory
  - Database

- You can select the type of persistence store from a drop down list. When the Database is selected, you must define the properties for database server connection. See Database Connection Properties For OAuth Persister Database Type on page 417.

Database Connection Properties For OAuth Persister Database Type

Driver

- Specifies the database jdbc driver when Database is used as OAuth persistence store.
Table 84  OAuth Server Properties

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>JDBC URL</td>
<td>Specifies the jdbc url for the database server when Database is used as OAuth persistence store.</td>
</tr>
<tr>
<td>JDBC User Name</td>
<td>Specifies the user to connect to the database server when Database is used as OAuth persistence store.</td>
</tr>
<tr>
<td>JDBC Password</td>
<td>Specifies the password of the user to connect to the database server when Database is used as OAuth persistence store.</td>
</tr>
</tbody>
</table>

Properties For OAuth Adapters

<table>
<thead>
<tr>
<th>Resource Name</th>
<th>Specifies the directory from where the custom adapters loads the resources such as properties file used by adapters. This directory location is relative to the ASG_HOME.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>For example, if the value is specified as /examples/OAuth/resources, the custom adapter looks for the resources such as properties file in the directory ASG_HOME/OAuth/Adapters/resources.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Owner Adapter</th>
<th>Specifies the class that provides the Owner Adapter interface. This adapter will be used to authenticate the resource owner and provide the login and access grant pages. See Owner Service Provider Interface on page 423 for details.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>For example, for file based owner adapter interface, value specified as: com.tibco.asg.oauth.identity.provider.file.OwnerAdapterService.</td>
</tr>
<tr>
<td></td>
<td>- The jar file that contains this adapter implementation must be placed in a directory in the classpath set in the ASG_HOME/bin/asg-engine.tra file. See How to Deploy Service Provider Interface on page 432 for details.</td>
</tr>
</tbody>
</table>
Table 84  OAuth Server Properties

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Client Adapter</td>
<td>Specifies the class that provides the Client Adapter interface. This adapter will be used to authenticate the client and to retrieve the client attributes. See Client Service Provider Interface on page 426 for details. For example, for file based client adapter interface, value specified as: com.tibco.asg.oauth.identity.provider.file.ClientAdapterFactory</td>
</tr>
<tr>
<td></td>
<td>The jar file that contains this adapter implementation must be placed in a directory in the classpath set in the ASG_HOME/bin/asg-engine.tra file. See How to Deploy Service Provider Interface on page 432 for details.</td>
</tr>
<tr>
<td>Scope Adapter</td>
<td>Specifies the class that provides the Scope Adapter interface. This adapter will be used to retrieve the scope description and the scope form a specific resource for a given owner. See Scope Service Provider Interface on page 429 for details. For example, for file based scope adapter interface, value specified as: com.tibco.asg.oauth.identity.provider.file.ScopeAdapterFactory</td>
</tr>
<tr>
<td></td>
<td>The jar file that contains this adapter implementation must be placed in a directory in the classpath set in the ASG_HOME/bin/asg-engine.tra file. See How to Deploy Service Provider Interface on page 432 for details.</td>
</tr>
</tbody>
</table>

Portal Engine Integration Properties

| Engine URL | Specifies the URL of the portal engine. For example, the engine URL can be specified as: http://portal_host_name:9122 |
Integration with Portal Engine (Optional)

By default, the client adapter is used by the portal engine. To integrate with the portal engine, you must set the following properties in `ASG_CONFIG_HOME/asg.properties` file as below:

- Set the client adapter property used by portal engine as below:
  - `oauth.client.adapter.class=com.tibco.asg.portal.engine.oauth.identity.provider.ClientAdapterImpl`
- Set the following property to define the portal engine url:
  - `tibco.clientVar.portal.engine.url`

For example, `tibco.clientVar.portal.engine.url=http://portal_host_name:9122`

You can set the Client Adapter and Engine URL using the API Exchange Gateway user interface.
Chapter 13  OAuth Service Provider Interface

This chapter describes the service provider interfaces supported by API Exchange Gateway for OAuth server.

Topics

- Overview on page 422
- Owner Service Provider Interface on page 423
- Client Service Provider Interface on page 426
- Scope Service Provider Interface on page 429
- How to Deploy Service Provider Interface on page 432
Overview

API Exchange Gateway OAuth server allows extending the OAuth authorization processing via java based service provider interfaces. The service provider interfaces are implemented in plugins which are loaded during startup. This functionality allows users to implement custom adapters.

Following service provider interfaces are supported:

- Owner service provider interface
- Client service provider interface
- Scope service provider interface
Owner Service Provider Interface

The owner service provider interface is used by OAuth server to authenticate resource owner during the authorization code flow. It enables the OAuth server to authenticate the resource owner and obtain attributes of the owner. In addition to authenticate the owner, the owner service provider interfaces are responsible for redirecting to login and access grant page.

Owner Service Provider Interface (SPI) Flow

Figure Owner SPI Flow on page 423 illustrates the flow for owner service provider interface.
Owner Service Provider Interface (SPI) Java API

Following is the Java API of the owner service provider interface:

```java
/**
 * OwnerAdapter is the interface use by OpenID Provider to authenticate
 * the resource owner.
 * <p>
 * A OwnerAdapter may be implemented using LDAP, database, 3rd party ISP, or a
 * combination of those.
 */
public interface OwnerAdapter {

    /**
     * This method is called when the instance of the adapter is first loaded.
     * The properties is a map of properties from SecurityRuntime.cfg.
     * The adapter may initialize itself using these properties.
     *
     * @param properties a map of properties from SecurityRuntime.cfg.
     */
    public void init(Map<String, String> properties);

    /**
     * Authenticate the owner with the specify username and password.
     *
     * @param username username to authenticate.
     * @param password password to authenticate.
     * @return a OwnerResult that has the result of the authentication.
     * @see OwnerResult which will has the owner profile or error from the authentication.
     */
    public OwnerResult authenticateOwner(String username, String password);

    /**
     * Process login redirects owner to a login page for resource owner to login.
     * The login page could be a form with j_username and j_password which will be posted to
     * the resumeUrl. When resumeUrl received the post request, it will
     * authenticate the j_username and j_password with authenticateOwner.
     * If authenticateOwner failed, processLogin is called again.
     *
     * @param request servlet request of the incoming request
     * @param response servlet response of the incoming request
     * @param resumeUrl the url to return to after login is done.
     */
```
/**
 * @throws ServletException
 * @throws IOException an exception if failed to redirect.
 */

public void processLogin(HttpServletRequest request,
        HttpServletResponse response, String message, String resumeUrl)
        throws ServletException, IOException;

/**
 * Process grant access redirects owner to a grant access for resource owner to
 * grant access to the client based on the scopes.
 *
 * The login page could be a form with j_username and j_password which will be posted to
 * the resumeUrl. When resumeUrl received the post request, it will
 * authenticate the j_username and j_password with
 * authenticateOwner.
 * If authenticateOwner failed, processLogin is called again.
 *
 * @param request servlet request of the incoming request
 * @param response servlet response of the incoming request
 * @param client the client to grant access to.
 * @param scopes an array of discription of scopes that the client wish to access
 * @param resumeUrl the url to return to after login is done.
 *
 * @throws ServletException
 * @throws IOException an exception if failed to redirect.
 */

public void processGrantAccess(HttpServletRequest request,
        HttpServletResponse response, String client, String[]
        scopes, String resumeUrl)
        throws ServletException, IOException;
Client Service Provider Interface

The client service provider interface is used by OAuth server to authenticate a client when the client is requesting the access token during authorization code or client credential flow. It enables the OAuth server to authenticate the client and obtain attributes of the client.

Client Service Provider Interface (SPI) Flow

Figure Client SPI Flow on page 426 illustrates the flow for client service provider interface.

Figure 30  Client SPI Flow

![Client SPI Flow Diagram]

Client Service Provider Interface (SPI) Java API

Following is the Java API of the client service provider interface:

```java
/**
 * ClientAdapter is the interface use by OpenID Provider to authenticate client.
 * It is also for OpenID Provider to retrieve the clients scopes.
 * <p/>
 * A ClientAdapter is implemented by Portal Service who manages the client registration.
 * *
 */

public interface ClientAdapter {

/**
 */
```
* This method is called when the instance of the adapter is first loaded.
* The properties is a map of properties from SecurityRuntime.cfg.
* The adapter may initialize itself using these properties.
* 
* @param properties a map of properties from SecurityRuntime.cfg.
* */
public void init(Map<String, String> properties);

/**
  * Authenticate the client with the specify id and secret.
  *
  * @param clientId client id to authenticate.
  * @param secret secret to authenticate.
  * @return a DirectoryResult that has the result of the authentication.
  * @throws AdapterException an exception if failed to retrieve the scopes of the client.
  */
public ClientResult authenticateClient(String clientId, String secret);

/**
  * Retrieve scopes for the specify client.
  *
  * @param clientId the clientId to retrieve the scope.
  * @return return a map that contains information of the scopes.
  * scope -> array of scopes
  * description -> array of descriptions
  * @throws AdapterException an exception if failed to retrieve the scopes of the client.
  */
public Map<String, String> getClientAttributes(String clientId) throws AdapterException;
**getAttributes**  The required attributes for getAttributes are as below:

Table 85  Required Attributes For getAttributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dn</td>
<td>Specifies the distinguish name for the client.</td>
</tr>
<tr>
<td>oidcClientID</td>
<td>Specifies the unique Id for the client.</td>
</tr>
<tr>
<td>oidcClientRedirectURI</td>
<td>Specifies the redirect or callback url.</td>
</tr>
<tr>
<td>scopes</td>
<td>Specifies a list of comma separated scopes.</td>
</tr>
</tbody>
</table>

**Sample attributes:**

oidcClientIDplayground=dn=oidcClientIDplayground,ou=client,dc=tibco,dc=com scopes=public playground

Following table shows the sample attributes:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Sample Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>oidcClientID</td>
<td>playground</td>
</tr>
<tr>
<td>dn</td>
<td>oidcClientIDplayground</td>
</tr>
<tr>
<td>ou</td>
<td>client</td>
</tr>
<tr>
<td>dc</td>
<td>tibco</td>
</tr>
<tr>
<td>scopes</td>
<td>public playground</td>
</tr>
<tr>
<td>oidcClientRedirectURI</td>
<td><a href="https://redirectHost/site/client/redirectEndpoint">https://redirectHost/site/client/redirectEndpoint</a></td>
</tr>
<tr>
<td>oidcAppName</td>
<td>Playground</td>
</tr>
<tr>
<td>oidcAppLogoURL</td>
<td><a href="https://redirectHost/site/client/logo">https://redirectHost/site/client/logo</a></td>
</tr>
<tr>
<td>oidcClientEmail</td>
<td><a href="mailto:playground.asg@tibco.com">playground.asg@tibco.com</a></td>
</tr>
</tbody>
</table>
Scope Service Provider Interface

The scope service provider interface is used by Authorization server to retrieve the description of the scopes and retrieving the scopes for a specified owner and resource to access.

Scope Service Provider Interface (SPI) Flow

Figure Scope SPI Flow on page 429 illustrates the flow for scope service provider interface.

Figure 31 Scope SPI Flow

Scope Service Provider Interface (SPI) Java API

Following is the Java API of the scope (token) service provider interface:
/**
 * ScopeAdapter is the interface used by OpenID Provider to retrieve client's
 * scope and scope for a specific resource for a user.
 *
 */

public interface ScopeAdapter {

/**
 * This method is called when the instance of the adapter is first
 * loaded.
 * The properties is a map of properties from SecurityRuntime.cfg.
 * The adapter may initialize itself using these properties.
 *
 * @param properties a map of properties from SecurityRuntime.cfg.
 */

public void init(Map<String, String> properties);

/**
 * Returns descriptions for specified scopes
 *
 * @param scopes the scopes to retrieve the descriptions for.
 * @return the descriptions for specified scopes
 * @throws AdapterException an exception if failed to retrieve the
 * scopes' descriptions.
 */

public String[] getDescriptions(String[] scopes)
throws AdapterException;

/**
 * Retrieve scopes for the resource a specified owner.
 *
 * @param owner the owner to resource.
 * @param resource the resource for the scope to retrieve.
 * @param resourceAttributes attributes of the resources
 * @return scopes for the resource a specified owner.
 */
* @throws AdapterException an exception if failed to retrieve the scopes of
  * the resource.
  */

public String[] getResourceScopes(String owner, String resource,
Map<String, String> resourceAttributes)
  
throws AdapterException;

}
How to Deploy Service Provider Interface

If you want to enable the OAuth server to use the service provider interfaces, you must deploy the API Exchange Gateway instance where the Identity Service Provider of the OAuth server is running. This section explains the steps required to deploy the service provider interfaces.

Follow the steps as below:

**Task A  Copy the SPI Jar Files**

- Copy the jars containing the implementation class of the implemented adapters and their dependencies to ASG_HOME/lib/ext/tpc1 directory.

**Task B  Set OAuth Adapter Properties**

Set the OAuth Adapter properties using API Exchange Gateway user interface. OAuth adapter properties settings are described in the OAuth Server Properties on page 416. Refer to section Properties For OAuth Adapters on page 418.

You can set the OAuth server adapter properties in the ASG_CONFIG_HOME/asg.properties file. Refer to Properties For OAuth Adapters on page 86 for properties details.
Chapter 14  Gateway Management Features

This chapter explains the functionality of TIBCO API Exchange Gateway management components.

Topics

- Central Logger, page 434
- Enabling JMS Channel For Central Logger, page 439
- Configuration Setup For JMS Channel, page 439
- Global Throttle Manager, page 444
- Cache Clearing Manager, page 447
- Reporting, page 448
Central Logger

Overview

The TIBCO API Exchange Gateway provides centralized via the Central Logger component. Each Gateway server instance publishes events during its operation. The Central Logger component receives the events from the management bus as messages. The Central Logger stores the messages in the database or in a file. The Gateway server instance aggregates events in memory and publishes the messages at configured intervals in order to reduce the disk load during high transaction rates.

Pre-requisites

The Central Logger component audit logs the transactions in a database or a file. In production systems, its recommended to use a database server. Make sure to configure and setup a database server for the functionality of the central logger component. See Database Setup and Configuration for Central logger, page 434

Database Setup and Configuration for Central logger

This section guides you through the steps required to configure the database and set up the database drivers for TIBCO API Exchange Gateway Central Logger component.

Database Location

Instructions in this section assume you are working with a local database for testing purposes. Adapt the instructions if you are working with a remote database. For example, in production environments, you might have to ask a database administrator to create a database and a database user for you.

Make sure that you have access to a running database server instance required for Central Logger component.

Task C  Setup Database Driver

- Copy the appropriate JDBC driver jar file to ASG_HOME/lib/ext/tpcl directory. See the product readme file for the supported versions of database drivers.

For example, database jar file for MySql database is:
mysql-connector-java-5.1.18-bin.jar

For example, database jar file for Oracle database is:
ojdbc6.jar
For example, database jar for **SQL Server** database is: sqljdbc4.jar

**Task D  Create a database**

Depending upon your environment, you might have to ask your database administrator to create a database or use an existing database. For example, for **MySql** database server, you can create a local database for your testing purposes. For production environments using **Oracle** database server, ask your database administrator to create a database for TIBCO API Exchange Gateway central logger component.

This section lists the steps to create a database for testing purposes using **MySql** database server.

1. Verify that the **MySql** database server is running.
2. Log on to the database server using the command:
   ```
   mysql -uroot -p
   ```
3. Enter the password when prompted. Enter password:
4. Type the following command at the mysql command prompt to create a new database:
   ```
   create database asgstat;
   ```
5. Verify that the **asgstat** database is created.


**Task E  Create a database user**

For testing purposes, this section explains the steps required to create the database user with appropriate privileges in a database, using the **MySql** database server.

For production systems using **Oracle** and **SQL Server** database server, work with your database administrator.

1. Verify that you are connected to the MySql database server as a root user. If not, type the command:
   ```
   mysql -uroot -p
   ```
2. Type the following command at the mysql command prompt to create a new user:
3. To grant the appropriate privileges to the database user, type the following command at the mysql command prompt:
   grant create, select, insert, update on asgstat.* to asguser@'%' identified by 'asgpass';

4. Type the following command to reload the privileges from the grant tables in the database:
   flush privileges;

**For Oracle Database**

Ask your database administrator to create a database user (for example `asguser`) and grant the `connect, resource` privileges to this user.

**For MS SQL Server Database**

Ask your database administrator to create a database user (for example `asguser`) and grant `create, select, insert, update` privileges to this user.

**Task F  Setup the database schema**

This section explains the steps to create the database tables in the database required for the TIBCO API Exchange Gateway Common Logger component.

**For MySQL Database**

1. Navigate to the following directory:
   
   \ ASG_HOME/\templates/database/mysql

2. Type the following command at the command prompt:
   
   mysql -D asgstat -uasguser -pasgpass < createAsgTransactions.sql
   mysql -D asgstat -uasguser -pasgpass < createAsgKpis.sql

**For Oracle Database**

1. Navigate to the following directory:
   
   \ ASG_HOME/\templates/database/oracle

2. Type the following command at the command prompt: (Replace SID with the actual oracle database SID name)
   
   sqlplus asguser/asgpass@SID @createAsgTransactions.sql
   sqlplus asguser/asgpass@SID @createAsgKpis.sql
For SQL Server Database

1. Navigate to the following directory:

   \ASG_HOME\templates\database\sqlserver

2. Type the following command at the command prompt:

   isql -Usa -d asgstat -i createAsgTransactions.sql
   isql -Usa -d asgstat -i createAsgKpis.sql

Task G Setup the database connection parameters

This section explains the steps required to setup the parameters to connect to the database.

1. Open the \ASG_CONFIG_HOME\asg/asg_cl.properties file for editing.

2. Edit the following parameters to provide the values to connect to the appropriate database:

   tibco.clientVar.CL/Database/Driver=database driver type
   tibco.clientVar.CL/Database/Url=database url
   tibco.clientVar.CL/Database/Username=database user name
   tibco.clientVar.CL/Database/Password=database password

   For example, values are shown below for MySql database:

   tibco.clientVar.CL/Database/Driver=com.mysql.jdbc.Driver
   tibco.clientVar.CL/Database/Url=jdbc:mysql://localhost:3306/asgstat
   tibco.clientVar.CL/Database/Username=asguser
   tibco.clientVar.CL/Database/Password=asgpass
   tibco.clientVar.CL/Database/Schema=asgstat

   For example, values are shown below for Oracle database:

   tibco.clientVar.CL/Database/Driver=com.oracle.jdbc.Driver
   tibco.clientVar.CL/Database/Url=jdbc:oracle:thin:@localhost:1521:ORCL
   tibco.clientVar.CL/Database/Username=asguser
   tibco.clientVar.CL/Database/Password=asgpass
   tibco.clientVar.CL/Database/Schema=asgstat

   For example, values are shown below for SQL Server database:

   tibco.clientVar.CL/Database/Driver=com.microsoft.sqlserver.jdbc.SQLServerDriver
   tibco.clientVar.CL/Database/Url=jdbc:sqlserver://localhost:1433
   tibco.clientVar.CL/Database/Username=asguser
   tibco.clientVar.CL/Database/Password=asgpass
   tibco.clientVar.CL/Database/Schema=asgstat

For the database Url field, replace localhost with the host name where database server runs, if needed.
Run Time Properties For Central Logger

See Run Time Properties of Central Logger on page 87 for details.

Enable Reporting to Central Logger

By default, the reporting to central logger component is not enabled. You can enable the reporting as follows:

1. Open the `ASG_CONFIG_HOME/asg/asg.properties` file for editing.
2. Look for the following section in the file:
   
   ```
   # Turn on or off reporting to CL
   tibco.clientVar.ASG/Logging/reportingEnabled=false
   ```
3. To enable the reporting, set the value as:
   
   ```
   tibco.clientVar.ASG/Logging/reportingEnabled=true
   ```
4. Save the file and close the editor.

Run Central Logger

This section explains the steps to run the central logger.

1. Open a terminal window.
2. Navigate to `ASG_HOME/bin` directory.
3. Type the following command to start the Central logger:

   ```
   ./asg-engine -u asg-cl -a ASG_Configuration_Name
   ```

   where `ASG_Configuration_Name` is the name of the configuration used by the gateway core engine.

Before you run the Central Logger, ensure:

- The gateway core engine is running in cache agent enabled mode. This means that the core engine is running with `asg-caching-core` processing unit.
- The gateway core engine is running with the same configuration as will be used by the central logger. This means that the core engine is running with `ASG_Configuration_Name` configuration.
Enabling JMS Channel For Central Logger

The Central logger component of the API Exchange Gateway receives the messages from the gateway core engine and logs the transaction data records to a database. It writes the records in bulk after a fixed interval. The Central logger runs as a separate engine than the gateway core engine.

By default, the messages from the gateway core engine are sent to the Central logger using Rendezvous transport. You can also enable the JMS transport for the Central logger to send the messages from the gateway core engine to Central logger.

When the Rendezvous transport is used by the gateway core engine, the messages are not guaranteed to be received by the Central logger. If the Central logger instance is not running, the messages sent by the gateway core engine are lost. To improve the reliability and guaranteed delivery of the messages, API Exchange Gateway supports the JMS transport as a communication channel between the core engine and the Central logger.

Configuration Setup For JMS Channel

This section explains the configuration steps required to use JMS transport for the gateway core engine and the Central Logger.

How to Enable JMS Channel

By default, the JMS channel for the Central logger is disabled using the property defined in the asg_core.cdd and asg_c1.cdd files.

To enable the JMS channel for the Central logger, do the following steps:

1. Navigate to the ASG_HOME/bin directory.

2. Open the asg_core.cdd file in a text editor and edit it as below:

   a. Locate the be.channel.deactivate property as below:
   
   `<property name="be.channel.deactivate" value="/DefaultImplementation/Channels/SouthboundEsb0Channel, /DefaultImplementation/Channels/SouthboundEsb1Channel, /DefaultImplementation/Channels/SouthboundEsb2Channel, /DefaultImplementation/Channels/North_ESBChannel, /DefaultImplementation/Channels/North_HttpTPChannel, /ASG/Channels/SOAPJMSChannel_North, /ASG/Channels/SOAPJMSChannel_South, /Common/Channel/CentralLoggerJMS"/>

   b. Remove the following entry from the be.channel.deactivate property value:
c. Ensure that the value of `be.channel.deactivate` property looks as below:
   ```xml
   <property name="be.channel.deactivate" value="/DefaultImplementation/Channels/SouthboundEsb0Channel,/DefaultImplementation/Channels/SouthboundEsb1Channel,/DefaultImplementation/Channels/SouthboundEsb2Channel,/DefaultImplementation/Channels/North_ESBChannel,/DefaultImplementation/Channels/North_HTTPChannel,/ASG/Channels/SOAPJMSChannel_North,/ASG/Channels/SOAPJMSChannel_South"/>
   ```

d. Save the changes to the file.

3. Open the `asg_cl.cdd` file in a text editor and edit it as below:
   a. Locate the `be.channel.deactivate` property as below:
      ```xml
      <property name="be.channel.deactivate" value="/Common/Channels/CentralLoggerJMS"/>
      ```
   b. Remove the following entry from the `be.channel.deactivate` property value:
      ```xml
      /Common/Channel/CentralLoggerJMS
      ```
   c. Ensure that the `be.channel.deactivate` property value looks as below:
      ```xml
      <property name="be.channel.deactivate" value=""/>
      ```
   d. Save the changes to the file.

**Set JMS Transport for Central Logger**

You must set the JMS transport for the Central Logger in the `asg.properties` file as below:

1. Navigate to the `ASG_CONFIG_HOME` directory.
2. Open the `asg.properties` file in a text editor.
3. Set the value of following property to JMS:
   ```
   tibco.clientVar.ASG/Logging/transport=JMS
   ```
4. Save changes to the file.
5. Open the `asg_cl.properties` file in a text editor.
6. Enable the JMS transport as follows:
   ```
   tibco.clientVar.ASG/Logging/transport=JMS
   ```
7. Save changes to the file.

The default value of logging transport is RV which means that Rendezvous is used for communication between the gateway core engine and Central Logger. Setting the value to JMS specifies that JMS transport is used for communication between the gateway core engine and Central Logger.

**Configuring JMS Transport Properties**

The JMS Transport Properties, page 441 table shows the properties you must define to use the JMS transport for the Central Logger. The JMS transport properties are defined in the gateway core engine properties `asg.properties` file as well as in the Central logger properties `asg_cl.properties` file.

To set the JMS transport properties, do following steps:

1. Navigate to the `ASG_CONFIG_HOME` directory.
2. Open the `asg.properties` file in a text editor.
3. Set the properties defined in the JMS Transport Properties, page 441 table.
4. Save the changes to the file.
5. Open the `asg_cl.properties` file in a text editor.
6. Set the properties defined in the JMS Transport Properties, page 441 table.
7. Save the changes to the file.

**Table 86: JMS Transport Properties**

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
</table>
| tibco.clientVar.Common/Connections/JMS/CL_JMSClonection/JMSProviderURL | Specifies the URL to connect to the Enterprise Message Service (EMS) or a JMS server.  
Example: tcp://localhost:7222 |
| tibco.clientVar.Common/Connections/JMS/CL_JMSClonection/JNDIContextURL | Specifies a JNDI connection URL to look up a JMS server.  
Example: tibjmsnaming://localhost:7222 |
### Table 86  JMS Transport Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tibco.clientVar.Common/Connections/JMS/CL_JMSCOnnection/TopicConnectionFactoryName</td>
<td>Specifies the name of TopicConnectionFactory object stored in JNDI. This object is used to create a topic connection with JMS server for the Central logger. Default is TopicConnectionFactory</td>
</tr>
<tr>
<td>tibco.clientVar.Common/Connections/JMS/CL_JMSCOnnection/QueueConnectionFactoryName</td>
<td>Specifies the name of QueueConnectionFactory object stored in JNDI. This object is used to create a queue connection with JMS server for the Central logger. Default is QueueConnectionFactory</td>
</tr>
<tr>
<td>tibco.clientVar.Common/Connections/JMS/CL_JMSCOnnection/JNDIUsername</td>
<td>Specifies the user name to use when logging into the JNDI server. If the JNDI provider does not require access control, this field can be empty. Example, admin</td>
</tr>
<tr>
<td>tibco.clientVar.Common/Connections/JMS/CL_JMSCOnnection/JNDIPassword</td>
<td>Specifies the password for logging into the JNDI server. If the JNDI provider does not require access control, this field can be empty.</td>
</tr>
<tr>
<td>tibco.clientVar.Common/Connections/JMS/CL_JMSCOnnection/JMSUsername</td>
<td>Specifies the user name to use to authenticate to the JMS server.</td>
</tr>
<tr>
<td>tibco.clientVar.Common/Connections/JMS/CL_JMSCOnnection/JMSPassword</td>
<td>Specifies the password to use to authenticate to the JMS server.</td>
</tr>
<tr>
<td>tibco.clientVar.Common/Connections/JMS/CL_JMSCOnnection/TransactionReportDestinationName</td>
<td>Specifies the name of the JMS destination to which the transaction reports are sent to the Central Logger by the gateway core engine. For example, asg.cl.transaction.queue</td>
</tr>
</tbody>
</table>
Table 86  JMS Transport Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tibco.clientVar.Common/Connections/JMS/CL_JMSConnection/TransactionReportDestinationType</td>
<td>Specifies the type of the JMS destination to which the transaction reports are sent to the Central Logger by the gateway core engine. Possible values are queue or topic. Default value is queue. The Central logger always listens on a queue. If the value of destination type set to topic, the JMS administrator must configure a bridge between the topic and the queue.</td>
</tr>
</tbody>
</table>

You must restart the gateway core engine and Central logger instance after setting the properties in asg.properties and asg_cl.properties file, if they are running.
Global Throttle Manager

The Global Throttle Manager manages the Façade Throttle Manager and Service Throttle Manager. The Global Throttle Manager reports throttle usage to the Central Logger.

The Global Throttle Manager component maintains the state of all the global throttles in both Façades (Façade Throttles) and Routers (Service Throttles). The Global Throttle Manager exchanges the states of global throttles with active Façade Throttle Managers and Service Throttle Managers.

The Global Throttle Manager component provides the mechanism to evenly distribute the global throttles among TIBCO API Exchange Gateway server instances.

The Global Throttle Manager component allows you to implement simple group throttles. Below are few examples of throttles types that are used in the Façade or Router as global throttles:

- Commercial throttles implement gross usage agreements.
- Partner throttles act to limit a partner’s impact on internal services.
- Operation throttles implement fine-grained usage agreements.
- Technical throttles protect service interfaces.

Throttle Calculation

Global Throttle Manager calculates the throttles as follows:

For Quota and High-Water-Mark throttles, it uses the following formula to get the max count per engine:
MaxCountPerEngine = ThrottleMaxCount/maxActiveEngines

For Rate and Error throttles, the calculation is done as follows:
MaxCountPerEngine =
(ThrottleMaxCount/maxActiveEngines)*(UpdateIntervalSec/Throttle Interval)

— ThrottleMaxCount and Throttle Interval are defined in the throttle configuration.

— UpdateIntervalSec is the time interval in seconds for sending throttle updates to Global Throttle Manager. The default value is 10 seconds and can be edited in the ASG_CONFIG_HOME/asg.properties file. This is defined by the following property:
   tibco.clientVar.ASG/Throttle/UpdateIntervalSec=10

The value of MaxCountPerEngine is always rounded up. For example, 1.1 will be 2 and 1.9 will also be 2.

For non zero throttles, this rounding means that the MaxCount Per Engine will always be at least 1. For example, if there are 20 active engines and the calculated value of MaxCountPerEngine is 0.4, then the effective throttle limit is calculated as:
Ceiling(0.4)*active engines = 1*20 = 20

For this case, the value of MaxCountPerEngine is not 8. (0.4 * 20)

This indicates that if there are many active engines and the throttles are defined in this setup, then the MaxCount can never be less than the number of active engines.

Run Global Throttle Manager

This section explains the steps to run the Global throttle manager.

1. Open a terminal window.
2. Navigate to ASG_HOME/bin directory.
3. Type the following command:
   asg-engine -u asg-gtm -a ASG_Configuration_Name

   where ASG_Configuration_Name is the project configuration of the API Exchange Gateway.

For example, if you want to run the global throttle manager for the BookQuery configuration, type the following command:
   asg-engine -u asg-gtm -a BookQuery
where BookQuery is the current configuration.

- Make sure that the cache agent and the gateway core engine are running before you start the global throttle manager.
- If you are running multiple instances of the global throttle manager instances, type the following command to run the global throttle manager:
  ```
  asg-engine -u asg-gtm -a ASG_Configuration_Name -n Global_Throttle_Manager_Instance_Name
  ```
  where *ASG_Configuration_Name* is the project configuration of the API Exchange Gateway and *Global_Throttle_Manager_Instance_Name* is the name of the global throttle manager instance.
**Cache Clearing Manager**

The Cache Clearing Manager provides mechanism to clear the cache. The Cache Clearing Manager manages the eviction of entries in the associative cache, especially for the entries that are not often referenced.

TIBCO API Exchange Gateway server stores the responses from side-bound service requests and is used for future look-ups. Cached objects have a time to live, which is evaluated every time an entry is retrieved from the associative cache and the entry gets evicted on lookup if the time to live (TTL) has expired. Cache entries with a relatively short time to live (TTL) that are not often read, might pollute the associative cache using system resources at no benefit. For that reason the Cache Clearing Manager evaluates all cache entries on a scheduled time basis and evicts these entries who’s time-to-live has expired.

**How To Run Cache Clearing Agent**

This section explains the steps to run the cache clearing manager:

1. Open a terminal window.
2. Navigate to ASG_HOME/bin directory.
3. Type the following command:
   
   ```
   ./asg-engine -u asg-cache-clean-up
   ```

   It's recommended to start the cache clearing manager after the cache agent is running successfully. If the cache clearing agent starts before the cache agent, there could be a potential possibility of additional cache agents not being able to connect to the cache provider using the discovery listener and function properly.
The Central Logger component stores reporting information to its database and optionally to a file. The Apache HTTP server also generates access logs.

The data from the central logger database can be used for reporting. It contains the following data:

- **High level Transaction Auditing**
  - One entry for every received request
  - Timing information
  - Service, operation and partner identities
  - Status of overall transaction

- **Detailed level Transaction Auditing**
  - Request payload before and after every transformation
  - Multiple identities associated with each transaction
  - The list of stages through which the transaction passed
  - Ad-hoc messages generated during transaction processing

- **Key performance indicators**
  - Count of transactions received in a given time interval
  - Per partner, service and operation
  - Default time intervals one minute, five minutes and one hour

### TIBCO Spotfire Integration

TIBCO API Exchange Gateway provides a sample TIBCO Spotfire analysis based on data captured by the Central Logger component. The sample shows the following metrics:

- Transaction rate per partner and service
- Throttle violation, Service quality and Service timeouts failures
- Load Visualization for operations and management
  - Volume
  - Latency
  - Peaks
Spotfire Configuration

This section describes the steps for TIBCO Spotfire Professional configuration. The Spotfire client will retrieve the data stored in TIBCO API Exchange Gateway database to display the reports.

Task A Configure TIBCO Spotfire Server and Client

This section assumes that you have a running instance of Spotfire server. Ask the Spotfire administrator to create a username and password to allow access on the Spotfire server instance.

Spotfire User Permissions

Make sure that the Spotfire Administrator creates a user to be part of the Library Administrator group on the Spotfire server instance. Ensure that the Administrator grants the library administration permissions to this user. This allows the users to:

- Create library contents (such as data sources, information links).
- Import and Export the library content.
- Store the analysis files.

1. Install TIBCO Spotfire Professional software. Refer to the product readme for the supported version.
2. Launch the TIBCO Spotfire Professional.
3. Connect to the appropriate TIBCO Spotfire server instance by providing the correct URL in the Server field.
4. Download the software package updates from the server, if prompted.
5. Verify you have successfully connected to the TIBCO Spotfire server instance for TIBCO API Exchange Gateway and updated your TIBCO Spotfire client to the version that is deployed on the TIBCO Spotfire server.

Task B Setup a Spotfire Data source

1. Within TIBCO Spotfire Professional, open the Information Designer from the menu bar as: Tools -> Information Designer.
2. Click the Setup Data Source link.
3. Input the Data Source fields, as appropriate.
For **MySQL database**, example values are shown as below:

Name: *asgstat*
Type: **MySQL5**
Connection URL: `jdbc:mysql://database host:port/database`
(where database host is the machine database server runs, by default the port is 3306, and database is the database name such as asgstat).
No. of connections:
— Min: 1
— Max: 10
Username: *asguser*
Password: *asgpass*

For **Oracle database**, example values are shown as below:

Name: *asgstat*
Type: **Oracle**
Connection URL: `jdbc:oracle:thin:@database host:port:service name`
(where database host is the machine database server runs, by default the port is 1521, and the service name is the Oracle service name)
No. of connections:
— Min: 1
— Max: 10
Username: *asguser*
Password: *asgpass*

For **SQL Server database**, example values are shown as below:

Name: *asgstat*
Type: **SQL Server**
Connection URL: `jdbc:sqlserver://database host:1433/database`
(where database host is the machine database server runs, by default the port is 1433, and database is the database name such as asgstat).
No. of connections:
— Min: 1
— Max: 10
Username: asguser
Password: asgpass

4. Click Save button.

5. Click Save button again, if pop window appears.

6. Verify that this data source appears in the left panel of the Information Designer dialog with a + sign next to it.

Task C  Create a Spotfire Information Model for TIBCO API Exchange Gateway Central Logger

1. Within TIBCO Spotfire Professional, open the Information Designer from the menu bar as: Tools -> Information Designer.

2. In the left panel of the Information Designer dialog, expand the data source name by clicking on the + sign. Expand the asguser schema.

3. Verify that following database tables are listed from the Central Logger database schema:
   - ASG_KPI
   - ASG_LOG_MESSAGES
   - ASG_THROTTLE_MESSAGES
   - ASG_THROTTLE_USAGE
   - ASG_TRANSACTION_DETAILS
   - ASG_TRANSACTION_KEYS
   - ASG_TRANSACTION_MESSAGES
   - ASG_TRANSACTIONS

4. Shift-click on the first table and the last table to select all tables. With all the tables selected, right-click on your mouse and select the option Create default Information Model from the context menu. Click the OK button on the next new dialog window.

5. On the Create Default Information Model Settings window, ensure that the radio button Automatically assign a new name to the created item is selected. Click the OK button.

6. Verify that the Information Links has been created for each table of the Central Logger database, which appears in the left pane of the Information Designer on the Elements tab.

7. Click the Close button to close the Information Designer.
Task D  Deploy the default TIBCO API Exchange Gateway Audit Trail DXP file on TIBCO Spotfire server

1. Within TIBCO Spotfire Professional, open the
   \texttt{ASG\_HOME/templates/spotfire/ASG\_LogData.dxp} Spotfire visualization
   file by selecting the menu option File -> Open.

2. If the analysis file opens with \textit{Missing Information Link} warnings, follow the steps:
   
   — Select the \textbf{Browse for the missing information link} radio button and click
     the OK button.
   
   — In the \textbf{Select Information Link} dialog window, select the
     \texttt{ASG\_TRANSACTIONS} information link (with the paperclip icon next to it)
     and click the OK button.
   
   — Repeat these steps for the following missing information links:
     \begin{itemize}
     \item \texttt{ASG\_TRANSACTION\_DETAILS}
     \item \texttt{ASG\_TRANSACTION\_MESSAGES}
     \item \texttt{ASG\_KPI}
     \item \texttt{ASG\_THROTTLE\_USAGE}
     \end{itemize}

3. Note that the analysis file opens without any errors. This file does not show
   any data as the ASG Central Logger database is empty.

4. Save the \texttt{ASG\_LogData.dxp} analysis file in the Spotfire Server by selecting
   menu option File -> Save as -> Library Item. In the Save As Library Item
   dialog window that opens, click on the Finish button.

5. Click the Close button in the dialog window.

6. After the tests are run, use the menu option File -> Reload Data to refresh the
   data. Verify that the audit trail data from the database is loaded in the
   visualization file.

7. Select File -> Exit to close the Spotfire Professional.
Chapter 15  Basic Deployment

This chapter explains the concepts and process for basic deployment of TIBCO API Exchange Gateway engine and its components.

Topics

- Overview, page 454
- Deploying TIBCO API Exchange Gateway Processing Units, page 455
- Deploy Using Monitoring And Management Server, page 456
- Configuring for TIBCO BusinessEvents DataGrid WKA Discovery, page 464
- Configuring Cluster Discovery and Internal Communication, page 465
- Site Topology Reference, page 472
- Working With Site Topology Editor, page 480
Overview

After the configuration is completely defined using the TIBCO API Exchange Gateway user interface, you should test the configuration by running the gateway engine and its components at the command line.

This section explains the process and configuration required to deploy the gateway core engine and its components in a single server or distributed environment.
Deploying TIBCO API Exchange Gateway Processing Units

The full deployment of API Exchange Gateway contains the deployment of following components, which can be done on a single server or can be distributed across multiple servers.

<table>
<thead>
<tr>
<th>Component</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gateway core engine</td>
</tr>
<tr>
<td>Central Logger</td>
</tr>
<tr>
<td>Global Throttle Manager</td>
</tr>
<tr>
<td>Cache Manager (Optional)</td>
</tr>
<tr>
<td>Cache Clearing Manager (Optional)</td>
</tr>
</tbody>
</table>

See High Availability Deployment Of Runtime Components on page 485 for deployment details of runtime components.

Requirements For Deployment

TIBCO API Exchange Gateway software packages the required files for deployment of its engine and other processing units, by default. Typically, a deployment requires one or more Enterprise Archive (EAR) files and one or more Cluster Deployment Descriptor (CDD) files. You must ensure the third party jars are available at runtime, if applicable.

In case, if any customizations are made in the ASG_DefaultImplementation project of the gateway, you will need to configure the CDD file settings and build the EAR file.

Deployment Options

You can deploy the gateway core engine and the other processing units in the following ways:

- Using TIBCO API Exchange Gateway Monitoring and Management Server. This is the recommended way. See Deploy Using Monitoring And Management Server, page 456.

- Run the gateway core engine and run time components at the command-line. See Running Processing Units At Command Line on page 484.
Deploy Using Monitoring And Management Server

The Monitoring and Management Server component allows you to deploy the cache-based API Exchange Gateway engines, and then monitor and manage a deployed cluster. It works with the following sub-components:

- **Site topology editor**
  The site topology editor is a canvas-based editor, which allows users to set the deployment configuration of the cluster by specifying its hosts and deployment mappings. See Working With Site Topology Editor on page 480.

- **MM Console**
  MM Console is a web-based dashboard that enables you to monitor the deployment and perform various operations on the cluster.

### Deployment Pre-Requisites

The Monitoring and Management Server requires a site topology file, which contains all the deployment time information. See Site Topology Overview, page 470 for details on site topology file.

TIBCO API Exchange Gateway packages a sample site topology file to deploy the gateway caching enabled core engine and global throttle manager on a single host.

**Sample Site topology file**

You can find a sample of the site topology file shipped with the product installation. This file is located as:

`ASG_HOME/mm/bin/asg_default.st`

### Configuration Settings Before You Deploy

This section explains the configuration settings required for the Monitoring and Management server.
Configure JMX properties in Gateway Engine TRA File

After the gateway engines are started, and join the cluster, they use JMX MBeans to expose monitoring and management information to the MM server, and to allow remote method invocation. The JMX port number must be specified before the engine’s JVM starts. A variable for the port number is provided in the TRA file so that the actual value can be specified before the engine starts.

To Configure JMX Properties

JMX properties are provided in the shipped ASG_HOME/bin/asg-engine.tra file but are commented:

```
#java.property.be.engine.jmx.connector.port=%jmx_port%
#java.property.be.engine.jmx.connector.authenticate=false
```

1. Open the following file for editing:
   
   ASG_HOME/bin/asg-engine.tra

2. Uncomment the following properties:
   
   ```
   java.property.be.engine.jmx.connector.port=%jmx_port%
   java.property.be.engine.jmx.connector.authenticate=false
   ```

To Enable Monitoring and Management

To expose JMX for monitoring and management (without authentication), uncomment this property:

```
java.property.be.engine.jmx.connector.port=%jmx_port%
```

Ensure that the value of the port property is set to this literal value: %jmx_port%. The actual value is substituted at runtime. When you use the MM console to start the gateway engines remotely, MM reads the port number from the PU configuration settings in the site topology file.

When more than one PU (engine) is deployed to the same host, ensure that a different JMX port is used for each of the PUs, in the site topology file.

To Enable JMX MBeans Authentication

The following property enables authentication:

```
java.property.be.engine.jmx.connector.authenticate=true
```
Edit MM.cdd File

The CDD file used by the MM server is the \texttt{MM.cdd}. In this file, you must specify the path to the site topology file. The \texttt{MM.cdd} file is a XML file and can be edited using any XML or text editor.

Before you make any changes to the \texttt{MM.cdd} file, make sure to make a back up of this file.

1. Navigate to the \texttt{ASG_HOME/mm/bin} directory.
2. Edit the MM.cdd file and change the value of the \texttt{be.mm.topology.file} property, if required. This property must be set to the fully qualified path of the site topology file for the cluster to be monitored. Specify the location of the site topology file you want to be loaded by the MM server.

   By default, the property is set to the sample of a site topology file, as shown below:

   \[
   \text{<property name="be.mm.topology.file" value="ASG_HOME/mm/bin/asg_default.st"/>}
   \]

   The sample site topology file is located as:

   \texttt{ASG_HOME/mm/bin/asg_default.st}

   Whenever you make changes to the \texttt{MM.cdd} file you must restart the MM server so that it uses the updated values.

Edit Site Topology File in a Text Editor

The site topology file contains deploy time information such as what processing units to deploy to specific computer/hosts in your environment. You need to know information about the computers that will host the agents you intend to deploy, for example the information about the machines’ operating system and IP address.

An annotated site topology file that can be used as template is located at:

\texttt{ASG_HOME/mm/bin/asg_default.st}

You only need to edit the topology file if the defaults don't serve your purposes. This happens typically when the gateway engines are to be deployed on a different host than the one where the Monitoring and Management server is running.

For example, edit the following settings in the \texttt{ASG_HOME/mm/bin/asg_default.st} file for your deployment (if the default values are not correct):
<host-resources>
  <host-resource id="HR_0">
    <hostname>127.0.0.1</hostname>
    <ip>127.0.0.1</ip>
    <user-credentials password="asg" username="asg"/>
    <os-type>Windows</os-type>
  </host-resource>
See Site Topology Reference, page 472 for details on the XML elements defined in the topology file.

⚠️ Make sure that you take a backup copy of the topology file before you make any changes.

### Install and Configure Software for Remote Start and Deployment

The following table shows which software utilities can be used to perform remote operations. Information about installation and configuration of each software utility follows. Use the software utility’s documentation for more details.

<table>
<thead>
<tr>
<th>Software</th>
<th>Deployment</th>
<th>Remote Start, Stop</th>
<th>Remote Method Invocation</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSH</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>PsTools (Windows)</td>
<td>Yes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TIBCO Hawk</td>
<td>Yes</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>JMX (required)</td>
<td></td>
<td></td>
<td>Yes</td>
</tr>
</tbody>
</table>

Although it is possible to use more than one utility for the machines in the cluster, it is recommended that you use only one across all the machines. Ensure that the software is installed and running on all relevant machines.

⚠️ The software you use on each machine in the monitored cluster is specified in the cluster’s site topology file. See Start PU Method Setting on page 477.

**SSH** Only SSH software enables MM to deploy TIBCO API Exchange Gateway engine to the predefined hosts, that is, those configured in the monitored cluster’s site topology file. SSH can also be used to start remote engines.
The SSH utility is available on UNIX machines by default. No action is required. On Windows machines you must install an SSH server, as explained next.

### Installing and Configuring an SSH Server

If you want to use SSH on Windows machines, you must download the software and install it. Many SSH servers are available. For Windows, OpenSSH and Copssh are supported.

### Configuring OpenSSH

TIBCO has tested with OpenSSH software. See TIBCO API Exchange Gateway readme file for specific versions that are supported.

If you use the OpenSSH server, note the following when installing OpenSSH:

- The OpenSSH package is not a part of the default Cygwin installation. During its installation, ensure that you select the OpenSSH package. Also select the option `Select required packages (RECOMMENDED)` to install all the required packages to satisfy the dependencies. See openSSH server documentation for details.

- Accept the default username suggested when configuring the OpenSSH server and provide a password for the username.

- For deployment and starting PUs, use an username that is ssh enabled to login to the remote host. For example, ssh username@hostname. The credentials of the username can be specified in the host settings of the site topology file, Host Settings User and Password fields. See Host Settings on page 475.

  However, if you choose to use a different username, you must ensure that the username is added to the SSH server.

### Configure User Authorization for Administrator and User Roles

MM authorization uses two preconfigured roles. These roles are specified in the provided passwords file that is used for file-based authentication:

```
ASG_HOME/mm/config/users.pwd
```

The file as shipped contains the following entries:

```
jdoe:A31405D272B94E5D12E9A52A665D3BFE:MM_ADMINISTRATOR;
mm_user:11b2016b63c99ef7ab6d6d716be7b78e:MM_USER;
admin:21232f297a57a5a743894a0e4a801fc3:MM_ADMINISTRATOR;
```

If you add more users ensure that they have the appropriate role. Note that role names are case sensitive:

- **MM_ADMINISTRATOR** Users with this role can execute methods, for example to deploy, start, and stop engines, and invoke method operations.
**MM_USER** Users with this role can view MM Console, but cannot deploy, start, or stop engines, or invoke method operations.
Configure MM Console Properties

Properties dealing with the Console are configured in the following file:

```
ASG_HOME/MM/web-root/app_config.xml
```

Update the file as needed to set values for the following properties.

Table 88  MM Console Configuration Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>debugMode</td>
<td>Set to true to enable more detailed error messages. Default is false.</td>
</tr>
<tr>
<td>Demo Mode</td>
<td>If demo mode is enabled, chart updates are faked with random values based on the most recent value. Default is false.</td>
</tr>
<tr>
<td>updateInterval</td>
<td>Defines the time interval (in seconds) between two consecutive calls from MM Console to the MM server. The UI is refreshed after each update interval: the panes and tables with statistics are populated with the newly received data, and the topology tree is updated with the last state of the cluster. Default is 5.</td>
</tr>
<tr>
<td>failedPaneThreshold</td>
<td>Maximum ratio of failed pane updates to number of displayed panes, before a system crash is assumed. If the number of failed panes exceeds the threshold an error displays in the console, <em>Lost connection to data server</em>. The user clicks OK and is logged out. Default is 0.2.</td>
</tr>
<tr>
<td>logoURL</td>
<td>Path to the image file for the company logo (or other image as desired). The image file must be stored within the BE_HOME/MM/web-root folder. The logoURL value is the relative location of the image file within the web-root folder. For example, if the image is in this location: web-root/images/logo.jpg, then the value of logoURL would be images/logo.jpg. The image displays in the upper left corner. The images size must be no more than 32 by 32 pixels.</td>
</tr>
<tr>
<td>chartStyles</td>
<td>You can configure preferences such as colors used for various chart elements. Follow the documentation in the file for each element.</td>
</tr>
</tbody>
</table>
Configuring for TIBCO BusinessEvents DataGrid WKA Discovery

If you are using the TIBCO BusinessEvents DataGrid cache provider, and you have configured ASG_DefaultImplementation project to discover cluster members using well-known addresses (WKA) then you must make some additional changes to the ASG_DefaultImplementation project CDD so that MM can monitor and manage the cluster.

For more details about WKA discovery in a TIBCO BusinessEvents DataGrid cluster refer to the sections Configuring the TIBCO BusinessEvents DataGrid Discover URL, page 466 and Configuring Cluster Discovery and Internal Communication, page 465.

To Configure ASG_DefaultImplementation Project’s CDD

1. Start TIBCO API Exchange Gateway Studio.
2. Open the ASGDefaultImplementation project.
3. Open the asg_core.cdd CDD file in the CDD editor.
4. Add the following property to the cluster properties sheet.
   
   be.mm.cluster.as.listen.url MMHostIP: Port

   Specify the IP of the computer hosting the MM server, and an unused port.

5. Add the value of the be.mm.cluster.as.listen.url property to the list of addresses in the be.engine.cluster.as.discover.url property. The discovery property should be set at the cluster level (so the value is identical for all potential cluster members).

   The discovery URL for well-known address configuration uses the following format:

   tcp://ip:port[ip:port]*

6. Save.
Configuring Cluster Discovery and Internal Communication

When you add a CDD file and select Cache OM type, you must configure how the members of the cache cluster discover each other at runtime and communicate with each other once the cluster is established.

To Configure a TIBCO BusinessEvents DataGrid Cluster (Metaspace)

An active LAN connection (device enabled and network cable plugged in) is required for TIBCO BusinessEvents DataGrid to work.

1. Add a CDD file or open the CDD file you added.
2. Select the Cluster tab > Properties node on the left and on the right, add the following two properties, as needed:
   - be.engine.cluster.as.discover.url
   - be.engine.cluster.as.listen.url

The properties can be omitted if you use PGM multicast with default values.

See the following sections for details on configuring these properties:
— Configuring the TIBCO BusinessEvents DataGrid Discover URL, page 466
— Unicast (Well-Known Address) Cluster Member Discovery, page 467
— Configuring the TIBCO BusinessEvents DataGrid Listen URL, page 468

3. If you use unicast (well-known address) discovery, and you use TIBCO API Exchange Gateway Monitoring and Management server for monitoring and management, you must also do the following (in the asg_core.cdd CDD file):
   a. Add the following property to the cluster properties sheet.
      - be.mm.cluster.as.listen.url MMHostIP: Port
      Specify the IP of the computer hosting the MM server, and an unused port.
   b. Add the value of the be.mm.cluster.as.listen.url property to the list of addresses in the be.engine.cluster.as.discover.url property, which should be present at the cluster level (so the value is identical for all potential cluster members).

The discover URL for well-known address configuration uses the following format:
   tcp://ip:port[;ip:port]*
Configuring the TIBCO BusinessEvents DataGrid Discover URL

When a cluster starts up, and also when new members join a cluster, a discovery process enables the members to discover each other. The discover URL specifies how an engine (node) listens for discovery requests from nodes attempting to join the cluster.

After the discovery is complete, the members communicate internally using a listen URL (explained in Configuring the TIBCO BusinessEvents DataGrid Listen URL on page 468).

Two types of discovery are available:

- Multicast discovery (PGM)
- Unicast discovery (TCP), also known as "well-known address" discovery

Configuration for both discovery methods is explained below.

If No Other Cluster Members are Started

If a newly started node does not discover any running cluster nodes, the behavior is different depending on the type of discovery used:

- If multicast discovery is used, the newly started node becomes the first node of a newly started cluster.
- If unicast (well-known-address) discovery is used there are two cases:
  - If the address of the newly started node is not in the discover URL’s list then it continues to wait for other well-known nodes to start, and a warning is written to the console while it waits.
  - If the address of the newly started node is in the discover URL’s list, then it becomes the first node of a newly started cluster.

Multicast (PGM) Cluster Member Discovery

The discover URL for multicast discovery uses PGM (Pragmatic General Multicast) protocol.

The discovery property is be.engine.cluster.as.discover.url. For multicast discovery, the value is a URL with the following format:

```
tibpgm://destinationPort/network/
```

The default value for the URL equates to the following value:

```
//7888;/239.8.8.9/
```
Specify the parameters as follows.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>destinationPort</td>
<td>Specifies the destination port used by the PGM transport. Must be the same value on all machines in the cluster. Default value is 7888</td>
</tr>
<tr>
<td>network</td>
<td>Specifies the IP address of the interface to be used for sending multicast packets, and the multicast group address to be used. The format is as follows: interface;multicast group address The value for interface is unique to a node. It must also be the same in both the discovery and the listen URLs for a node. If there are multiple interfaces on one machine, specify the interface you want to use and do not rely on the default value. The value for multicast group address must be the same on all machines in the cluster. The default value for interface is the first available interface provided by the operating system hosts file for the machine. <strong>Note</strong> If the desired interface is not listed in the hosts file then PGM picks the first available interface in the file. (On most operating systems, this file is called the /etc/hosts file.) If the first interface is the loopback interface (127.0.0.1) then PGM fails to start. In this case you would see a stacktrace exception in the log file such as the following: SYS_ERROR (multicast_error - (8) grp_iface not a valid multicast interface) To resolve this issue, either modify the hosts file, or provide the desired interface explicitly in the network argument. The default value for multicast group address is the multicast group address as 239.8.8.9.</td>
</tr>
</tbody>
</table>

**Unicast (Well-Known Address) Cluster Member Discovery**

If you cannot or do not wish to use multicast discovery in your environment, then configure unicast discovery, also known as "well-known address" or WKA discovery. These "well-known addresses" enable a newly started node to discover existing members. Unicast discovery uses the TCP protocol.
The discovery property is `be.engine.cluster.as.discover.url`. For unicast discovery, the value is a semicolon-separated list comprising a sub-set of all the listen URLs (which are different for each PU), using this format:

```
tcp://ip:port[;ip:port]*/
```

### One cluster node in the WKA list must be running at all times
At least one cluster node specified in the well-known address list must be running at all times, so that other new members can join the cluster (metaspace). If all nodes specified in the well-known address list stop, then other nodes that are still running continue to function, but they print warnings to the console and no new members can connect to this cluster.

### For WKA discovery, make discover URL a cluster-level property and listen URL a PU-level property
The discover URL property (`be.engine.cluster.as.discover.url`) must be present and configured identically for all potential cluster members. Therefore add this property at the cluster level of the CDD file. The listen URL property (`be.engine.cluster.as.listen.url`) must be present and configured differently for each possible cluster member. Therefore add this property at the PU level.

### Configuring the TIBCO BusinessEvents DataGrid Listen URL
The listen URL is used for direct communication between the members of the metaspace. It is configured the same way for multicast and for unicast discovery (see Configuring the TIBCO BusinessEvents DataGrid Discover URL on page 466). The listen URL value must be different for each cluster member, so configure it at the PU level.

The listen URL uses this format:

```
tcp://interface:port[-EndPort |]*/
```

The cluster member binds to the specified interface and the specified port when creating the TCP socket. Specify the parameters as follows.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>interface</code></td>
<td>To specify a value, use the desired IP address. The value for <code>interface</code> must be the same in both the discovery and the listen URLs for a node. If there are multiple interfaces on one machine, specify the interface you want to use and do not rely on the default value. The default value for <code>interface</code> is the first available interface provided by the operating system for the machine.</td>
</tr>
</tbody>
</table>
Multiple Nodes on One Machine

If multiple nodes (engines) are running on one machine, identify each uniquely. Use the same value for `interface`, but a different value for `port` for each node.

Auto-incrementing Within a Range of Ports

If a machine has blocked some ports in the default range, or if you want to use a different range, you can configure the listen URL to start with a specified IP address and port, and optionally provide an upper limit. If the specified port is not available, TIBCO API Exchange Gateway auto-increments the port until it finds an available port, up to the specified upper limit, if any. To specify a specific range use this format:

```
tcp://interface:Port-EndPort/
```

For example, given the following listen URL, TIBCO API Exchange Gateway attempts to open port 8000 and if it is not available it tries the next port number, until it finds an available port, up to 9000 (inclusive). If none is available, it keeps retrying. Make some ports in the specified range available so that the cluster nodes can start.

```
tcp://interface:8000-9000/
```

To specify a range with the upper limit of unsigned short minus one, use this format:

```
tcp://interface:Port-*/
```
Site Topology Overview

The site topology file contains deployment time information such as which processing units to deploy to specific hosts in your environment. You need to know information about the computers that will host the agents you plan to deploy, for example the operating system and IP address.

The MM server uses the SSH software as the remote invocation software to start remote processes on unix machines.

- Changes to the EAR file do not affect the topology configuration. However if the cluster, processing unit, or agent definitions in the CDD file change, you must recreate the site topology file using the updated CDD.
- If you change the site topology, you must restart the MM server.

1. Configure Cluster Properties

2. Add Deployment Units (DUs)  Add DUs as needed. For each DU, specify the following:
   - The deployment location of the CDD and EAR files. MM copies the files to the specified location at deploy time.
   - One or more processing unit configurations (PUCs). You’ll configure the PUCs in the next step.

3. Add Processing Unit Configurations (PUCs) to DUs  For each PUC, select one processing unit (PU) from the list of PUs defined in the CDD file. Set deploytime properties such as the JMX ports used by MM to communicate with the deployed engine.

4. Add Hosts  Here you specify the host configuration, including the software used on the remote hosts to start remote gateway engines (processes). You must map the DU’s to the hosts where you want to deploy them. Multiple hosts can reuse the same deployment unit configuration.

In the topology file, you reference two locations for the CDD and EAR files. The files in each location must be exact copies:

- Master CDD and EAR files: You specify the location of the master CDD and EAR files. These copies must be manually copied to the specified location on the MM server host, for use in deployment.
- Deployed CDD and EAR files: In the deployment unit settings, you specify where MM will place the CDD and EAR files when it performs deployment to a remote host.
The project and master CDD can have the same location if you configure the site topology file and run the MM server in the same host. These two sets of fields are available in case you are configuring the topology file on a different machine from the MM server machine.

- All locations specified must already exist. The software does not create directories.
- Use the correct path for the operating system of the host (For example, linux versus windows).
- **Limitation: One CDD and EAR file per Cluster Machine:** Currently deployment is at the Machine level and each machine can have only one copy of the deployed CDD and EAR files. If you specify multiple DUs for the same host, problems may occur because CDD and EAR files are copied only to the location specified for the first DU.

**Deployment-Specific Processing Units**

In general, you can reference one processing unit multiple times to create different processing unit configurations (PUCs). However processing units that have deployment-specific settings cannot be used in this flexible manner.

**Agent-Instance-Specific Properties**

If a processing unit contains agent-instance-specific properties (such as agent key and priority settings), you must use it in only one PUC, which must be used in only one DU, that must itself used only once in the deployment.

**Host-Specific Processing Units**

Processing units can have host-specific settings. If a deployment unit contains a PUC that references such a processing unit, you must link it only to the appropriate host, for deployment.
Site Topology Reference

Site Settings

Table 89  Site Topology — Site Settings

<table>
<thead>
<tr>
<th>Property</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Site Name</td>
<td>Site name. Default value is the name of the site topology file.</td>
</tr>
<tr>
<td>Description</td>
<td>Description of the site, as desired.</td>
</tr>
</tbody>
</table>

Cluster Settings

Table 90  Site Topology — Cluster Settings

<table>
<thead>
<tr>
<th>Property</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cluster Name</td>
<td>Read-only field displaying the cluster name specified in the CDD. This name is set in the Cluster Name field of the CDD editor.</td>
</tr>
<tr>
<td>Project CDD</td>
<td>Location and name of project CDD. This is the location used by TIBCO API Exchange GatewayStudio for configuration of the site topology.</td>
</tr>
<tr>
<td>Master CDD</td>
<td>Location and name of the master CDD. This is the location used by the MM server.</td>
</tr>
<tr>
<td>Master EAR</td>
<td>Location and name of the master EAR. This is the location used by the MM server.</td>
</tr>
</tbody>
</table>
Deployment Unit Settings

Table 91  Site Topology — Deployment Unit Settings

<table>
<thead>
<tr>
<th>Property</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deployment Unit Name</td>
<td>Name of the deployment unit. Name must be unique.</td>
</tr>
<tr>
<td></td>
<td>It can be helpful to include the operating system of the host to which you will deploy this DU in the DU name. If a DU contains any host-specific settings, it’s also a good idea to put the host name in the DU name.</td>
</tr>
<tr>
<td></td>
<td><strong>Note</strong> Paths in different operating systems are specified using different tokens. Even if the DUs are identical in all other respects, you must create different DUs for different operating systems.</td>
</tr>
<tr>
<td></td>
<td>Default value is DU_n where n is a number that increments each time you add a DU to the diagram.</td>
</tr>
<tr>
<td>Deployed CDD</td>
<td>Absolute file path to the location in the remote host to where MM server will deploy the copy of the master CDD referred by this DU.</td>
</tr>
<tr>
<td>Deployed EAR</td>
<td>Absolute file path to the location where the MM server will deploy the copy of the master EAR used by this DU.</td>
</tr>
<tr>
<td>Processing Unit Configurations</td>
<td>Displays a list of processing unit configurations.</td>
</tr>
</tbody>
</table>

Processing Unit Settings

Table 92  Site Topology — Processing Unit Settings

<table>
<thead>
<tr>
<th>Property</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processing Unit Configuration Name</td>
<td>The name that identifies this configuration of the processing unit, as specified in the Processing Unit setting (see below). The name must be unique within a DU.</td>
</tr>
<tr>
<td></td>
<td>The processing units settings are configured in the CDD.</td>
</tr>
</tbody>
</table>
Use As Engine Name

Check this checkbox to use the value of the Processing Unit Configuration Name field as the engine name.

It is recommended that you use the same choice across all processing units in the cluster.

Processing Unit

Select the processing unit you want to use. Only processing units configured in the CDD selected as the Project CDD appear in the list. The same processing unit can be used in multiple PUCs.

Number of Agents

Displays the number of agents in the selected processing unit. (Not present in the site topology XML file.)

JMX Port

JMX port used by MM to perform monitoring and management. Required.

When more than one PU is deployed on the same host (in one DU or multiple DUs), you must ensure the JMX port in each of these PUs is different.
Host Settings

The host settings are defined under `<host-resources>` tag by the following variables:

Table 93  Site Topology — Host Settings

<table>
<thead>
<tr>
<th>Property</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>General Settings</td>
<td></td>
</tr>
<tr>
<td>hostname</td>
<td>Name of the computer hosting the TIBCO API Exchange Gateway deployment (including the domain extension). Used for remote access. Used to identify the host in the MM user interface. Required. To validate the hostname, ping the host using this name from the MM server machine. <strong>Note:</strong> Specify the exact name of the host. Errors in the host name result in the host appearing in the MM Console UI as an predefined machine. Do not, for example, use localhost.</td>
</tr>
<tr>
<td>ip</td>
<td>IP address of the host machine. Used for remote access. Required. You can use 127.0.0.1 (localhost loop back IP address) for engines running on the same machine as the MM server.</td>
</tr>
</tbody>
</table>
### Table 93  Site Topology — Host Settings (Cont’d)

<table>
<thead>
<tr>
<th>Property</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>username</strong></td>
<td>User name to log onto the host machine. The user credentials are used for remote deployment and execution, including starting a process unit. At runtime, a dialog box pops up to authenticate the user, for example when deploying a PU. If you provide a username and password here, then the dialog is prepopulated with these values. You can enter different values at runtime as needed. If you do not provide the credentials here, then you must provide them at the pop-up dialog. You can specify a local user or a domain user. Enter details for the user you specified for the remote connection utility you are using. For example, if you use SSH utility, you specify the username you use for SSH utility.</td>
</tr>
<tr>
<td><strong>password</strong></td>
<td>Password of the user referenced in the User Name field. The password is encrypted. See notes in User Name section.</td>
</tr>
<tr>
<td><strong>os-type</strong></td>
<td>Operating system of the host machine. See the product readme for a list of supported platforms.</td>
</tr>
</tbody>
</table>
### Start PU Method Setting

Choose the method that MM will use to start this processing unit on remote machines:

- Use Hawk
- Use PsTools
- Use SSH. If you choose Use SSH, and do not want to use the default port number of 22, then also enter the port. The host must accept a secure connection through this port. Using the default port is generally recommended because it is also the default port used by most Linux SSH servers.

Note that a username and password for the remote machines are required for MM to connect (see notes for User Name and Password fields).

See [Install and Configure Software for Remote Start and Deployment on page 459](#) for details on each option.

Default is SSH. Default SSH port number is 22.

<table>
<thead>
<tr>
<th>Property</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start PU Method Setting</td>
<td></td>
</tr>
</tbody>
</table>

---

*Table 93  Site Topology — Host Settings (Cont’d)*
Working with Cluster Explorer

Active and inactive nodes are shown in Cluster Explorer for a quick view of system health. Using Cluster Explorer, you can use functionality available at various nodes on the left, and you can view information about that node level on the right.

The Cluster Explorer shows the hierarchy of cluster members. Inactive agents (which could be standby agents or failed agents) are dimmed.

The structure of the cluster member hierarchy is as follows:

Site
Cluster
  Machine (host name)
    Process (Processing Unit or JVM process ID)
    Agent (ActiveService Gateway Class agent)
Cache Objects

Where:
- Site is the root.
- Cluster shows the name of the cluster being monitored.
- Machine shows one or more machines within the cluster. They run the cluster processes (process units or engines).
- Process shows each of the JVM processes (gateway engines) running on a machine. The label for a process that was predefined in the topology file is the process unit ID assigned in the file, concatenated with the process ID enclosed in parentheses. The label for an unpredefined process is the JVM process ID.
- Agent lists all agents, regardless of their type, running in the JVM process.
- The Cache Objects panel shows all the objects stored in the cache, without regard to their physical location in the TIBCO API Exchange Gateway cluster.
- Machines, TIBCO API Exchange Gateway engines, and agents are all members of the TIBCO API Exchange Gateway cluster.

Navigating Cluster Explorer

To navigate the cluster explorer, do any of the following:
- Expand Cluster Explorer and select the member you want to work with or whose metrics you want to see. Metrics display on the right.
- Click an inactive cluster member to display the last available health metrics for that member.
• Click the minimize button in the Cluster Explorer title bar to minimize the explorer panel.

Starting, Stopping, Pausing, and Resuming Gateway Engines

To Start, Stop, Pause, or Resume an Engine
1. From Cluster Explorer, select the engine you want to start, stop, pause, or resume. (You resume a paused engine.)
2. Click the appropriate icon button: **Start, Stop, Pause, or Resume**
3. Verify the login details and click **OK**.
Working With Site Topology Editor

The site topology editor is built within TIBCO API Exchange Gateway Studio. Before you begin ensure that you have a valid CDD file. The processing units that you will deploy to the various hosts are defined in the CDD.

See Site Topology Overview on page 470 for important information. See Site Topology Reference on page 472 for detailed information on the settings.

To Add a Site Topology Diagram

1. Open the project in TIBCO API Exchange Gateway Studio. Select the project root, right-click and select New > Other > TIBCO BusinessEvents > Site Topology.
2. At the New Site Topology Wizard, enter a unique Cluster Topology name and optional description.
3. Select the Cluster Deployment Descriptor (CDD) that contains the PU definitions and other details you want to use. Only CDD files within the studio project you are configuring are available for selection.
4. Click Finish. You see the site topology editor, showing the cluster bar icon and site globe icon, ready for you to build the site topology diagram:

To Configure the Site Topology

1. On the canvas, click the site icon (the globe). In the Site Properties tab you can change the site name and description as desired. Other fields are view-only.
2. Click the blue bar, which represents the cluster. In the Cluster Properties tab, specify the following:

   — The location of the Project CDD, which must be available to the TIBCO API Exchange Gateway Studio Explorer, represents the CDD file you selected in the wizard.

   — The location of the Master CDD and EAR on the MM server. The MM server reads these files and copies them to the remote deployment locations specified in the DUs.

   If the MM server is on a different machine from the machine where you are running TIBCO API Exchange Gateway Studio, you must copy the master files to the specified location so they are available for use by MM.

   In the topology file, you reference three locations for the CDD and EAR files. The files in each location must be exact copies:

   • Project CDD file: In the cluster configuration tab, you specify a locally available copy of the project CDD, used only at design-time for configuring the topology file in TIBCO API Exchange Gateway Studio.

   • Master CDD and EAR files: Also in the cluster configuration tab, you specify the location of the master CDD and EAR files. These copies must be manually copied to the specified location on the MM server, for use in deployment.

   • Deployed CDD and EAR files: In the Deployment Unit settings, you specify where MM will place the CDD and EAR files when it performs deployment.

   The project and master CDD can be in the same location if you are using one machine to configure the topology file and to run MM server. These two sets of fields are available in case you are configuring the topology on a different machine from the MM server machine.

   • All locations specified must already exist. The software does not create directories.

   • Use the correct path delimiter for the operating system of the host machines.

3. Add one or more deployment units: In the Site Topology section of the palette to the right of the canvas, click the deployment unit icon and then click the canvas. A DU icon appears on the canvas. Click again to add more DUs. Right-click the canvas to stop adding units. (If the palette is not visible, click Window > Show View > Palette or Window > Reset Perspective.)

   A connection arrow appears automatically, connecting each deployment unit to the cluster.
4. Click each DU in turn and configure the Deployment Unit Properties tab settings.
   - In the Deployed CDD and Deployed EAR fields, specify the directory where MM will put the files when it deploys this DU to the host machine.
   - Click Add and add one or more processing unit configurations (PUCs) to the deployment unit.

See Site Topology Reference: Deployment Unit Settings on page 473 for details.

5. Configure processing unit configurations (PUCs): In the DU property sheet, double click one of the listed PUCs (or click the PUC icon shown in the diagram). The Processing Unit Configuration properties appear. Configure the PUC as follows (and configure the rest of the PUCs in a similar way):
   - As desired replace the default PUC name with a name of your choice.
   - As desired, select the option to use the PUC name as the engine name.
   - Select the processing unit to use for this configuration. The list displays the PUs defined in the CDD. You can use one PU in multiple DUs, as appropriate. When you select a PU, the number of agents defined for it displays. (No agent level configuration is done in the site topology editor.)
   - Set the JMX port for MM to perform monitoring and management. When multiple PUs are running on one host, each PU must have a different JMX port. You can reuse ports on different hosts, however.

See Site Topology Reference: Processing Unit Settings on page 473 for details, especially on JMX port.

6. Add one or more hosts. In the Site Topology section of the palette, click the Host icon, and then click the canvas. A host icon appears on the canvas. Click again to add more hosts. Right-click in the canvas area to stop adding hosts.

7. Click each host icon in turn and configure the Properties tab.
   - In the General tab, configure the host name (including the domain extension), IP, and as needed, the user name and password, and operating system.
   - In the Start-PU-Method tab, select an option to use for MM to start a processing unit on this host.

See Site Topology Reference: Host Settings on page 475 for details.
8. Connect each host to one or more deployment units:
   — In the Links section of the palette, click the Connect icon.
   — Click a host and then the title bar of the deployment unit you want to deploy on that host.

   Right-click to stop connecting.

   To remove a connection, right-click to stop connecting, then right-click a connection arrow and click the Delete option.

9. Save.

The canvas has a property sheet too: click an empty area of the canvas to see the number of deployment units and number of processing units in the site topology.

**Specifying the Site Topology Files for the MM Server**

After creating the site topology file, you should specify the location of this file using the property `be.mm.topology.file` in the `MM.cdd` file. The MM server loads the site topology file specified in the property `be.mm.topology.file` in the `MM.cdd` file.

If a site topology file (with the same name) is present under `ASG_HOME\mm\config` and is also specified using the property `be.mm.topology.file`, then the file specified by the property `be.mm.topology.file` will be parsed and loaded by the MM server.
Running Processing Units At Command Line

See Processing Units of TIBCO API Exchange Gateway Engine on page 57 for the processing units details of API Exchange Gateway.

You can run any processing unit as follows:

1. Navigate to ASG_HOME/bin directory.
2. Type the command as below:
   asg-engine -u asg_processing_unit_name -a asg_config_name

   where,
   - `asg_processing_unit_name` is the name of the processing unit name.
   - `asg_config_name` is the name of the project configuration. The project configuration is not required for the gateway management components such as global throttle manager, central logger, cache clearing manager.

   For example, run the asg-gtm processing unit as follows:
   asg-engine -u asg_gtm -a ASG_Configuration_Name

   where `ASG_Configuration_Name` is the project configuration of the API Exchange Gateway.
Chapter 16  High Availability Deployment Of Runtime Components

This chapter describes the fault tolerance and high availability configuration setup of runtime components of API Exchange Gateway software.

Topics

- Introduction on page 486
- Configuration For High Availability Setup on page 495
- Configuring Load Balancer on page 496
- Configuring Apache Modules For Core Engines on page 499
- Configuring Fault Tolerance Parameters on page 503
- Configuring Gateway Core Engines on page 504
- Configuring Cache Agent on page 504
- Configuring Cache Cleanup Agent on page 506
- Configuring Global Throttle Manager on page 507
- Configuring Central Logger on page 509
- Rendezvous Session Connection Parameters For Apache Module on page 516
- Rendezvous Session Connection Parameters For Gateway Core Engine on page 517
- Rendezvous Session Connection Parameters For Gateway Core Engine and Global Throttle Manager Communication on page 519
- Rendezvous Session Connection Parameters For Gateway Core Engine and Central Logger Communication on page 521
Introduction

This chapter describes the deployment of runtime components in a high availability setup. The runtime components are gateway core engine, cache agents, global throttle manager, central logger and cache cleanup agents. You should deploy the runtime components in such a way that they should be highly available in production systems to achieve maximum functionality of the gateway.

API Exchange Gateway Engine is deployed as a cluster of core engines that together act as a single logical gateway. The core engines in the cluster can run on a single server or in a distributed environment across multiple physical or virtual servers.

Typically for production deployment requirements, you will require to add additional instances of gateway core engines. The architecture of API Exchange Gateway Engine has been designed so that when multiple core engine instances are deployed in a gateway cluster, the key management functions such as global throttle management, cache management, cache cleanup management and central logging are coordinated across all core engine instances.

However, as the levels of transactions increase, it is likely that there will be a corresponding increase in management activity. To avoid the possible impact of management activity upon the core engines of the API Exchange Gateway, the management components (throttle manager, cache cleanup manager, and central logger) and the TIBCO Spotfire Servers should be moved onto separate servers.

If multiple instances of the core engine are deployed in a cluster, you must start the cache agent instances explicitly once you start the core engines.

TIBCO Rendezvous is used for the communication between most of the run-time components of the gateway. The run-time components share a single set of configuration files.

The set of configuration files should be stored on a shared file system so that they are accessible for reading to each of the runtime components.

Overview

This section gives an overview and configuration setup of the deployment of runtime components with multiple instances for load balancing and high availability.

The administrators deploy two or more instances of gateway core engines in production to achieve high availability through load balancing. That is, a load balancer routes request messages to multiple core engine instances. The core engines together can handle more messages than just one core engine instance.
running. Also, such deployment makes the run time components highly available to process the requests with minimum or no down time. The core engine instances, thus, share the load of requests when large number of requests are received from the clients.

All the runtime components of the API Exchange Gateway are deployed in the same cluster except the central logger component. The central logger instance must be running in a separate cluster.

**Overview Of Deployment**

Figure Overview of Runtime components For High Availability on page 488 shows a high level overview of a deployment model of runtime components in a cluster for high availability setup.
Figure 32  Overview of Runtime components For High Availability
Operational Layer Components

This section briefly describes the information for load balancing setup of gateway core engines and cache agents.

Load Balancing Gateway Core Engines

All the instances of the core engine automatically behave in a fault tolerant manner. The load balancer distributes the load of requests within all active agents in the same group as per the configuration setup described in Configuring Load Balancer on page 496 section. If any core engine instance fail, the load balancer distributes the load between the remaining active core engine instances in the group.

There is no discovery protocol between Apache server and the gateway core engine. If one of the gateway core engine instance goes down, the Apache server is not able to determine that this instance is not available and keeps sending the requests to the gateway core engine which results in all those requests timing out. For this reason, it is recommended that you should not use a single Apache server to send request messages to multiple instances of the core engines.

For the Apache server and core engine configuration setup, you must consider following tips:

- There should be a single Apache server per core engine instance. For example, if you plan to run two instances of gateway core engine instances, you must setup two Apache servers, one Apache server for each core engine instance.

- You must configure health monitor of the load balancer to call the gateway ping operation so that the load balancer can determine which instances of the gateway core engines are up and running. See Configuring Load Balancer, page 496 for configuration details. If any of the Apache server or the gateway core engine instance goes down, the load balancer considers it as a failure to forward the request and routes the request to the second active instance of Apache server as configured in the load balancer group.

Cache Agents

API Exchange Gateway supports in memory caching of the data. Fault tolerance of cache agents is handled transparently by the object management layer. For fault tolerance of cache data, the only configuration task is to define the number of backups you want to keep. Use of a backing store is not needed as the cache agents are only used to implement the association cache, which is automatically rebuilt after complete failure as new transactions are handled by the API Exchange Gateway.
Gateway Management Layer Components

The components of the Gateway Management Layer should be deployed once in an active-passive configuration setup. The central logger and the global throttle manager need to have a single running instance at all times to ensure that the gateway core engine operates without loss of functionality.

Therefore the central logger and global throttle manager are deployed in fault tolerant configuration with one active engine (central logger instance or global throttle manager instance) and one or more standby agents on a separate host servers. Such fault-tolerant engine setup can be configured in the cluster deployment descriptor (CDD) file by specifying the maximum number of one active agent for either of the agent classes and by creating multiple processing unit configurations for both the global throttle manager and the central logger agent. Deployed standby agents maintain a passive Rete network. They do not listen to events from channels and they do not update working memory. The standby agents take over from the active instance in case it fails.

The Cache Cleanup agent of the Gateway Management Layer have no direct impact on the functionality of an operating API Exchange Gateway Engine instance. It is recommended that multiple versions of cache cleanup agents are deployed across host servers with one active instance running. If the running instance goes down, one of the other instances is started to regain full gateway functionality.

Example Deployment Model

Figure Deployment of Runtime Components In a Cluster on page 492 illustrates the deployment of runtime components on the machines in a cluster environment for high availability.
Figure 33  Deployment of Runtime Components In a Cluster
It is recommended to have the runtime components deployed in a cluster as follows:

- **Gateway Core Engines** - You must deploy all the instances of the gateway core engines as Active Active engine instances.

- **Cache Agents** - Fault tolerance of cache agents is handled transparently by the object management layer. At least two cache agents should be deployed to provide the fault tolerance of the cached data.

- **Cache Cleanup Agents** - You must deploy multiple instances of the cache cleanup agents with one instance as Active (primary) and the other instances are deployed as Passive (secondary).

- **Global Throttle Manager** - You must deploy multiple instances of the global throttle manager with one instance as Active (primary) and the other instances are deployed as Passive (secondary).

- **Central logger** - You can deploy multiple instances of the central logger with one instance as Active (primary) and the other instances are deployed as Passive (secondary).

The figure Deployment of Runtime Components In a Cluster on page 492 illustrates an example deployment model in which the components are deployed as follows:

- All the instances of the runtime components of the gateway, except the central logger are deployed on the machines in one cluster (Cluster A). The two instances of the central logger are deployed on the machines in a second cluster (Cluster B).

- On machine A, the Apache server with Apache module instance 1 runs.

- On machine B, the Apache server with Apache module instance 2 runs.

- On machine C in Cluster A, the components are deployed as below:
  - The gateway core engine instance 1 runs as an Active instance.
  - One instance of the cache agents runs.
  - The cache cleanup agent instance 1 runs as an Active instance.

- On machine D in Cluster A, the components are deployed as below:
  - The gateway core engine instance 2 runs as an Active instance.
  - Second instance of the cache agents runs.
  - The cache cleanup agent instance 2 runs as an Passive instance.
On machine E in Cluster A and Cluster B, the components are deployed as below:

- One instance of the global throttle manager runs as an Active instance.
- One instance of the central logger runs as a Passive instance.

On machine F in Cluster A and Cluster B, the components are deployed as below:

- Second instance of the global throttle manager runs as a Passive instance.
- Second instance of the central logger runs as an Active instance.

All the runtime components communicate using the Rendezvous channel. In the example deployment model as shown in Deployment of Runtime Components In a Cluster on page 492, it's assumed that all the machines running various instances of runtime components are in the same subnet.

You can add more instances of the components as required. The gateway core engine instances, if added, must be configured as Active for load balancing. The instances of cache cleanup agent, global throttle manager and central logger, if added, must be configured as Passive for fault tolerance.

Configuration For High Availability Setup

This section describes the configuration of multiple processing units for load balancing and fault tolerance to achieve high availability and high throughput.

### Configuration Files

The high availability configuration of runtime components of the gateway can be setup in the `asg_core.cdd` and `asg_cl.cdd` files respectively. An agent class is an agent type, defined in the CDD file that deploys as an agent instance.

The `asg_core.cdd` file defines the configuration for the following processing units:

- **default, asg-core, asg-caching-core** - These processing units refer to the gateway core engine. The configuration settings of `core-class` (Inference) agent class defines the runtime behaviour of `default`, `asg-core`, `asg-caching-core` agent processing units.

- **asg-cache** - This processing unit refers to the cache agent. The configuration settings of `cache-class` (Cache) agent defines the runtime behaviour of `asg-cache` processing unit.

- **asg-cache-cleanup** - This processing unit refers to the cache cleanup agent. The configuration settings of `cache-cleanup-esp` (Query) and `cache-cleanup-scheduler` (Inference) agent classes define the runtime behaviour of `asg-cache-cleanup` processing unit.
• `asg-gtm` - This processing unit refers to the global throttle manager. The configuration settings of `gtm-class` (Inference) agent class defines the runtime behaviour of `asg-gtm` processing unit.

`asg_cl.cdd` file defines the configuration for the following processing unit:

• `asg_cl` - This processing unit refers to the Central Logger. The configuration settings of `BusinessEvents_Archive` (Inference) agent class define the runtime behaviour of `asg_cl` processing unit.

To setup the deployment configuration of runtime components for high availability, do following steps:

• Configure an IP based load balancer. See Configuring Load Balancer on page 496.

• Configure Apache module per core engine instance. See Configuring Apache Modules For Core Engines on page 499.

• Configure machines for cluster. See Cluster Configuration For Runtime Components on page 500.

• Configure the gateway core instances. See Configuring Gateway Core Engines on page 504.

• Configure the cache agent instances. See Configuring Cache Agent on page 504.

• Configure the cache cleanup agent instances. See Configuring Cache Cleanup Agent on page 506.

• Configure the global throttle manager instances. See Configuring Global Throttle Manager on page 507.

• Configure the central logger instances. See Configuring Global Throttle Manager on page 507.

**Configuring Load Balancer**

You must use a HTTP load balancer with API Exchange Gateway. For example, you can use the F5 load balancer. This section describes the configuration steps for F5 load balancer. If you use a different load balancer, you should refer to the documentation of the load balancer to complete the following tasks.

**Create A Health Monitor For Gateway Core Engines**

To create a monitor for the load balancer, perform the following steps:

3. Go to the Health Monitor tab of the navigation pane.
4. Expand Monitors node under Local Traffic on left. Click on "+" to create a new monitor.

5. Verify that the New Monitor screen appears.

6. For General Properties section, input the values for the following fields:
   a. Name: Enter the name of the monitor. (For example, asgping)
   b. Type: Select the type as HTTP from drop down list.

7. For Configuration field, select BASIC from drop down list.

8. Input the values for the following fields under Configuration section as below:
   a. interval - desired interval: Set this value in seconds (for example, 2)
   b. timeout - desired timeout: Set this value in seconds (for example, 15)
   c. Send String - Set the value to a string to send to API Exchange Gateway Engine in the http URL. GET /ping.
   d. Receive String - Set the value to a string, expected to receive as a response from API Exchange Gateway Engine. Set this as "ASG is alive" (without quotes).

9. Click on Finished button.

**Create A Load Balancing Pool**

This task is required to create a pool to load balance HTTP connections. You can use the configuration utility to create a load balancing pool.

To create a pool, do following steps:

1. Go to the Main tab of the navigation pane.

2. On the left, under Local Traffic > Virtual Servers, select Pools node. Click on "+" to create a new pool.

3. Verify that the Pools screen opens.

4. In the upper-right corner of the screen, click Create. Verify that New Pool screen opens.

5. Type the name of the pool in the Name field. (for example, asg_http_pool.)

6. For Configuration field, select BASIC from drop down list.
7. Under Configuration section, Go to Health Monitors sub-section.
   a. Add a health monitor as follows:
      — Select an existing health monitor from the Available field. For example, select asgping health monitor as created in Create A Health Monitor For Gateway Core Engines on page 496 section.
   b. Click the Move button (<<) to move the monitor from the Available field to the Active field. For example move the asgping health monitor from the Available box to the Active box.
   c. Verify that the asgping health monitor appears under Active box.
8. Under Resources setting, select an appropriate algorithm from the drop down listbox for Load Balancing Method field. For example, you can select Round Robin.
9. Add the pool members as below:
   a. Select the New Address option.
   b. In the Address box, type the IP address of the machine where the gateway core engine runs.
   c. In the Service Port field, enter the service port of http module on that machine. (for example, type 80, or select HTTP).
   d. Click Add.
   e. You can add a pool member for each server in the pool using steps b, c, and d, if needed.
10. Click Finished button.

Create a Virtual Server
This task is required to create a virtual server. The virtual server processes the HTTP traffic and send it to the pool.
You can use the Configuration utility to create the virtual server. To create a virtual server, do following:
1. Go to the Main tab of the navigation pane.
2. Expand Local Traffic node. Click Virtual Servers.
3. Verify that the Virtual Servers screen opens.
4. In the upper-right corner of the screen, click Create. Verify that the New Virtual Server screen opens.

If the Create button is not available, make sure that you have permission to create a virtual server. Ask your administrator to grant permissions to create virtual server for your role.

5. In the Name box, type a name for the virtual server (example vs_http).
6. In the Destination box, verify that the type of virtual server is Host.
7. In the Address box, type an IP address for the virtual server.
8. In the Service Port box, type 80, or select HTTP from the list.
9. In the Configuration area of the screen, locate the HTTP Profile setting and select http. This assigns the default HTTP profile to the virtual server.
10. In the Resources section of the screen, locate the Default Pool setting and select the name of the HTTP pool created in the Create A Load Balancing Pool on page 497 section.
11. From the Default Persistence profile setting, select profile_none as the profile.
12. Click Finished.

**Configuring Apache Modules For Core Engines**

You must configure the Rendezvous subject for each core engine instance and Apache module so that the requests from the Apache server are forwarded to the correct instance of the core engine.

- If the Rendezvous daemon is running with non-default parameters on the machines where Apache server and the gateway core engine runs, you must configure the Rendezvous session connection parameters as described in Rendezvous Session Connection Parameters For Apache Module on page 516 and Rendezvous Session Connection Parameters For Gateway Core Engine on page 517.

**Set the Rendezvous Connection Parameters For Apache Module**

For each machine running the Apache server, set the Rendezvous parameters for Apache module as below:

2. Edit the mod_ASG.conf file.
3. Set the Asgsub\text{ject} parameter, which must be defined unique for each machine. For example, this can be set be below:
\texttt{AsgSubject \ ASG\_CoreEngine1\_Subject1}

4. You must set the Rendezvous connection parameters as described in the table \textit{Rendezvous Session Connection Parameters For Apache Module on page 516}, if the Rendezvous daemon is running with non-default session parameters.

5. Save the changes in the file.

\textbf{Set the Rendezvous Connection Parameters For Gateway Core Engine}

For each machine running the gateway core engine instance, set the properties for Rendezvous connection as below:

1. Navigate to the \texttt{ASG\_CONFIG\_HOME} directory.
2. Edit \texttt{asg\_properties} file.
3. Set the \texttt{tibco\_clientVar\_ASG/\_modRV/\_north\_request} property, which must be defined unique for each machine. For example, this can be set as below:
\texttt{tibco\_clientVar\_ASG/\_modRV/\_north\_request=ASG\_CoreEngine1\_Subject1}

The value of \texttt{tibco\_clientVar\_ASG/\_modRV/\_north\_request} property must match with the Asgsub\text{ject} defined in the \texttt{mod\_ASG\_conf} file for that machine.

4. You must set the Rendezvous connection parameters as described in the table \textit{Rendezvous Session Connection Parameters For Gateway Core Engine on page 517}, if the Rendezvous daemon is running with non-default session parameters.

5. Save the changes in the file.

\textbf{Cluster Configuration For Runtime Components}

This section explains the configuration to setup the cluster properties. A cluster defines the machines running the gateway core engines, global throttle manager, cache agent and cache cleanup agents. You must define all run time components in the same cluster except the central logger component. The central logger instance must be deployed in a separate cluster.
The cluster properties are defined in the `asg_core.cdd` and `asg_cl.cdd` files.

- All the runtime components (gateway core engine, global throttle manager, cache agent and cache cleanup agent) must run in one cluster.
- The central logger instance must run in a separate cluster.

To define the machines in a cluster, you must set the discover URL for the cluster.

**Discover URL**

The discover URL specifies how the gateway engine (node) listens for discovery requests from nodes attempting to join the cluster. When a cluster starts up, and also when new members join a cluster, a discovery process enables the members to discover each other. The discover URL specifies how an gateway engine (node) listens for discovery requests from nodes attempting to join the cluster.

The discovery URL for well-known address configuration uses the following format:

```
tcp://machine1_ipaddress:port;machine2_ipaddress:port;machine3_ipaddress:port/
```

where `machine1`, `machine2`, `machine3` belong to same cluster.

For each machine in the cluster, the discover URL must have the IP addresses of the machines with respective ports which belong to the same cluster. These machines in the cluster run the components (gateway core engine, cache agent, global throttle manager, cache cleanup agent).

**Configure Discover URL**

The discover URL for a cluster is defined in the `Properties` section of the `Cluster` tab in the `ASG_HOME/bin/asg_core.cdd` and `ASG_HOME/bin/asg_cl.cdd` files. The discover URL is defined on each machine where the runtime component runs.

- To set the discover URL for a cluster containing the machines where gateway core engine, cache agent, cache cleanup agent, global throttle manager instances are running, you must edit the `asg_core.cdd` file on each machine in the cluster. See Edit `asg_core.cdd` File To Set Discover URL (using text editor) on page 502.

- To set the discover URL for a cluster containing the machines where central logger instances are running, you must edit the `asg_cl.cdd` file on each machine in the cluster. See Edit `asg_cl.cdd` File To Set Discover URL (using text editor) on page 502.
Edit asg_core.cdd File To Set Discover URL (using text editor)

To edit the discover URL and listen URL, perform the following steps:
1. Open the ASG_HOME/bin/asg_core.cdd file for editing.
2. Edit the following properties and set the value to the actual IP addresses of the machines in a cluster, and an unused port.

For example, if the gateway core engine instance 1 is running on Machine C, gateway core engine instance 2 is running on Machine D, cache agent instance 1 is running on Machine C, cache agent instance 2 is running on Machine D, cache cleanup agent instance 1 is running on Machine C, cleanup agent instance 2 is running on Machine D, global throttle manager instance 1 is running on Machine E, global throttle manager instance 2 is running on Machine F, then set the URL as below, where port1 is an unused port:

   <property-group comment="" name="cluster">
     <property name="be.engine.cluster.as.discover.url" value="tcp://Machine C_IP_address:port1;Machine D_IP_address:port1;Machine E_IP_address:port1;Machine F_IP_address:port1"/>
   </property-group>

3. Save the changes to asg_core.cdd file.

Edit asg_cl.cdd File To Set Discover URL (using text editor)

To set the discover URL for a cluster containing the machines where the central logger instances are running, you must edit the ASG_HOME/bin/asg_cl.cdd file on each machine as below:

1. Open the ASG_HOME/bin/asg_cl.cdd file for editing.
2. Edit the following properties and set the value to the actual IP addresses of the machines in a cluster, and an unused port.

For example, if the central logger instance 1 is running on Machine E, central logger instance 2 is running on Machine F, then set the URL as below:

   <property-group comment="" name="cluster">
     <property name="be.engine.cluster.as.discover.url" value="tcp://Machine E_IP_address:port2;Machine F_IP_address:port2"/>
   </property-group>

3. Save the changes to the asg_cl.cdd file.

To edit the asg_core.cdd and asg_cl.cdd files using API Exchange Gateway Studio to set the discover URL, see Editing CDD file using API Exchange Gateway Studio on page 513.
Configuring Fault Tolerance Parameters

For the gateway core engines, you should run the multiple instances across the servers in a load balanced setup. The configuration for load balanced setup is defined in the `asg_core.cdd` file. See Configuring Gateway Core Engines, page 504.

For the cache cleanup agent, global throttle manager and central logger components, the instances must be deployed as one active engine and one or more stand by agents that run on a separate server. See Configuring Cache Cleanup Agent on page 506, Configuring Global Throttle Manager on page 507 and Configuring Central Logger on page 509.

The maximum number of active instances are configured as an agent class configuration parameter. An agent class is an agent type, defined in the CDD file that deploys as an agent instance.

Following parameters define the fault tolerant configuration of run time components:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Max Active</td>
<td>• Specifies the maximum number of active agents. This value is used for fault tolerance. Deployed agents that are acting as standbys can take over from active instances that fail. In many cases, there is no need to keep standby instances.</td>
</tr>
<tr>
<td>Priority</td>
<td>• Specifies the priority of the agent for fault tolerant setup of agents. The priority is set at the processing unit level.</td>
</tr>
<tr>
<td></td>
<td>• The priority indicates the order in which standby agents become active, and conversely, the order in which active agents become standbys, when new agents join the cluster. The lower the number, the higher the agent is in the activation priority list. For example, an agent with priority 2 has a higher priority than an agent with a priority of 6. This value determines the order of each instance of an agent class for startup, as well as fail over and fail back in fault tolerance situations.</td>
</tr>
<tr>
<td></td>
<td>• If the priority values are same for agents in fault tolerant setup (active passive), the agent which is started first gets the higher priority to become active agent instance.</td>
</tr>
</tbody>
</table>

This section explains the steps to set the maximum number of active instances and priority required for the high availability of runtime components.
Configuring Gateway Core Engines

You must run the multiple gateway core engine instances as Active to achieve load balancing. Out of the box, API Exchange Gateway provides the configuration parameters for load balancing, which are set in the ASG_HOME/bin/asg_core.cdd file. You can run multiple instances of the core engine using the configuration parameters set in the ASG_HOME/bin/asg_core.cdd file. By default, you can run unlimited number of instances.

For the gateway core engine Active Active instances, Max Active and Priority parameters are not applicable.

Example Settings For Gateway Core Engine Instance

Just as a reference, this section shows the sample settings for the gateway core engines in the ASG_HOME/bin/asg_core.cdd (XML file). By default, no changes are required to run multiple instances of the gateway core engine.

```xml
<inference-agent-class id="core-class">
  <load>
    <max-active/>
  </load>
</inference-agent-class>
```

Configuring Cache Agent

The cache agents behave according to the cache object management configuration set in the Cluster tab of the CDD file. Refer to CDD Cluster Tab Cache OM Settings on page 505 table for the object management configuration parameters.
Out of the box using the default configuration in the `asg_core.cdd` file, you can run more than one instance of cache agent.

Table 95  CDD Cluster Tab Cache OM Settings

<table>
<thead>
<tr>
<th>Property</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cache Agent Quorum</td>
<td>Specifies a minimum number (quorum) of storage-enabled nodes that must be active in the cluster when the system starts up before the other agents in the cluster become fully active. The property does not affect the running of the deployed application after startup (though a message is written to the log file if the number of cache agents running falls below the number specified in this property). Default is 1.</td>
</tr>
<tr>
<td>Number of Backup Copies</td>
<td>The number of backup copies (also known as the backup count) specifies the number of members of the distributed cache service that hold the backup data for each unit of storage in the cache. Value of 0 means that in the case of abnormal termination, some portion of the data in the cache will be lost. A backup count of 1 means one server plus one backup are needed, that is, two cache agents. Default is 1.</td>
</tr>
</tbody>
</table>

Example Settings For Cache Agent Instance

This section shows the sample settings for cache agents in the `ASG_HOME/bin/asg_core.cdd` (XML) file. You can use the sample settings as a reference by editing the `ASG_HOME/bin/asg_core.cdd` (XML) file in a text editor.

```
<object-management>
    <cache-agent-quorum>1</cache-agent-quorum>
    <backup-copies>1</backup-copies>
</object-management>
```
Configuring Cache Cleanup Agent

To run the cache cleanup agent instances in a fault tolerant mode, you must set the Max Active property. Optionally, you can configure Agent Priority parameter. The Max Active and Priority parameters are defined in the asg_core.cdd file. See Fault Tolerant Configuration Parameters on page 503 for description of parameters.

To set the number of the active instances and priority for the cache cleanup agent instances, perform the following steps:

1. Navigate to the ASG_HOME/bin directory.
2. Edit the asg_core.cdd file. You can edit the file either using a text editor or using API Exchange Gateway Studio. See Editing Cluster Deployment Descriptor (CDD) File on page 513.
3. If you use the API Exchange Gateway Studio, set Max Active property for cache cleanup agents as below:
   a. Select Agent Classes tab.
   b. Select the cache-cleanup-scheduler (Inference) agent.
   c. On the right side, set the following property as below:
      Max Active 1
   d. Select the cache-cleanup-esp (Query) agent.
   e. On the right side, set the following property as below:
      Max Active 1
4. Save the changes.
5. Optionally, you can set the Priority for the processing units as below using API Exchange Gateway Studio:
   a. Select Processing Units tab.
   b. Select the asg-cache-cleanup node.
   c. On the right side, in the Agents section, set the priority for following agents:
      — cache-cleanup-scheduler
      — cache-cleanup-esp
   d. Double click the Priority column to set a value, if required.
6. Save the changes.
Example Setting For Cache Cleanup Agent Instance

This section shows the sample settings for cache cleanup agents in the ASG_HOME/bin/asg_core.cdd (XML) file.

You can use the sample settings as a reference to set the values by editing the ASG_HOME/bin/asg_core.cdd (XML) file in a text editor:

```xml
<query-agent-class id="cache-cleanup-esp">
  <load>
    <max-active>1</max-active>
  </load>
</query-agent-class>

<inference-agent-class id="cache-cleanup-scheduler">
  <load>
    <max-active>1</max-active>
  </load>
</inference-agent-class>

<processing-unit id="asg-cache-cleanup">
  <agents>
    <agent>
      <ref>cache-cleanup-esp</ref>
      <key/>
      <priority/>
    </agent>
    <agent>
      <ref>cache-cleanup-scheduler</ref>
      <key/>
      <priority/>
    </agent>
  </agents>
</processing-unit>
```

Configuring Global Throttle Manager

To run the global throttle manager instances in a fault tolerant mode, you must set the Max Active property. Optionally, you can configure Agent Priority parameter. The Max Active and Priority parameters are defined in the asg_core.cdd file. See Fault Tolerant Configuration Parameters on page 503 for description of parameters.
To set the number of the active instances and priority for the global throttle manager instances, perform the following steps:

1. Navigate to the `ASG_HOME/bin` directory.
2. Edit the `asg_core.cdd` file. You can edit the file either using a text editor or using API Exchange Gateway Studio. See Editing Cluster Deployment Descriptor (CDD) File on page 513.
3. Set the Max Active property as below for the Agent Classes > gtm-class (Inference) agent. See Set Max Active on page 515 using API Exchange Gateway Studio.
   
   Max Active 1

4. Save the changes.

5. Optionally, you can set the priority for the `asg-gtm` processing unit using API Exchange Gateway Studio as follows. See Set Priority on page 515.
   a. Go to Processing Units tab.
   b. Select the `asg-gtm` node.
   c. Go to Agents section, and select the row with gtm-class agent.
   d. Set a value for Priority. For example, you can set this value to 2.

6. Save the changes.

**Example Setting For Global Throttle Manager Instance**

This section lists the sample settings for the global throttle manager instance in the `ASG_HOME/bin/asg_core.cdd` (XML) file.

You can use the sample settings as a reference to set the values by editing the `ASG_HOME/bin/asg_core.cdd` (XML) file in a text editor:

```xml
<inference-agent-class id="gtm-class">
  <load>
    <max-active>1</max-active>
  </load>
</inference-agent-class>

<processing-unit id="asg-gtm">
  <agents>
    <agent>
      <ref>gtm-class</ref>
      <key/>
    </agent>
  </agents>
</processing-unit>
```
Configuring Central Logger

To run the central logger instances in a fault tolerant mode, you must set the Max Active property. Optionally, you can configure Agent Priority parameter. The Max Active and Priority parameters are defined in the ASG_HOME/bin/asg_cl.cdd file. See Fault Tolerant Configuration Parameters on page 503 for description of parameters.

To set the number of the active instances and priority for the central logger instances, perform the following steps:

1. Navigate to the ASG_HOME/bin directory.

2. Edit the asg_cl.cdd file. You can edit the file either using a text editor or using API Exchange Gateway Studio. See Editing Cluster Deployment Descriptor (CDD) File on page 513. If you use API Exchange Gateway Studio, perform the following steps:
   a. Open the ASG_HOME/bin/asg_cl.cdd file using the steps described in Editing CDD file using API Exchange Gateway Studio on page 513.
   b. Select Agent Classes tab.
   c. Select the BusinessEvents_Archive (Inference) node.
   d. On the right side, set the following property as below:
      Max Active 1
   e. Save the changes.

3. Optionally, you can set the priority for the asg-cl processing unit using API Exchange Gateway Studio as follows. See Set Priority on page 515.
   a. Go to Processing Units tab.
   b. Select the asg_cl node.
   c. Go to Agents section, and select the row with BusinessEvents_Archive agent.
   d. Double click the Priority column to set a value. For example, you can set this value to 5.
   e. Save the changes.
Example Settings For Central Logger Instance

This section lists the sample settings for the central logger instance in the ASG_HOME/bin/asg_cl.cdd (XML) file.

You can use these sample settings as a reference to set the values for max-active and priority by editing the ASG_HOME/bin/asg_cl.cdd (XML) file in a text editor:

```xml
<inference-agent-class id="BusinessEvents_Archive">
    ..... 
    ..... 
    <load>
        <max-active>1</max-active>
    </load>

    <processing-unit id="asg-cl">
        <agents>
            <agent>
                <ref>BusinessEvents_Archive</ref>
                <key/>
                <priority>5</priority>
            </agent>
        </agents>
    </processing-unit>
</inference-agent-class>
```

4. Save the changes to the file.

Configure Rendezvous Session Connection Parameters

You must configure the Rendezvous session connection parameters for the gateway core engine instances to communicate with the global throttle manager instances and the central logger instances, in case, the Rendezvous daemon runs with the non-default session parameter settings on the machines where these components are running.

The parameters are defined in the ASG_CONFIG_HOME/asg.properties file and ASG_CONFIG_HOME/asg_cl.properties file. See following sections for the list of connection parameters:

- Rendezvous Session Connection Parameters For Gateway Core Engine and Global Throttle Manager Communication on page 519
• Rendezvous Session Connection Parameters For Gateway Core Engine and Central Logger Communication on page 521

The instances of the runtime components illustrated in the deployment Deployment of Runtime Components In a Cluster on page 492 assumes the following points:

• The gateway core engine instances and the global throttle manager instances are running on machines which are in the same subnet.

• The gateway core engine instances and the central logger instances are running on machines which are in the same subnet.
Appendix A

This appendix describes the following topics:

- Editing Cluster Deployment Descriptor (CDD) File on page 513
- Rendezvous Session Connection Parameters For Apache Module on page 516
- Rendezvous Session Connection Parameters For Gateway Core Engine on page 517
- Rendezvous Session Connection Parameters For Gateway Core Engine and Global Throttle Manager Communication on page 519
- Rendezvous Session Connection Parameters For Gateway Core Engine and Central Logger Communication on page 521
Editing Cluster Deployment Descriptor (CDD) File

You define the cluster member machines, processing units, and agents in the Cluster Deployment Descriptor (CDD) which is an XML file. The CDD file is configured in the CDD editor in API Exchange Gateway Studio.

You can edit any CDD file in the following ways:

- Using text editor.
- Using API Exchange Gateway Studio.

Editing CDD file using Text Editor

The CDD file is a XML file, you can use any text or XML editor to edit the file. To edit any property in the CDD file, locate the property and set the new value in the value field.

To edit the CDD file, do following:

1. Open the CDD file in a text editor.
2. Set the value of the property, as needed. For example, to edit the value of discover URL for a cluster, you can set the property as below:

   ```xml
   <property name="be.engine.cluster.as.discover.url" value="tcp://Machine1_IPAddress:port1;Machine2_IPAddress:port1;Machine3_IPAddress:port1" />
   ```

3. Save the changes to the file.

Editing CDD file using API Exchange Gateway Studio

You can edit a CDD file using the gateway studio as follows:

1. Navigate to the ASG_HOME/bin directory.
2. Copy the CDD file (for example, asg_core.cdd) file to ASG_HOME/projects/ASG_DefaultImplementation folder.
3. Navigate to the ASG_HOME/studio/eclipse directory.
4. Type the following command to start the TIBCO API Exchange Gateway Studio:

   **Windows platform**
   
   studio.exe

   **Unix platform**
   
   ./studio
5. If you are prompted, select or create the Eclipse workspace directory where your project files will be stored. (If you check the option to use this workspace as a default, you are not prompted again.)

6. Click OK.

7. Close the Welcome screen.

8. From the File menu select Import.

9. In the Import Select wizard, select an import source as General > Existing Projects into Workspace and click Next. You see the Import Projects dialog.

10. In the Import Projects dialog Select root directory field, browse to and select the project: ASG_HOME/projects/ASG_DefaultImplementation.

11. Click Finish.

12. In the Studio Explorer, expand the ASG_DefaultImplementation project node. Verify that you see the CDD file (for example, asg_core.cdd) file.

13. Double-click the CDD file (for example asg_core.cdd) file.

14. Select the appropriate tab to edit the properties. For example, see Set Discover URL on page 514, Set Max Active on page 515, Set Priority on page 515.

15. Save the changes to the file.

16. Back up the original CDD file (for example, asg_core.cdd) file in the ASG_HOME/bin directory.

17. Copy the modified CDD file (for example asg_core.cdd) file from ASG_HOME/projects/ASG_DefaultImplementation to the ASG_HOME/bin directory.

If you select the "Copy projects into workspace" option during the import of the project, then the modified asg_core.cdd file exists in the workspace directory. Make sure to copy the asg_core.cdd file from workspace directory to the ASG_HOME/bin directory.

Task E  Set Discover URL

a. To set the discover URL for a cluster, select cluster tab > Properties on the left. On the right, expand cluster node to edit the following properties:
   be.engine.cluster.as.discover.url

b. Set the values to the actual IP address of the machine, and an unused port. For example,
be.engine.cluster.as.discover.url=tcp://Machine1_IPAddress:port1;Machine2_IPAddress:port1;Machine3_IPAddress:port1;Machine4_IPAddress:port1/

c. Save the changes to the resource.

**Task F  Set Max Active**

This task explains the steps to setup the Max Active property for an agent.

a. Select Agent Classes tab.

b. Select an agent node for which you want to set the Max Active property.

For example, to set the Max Active property for global throttle manager engine, select the gtm-class(Inference) node.

c. On the right side, click the following property to set a value as below:

   Max Active 1

d. Save the changes to the resource.

**Task G  Set Priority**

This task explains the steps to setup the Priority property for a processing unit.

a. Select Processing Units tab.

b. Select the processing unit for which you want to setup the Priority property.

For example, to set the Priority property for global throttle manager processing unit, select the asg-gtm node.

c. On the right side, in the Agents section, select the row with gtm-class agent.

d. Double click the Priority column to set a value. For example, you can set this value to 2.

e. Save the changes to the resource.
Rendezvous Session Connection Parameters For Apache Module

This section lists the parameters required for Apache module to connect to the gateway core engine using Rendezvous daemon. The parameters are defined in the `mod_ASG.conf` file located under `ASG_HOME/modules/http_server/apache` directory.

Table 96  Rendezvous Session Connection Parameters For Apache Module

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AsgService</td>
<td>Specifies the Rendezvous daemon service parameter.</td>
</tr>
<tr>
<td></td>
<td>Default value is: 7500</td>
</tr>
<tr>
<td>AsgNetwork</td>
<td>Specifies the Rendezvous daemon network parameter.</td>
</tr>
<tr>
<td></td>
<td>Default value is: loopback 1o</td>
</tr>
<tr>
<td>AsgDaemon</td>
<td>Specifies the Rendezvous daemon parameter. This parameter is set as the tcp port.</td>
</tr>
<tr>
<td></td>
<td>Default value is: tcp:7500</td>
</tr>
</tbody>
</table>
| AsgSubject    | • Specifies the subject to which the Rendezvous daemon sends the client request. Default value is: _LOCAL.asg.north.request
|               | You can change the default value, as required.                              |
|               | • This value should match with the value set in the `tibco.clientVar.ASG/modRV/north_request` parameter of the `ASG_CONFIG_HOME/asg.properties` file. |
Rendezvous Session Connection Parameters For Gateway Core Engine

This section lists the properties required for the gateway core engine to listen to the messages from the Apache module using Rendezvous daemon. The properties are defined in the ASG_CONFIG_HOME/asg.properties file.

Table 97  Rendezvous Session Parameters For Apache Module And Gateway Engine Communication

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tibco.clientVar.ASG/modRV/facade_request</td>
<td>Specifies the subject name used by the Rendezvous daemon to listen to the requests from Apache module. The gateway core engine listens to the requests on the same subject.</td>
</tr>
<tr>
<td></td>
<td>• This property value must match with the subject value specified in the Apache module configuration file (mod_ASG.conf) for Apache server.</td>
</tr>
<tr>
<td></td>
<td>Default value is: _LOCAL.asg.north.request</td>
</tr>
<tr>
<td>tibco.clientVar.ASG/modRV/RvDaemon</td>
<td>Specifies the value of Rendezvous daemon for the gateway core engine to connect and listen for the requests from the Apache module.</td>
</tr>
<tr>
<td></td>
<td>Default value is: tcp:7500</td>
</tr>
<tr>
<td>tibco.clientVar.ASG/modRV/RvNetwork</td>
<td>Specifies the value of Rendezvous network for the gateway core engine to connect and listen for the requests from the Apache module.</td>
</tr>
<tr>
<td></td>
<td>This property value must match with the network value specified in the Apache module configuration file for Apache server.</td>
</tr>
</tbody>
</table>
Table 97  Rendezvous Session Parameters For Apache Module And Gateway Engine Communication

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tibco.clientVar.ASG/modRV/RvService</td>
<td>Specifies the value of Rendezvous service for the gateway core engine to connect and listen for the requests from the Apache module. Default value is: 7500</td>
</tr>
</tbody>
</table>
Rendezvous Session Connection Parameters For Gateway Core Engine and Global Throttle Manager Communication

This section lists the properties required for Rendezvous communication between the gateway core engine and the global throttle manager. The properties must be defined in the `ASG_CONFIG_HOME/asg.properties` file.

Table 98  Rendezvous Session Parameters For Core Engine And Global Throttle Manager Communication

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| tibco.clientVar.Common/Connections/RV/SubjectPrefix | Specifies the prefix for Rendezvous subject names used by the gateway core engine to communicate with other components.  
Default value is: TIBCO.ASG.INTERNAL |
| tibco.clientVar.ASG/GTM/RV/RvDaemon             | • Specifies the value of Rendezvous daemon for the gateway core engine to connect to the global throttle manager.  
Default value is: tcp:7500  
• By default, this property is not defined in the `ASG_CONFIG_HOME/asg.properties` file. You must explicitly add this. |
| tibco.clientVar.ASG/GTM/RV/RvNetwork            | • Specifies the value of Rendezvous network for the gateway core engine to connect to the global throttle manager.  
• By default, this property is not defined in the `ASG_CONFIG_HOME/asg.properties` file. You must explicitly add this. |
### Table 98  Rendezvous Session Parameters For Core Engine And Global Throttle Manager Communication

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| `tibco.clientVar.ASG/GTM/RV/RvService` | • Specifies the value of Rendezvous service for the gateway core engine to connect to the global throttle manager.  
  Default value is: 7500  
  • By default, this property is not defined in the `ASG_CONFIG_HOME/asg.properties` file. You must explicitly add this. |
Rendezvous Session Connection Parameters For Gateway Core Engine and Central Logger Communication

This section lists the properties required for Rendezvous communication between the gateway core engine and the central logger. The properties are defined in the `ASG_CONFIG_HOME/asg.properties` and `ASG_CONFIG_HOME/asg_cl.properties` files.

**Parameters for Rendezvous connection From Gateway Core Engine To Central Logger:**

The gateway core engine uses the Rendezvous daemon to send messages to the central logger. Set the properties for the gateway core engine to communicate with the Rendezvous daemon for the central logger in `ASG_CONFIG_HOME/asg.properties` file as below:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tibco.clientVar.Common/Connections/RV/SubjectPrefix</td>
<td>Specifies the prefix for Rendezvous subject names used by the gateway core engine to communicate with the other components. Default value is: TIBCO.ASG.INTERNAL</td>
</tr>
</tbody>
</table>
| tibco.clientVar.ASG/CL/RV/RvDaemon | - Specifies the value of Rendezvous daemon for the gateway core engine to send messages to the central logger. Default value is: tcp:7500  
- By default, this property is not defined in the `ASG_CONFIG_HOME/asg.properties` file. You must explicitly add this. |
Parameters for Rendezvous connection For Central Logger

The central logger uses the Rendezvous to listen to the messages from the gateway core engine. Set the properties for the central logger to communicate with the Rendezvous daemon in \texttt{ASG\_CONFIG\_HOME/asg\_cl\_properties} file as below:

\begin{table}[h]
\centering
\renewcommand{\arraystretch}{1.2}
\begin{tabular}{|l|l|}
\hline
Parameter & Description \\
\hline
tibco.clientVar.Common/Connections/RV/SubjectPrefix & Specifies the prefix for Rendezvous subject names used by the gateway engine to communicate with the central logger and the global throttle manager.  
& Default value is: \texttt{TIBCO.ASG.INTERNAL} \\
\hline
\end{tabular}
\end{table}

Table 99  Rendezvous Session Parameters For Core Engine To Communicate With Central Logger

\begin{table}[h]
\centering
\renewcommand{\arraystretch}{1.2}
\begin{tabular}{|l|l|}
\hline
Parameter & Description \\
\hline
tibco.clientVar.ASG/CL/RV/RvNetwork & \begin{itemize}
\item Specifies the value of Rendezvous network for the gateway core engine to send messages to the central logger. 
\item By default, this property is not defined in the \texttt{ASG\_CONFIG\_HOME/asg\_properties} file. You must explicitly add this.
\end{itemize} \\
\hline
tibco.clientVar.ASG/CL/RV/RvService & \begin{itemize}
\item Specifies the value of Rendezvous service for the gateway core engine to send messages to the central logger. Default value is: 7500 
\item By default, this property is not defined in the \texttt{ASG\_CONFIG\_HOME/asg\_properties} file. You must explicitly add this.
\end{itemize} \\
\hline
\end{tabular}
\end{table}

Table 100  Rendezvous Session Connection Parameters For Central Logger
### Table 100  Rendezvous Session Connection Parameters For Central Logger

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| **tibco.clientVar.ASG/CL/RV/RvDaemon** | - Specifies the value of Rendezvous daemon for the central logger to listen to the messages from the gateway core engine.  
  
  Default value is: tcp:7500  
  - By default, this property is not defined in the `ASG_CONFIG_HOME/asg_cl.properties` file. You must explicitly add this. |
| **tibco.clientVar.ASG/CL/RV/RvNetwork** | - Specifies the value of Rendezvous network for the central logger to listen to the messages from the gateway core engine.  
  - By default, this property is not defined in the `ASG_CONFIG_HOME/asg_cl.properties` file. You must explicitly add this. |
| **tibco.clientVar.ASG/CL/RV/RvService** | - Specifies the value of Rendezvous service for the central logger to listen to the messages from the gateway core engine.  
  
  Default value is: 7500  
  - By default, this property is not defined in the `ASG_CONFIG_HOME/asg_cl.properties` file. You must explicitly add this. |
For the communication between the gateway core engine and the central logger components, the Rendezvous session connection parameters in the 
\textit{ASG\_CONFIG\_HOME/asg.properties} file and
\textit{ASG\_CONFIG\_HOME/asg\_cl.properties} file must match.
Chapter 17  Advanced Features

This chapter describes some advanced functionality of TIBCO API Exchange Gateway.

Topics

- Cache Agent, page 526
- Hot Deployment Overview, page 527
- Extension Mechanism, page 528
Cache Agent

The cache agent stores cache data and are responsible for object management. A processing unit can have one cache agent only.

Using cache clustering technology, object data is kept in memory caches, with redundant storage of each object for reliability and high availability. Cache data is shared across all the gateway engines participating in the cluster. The purpose of cache agents is to store and serve cache data for the cluster.

The cache agents are used to implement the association cache and response cache functionality. The size of a cache can be unlimited or limited. Performance is best when all the data is in cache.

Only use an unlimited cache if you deploy enough cache agents to handle the data. Otherwise out of memory errors may occur.

How to run Cache Agent

You can start the cache agent as follows:

1. Open a terminal window.
2. Navigate to ASG_HOME/bin directory.
3. Type the following command to start up the cache agent:

   ./asg-engine -u asg-cache

Following command starts up the gateway core engine in cache agent enabled mode:

   ./asg-engine -u asg-caching-core
Hot Deployment Overview

You can make certain changes to a set of TIBCO API Exchange Gateway configuration, without having to shut down the gateway engine. This is known as hot deployment.

Hot deployment process suspends north inbound channels and waits for the time delay specified before hot deploying the configuration.

Due to a potential to cause an unexpected result to the transaction in process, it is strongly recommended that the hot deployment is used only in a development and testing environment.

Enabling Hot Deployment

By default, the hot deployment is not enabled. To enable the hot deployment feature, perform the following steps:

1. Navigate to the ASG_CONFIG_HOME/asg directory.
2. Edit the asg.properties file to change the value of the following property to true:
   
   tibco.clientVar.ASG/Deployments/AllowHotUpdate=true

3. Deploy and restart the gateway engines that need to be hot deployed engines. For example, asg_core or asg_caching_core.

Invoking Hot Deployment

Hot deployment can be invoked from an MBean client by invoking the refreshConfig method in com.tibco.asg domain for the appropriate gateway engine.

   refreshConfig method uses following parameters:

   • The first parameter is the location of the configuration directory used by the gateway engine.
   • The second parameter is the delay in milliseconds. If the delay value is -1, the gateway engine does not shutdown on any error during hot deployment. Otherwise, the gateway engine shuts down on error.
Extension Mechanism

Extension mechanism capability allows you to add custom stages in the default transaction processing pipeline. The custom stage is developed using the rules language within TIBCO API Exchange Gateway Studio.

Following features are implemented using the extension mechanism:

**Association Cache**
The gateway engine provides a mechanism to cache the previous acquired information retrieved from the external systems during the lookups. It uses that information later to optimize the time taken for routing of the requests.

**Response Cache**
The gateway engine has the capability to store the responses of requests in the cache clusters. It uses these responses for later requests. Response cache is implemented using association caches. This functionality allows the gateway engine to process the requests faster and also off load the service endpoints.

**Sequential Orchestration**
TIBCO API Exchange Gateway supports sequential orchestration. Sequential orchestration allows you to access multiple service endpoints by making a number of sequential calls to fulfill or authorize a request. With sequential orchestration, there is effectively a single outbound service invocation, preceded by one or more sidebound service invocations.

Sequential orchestration may use the associative and responses cache features to accelerate the processing of future requests, which helps to minimize the load on back-end systems.

Sequential orchestration allows you to access the external systems when one or more service requests are pipelined.

**Field Translation**
The gateway engine has the ability to call the external services to perform the translations of certain fields required for the main request processing. For example, if the requestor sends an product id in the request but the backend service requires the product name, then the gateway engine calls a east side service to translate the product id into product name. This translated value (product name in this case) is replaced in the main request payload and is used to invoke the back end service.
This cross referencing for lookups and data enrichments can use the association cache functionality for the faster processing of requests.

**Content based Authorization**

TIBCO API Exchange Gateway supports the partner authorization based on the content of the incoming request message. This functionality allows the partners to authenticate the references of the partner (for example, customers of the partners) which are sent in the content of the message.

For content based authorization, you have a single request which contains one or more customer references. API Exchange Gateway supports the authorization of each of the request individually by parsing the content of the request message.

Content based authorization uses Extension Mechanism capability which allows you to use the association cache functionality.

**LDAP Based Authorization for Partner References**

TIBCO API Exchange Gateway allows you to authorize the partner’s references in the content of the request message with a LDAP system. API Exchange Gateway provides a set of catalog and custom rule functions to support this functionality.
Appendix B  Configuration Tasks

This appendix explains few configuration tasks required for some functionality of the product.

Topics

- Cross-Origin Resource Sharing (CORS) Filter Properties, page 532
- Configure JMS Destinations for Southbound Service Operations, page 534
- Configure Async Mode for Southbound JMS Service, page 535
- Configure Retry parameters for HTTP / HTTP(s) Transport, page 536
- Enable detail level logging for Gateway, page 537
- Configure TIBCO Enterprise Message Service, page 538
- Configuring JMS northbound transport for XML, page 540
- Changing the Stack Size, page 543
- Modify Unicast Discovery URL in CDD file, page 544
- Generate Private Keys And Public Certificates with OpenSSL, page 547
Cross-Origin Resource Sharing (CORS) Filter Properties

TIBCO API Exchange Gateway can be configured to add servlet filters to its Tomcat engine. You can enable CORS by adding one or more CORS filter to the Tomcat server instance that runs the HTTP channels.

To add a CORS filter, set the following properties either in $ASG_HOME/bin/asg_core.cdd$ file or in the $ASG_HOME/bin/asg_engine.tra$ file:

The CDD file can be edited in the following ways:

- Using text editor.
- Using API Exchange Gateway Studio.

This section explains how to add the CORS filter properties in the $asg_core.cdd$ file using a text editor. Do the following:

1. Open the $ASG_HOME/bin/asg_core.cdd$ file for editing in a text editor.
2. Set the following properties as below:

   ```
   be.http.filter.cors.class=org.apache.catalina.filters.CorsFilter
   be.http.filter.cors.urlpattern.arbitraryUniqueName=urlPattern
   ```

   where,

   - `cors` identifies an instance of the filter. It is a non-empty string which should contain no dot and no equals sign.
   - `org.apache.catalina.filters.CorsFilter` is the name of the class that implements the filter. For CORS, this must be `org.apache.catalina.filters.CorsFilter`
   - `arbitraryUniqueName` is a non-empty string chosen arbitrarily by the user. This is used to make the name of the property unique.
     - `arbitraryUniqueName` cannot contain the equal sign.
     - `arbitraryUniqueName` cannot be shared by multiple properties that have the same `be.http.filter.filterName.urlpattern.prefix`.
   - `urlPattern` is the URL pattern that defines where the CORS filter is applied.
3. Save the changes to the file.

- Properties start with `be.http.filter.filterName` where `filterName` is the filter name.
- `filterName` cannot contain a dot character "."
- `filterName` is used both by API Exchange Gateway and by Tomcat server to find all the declarations that apply to the same filter.
- The type of Tomcat filter is defined by the value of the property `be.http.filter.filterName.class`

**Adding URL patterns**

To add additional url patterns, you can set following additional properties:

- `be.http.filter.cors.urlpattern.2`
- `be.http.filter.cors.urlpattern.3`

Ensure that the property names are unique when you add the properties for additional url patterns by adding a unique `arbitraryUniqueName`.

**Example Value**

*(URL Pattern)*

- `be.http.filter.cors.urlpattern.2=/some/Url*`
- `be.http.filter.cors.urlpattern.3=/anotherUrl`

**Add Filter Parameters**

To add filter parameters, set the properties by following the structure:

`be.http.filter.cors.param.paraName=paramValue`

where:

- `cors` identifies an instance of the filter.
- `paraName` is the parameter name. This should not contain no equals sign.
- `paramValue` is the parameter value.

**Example Value**

*(Filter Parameters)*

- `be.http.filter.cors.param.cors.allowed.origins=*`
- `be.http.filter.cors.param.cors.allowed.methods=GET,POST,HEAD,OPTIONS,PUT`

- Refer to [http://tomcat.apache.org/tomcat-7.0-doc/config/filter.html#CORS_Filter](http://tomcat.apache.org/tomcat-7.0-doc/config/filter.html#CORS_Filter) for a list of available parameter names and values.
Configure JMS Destinations for Southbound Service Operations

By default, the southbound service operations are sent to a single default queue for all service operations per ESB channel defined for a backend service. Users can configure a JMS destination name and type for a given channel for southbound service operation of ESB type.

The default queue are explained in Configure TIBCO Enterprise Message Service, page 538. You can override the default JMS destinations by configuring the destination and type.

To configure a destination and type for an ESB service type, perform the following steps:

1. Start the GUI, if not already started. See Starting GUI, page 166 for details.
2. Add a new service of ESB Type. See Add A New Target Operation, page 195 for details.
3. Set the following parameters to configure a destination for the service:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESB Channel</td>
<td>Define the number of predefined ESB channels.</td>
</tr>
<tr>
<td>Destination Name</td>
<td>Define a name of the custom queue or topic destination for JMS channel. This topic or queue should exist on the EMS server. For example, type a queue name as <code>asg.custom.requestQ</code> For example, type a topic name as <code>asg.custom.requestT</code></td>
</tr>
<tr>
<td>Destination Type</td>
<td>Select the type of the destination for JMS channel from the drop down list box. Valid values are QUEUE or TOPIC. If no value is given, then the QUEUE type is used as default. QUEUE type sets the JMS destination as queue. TOPIC type sets the JMS destination as topic.</td>
</tr>
</tbody>
</table>

4. Save the service configuration.
Configure Async Mode for Southbound JMS Service

TIBCO API Exchange Gateway provides the capability to send a JMS southbound service asynchronously. Users can configure async mode for southbound service request using the services tab on configuration GUI. When the southbound request is sent to the backend service in async mode, no response is expected from the southbound service, and the gateway does not wait for any southbound response from the backend service.

After the async request is sent to the backend service, a default northbound response payload is created and sent back as the northbound response. The default northbound response is shown as below:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<response>Request sent asynchronously.</response>
```

Users can use the reverse southbound mapping to customize the default northbound response.

Set the async mode for a service using the configuration GUI as below:

1. Start the GUI, if not already started. See Starting GUI, page 166 for details.
2. Add a new service of ESB Type. See Add A New Target Operation, page 195 for details.
3. Set the following field for the service as follows:
   Mode: ASYNC (select from the drop down list box)
4. Save the service configuration.
TIBCO API Exchange Gateway provides the retry mechanism in case when it fails to send the southbound request message to the backend service using http or transport.

Users can configure the parameters for retry of the request messages, in case, the request message is not delivered due to timeout or any network problems. The retry parameters for http/http(s) service type are available on the services tab of the configuration GUI.

Set the retry parameters for a http/http(s) service using the configuration GUI as below:

1. Start the GUI, if not already started. See Starting GUI, page 166 for details.
2. Add a new service of http/http(s) type. See Add A New Target Operation, page 195 for details.
3. Set the following fields:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Timeout</td>
<td>Set the value in milliseconds which is used as a timeout to use when invoking the backend service. The indicates the maximum time to wait before the northbound response is returned from a backend service.</td>
</tr>
<tr>
<td>Retry Count</td>
<td>Set the value to use as a number of retries for http connection.</td>
</tr>
<tr>
<td>Retry Interval</td>
<td>Set the value to use as an interval between http connection retries.</td>
</tr>
<tr>
<td></td>
<td>If set as 0, it means no retry is done between http connection retries.</td>
</tr>
<tr>
<td>Retry Timeout</td>
<td>Set the value in milliseconds which is used as a timeout value on each attempt of http connection.</td>
</tr>
<tr>
<td></td>
<td>If set as 0, it means no time to wait for each attempt of http connection.</td>
</tr>
</tbody>
</table>

4. Save the service configuration.
Enable detail level logging for Gateway

This section explains the steps required to enable the detail level logging for basic authentication.

1. Open a terminal window.
2. Navigate to $ASG_CONFIG_HOME/asg$ directory.
3. Edit the $asg.properties$ file.
4. Search the following property in the file:
   
   $tibco.clientVar.ASG/Logging/MinLogLevel=1$

5. Set the highest log level by changing the value of the property as below:
   
   $tibco.clientVar.ASG/Logging/MinLogLevel=0$

6. Search the following property to enable the detail logging for common logger logging by:
   
   $tibco.clientVar.ASG/Logging/clLogLevel=0$

7. Enable detailed logging by changing the value of the property as below:
   
   $tibco.clientVar.ASG/Logging/clLogLevel=1$
Configure TIBCO Enterprise Message Service

This section describes the steps to setup and configure TIBCO Enterprise Messaging services destination required for API Exchange Gateway runtime.

This section assumes that API Exchange Gateway has access to a running instance of TIBCO Enterprise Messaging services.

1. Make sure TIBCO EMS server is running.
2. Create the following queues in the TIBCO EMS server. For example using `tibemsadmin` tool:
   ```
   create queue asg.out.request
   create queue asg.out.request.reply.0.0
   create queue asg.out.request.reply.0.1
   create queue asg.out.request.reply.0.2
   create queue asg.in.request
   create queue asg.in.request.reply.0
   create queue asg.custom.request
   create queue asg.custom.request.reply.0
   ```

   The default configuration assumes that TIBCO Enterprise Message Service is running on the localhost and port 7222. Depending on the host and port where TIBCO Enterprise Message Service runs, you can edit the TIBCO Enterprise Message Service configuration parameters in the `ASG_CONFIG_HOME/asg/asg.properties` file.

   For example, the steps are listed using the TIBCO Enterprise Message Service `tibemsadmin` tool:
   1. Navigate to the TIBCO Enterprise Message Service installation directory.
   2. Change to the `EMS_HOME/bin` directory.
   3. Start the `tibemsadmin` tool, as below:
      ```
      tibemsadmin server "tcp://host:port" user adminuser
      ```
   4. Type the following command to create queues:
      ```
      create queue asg.in.request
      create queue asg.in.request.reply.0
      create queue asg.out.request
      create queue asg.out.request.reply.0.0
      create queue asg.out.request.reply.0.1
      create queue asg.out.request.reply.0.2
      create queue asg.custom.request
      create queue asg.custom.request.reply.0
      ```
5. Exit the **tibemsadmin** utility.
Configuring JMS northbound transport for XML

This section describes the steps to configure the JMS transport on the northbound side for XML message.

TIBCO Designer Configuration

In this scenario, TIBCO BusinessWorks is used as a client to send the XML payload to the API Exchange Gateway engine using the JMS transport.

1. Open the TIBCO Designer and create a new project.

2. Create a new JMS Application Properties activity resource. Type a name for this resource (for example, JMSProperty1) Add a new property for this resource as follows:
   - PropertyName: Operation
   - Type: string
   - Cardinality: required

3. Create a new Process Definition. Type a name for this process (for example, SendJMSMessage).

4. Create a new activity JMS Queue Requestor resource in the process. Type a name for this resource (for example, SendJMSRequest). Configure the JMS Queue Requestor resource as follows:
   - Configure the JMS Connection parameter. Select a pre-configured JMS connection resource.
   - Click the Advanced tab. For the JMS Application Properties field, select the configured JMS Application Properties resource (for example, JMSProperty1). Click Apply.
   - Click the Input tab. Verify that the Operation field appears under OtherProperties. Specify a value for the Operation. This value of the operation field matches the SOAP Action parameter of the operation configuration in the Operations tab of the gateway configuration UI.
   - Click Apply and Save the project.

Operations Configuration in Gateway UI

1. Start the gateway UI.

2. Click Operations tab. Configure a new operation as below:
   - Operation Name: Any logical name. For example, getLocationBW.
• SOAP Action: Specify the value of this parameter to prepend /ESB followed by the value of Operation, a JMS application property.

  For example, if the value of Operation (JMS application property) is specified as getLocation, type the value of SOAP Action parameter as /ESB/getLocation. Operation is identified by Operation/SOAP Action.

3. Click Save to save the configuration.

Enable the ESB Channels in the CDD file

By default, the ESB channels are disabled. To enable the channels, perform the following steps:

1. Open the ASG_HOME/bin/asg_core.cdd file for editing in a text editor.

2. Search the following property.

   <property-group comment="" name="Channel">
       <property name="be.channel.deactivate" value="/DefaultImplementation/Channels/SouthboundEsb0Channel,/DefaultImplementation/Channels/SouthboundEsb1Channel,/DefaultImplementation/Channels/SouthboundEsb2Channel,/DefaultImplementation/Channels/North_ESBChannel,/DefaultImplementation/Channels/North_HTTPChannel"/>
   </property-group>

3. To enable the channels, rename the property name to be.channel.deactivate.test as below:

   <property-group comment="" name="Channel">
       <property name="be.channel.deactivate.test" value="/DefaultImplementation/Channels/SouthboundEsb0Channel,/DefaultImplementation/Channels/SouthboundEsb1Channel,/DefaultImplementation/Channels/SouthboundEsb2Channel,/DefaultImplementation/Channels/North_ESBChannel,/DefaultImplementation/Channels/North_HTTPChannel"/>
   </property-group>

4. Save the file.

Update TIBCO Enterprise Message Service Libraries

To include the TIBCO Enterprise Message Service libraries, perform only one of the following steps:

• Open the ASG_HOME/bin/asg-engine.tra file for editing. Set the tibco.env.EMS_HOME property to the TIBCO Enterprise Message Service installation home.

  For example,
tibco.env.EMS_HOME=/home/asg/tibcoasg/ems/5.1

- Copy the jms.jar and tibjms.jar files from TIBCO Enterprise Message Service installation (EMS_HOME/lib) to ASG_HOME/lib/ext/tpcl
Changing the Stack Size

To address the high memory usage requirements of TIBCO API Exchange Gateway server, it is recommended to set a smaller stack size, for example 256 KB. This can be done in the following ways:

You must have super user privileges to change the stack size.

To Permanently Change the Stack Size

1. Open the following file for editing:
   
   /etc/security/limits.conf

2. Add the following lines (example values shown):

   * soft stacksize 256
   * hard stacksize 256

Refer to your Operating System’s manual for details and set the appropriate values.

To Temporarily Change the Stack Size

You can change the stack size for the current session at the command line. To do so type a command like the following at a command prompt:

   ulimit -s 256

The above command sets the stack size to 256 KB for the current session only.
Modify Unicast Discovery URL in CDD file

When API Exchange Gateway server configuration is manually copied to any server from where it was installed and configured, it requires to edit the discover URL and listen URL.

**Discover URL**

The discover URL specifies how the gateway engine (node) listens for discovery requests from nodes attempting to join the cluster. When a cluster starts up, and also when new members join a cluster, a discovery process enables the members to discover each other. The discover URL specifies how an gateway engine (node) listens for discovery requests from nodes attempting to join the cluster.

The discovery URL for well-known address configuration uses the following format:

```
tcp://ip:port[;ip:port]*
```

After the discovery is complete, the members communicate internally using a listen URL.

**Listen URL**

The listen URL is used for direct communication between the members of the metaspace. The listen URL value must be different for each cluster member.

The listen URL uses this format:

```
tcp://interface:port[-EndPort |]*
```

The cluster member binds to the specified interface and the specified port when creating the TCP socket. Specify the parameters as follows.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>interface</strong></td>
<td>To specify a value, use the desired IP address.</td>
</tr>
<tr>
<td></td>
<td>The value for <strong>interface</strong> must be the same in both the discovery and the listen URLs for a node. If there are multiple interfaces on one machine, specify the interface you want to use and do not rely on the default value.</td>
</tr>
<tr>
<td></td>
<td>The default value for <strong>interface</strong> is the first available interface provided by the operating system for the machine.</td>
</tr>
</tbody>
</table>
How to edit the Discover URL and Listen URL (using text editor)

To edit the discover URL and listen URL, perform the following steps:

1. Open the ASG_HOME/bin/asg_core.cdd file for editing.
2. Edit the following properties and set the value to the actual IP address of the machine, and an unused port.

For example:

```
<property name="be.engine.cluster.as.discover.url" value="tcp://127.0.0.1:6000/"/>
<property name="be.engine.cluster.as.listen.url" value="tcp://127.0.0.1:6000-*"/>
```
3. Save the file.

How to edit the Discover URL and Listen URL (using API Exchange GatewayStudio)

You can edit the discover and listen URL in the Cluster Deployment Descriptor (asg_core.cdd) file as follows:

1. Navigate to the ASG_HOME/bin directory.
2. Copy the asg_core.cdd file to ASG_HOME/projects/ASG_DefaultImplementation folder.
3. Navigate to the ASG_HOME/studio/eclipse directory.
4. Type the following command to start the TIBCO API Exchange Gateway Studio:
   ```
   ./studio
   ```
5. If you are prompted, select or create the Eclipse workspace directory where your project files will be stored. (If you check the option to use this workspace as a default, you are not prompted again.)
6. Click OK.
7. Close the Welcome screen.

---

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Notes</th>
</tr>
</thead>
</table>
| `port`           | To specify a single port use the port number in the listen URL, as shown in this example:  
                   | `tcp://interface:6000/`  
                   | The default value is the first available port in the 50000+ range. |
8. From the File menu select **Import**.

9. In the Import Select wizard, select an import source as **General > Existing Projects into Workspace** and click **Next**. You see the Import Projects dialog.

10. In the Import Projects dialog **Select root directory** field, browse to and select the project: `ASG_HOME/projects/ASG_DefaultImplementation`.

11. Click **Finish**.

12. In the Studio Explorer, expand the `ASG_DefaultImplementation` project node. Verify that you see the `asg_core.cdd` file.


14. Select Cluster tab > **General** on the left. On the right, expand cluster node to edit the following properties:
   - `be.engine.cluster.as.discover.url`
   - `be.engine.cluster.as.listen.url`

15. Set the values to the actual IP address of the machine, and an unused port.
   For example,
   ```
   be.engine.cluster.as.discover.url=tcp://127.0.0.1:6000/
   be.engine.cluster.as.listen.url=tcp://127.0.0.1:6000-*
   ```

16. Save the file.

17. Back up the original `asg_core.cdd` file in the `ASG_HOME/bin` directory.

18. Copy the modified `asg_core.cdd` file from `ASG_HOME/projects/ASG_DefaultImplementation` to the `ASG_HOME/bin` directory.

   If you select the "Copy projects into workspace" option during the import of the project, then the modified `asg_core.cdd` file exists in the workspace directory. Make sure to copy the `asg_core.cdd` file from workspace directory to the `ASG_HOME/bin` directory.
If you want to use SSL/TSL with the Apache HTTP server, you need to create an SSL certificate. This certificate is required for the authorization between the Apache HTTP server and client so that each party can clearly identify the other party. To ensure the integrity of the certificate, it must be signed by a party every user trusts.

This section describes the procedure for following tasks:

- To generate a self-signed certificate using the OpenSSL toolkit.
- To create your own certificate authority that you use to sign your own generated request by using the OpenSSL toolkit.

**Generating Self-Signed SSL Certificates**

**Create Private Key**

To create a private RSA key using the OpenSSL package to be used by the mod_ssl module of Apache HTTP serve, use the following command:

```
$ openssl genrsa -out asgserver01.key 1024
```

The above command generates a 1024 bit long RSA private key and stores the private key file in the `asgserver01.key` file.

As SSL is a PKI based encryption system, it requires a private key to reside on the server. The generated RSA private key `asgserver01.key` file is a digital file used to decrypt messages sent to the Apache HTTP server. This file has a public component that will be distributed (via a digital certificate file) to allow clients to encrypt messages before sending them to the server.

**Generate Certificate Signing Request (CSR)**

A Certificate Signing Request (CSR) is a digital file that contains the server's public key and the server's identity. Normally this file is sent to a Certifying Authority (CA) so that it can be converted into a real digital certificate. A digital certificate contains the server's RSA public key, its name (or identity), the name of the CA, and it is digitally signed by your CA. The clients that know the CA can verify the signature on that digital certificate, thereby obtaining the server's RSA public key. This enables the clients to send messages that only a server can decrypt.

To generate a certificate signing request (CSR) for a previously generated private key file, use the following command:
For a widely used production deployment when you want that the certificate is automatically accepted by all major client implementations, you will send the CSR file to an officially established Certificate Authority.

For testing purposes, you can sign your own public key which will be perfectly usable certificate.

To generate a self signed certificate for the previously generated certificate signing request (CSR) signed with the generated private key file, use the following command:

```bash
$ openssl x509 -in asgserver01.csr -out tibasg.crt -req -signkey asgserver01.key -days 365
```

### Generating SSL Keys and Certificates With Your Own your own Trusted CA

This section explains the simplified approach of generating the keys and certificates by using the `CA.pl` (or `CA.sh`) script that is shipped with OpenSSL tool kit.

The `CA.pl` (or `CA.sh`) script allows you to create your private Certificate Authority that you can use in turn to generate new private keys and certificates that are signed by your own private Certificate Authority. Use the `CA.pl` (per1) or `CA.sh` (shell) script that is shipped with OpenSSL.

Perform the following steps:

#### Create CA Hierarchy

This section explains the steps to create CA hierarchy for your private CA. This is a one time action. Once you have created your CA hierarchy, it will be used for every key/certificate pair you want to generate and sign with this CA.

Perform the following steps to create a CA hierarchy for your private CA:

1. Open a command prompt window.
2. Navigate to the directory.
3. Enter the following command (with the `-newca` parameter) to create a CA hierarchy:
CA.sh -newca

4. Follow the prompt and enter filename of the CA certificates which should also contain the private key.

5. Verify that the relevant files and directories are created in a directory.

Create Private Key and Certificate Signing Request (CSR)

After you have created a hierarchy for your own Certificate Authority (CA), you can use the same CA.sh script to create the private key and certificate signing request.

Perform the following steps to create the private key:

1. Open a command prompt window.
2. Navigate to the directory.
3. Enter the following command (with the -newreq parameter) to create a new certificate request (CSR):
   
   CA.sh -newreq

4. Verify that the output of this command contains both the private key and the certificate signing request. The private key is written to the file newkey.pem and the certificate request is written to the file newreq.pem.

5. Enter the following command (with the -sign parameter) to have the certificate signing certificate request being signed by the CA:
   
   CA.sh -sign

   The script expects the certificate request to be in the file newreq.pem. The new certificate is written to the file newcert.pem.

Create PKCS#12 archive (Optional)

Optionally, can also create a PKCS#12 archive. The PKCS#12 file is an archive file format that contains the user certificate, private key and CA certificate. The PKCS#12 file can be imported directly into a browser.

Do the following steps to create the PKCS#12 file:

1. Open a command prompt window.
2. Navigate to the directory.
3. Enter the following command (with the -pkcs12 parameter) to create a PKCS#12 file:
CA.sh -pkcs12 "ASG Server Demo Certificate"

ASG Server Demo Certificate is typically displayed in the browser list box. If you do not provide the ASG Server Demo Certificate argument, the name My Certificate is used by default.
Editing Cluster Deployment Descriptor (CDD) File

You can edit cluster deployment descriptor (CDD) file in the following ways:

**Using Text Editor**

To edit properties defined in the CDD file, perform the following steps:

1. Open the CDD file (for example, `ASG_HOME/bin/asg_core.cdd`) file in a text editor.

2. Edit the file, as needed. For example, you can set the following cluster properties to the actual IP address of the machine, and an unused port as below:

   For example:
   ```
   <property name="be.engine.cluster.as.discover.url" value="tcp://127.0.0.1:6000/>
   <property name="be.engine.cluster.as.listen.url" value="tcp://127.0.0.1:6000-*/"/>
   ```

3. Save the file.

**Using API Exchange Gateway Studio**

You can modify the properties defined in the CDD file using the studio. To do so, edit the CDD file (for example, `ASG_HOME/bin/asg_core.cdd`) file as follows:

1. Navigate to the `ASG_HOME/bin` directory.

2. Copy the `asg_core.cdd` file to `ASG_HOME/projects/ASG_DefaultImplementation` folder.

3. Navigate to the `ASG_HOME/studio/eclipse` directory.

4. Type the following command to start the TIBCO API Exchange Gateway Studio:
   ```
   ./studio
   ```

5. If you are prompted, select or create the Eclipse workspace directory where your project files will be stored. (If you check the option to use this workspace as a default, you are not prompted again.)

6. Click OK.

7. Close the Welcome screen.

8. From the File menu select Import.
9. In the Import Select wizard, select an import source as **General > Existing Projects into Workspace** and click **Next**. You see the Import Projects dialog.

10. In the Import Projects dialog **Select root directory** field, browse to and select the project: \(\text{ASG\_HOME/\_projects/ASG\_DefaultImplementation}\). Click **Next**.

11. Click **Finish**.

12. In the Studio Explorer, expand the \(\text{ASG\_DefaultImplementation}\) project node. Verify that you see the \(\text{asg\_core.cdd}\) file.

13. Double-click the \(\text{asg\_core.cdd}\) file.

14. Edit the properties, as needed. For example, to edit the cluster properties, do following:
   
a. Select Cluster tab > **General** on the left. On the right, expand cluster node to edit the following properties:
      
      - \(\text{be.\_engine.\_cluster.\_as.\_discover.\_url}\)
      - \(\text{be.\_engine.\_cluster.\_as.\_listen.\_url}\)

   b. Set the values to the actual IP address of the machine, and an unused port.

   For example,

   - \(\text{be.\_engine.\_cluster.\_as.\_discover.\_url}=\text{tcp://127.0.0.1:6000/}\)
   - \(\text{be.\_engine.\_cluster.\_as.\_listen.\_url}=\text{tcp://127.0.0.1:6000-*/}\)

15. Save the file.

16. Make a back up copy of the original \(\text{asg\_core.cdd}\) file in the \(\text{ASG\_HOME/\_bin}\) directory.

17. Copy the modified \(\text{asg\_core.cdd}\) file from \(\text{ASG\_HOME/\_projects/ASG\_DefaultImplementation}\) to the \(\text{ASG\_HOME/\_bin}\) directory.

   If you select the "Copy projects into workspace" option during the import of the project, then the modified \(\text{asg\_core.cdd}\) file exists in the workspace directory. Make sure to copy the \(\text{asg\_core.cdd}\) file from workspace directory to the \(\text{ASG\_HOME/\_bin}\) directory.
Connection Problem to TIBCO DataGrid

If the IP address of the machine where TIBCO API Exchange Gateway software is installed changes, the gateway engine might report the connection error to TIBCO Datagrid as it does not automatically update the IP address in the cdd file of the core engine.

Error:

When the gateway engine is started using `asg-engine` executable, following warning message is shown:

```
asg-caching-core Info [main] - [com.tibco.cep.runtime.service.dao.impl.tibas.ASDaoProvider]

asg-caching-core Warning [main] - [com.tibco.cep.runtime.service.dao.impl.tibas.ASDaoProvider]
```

Resolution:

Modify the `ASG_HOME/bin/asg_core.cdd` file to either use the new public IP address or just the loopback value. Restart the gateway engine after making the changes in the cdd file.

You can edit the file using an XML editor:

```
<property-group comment="" name="cluster">
  <property name="be.engine.cluster.as.discover.url" value="tcp://127.0.0.1:6000/"/>
  <property name="be.engine.cluster.as.listen.url" value="tcp://127.0.0.1:6000-*"/>
  <property name="be.mm.cluster.as.listen.url" value="tcp://127.0.0.1:6000-*"/>
</property-group>
```
Glossary

A

Always do this
In order to implement pop-up glossary entries in HTML, glossaries can contain only paragraphs like this, and letters dividing the entries alphabetically.

alert
A notification to an end-user, for example, scheduled alerts deliver portal headlines to a chosen device. See also real-time alert.

C

consumer
A service consumer is the initiator of a message exchange with a service provider.

D

deployment descriptor
An XML file that describes the configuration of a web application. It’s located in the WEB-INF directory of the application’s WAR file.

DMZ
An acronym for demilitarized zone, this term is used metaphorically for that part of a network between an inner and an outer fire wall. Machines placed in the DMZ may be available to authorized users outside the firewalls, whereas machines placed behind the DMZ are protected from outside access.

domain
In TIBCO Administrator, two kinds of domains are used. See also administration domain and application domain.

S

SOAP (Simple Object Access Protocol)
A basic web services standard for making web services available remotely. See also UDDI, WSIL, WSDL, WSRP

common page
A logical name-value binding to identify a portal page in page layout templates, regardless of where that portal page exists in the page tree. Provides for flexibility and ease of template maintenance.

community site
A type of portal site created and used by business users in support of team communications.
Index

A
animation preference for diagrams 480

B
be.engine.cluster.as.discover.url 466, 468

C
Cache Agent Quorum 505
Cluster Settings 472
Configuring MM Console Properties 462
customer support 22

D
Database Location 434
Deployment Unit Settings 473

H
Host Settings 475

I
Inactive Members 478, 480

J
java.property.be.engine.jmx.connector.authenticate 45
7
java.property.be.engine.jmx.connector.port 457, 457

L
Limitations in Use of Certain Processing Unit
Configurations 471
log files
location of (setting) 61

N
Number of Backup Copies 505

P
Processing Unit Settings 473
Project, Master, and Deployed CDD and EAR
Files 470

S
Site Settings 472
Site Topology Overview 470
Site Topology Reference 472
support, contacting 22
T

technical support 22
TIBCO_HOME 19

W

Working with Cluster Explorer 478, 480