

TIBCO® API Exchange Gateway User Guide

Software Release 2.3

September 2016

Document Updated: April 2018

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 and Two-Second Advantage are either registered trademarks or trademarks of TIBCO Software Inc. in the United States and/or other countries.

Enterprise Java Beans (EJB), Java Platform Enterprise Edition (Java EE), Java 2 Platform Enterprise Edition (J2EE), and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle Corporation 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 © 2004-2018 TIBCO Software Inc. All rights reserved.

Contents

Figures	24
TIBCO Documentation and Support Services	25
Introduction to TIBCO API Exchange Gateway	26
Design-time Components	27
Config UI	27
Studio	28
Runtime Components	28
Gateway Operational Layer	28
Gateway Management Layer	28
Deployment Architecture	30
Single Server Deployment Architecture	31
Distributed Deployment Architecture	33
Getting Started	36
Examples Overview	36
Examples	36
Configure an Endpoint Operation for TIBCO API Exchange Gateway	37
Creating a New Configuration	38
Configuring Partner Group	38
Configuring Partner Data	38
Configuring a Facade Operation	39
Configuring a Target Operation	39
Configuring an Authorization Configuration	40
Configuring Routing Configuration	40
Saving the Gateway Configuration	41
Testing the Gateway configuration	41
Running Apache HTTP Server if not running	41
Running Core Engine	41
Testing the Configured Operation and Target Operation	42
Working with Studio	42
Starting Studio	42
Loading the Default ASG_DefaultImplementation Project	43
Editing Validating and Building the Default ASG_DefaultImplementation Project	43
Adding or Edit a Resource in a Project	43
Validating a Project, Project Folder or Project Resource	44
Fixing Validation Errors	44
Building the Default ASG_DefaultImplementation Project	44

Debugging Project in Studio	45
Setting the Debug Perspective within Studio	48
Validation Tool (asg-validate)	48
Running asg-validate Using asg-tools	48
Log File for asg-validate	49
Limitations of asg-validate	49
Runtime Properties	49
Runtime Properties of Core Engine	49
Runtime Properties of Central Logger	77
Building an EAR File at the Command Line	83
Core Engine Configuration	85
Core Engine	85
Starting Core Engine	85
Processing Units of Core Engine	86
Configure Log Files Settings	87
Log File Name and Location	88
Number and Size of Log Files	88
Setting the Log File Configuration Settings in a CDD File	89
Logging Levels of Core Engine	89
Apache Module for TIBCO API Exchange Gateway	91
Installing Apache HTTP Server	91
Installing Apache HTTP Server with SSL	91
Configuring Apache HTTP Server Using HTTP Transport	91
On the Windows Platform	92
Configuration On the UNIX Platform	92
Running the Apache HTTP Server	93
On the Windows Platform	93
Running On the UNIX Platform	93
Secure Communications	93
Mutual SSL Authentication	93
Generate Keys and Certificates	94
Configure SSL on Apache HTTP Server	94
Configure Client Authentication with Digital Certificates on Apache HTTP Server	94
Configure Client Certificate Identification Details On Apache HTTP Server	95
Register Partners on Config UI	96
SSL Communications Configuration	96
Configuring One-Way SSL Authentication	96
Mutual SSL Authentication Configuration	97
Prerequisites for Mutual SSL Setup	97

Configuring Mutual SSL on Apache HTTP Server	97
Configuring Client Authentication with Digital Certificates on Apache HTTP Server	99
Using Firefox	100
Using Internet Explorer	101
Testing the Imported Certificate	101
Forwarding Client Certificate Identification Details on Apache HTTP Server to Core Engine	101
Registering Partners Using the Config UI	102
Configure the Apache Server for Basic HTTP Authentication	103
Configuring Apache HTTP Server for Basic Authentication	103
Enabling Basic Authentication on Apache HTTP Server	103
Creating a Password File for the Apache HTTP Server	104
Reloading the Configuration File for the Apache HTTP Server	104
On the UNIX Platform	105
Configuring a Client (Requester) for Basic Authentication (Example Use Case)	105
Configuring the Endpoint URL for Transport	105
Creating an Identity Resource	106
Configuring Identity For Transport	106
Configuring TIBCO API Exchange Gateway for Basic Authentication	106
Configure Apache Module for RVRD Setup through a Firewall (DMZ)	107
Setting up TIBCO API Exchange Gateway in a DMZ Environment	108
Configure Apache HTTP Server as Reverse Proxy	109
Directives	109
Setting up Reverse Proxy Server for Non-SSL Communication	110
Setting up Reverse Proxy Server for SSL Communication	110
Transport Communication	112
Facade Operation Requests	112
Central Logger	112
Global Throttle Manager	113
Rendezvous Transport Communication	113
Enabling Rendezvous Communication for TIBCO API Exchange Gateway	114
Backup Files	114
Editing asg_core.cdd File	115
Editing asg_cl.cdd File	115
Setting tibco.env.RV_HOME Property in TRA Files	116
Editing Properties Files	116
Configuration Setup	117
Setting Rendezvous Transport Properties	117
Rendezvous Session Connection Parameters for Apache Module	117
Rendezvous Session Connection Parameters for Core Engine	118

Rendezvous Session Connection Parameters For Core Engine and Central Logger Communication	119
Rendezvous Session Connection Parameters for Core Engine and Global Throttle Manager Communication	121
Secure Deployments with TIBCO Rendezvous	122
Configuration for Secure Rendezvous Daemon	123
Configuration Tips	123
Setup and Configure Rendezvous Daemons	124
Configuration Setup for Apache Module and TIBCO API Exchange Gateway	125
Install Apache Server	125
Install TIBCO API Exchange Gateway	125
Setting up and Configuring Apache Module	125
Setting up Apache Module on Machine 1	125
Configuring Apache Module on Machine 1	125
Sample Properties For Apache Module	127
Configuring the Core Engine Properties	128
Sample Properties for Core Engine	129
asg-password-obfuscator Utility	130
Enabling Facade HTTP Transport	131
Enable Facade HTTPS Transport	131
Setting SSL Properties	131
Download Tomcat Native Library	131
Downloading on Windows platform	131
Downloading on Linux platform	132
Downloading and Building APR	132
Building Tomcat Native Library	132
Setting LD_LIBRARY_PATH	133
Setting Content-Type for Error Response	133
Endpoint Ports	134
JMS Transport Communication	137
SOAP JMS Transport	138
Gateway as SOAP JMS Server	138
Gateway as SOAP JMS Client	138
Configuring SOAP JMS Transport	139
Enabling SOAP JMS Channels	139
Edit asg.properties File for JMS Server	140
JMS Server Connection Parameters	140
JMS Server Queue Names	141
Create queues on EMS Server	142
Create Users On JMS Server	142
Setting JMS Jars in ClassPath	143

Config UI Configuration	143
Configuring New Partner	143
Configuring Facade Operation	144
Configuring Partner Operation	144
Configuring Target Operation	145
SSL Support for JMS Transport	145
Set JMS Message Delivery and Acknowledgment Mode	146
JMS Message Delivery Modes	146
JMS Message Acknowledgment Mode	147
Non-Standard JMS Headers	149
Setting up JMS Properties	149
ESB Channel	151
Enabling ESB Channels	151
Edit asg.properties File for ESB Channel Properties	152
JMS Server Connection Parameters For ESB Channel	152
Create queues on EMS Server	155
Create Users On EMS Server	155
Config UI	156
Starting GUI	156
Accessing Config UI through HTTPS Transport	157
SSL Properties for Config UI	158
Changing Login Host and Port Information	160
Authentication Process for Config UI	160
Authentication Properties	161
Configuration Setup for Authentication Process	163
LDAP Server Authentication	163
Configuring asg-configui.tra File	163
Configuring web.xml File	164
File-Based Authentication	164
Configuring asg-configui.tra File (FILE)	164
Configuring web.xml File (FILE)	165
Authentication Property Files	165
Default Authentication	166
Enable Debug Logging for Config UI	167
Creating Properties File	167
Logging to stdout	167
Logging to a File	168
Using Properties File in the TRA File	168
Configuring Directory for Log Files	168

Manage a Gateway Project Configuration	169
Publish Project Configuration	169
Publishing Configuration	170
Change Log Level Settings	170
Changing Log Level Settings for Core Engine	171
Changing Log Level Settings for Central Logger	171
Updating Project Configuration	172
Validate Configuration	172
Project Configuration	172
MAPPING	173
Mapping	173
Schemas	174
Error Maps	174
SECURITY	175
WSS	175
KeyStores	176
Policy Mapping	177
Policy Binding	177
MONITORING	178
Monitors	178
KPI Groups	180
ROUTING	181
Facade Operations	181
Adding a New Facade Operation	181
Deleting an Operation	183
Characters Supported in Facade Operation URI	183
Target Operations	183
Adding a New Target Operation	184
Target Operation Groups	191
Routing	192
PARTNER	193
Partners	193
Partner Groups	194
Adding a Throttle for a Partner Group	194
Facade Access	195
Adding a New Facade Access	195
Partner API Key	196
Set Runtime Properties	196
Setting General Properties	197

Setting Monitoring Properties	197
Setting Database Properties	200
Setting Transport Properties	201
Security Properties	214
Transaction Pipeline processing	221
Request Pipeline Processing	221
Response Pipeline Processing	223
Parsing Step	224
Set the Partner Identity for Request	224
Set the Routing Key for Request	224
Enrich the Audit Trail Log for Request	225
Logging Request Headers	225
Sample XSLT	226
Validate the Request Content	227
Set Metric Increment for Content-Based Throttles	227
Set Sticky Key for Load-Balancing with StickyResourceAffinity	228
Overriding HTTP Headers	228
Parsing XSLT Documents	230
Parsing Output Document Schema	233
Mappings and Transformations	235
Mapping Types	237
Mapping Configuration	237
Rendezvous Mapping Type	237
XSLT Mapping Type	238
Assign to the Gateway Operation Endpoint	238
Assign to the Gateway Reference Endpoint	239
Transformations (XSLT Mapping)	239
Set error codes for content validation	240
Validation	240
Enabling Validation	240
Implementing Request Validation	241
Map the Protocol Headers in Request Context	242
Enumeration Orchestration	242
Response Transformation	243
Mapping Schemas	243
Mapping Container	243
Mapping XSLT Schema	244
Context Document	245
JSON XML Transformation	248

Converting XML Message to JSON Message	249
BookQuery Example Response Message	250
BookQuery Example XML Response	250
Example XSLT to Convert BookQuery XML Response to JSON Response	251
BookQuery Example JSON Transformed Response	251
Converting JSON Message to XML Message	252
BookQueryBE Example Request Message	253
BookQueryBE Example JSON Message	253
Example XSLT to Convert BookQueryBE JSON Request to XML Request	253
BookQueryBE Example XML Transformed Request	254
XSLT Functions for URL Encode and URL Decode	254
Decode() Function	254
Sample XSLT (Decode)	254
Sample Input XML(Decode)	255
Sample Output XML(Decode)	255
Encode() Function	255
Sample XSLT (Encode)	255
Sample Input XML (Encode)	256
Sample Output XML (Encode)	256
Using Encode() and Decode() Functions	256
XSLT Functions for Base64 Encode and Decode	257
textToBase64()	257
base64ToText()	257
Custom Java Functions	257
Java Function	257
XSLT File	258
Pass-Through Gateway	258
Starting Config UI	258
Enabling Default Operation	258
Configuring DefaultOperation Facade Operation	259
Configuring DefaultOperation Facade Operation (REST)	259
Configuring DefaultOperation Facade Operation (SOAP)	259
Configuring Target Operation for DefaultOperation	260
Configuring Routing Key for DefaultOperation	261
Configuring Facade Access for DefaultOperation	261
Pass-Through Special Characters in Query String	262
Proxy Server	262
Configuring HTTP Headers	263
Routing Overview	264

Routing Key	264
Routing Key using XSLT	265
How to Derive and Configure Routing Key	265
Define a Transformation File	265
Navigating to the ROUTING Tab	266
Uploading the Transformation (XSLT) File	266
Routing Configuration	267
Routing Configuration for a Target Operation	267
Routing Configuration for a Target Operation Group	268
Routing Use Case using XSLT	269
Configuration	269
Define a Transformation File	269
Uploading the Transformation File	270
Routing Configuration	271
Preferred Routing	271
Use Case for Preferred Routing	272
Example	272
Overriding Preferred Routing Key using XSLT	273
Example XSLT File	273
Target Operation Group	273
Overview	274
Types of Target Operations Group	274
Routing Algorithms for Target Operation Group	275
LoadBalanced	275
RoundRobin	275
Weighted RoundRobin	276
RoundRobin with Failover	278
Weighted RoundRobin with Failover	278
Sticky Resource Affinity	279
Target Operation Group Configuration	280
Configuring a Target Operation Group	280
Configuring a RoundRobin Target Operation Group	281
Configuring a WeightedRoundRobin Target Operation Group	282
Configuring a RoundRobinWithFailOver Target Operation Group	283
Configuring a WeightedRoundRobinWithFailOver Target Operation Group	284
StickyResourceAffinity Target Operation Group Configuration	285
Define Sticky Routing Key	285
Uploading Sticky Routing Key File	287
Configuring StickyResourceAffinity Type Target Operation Group	288

HealthCheck for Reference	289
HealthCheck Modes for a Target Operation	289
HealthCheck Methods for Timer Mode	290
HealthCheck Configuration for Target Operation	291
Configuration for Reset Mode of HealthCheck	291
Configuration for Timer Based HealthCheck	292
Configuration for HTTP HealthCheck Method	293
Configuration for HTTPS HealthCheck Method	293
Configuration for HealthCheckURL HealthCheck Method	293
Configuration for TCPEcho HealthCheck Method	294
Configuration for ContentVerification HealthCheck Method	294
Configuration for SampleRequest HealthCheck Method	294
Throttles Overview	296
Facade Throttles	296
Service Throttles	296
Throttle Types	297
Rate	297
Quota	297
High Water Mark	298
Error	298
Monitor Time Modifiers	299
Configuring Time Modifier Throttles	300
Throttle Chaining	301
Throttle Counter	301
Throttle UpdateInterval	301
Configuring Throttles	302
Configuration Parameters for Throttles	302
Creating a Throttle Policy Definition	303
Modifying the Existing Configuration to Add Throttles	303
Defining a Quota Throttle	303
Assigning a Throttle Policy to the Target Operation	304
Testing the Target Operation	304
Content Based Throttles	305
Configure Content-Based Throttles	305
Configuring Throttle	306
Define XSLT File	306
Uploading XSLT File	308
Payload Size Throttles	309
Payload Size Throttle Types	309

Configuring Payload Size Throttles	310
Configuring Monitor Counter	311
Traffic Shaping	312
Configuration	312
QueueCompactionInterval	312
Shared Throttles Overview	312
Configuration Setup for Shared Throttles	313
Configuring ActiveSpaces Metaspace Connection Properties	313
Enable Shared Throttles	314
Throttle Configuration Parameters	314
Example Use Case	314
Authentication and Authorization	316
User Authentication	316
Transport and Protocol Level Authentication	316
WS Security Services Authentication	317
Security Service Providers	317
Web Services Security (WSS) Properties	318
Types of Security Service Providers	318
Configuring LDAP Authentication Service Provider (LDAP ASP)	319
Configuring Trust Identity Provider	325
Configuring Subject Identity Provider	326
Configuring WSS Service Provider	328
Limitations	329
Web Services Security Authentication	329
Registering WSS resources with TIBCO API Exchange Gateway	330
Defining the WSS security operations	331
Configure Secure Services with TIBCO API Exchange Gateway	332
Altering List of Algorithms (Optional)	332
Define DSS Properties for Services	333
Properties For SSL Authentication (isAnonymous = true)	333
Properties For Mutual SSL Authentication (isAnonymous = false)	334
Configuring Services	336
Partner Authorization Overview	337
Operation Identification	337
Partner Identification	337
Partner API Key	339
Partner Authorization	340
Overview of Security Policies	341
Security Concepts	341

Types of Security Policies	344
Authentication	344
Authorization	346
Confidentiality	346
Integrity	346
CredentialMapping	347
Manage Policies	347
Configure Shared Resource	347
Registering Shared Resource with TIBCO API Exchange Gateway	348
Create Policy	349
Creating Policy File	350
Sample Template Policy Files	350
Registering Policy	351
Applying Policies	352
Policy Use Cases	353
Authentication Policies	353
Configuring Authentication Policies	353
Authentication Policies Types	354
Basic	354
UsernameToken	355
SAML	355
SiteMinder	356
OAuth	356
Schema for OAuth Policies	357
OAuth Policy File Fields	357
SPNEGO	358
Configuration Setup for Kerberos SPNEGO Authentication Policy	359
Creating a User Account in the Microsoft Active Directory for TIBCO API Exchange Gateway ...	359
Mapping the Service Principal Name (SPN) to a Microsoft User Account	359
Generating a Keytab File for an SPN	360
Authentication Using Custom Shared Resource	361
Implementing Custom Login Module	362
Abstract LoginModule	362
Custom LoginModule	362
Sample Custom LoginModule	362
Packaging and Deploying the Custom Login Module	366
Applying Authorization Policies	367
Configuring Authorization Policies	367
Authorization Policies Types	367

Role	367
Integrity Policies	369
Configuring Integrity Policies	369
Integrity Policies Types	370
Sign	370
Verify Signature	370
Confidentiality Policies	371
Configuring Confidential Policies	372
Confidentiality Policies Types	372
Encryption	372
Decryption	373
Credential Mapping Policies	373
Credential Mapping Policies Types	374
UsernameToken Credential Mapping	374
SAML Credential Mapping	375
Credential Mapping by OAuth Policy	375
Types of Security Shared Resources	376
Shared Resources Properties	377
Configuring LDAP Authentication Shared Resource	377
Configuring SiteMinder Service Provider	382
Configuring Trust Identity Provider	383
Configuring Subject Identity Provider	384
Configuring the Kerberos Service Provider	385
Configuring Custom Shared Resource	387
Shared Resources Properties Sample Files	388
LdapAsp.properties	389
SiteMinderAsp.properties	389
Kerberos SPNEGOAsp.properties	390
SubjectIsp.properties	390
TrustIsp.properties	390
IdentityIsp.properties	391
WssAsp.properties	391
CelmAsp.properties	392
Authentication using File-Based Identity Store	392
Configuring User Authentication Policy using File	393
Create a Shared Resource Properties File	393
Create XML File for Credentials	394
asg-password-hasher Tool	397
Creating Policy File	398

Sample Policy	398
Registering Policy	399
Applying Policy	400
Data Masking and Selective Log Policy	401
Configuration Setup For Log Policy	401
Setting up Properties	401
Create Log Policy Configuration File	401
Creating Data Masking Configuration	402
Masking Headers Data	402
Masking Query String Parameters Data	403
Masking Payload Data	403
Masking Data in Text Payloads	404
Masking Data in XML Payloads	405
Masking Data in JSON Payloads	406
Masking Data in Property Format Payloads	408
Sample Log Policy XML File	409
Log Policy Schema File	414
Upload Log Policy XML File	416
Supported Stages for Log Policy	417
Create Selective Logging Configuration	417
Selective Logging for Text Payloads	418
Selective Logging for XML Payloads	418
Selective Logging for JSON Payloads	419
Selective Logging for Property Format Payloads	419
AntiVirus Scan of Request and Response Payloads	419
Configuration Setup of McAfee Web Gateway	419
Downloading McAfee Web Gateway	420
Configuring McAfee Web Gateway for ICAP Server	420
Enabling ICAP Server	422
Configuring ICAP Client for Request and Response	423
Configuring McAfee Web Gateway for SSL (Optional)	424
Configure TIBCO API Exchange Gateway to Enable AntiVirus Scan	426
Setting Runtime Properties of ICAP Server	426
Enabling AntiVirus Scan for Request Payload of Facade Operation	427
Enabling AntiVirus Scan for Response Payload of Facade Operation	427
Enabling AntiVirus Scan for Request and Response Payloads of Facade Operation	427
OAuth Server	429
Capabilities of the OAuth Server	429
OAuth Client Policies	429

OAuth 2.0 Concepts	430
Benefits of using the OAuth Server	431
OAuth Server Components and Interactions	432
Components	432
Component Interactions	432
OAuth Flows	435
Authorization Code	435
Client Credential	436
Password Credential	436
Configuration Setup of OAuth Server Authorization	436
Setting OAuth Server Properties	436
Enable OAuth Authorization For Gateway (Set Adapter Properties)	437
Owner Adapter	437
File-Based Owner Adapter	437
owners.properties	438
LDAP	438
Client Adapter	440
File-Based Client Adapter	440
clients.properties	441
TIBCO API Exchange Manager	442
Scope Adapter	442
File-Based Scope Adapter	443
scopes.properties	443
Non-Default (Custom) Adapter For Owner Client and Scopes	444
Starting OAuth Server	444
Manage Access Token	444
OAuth Server Endpoint	444
Token Management APIs	445
OAuth Server Endpoints	445
Transport and Port	445
Request Access Token	446
Client Credential Flow	446
Password Credential Flow	446
Authorization Code Flow	447
Access Token Response Example	449
Access Token Error Example	449
Retrieve Access Token Details	449
Using Access Token	449
Validating Access Token Request	450

Sample Response (Retrieve Access Token)	450
Retrieve List of Tokens	450
Retrieve Token for Specific Owner	451
Refresh Token	451
Revoke Token	452
Accessing Token Persistence	453
Enabling OAuth for Application using TIBCO API Exchange Manager	455
Authorization API	457
Name	457
Description	457
Authorization Request	457
Authorization Response	458
Authorization Error	458
Token Request API	460
Name	460
Description	460
Access Token Request	460
Access Token Response	461
Access Token Request Error	461
Token Validation API	463
Name	463
Description	463
Token Validation Request	463
Token Validation Response	464
Token Validation Error	464
Retrieve Access Token	466
Retrieving all tokens	466
Retrieving tokens for specific owner	466
Revoke Token API	467
Name	467
Description	467
Revoke Token Request	467
OAuth Service Provider Interfaces	467
Owner Service Provider Interface	468
Owner Service Provider Interface (SPI) Flow	468
Owner Service Provider Interface (SPI) Java API	468
Client Service Provider Interface	469
Client Service Provider Interface (SPI) Flow	470
Client Service Provider Interface (SPI) Java API	470

Scope Service Provider Interface	471
Scope Service Provider Interface (SPI) Flow	471
Scope Service Provider Interface (SPI) Java API	472
Deploying Custom Adapters	473
Default Adapters	473
Client Adapter	473
Owner Adapter	473
Scope Adapter	474
Gateway Management Features	475
Central Logger	475
Overview	475
Database Setup and Configuration for Central Logger	475
Database Location	475
Task A Setup Database Driver	475
Task B Creating a Database	476
Task C Creating a Database User	476
Task D Setting up the Database Schema	477
For MySQL Database	477
For Oracle Database	477
For SQL Server Database	477
For DB2 Database	478
Task E Setting up the Database Connection Parameters	478
Runtime Properties For Central Logger	479
Enabling Reporting to the Central Logger	479
Running the Central Logger	479
Central Logger Database	479
Database Tables	480
Schema Details	480
ASG_TRANSACTIONS Table	481
ASG_TRANSACTION_DETAILS Table	483
ASG_TRANSACTION_MESSAGES Table	484
ASG_TRANSACTION_KEYS Table	485
ASG_THROTTLE_USAGE Table	485
ASG_THROTTLE_MESSAGES Table	486
ASG_KPI Table	487
ASG_LOG_MESSAGES Table	488
Write Transactions Data to File	488
Enabling Transaction Data to a File	489
Format of Transaction Data Log File	490

Recording Error Events to Central Logger	490
Publishing Error (Failed) Transactions	490
Correlation ID	491
Setting Correlation ID for HTTP Header	491
Enable JMS Channel for Central Logger	492
Configuration Setup for JMS Channel	492
Enabling JMS Channel for Central Logger	492
Setting JMS Transport for Central Logger	494
Configuring JMS Transport Properties	494
Global Throttle Manager	496
Throttle Calculation	497
Running Global Throttle Manager	497
Enabling AS Transport	498
Overview	498
Configuration	498
Cache Cleanup Agent	501
Running Cache Cleanup Agent	501
Reporting	502
TIBCO Spotfire Integration	502
Spotfire Configuration	502
Configuring TIBCO Spotfire Server and Client	503
Setting up a Spotfire Data source	503
Creating a Spotfire Information Model for Central Logger	504
Deploying Default TIBCO API Exchange Gateway Audit Trail DXP File on TIBCO Spotfire Server	505
Basic Deployment	507
Deploying TIBCO API Exchange Gateway Processing Units	507
Requirements For Deployment	507
Deployment Options	507
Deploy Using Monitoring And Management Server	507
Deployment Pre-Requisites	508
Configuration Settings Before You Deploy	508
Configuring JMX properties in Core Engine TRA File	508
To Configure JMX Properties	508
To Enable Monitoring and Management	508
To Enable JMX MBeans Authentication	509
Editing MM.cdd File	509
Editing Site Topology File in a Text Editor	509
Install and Configure Software for Remote Start and Deployment	510
Configure User Authorization for Administrator and User Roles	511

Deploying and Managing Processing Units	511
Overriding Global Variables in MM	513
Configure MM Console Properties	513
Configuring for TIBCO BusinessEvents DataGrid WKA Discovery	514
Configuring ASG_DefaultImplementation Project's CDD	515
Configure Cluster Discovery and Internal Communication	515
Configuring a TIBCO BusinessEvents DataGrid Cluster (Metaspace)	515
Configuring the TIBCO BusinessEvents DataGrid Discover URL	516
If No Other Cluster Members are Started	516
Multicast (PGM) Cluster Member Discovery	516
Unicast (Well-Known Address) Cluster Member Discovery	517
Configuring the TIBCO BusinessEvents DataGrid Listen URL	518
Site Topology Overview	519
References in Site Topology File	519
Deployment-Specific Processing Units	520
Site Topology Reference	520
Site Settings	520
Cluster Settings	520
Deployment Unit Settings	521
Processing Unit Settings	521
Host Settings	522
Working with Cluster Explorer	523
Navigating Cluster Explorer	524
Starting Stopping Pausing and Resuming Core Engines	524
Working With Site Topology Editor	524
Adding a Site Topology Diagram	524
Configuring the Site Topology	525
Specifying the Site Topology Files for the MM Server	526
Running Processing Units At Command Line	526
Deploying Gateway Components Using TIBCO Administrator	527
Advanced Features	531
Cache Agent	531
Running Cache Agent	531
Hot Deployment Overview	531
Enabling Hot Deployment	531
Invoking Hot Deployment	532
Extension Mechanism	532
Response Caching	533
Types of Response Caching	533

Facade Response Cache (Simple Cache)	533
Sample Request Header	534
Sample Response Headers	534
Target Backup Response Cache	535
Cache Response Key	535
Response Caching with Proxy Server	536
Enabling Response Caching	536
Response Caching Parameters	536
Clearing Cached Items	537
Overriding Cache Response Key and Parameters	537
Sample XSLT File	538
Performance Tuning Parameters	539
Large Payload Limit Settings	539
Enable Access Logs in HTTP Channel	539
High Availability Deployment Of Runtime Components	541
Overview	541
Operational Layer Components	542
Gateway Management Layer Components	543
Configuration For High Availability Setup	545
Configure Load Balancer	546
Creating A Health Monitor For Core Engines	546
Creating A Load Balancing Pool	546
Creating a Virtual Server	547
Configure Apache Modules for Core Engines	548
Setting the Rendezvous Connection Parameters For Apache Module	548
Setting the Rendezvous Connection Parameters for Core Engine	548
Cluster Configuration For Runtime Components	549
Configuring Discover URL	549
Editing asg_core.cdd File To Set Discover URL (using text editor)	549
Editing asg_cl.cdd File To Set Discover URL (using text editor)	550
Configuring Fault Tolerance Parameters	550
Configuring Core Engines	551
Configuring Cache Agent	551
Configuring Cache Cleanup Agent	552
Example Setting For Cache Cleanup Agent Instance	553
Configuring Global Throttle Manager	553
Example Setting For Global Throttle Manager Instance	554
Configuring Central Logger	554
Example Settings For Central Logger Instance	555

Configure Rendezvous Session Connection Parameters	555
Appendix A	556
Edit Cluster Deployment Descriptor (CDD) File	556
Editing CDD File using Text Editor	556
Editing CDD File using Studio	556
Setting Discover URL	557
Setting Max Active	557
Setting Priority	558
Configuration Tasks	558
Enabling Cross-Origin Resource Sharing(CORS) Filter Properties	558
Adding URL patterns	559
Adding Filter Parameters	559
Configuring JMS Destinations for Southbound Service Operations	559
Configuring Async Mode for Southbound JMS Service	560
Configuring Retry parameters for HTTP HTTP(s) Transport	561
Enabling detail level logging for Gateway	561
Configuring TIBCO Enterprise Message Service	562
Configuring JMS Northbound Transport for XML	563
Configuring TIBCO Designer	563
Configuring Operations	563
Enable the ESB Channels in CDD File	563
Updating TIBCO Enterprise Message Service Libraries	564
Change the Stack Size	564
Changing the Stack Size Permanently	564
Changing the Stack Size Temporarily	564
Modify Unicast Discovery URL in CDD file	564
Editing the Discover URL and Listen URL (using text editor)	565
Editing the Discover URL and Listen URL (using Studio)	565
Generate Private Keys And Public Certificates with OpenSSL	566
Generating Self-Signed SSL Certificates	567
Generating SSL Keys and Certificates With Your Own your own Trusted CA	567
Creating CA Hierarchy	568
Creating Private Key and Certificate Signing Request (CSR)	568
Creating PKCS#12 archive (Optional)	568
Editing Cluster Deployment Descriptor (CDD) File	569
Using Text Editor	569
Using Studio	569
Connection Problem to TIBCO DataGrid	570

Figures

Single Server Deployment	31
Deployment of Multiple Components in Distributed Environment	34
Apache HTTP Server in DMZ and Other Components in Secure Network	108
Rendezvous Transport Communication	114
Secure Deployment with Rendezvous	123
LDAP Server Properties Sample File:	166
Sample Property File for File Authentication	166
RoundRobin Routing	276
Weighted RoundRobin	278
StickyResourceAffinity Routing Algorithm	280
RoundRobin Target Operation Group Configuration	282
WeightedRoundRobin Target Operation Group Configuration	283
RoundRobinWithFailOver Target Operation Group	284
WeightedRoundRobinWithFailOver Target Operation Group	285
StickyResourceAffinity Target Operation Group Configuration	289
Throttle Types	299
Security Enforcement EndPoints	341
Authentication Policies	353
Authorization Policies	367
Sign Policy	369
VerifySignature Policy	369
Encryption Policy	371
Decrypt Policy	372
CredentialMapping Policies	374
OAuth Server Overview	431
Interactions Between Components	433
Owner SPI Flow	468
Client SPI Flow	470
Scope SPI Flow	472
Central Logger Database Schema	481
Overview of Runtime components For High Availability	542
Deployment of Runtime Components In a Cluster	544

TIBCO Documentation and Support Services

How to Access TIBCO Documentation

Documentation for TIBCO products is available on the TIBCO Product Documentation website, mainly in HTML and PDF formats.

The TIBCO Product Documentation website is updated frequently and is more current than any other documentation included with the product. To access the latest documentation, visit <https://docs.tibco.com>.

Product-Specific Documentation

Documentation for TIBCO products is not bundled with the software. Instead, it is available on the TIBCO Documentation site. To directly access documentation for this product, double-click the following file:

The following documents for this product can be found on the TIBCO Documentation site:

- Installation
- User's Guide

How to Contact TIBCO Support

You can contact TIBCO Support in the following ways:

- For an overview of TIBCO Support, visit <http://www.tibco.com/services/support>.
- For accessing the Support Knowledge Base and getting personalized content about products you are interested in, visit the TIBCO Support portal at <https://support.tibco.com>.
- For creating a Support case, you must have a valid maintenance or support contract with TIBCO. You also need a user name and password to log in to <https://support.tibco.com>. If you do not have a user name, you can request one by clicking Register on the website.

How to Join TIBCO Community

TIBCO Community is the official channel for TIBCO customers, partners, and employee subject matter experts to share and access their collective experience. TIBCO Community offers access to Q&A forums, product wikis, and best practices. It also offers access to extensions, adapters, solution accelerators, and tools that extend and enable customers to gain full value from TIBCO products. In addition, users can submit and vote on feature requests from within the [TIBCO Ideas Portal](https://community.tibco.com). For a free registration, go to <https://community.tibco.com>.

Introduction to TIBCO API Exchange Gateway

This section provides an overview of TIBCO API Exchange Gateway and describes the key components.

TIBCO API Exchange Gateway provides an event-driven web services platform. Using this platform, users can route the APIs requests from consumers to various target services exposed by an organization's internal services layer. Users can completely manage the requests to access the APIs. TIBCO API Exchange Gateway is an event-based routing engine, which processes the requests and responses at a high speed.

TIBCO API Exchange Gateway provides the following key features:

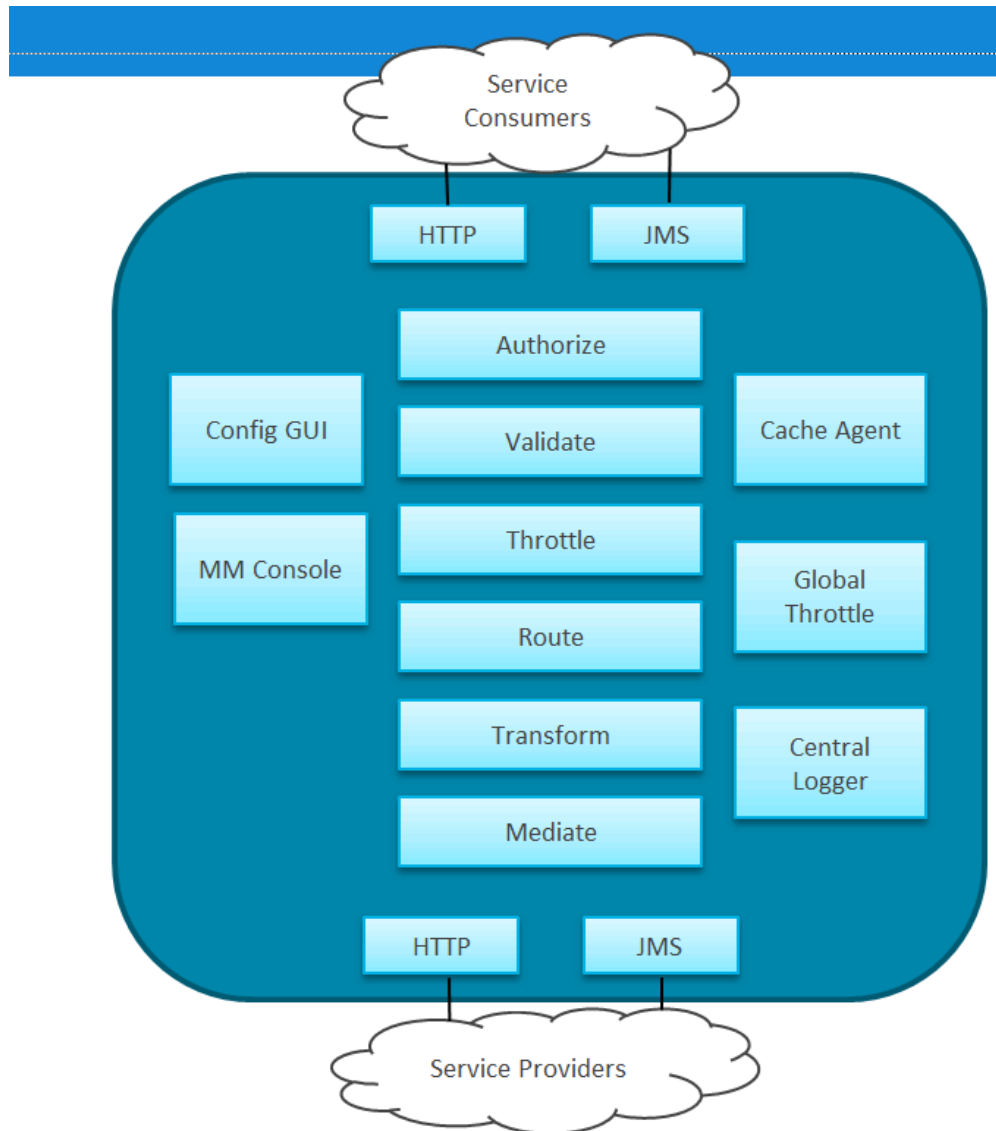
- Receives, routes, and forwards the requests at a high speed
- Routes the requests between any requester and any service endpoint
- Protects from overuse of target service endpoints
- Protects the access of target service endpoints from unauthorized partners
- Reports operation activity such as performance and fault monitoring
- Ensures service level agreements are met

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 an overview of the product functionality:

Functional Overview



Design-time Components

Using the design-time components, the users can perform the following tasks:

- Configure the data for a project used by the Core Engine at run time.
- Develop custom extensions to change the default behavior of the Core Engine.

Config UI

Config UI is used to configure the data required by the gateway at run time.

TIBCO API Exchange Gateway provides a web-based configuration user interface (UI). Using Config UI, users can enter information such as partner data, partner operations, partner groups, services, operations, mappings, throttles, error maps, schemas, and routing. This information is used by the Core Engine at run time for the various functions of the gateway.

Studio

Studio is used to extend the default functionality of the gateway.

Studio is a design-time environment. Using the Studio, users can design and develop custom extensions. Custom extensions can be integrated with the default implementation to customize the default behavior of the Core Engine.

Runtime Components

This section explains the run-time components.

TIBCO API Exchange Gateway provides the following run-time components:

- [Gateway Operational Layer](#)
- [Gateway Management Layer](#)

Gateway Operational Layer

The Gateway Operational layer provides the core functionality of the gateway.

The Gateway Operational layer consists of the following subcomponents:

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 can be used as follows:

- With cache enabled
The cache-enabled Core Engine does not require the Cache Agent.
- Without cache enabled
The Core Engine without cache enabled requires the Cache Agent to run separately.

Cache Agent

The Cache Agent stores the cache data for all objects of the cluster.

Apache HTTP Server (Optional)

You can use the Apache HTTP server as follows:

- Reverse Proxy with HTTP channel (use the Apache HTTP server as it is)
- Apache Module for RV channel (Optional)
The Apache module is used to terminate an incoming request through the HTTPS transport. This module communicates with the facade component to forward the requests for further processing. Optionally, a JMS server can be deployed to use the JMS transport.

Gateway Management Layer

The Gateway Management layer provides request tracking and logging.

It has the following subcomponents:

Central Logger

The Central Logger provides centralized logging of messages in a database.

Global Throttle Manager

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

Cache Cleanup Agent

The Cache Cleanup Agent component clears the cache based on the size and age of the cached values.

Monitoring and Management Server

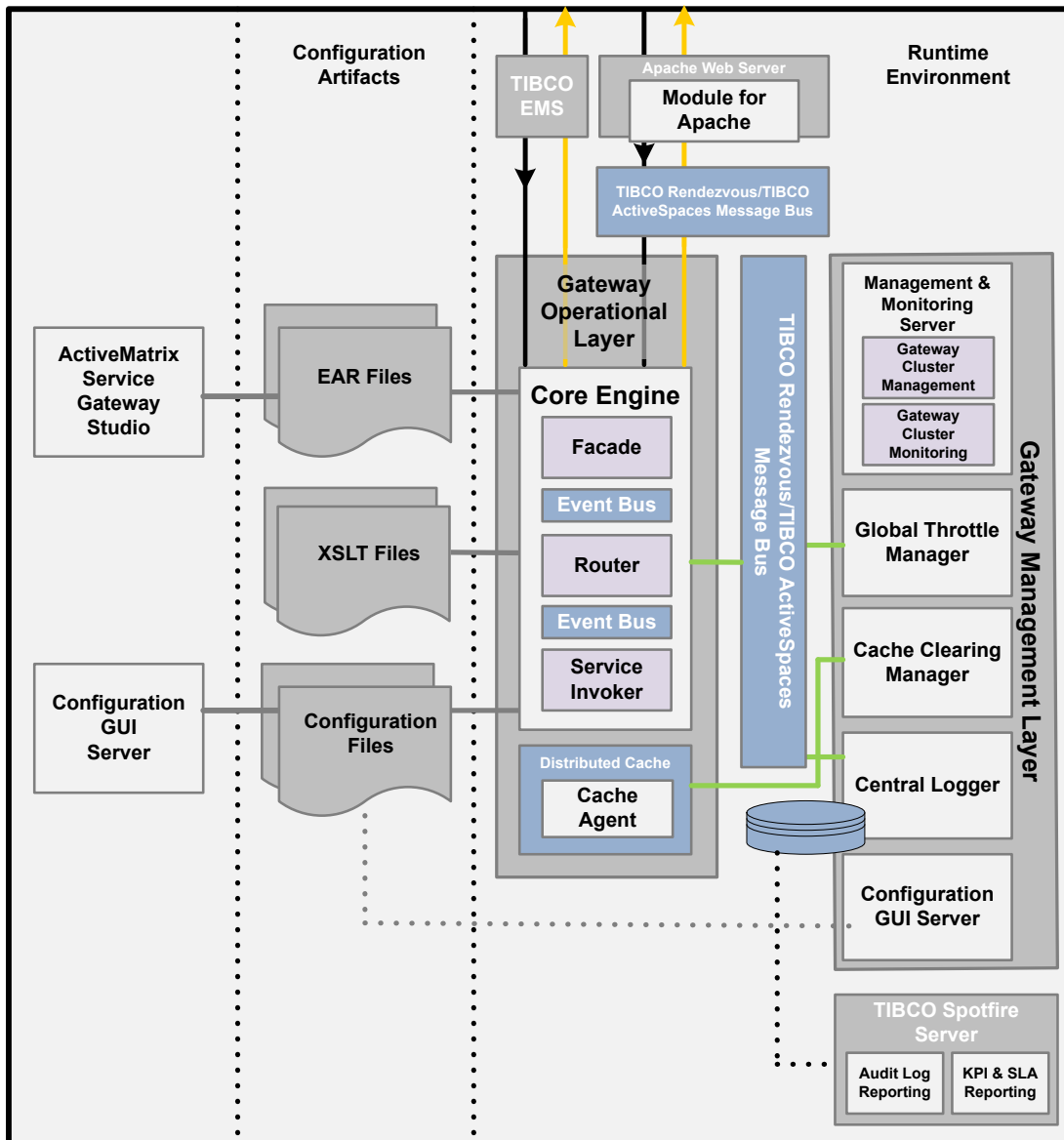
The Monitoring and Management Server is the central management component that monitors the status and manages the operational tasks of all components in a gateway cluster.

Gateway Reporting (Optional)

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

The primary software components are displayed in the following diagram:

Functional Components



Deployment Architecture

This section describes the various options to deploy the Core Engine and other components.

TIBCO API Exchange Gateway is deployed as a cluster of engines that 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/TIBCO ActiveSpaces is used for communication between most of the runtime components of both the gateway operational layer and the gateway management layer. The gateway operational layer and all its components share a single set of configuration files. Therefore, the configuration files should be stored on a shared storage device that is accessible to each of the runtime components.

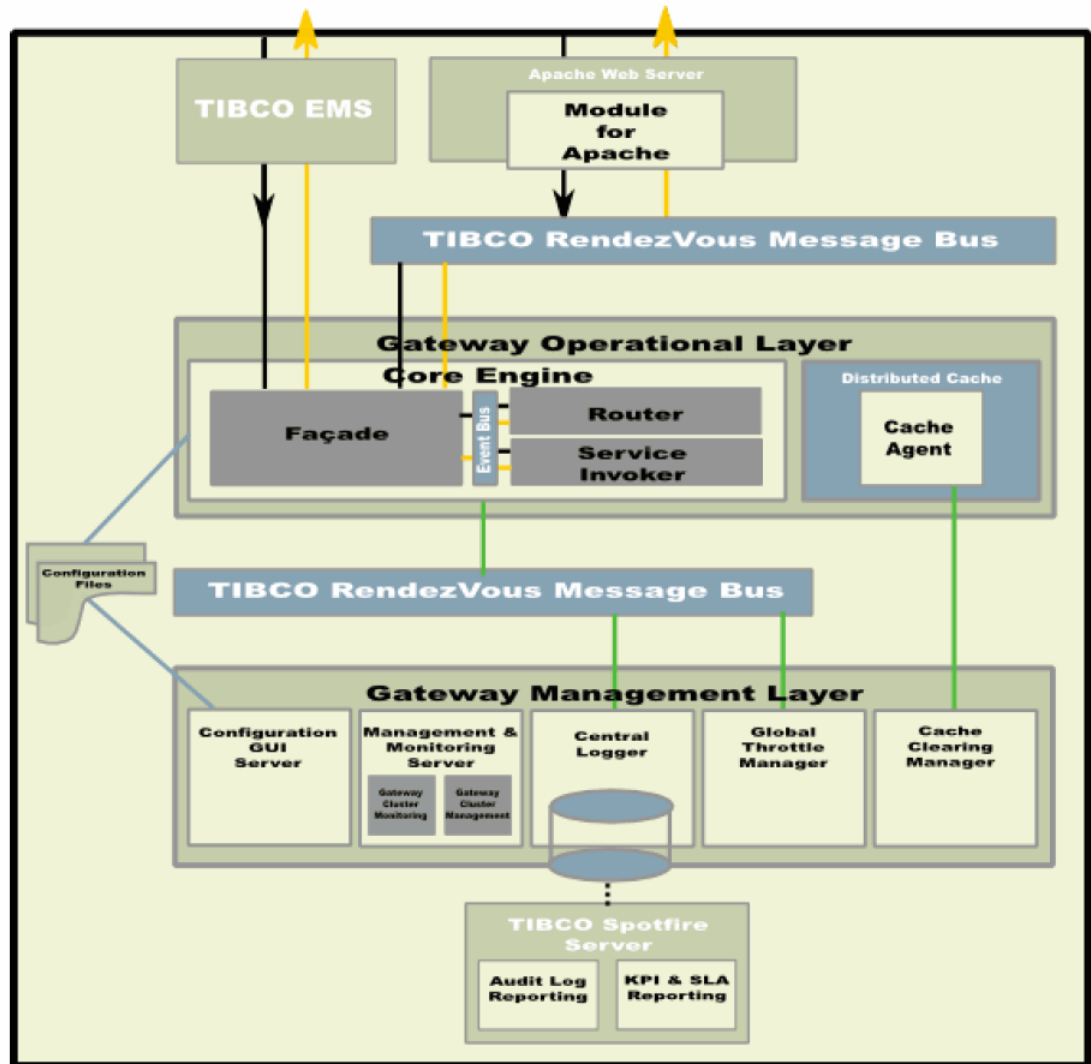
When multiple instances of the Core Engine are deployed in a cluster, multiple instances of cache-agents are also instantiated. This deployment has a single distributed cache that is shared across all the core engines to support association and response cache functionality.

Single Server Deployment Architecture

This section explains the deployment of gateway components as a single server instance.

In its simplest non-high available form, TIBCO API Exchange Gateway can run as a single server. The following figure displays the deployment of the software components as a single server instance:

Single Server Deployment



This configuration provides the entire functionality, including the optional operational reporting and analytics provided by TIBCO Spotfire® Server components. TIBCO Spotfire Professional client software running on a Windows workstation and TIBCO Spotfire WebPlayer Server software running on a Windows server are not visible in the diagram.

The protocol termination components, The Module for Apache HTTP Server (Optional) and TIBCO Enterprise Messaging Service for JMS transport (Optional), need to be deployed and managed with their standard operations management tools.

The Module for Apache HTTP Server that is part of 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 communication with the Core Engine. In case the Apache HTTP Server is deployed within a DMZ zone, you should configure TIBCO Rendezvous Routing Daemon. TIBCO Rendezvous Routing Daemon forwards the TIBCO Rendezvous messages from the DMZ network through the firewall to the internal network where TIBCO API Exchange Gateway components are deployed.

Runtime Components

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

- Core Engine
- Central Logger
- Global Throttle Manager
- Cache Cleanup Agent

The management layer components, the Central Logger and the Global Throttle Manager communicate with the Core Engine using Rendezvous messages at run time.

The Core Engine publishes the messages as events on the Rendezvous bus. The Central Logger component receives these messages from the Rendezvous bus and stores them in the Central Logger database at appropriate intervals. The Global Throttle Manager also uses the Rendezvous bus to report the throttle usage data to the Central Logger.

The Global Throttle Manager controls the throttle allocation for each Core Engine. The Global Throttle Manager receives throttle reports from the Core Engines over the Rendezvous bus. It also sends the throttle grants back to the Core Engines over the Rendezvous bus.

The Global Throttle Manager treats the throttle usage events as the heartbeat interval of a Core Engine. In the absence of a configurable number of consecutive heartbeats, the Global Throttle Manager treats the Core Engine as dead and distributes the throttle limits of the dead instance equally to all the live Core Engines.

The Cache Cleanup Agent does not use the Rendezvous bus to interact with the Core Engine. It connects directly to one of the Cache agents to clear the cache so that it does not grow too large. This cleanup of the cache is called cache flushing.

To deploy and start the cluster of runtime components, use one of the following methods:

- Command Line
At the command line, specify the component unit to start and optionally, a custom CDD file to use.
- TIBCO API Exchange Gateway Monitoring and Management Server



Use only one method for the entire gateway cluster you are deploying. For best results, use the Management and Monitoring Server to deploy the runtime components.

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 Core Engine (with or without caching agent), Global Throttle Manager, or Cache Cleanup Agent 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 the `asg_core.ear` and `asg_cl.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 are persisted in the configuration files on the shared storage device. These configuration files need to be reloaded by the runtime components to be effectuated.

The optional operational reporting and 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

The distributed deployment architecture describes the deployment of multiple instances of the Core Engines and other components of gateway.

TIBCO API Exchange Gateway supports a distributed deployment environment, in which multiple instances of the Core 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 Core Engine of the gateway cluster, all deployed on a single server host. Using this configuration you can 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](#).

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

Load Balancing

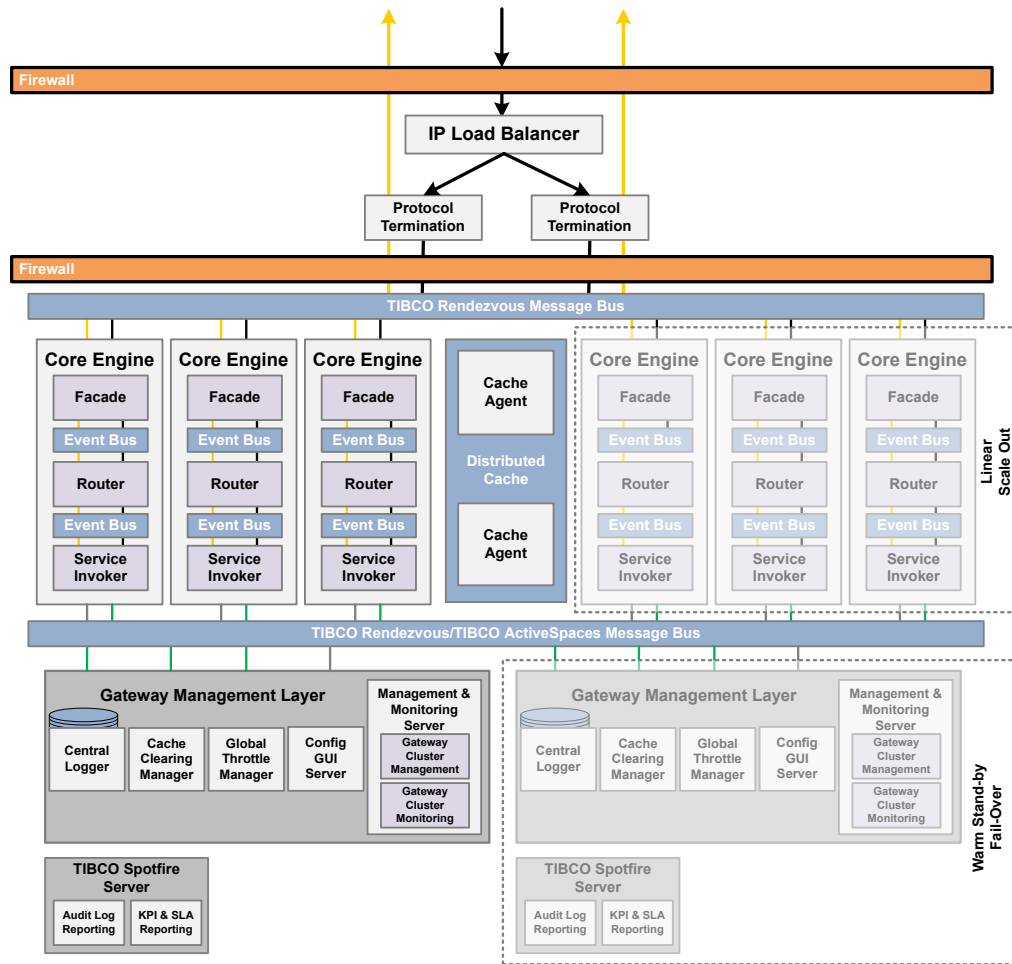
TIBCO API Exchange Gateway can be rapidly scaled up and down through the addition or removal of additional instances of the Core Engines to the gateway cluster.

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 Logger are coordinated across all the Core Engine instances. The components that provide the management functions do not need to be scaled to support the higher transaction volumes.

However, as transaction levels increase, it is likely that this is accompanied by a corresponding increase in management activity. To avoid the possible impact of the management activity on the Core Engines, these management components and the TIBCO Spotfire Servers should be moved onto separate servers.

The following diagram illustrates a simplified view of the scaled solution and depicts the deployment of various components in a distributed environment:

Deployment of Multiple Components in 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. This type of deployment 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 must be configured appropriately.

- For the JMS transport endpoints, load balancing of the requests across multiple Core Engines is achieved by setting up non-exclusive queues in the JMS server. This type of setup automatically balances the load of incoming messages across the JMS receivers of the Core Engine instances.
- For the HTTP transport endpoints, load balancing of the requests across multiple Core Engines is handled by the Module for the Apache HTTP server. If a comma-separated list of TIBCO Rendezvous/TIBCO ActiveSpaces subjects is configured for an Apache server location, the Module for the Apache HTTP server load balances the incoming requests across the list of TIBCO Rendezvous/TIBCO ActiveSpaces subjects. For each deployed Core Engine instance, a different TIBCO Rendezvous/TIBCO ActiveSpaces subject from the list must be configured to ensure that requests are handled only once by a single Core Engine instance.

When the protocol termination components 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 the Apache HTTP servers or JMS servers 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, the configuration setup of the components in the Gateway Operational Layer is different from the setup of 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 multiple engine instances running across sites and the host servers with the same configuration supporting a load balanced configuration. See [Load Balancing](#).

A fault-tolerant setup for JMS endpoints of TIBCO API Exchange Gateway leverages the fault-tolerant setup capabilities of the TIBCO Enterprise Message Service. See *TIBCO Enterprise Message Service™ User's Guide* for details.

Fault tolerance of Cache Agents is handled transparently by the object management layer. For the 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. Cache Agents are used only to implement the association cache. The association cache is automatically rebuilt after complete failure when new transactions are handled by TIBCO API Exchange Gateway. Therefore, Cache Agents do not require a backing store.

Gateway Management Layer

The components of the Gateway Management Layer must be deployed once in a primary-secondary group configuration. The Central Logger and the Global Throttle Manager must have a single running instance at all times to ensure that the Core Engine operates without loss of functionality.

Therefore the Central Logger and Global Throttle Manager must be deployed in fault-tolerant configuration with one active instance engine and one or more standby agents on 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 take over from an active instance in case it fails.

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

- Cache Cleanup Agent
- Configuration GUI Server
- Monitoring and Management Server

Deploy multiple versions of these components across host servers with one instance running. If the running instance goes down, start one of the other instances to regain complete gateway functionality.

Getting Started

This section provides information on how to get started with TIBCO API Exchange Gateway product.

Examples Overview

TIBCO API Exchange Gateway provides sample projects to demonstrate various capabilities of the gateway. Examples include:

- APIExchange
- BookQuery
- BookQueryBE
- Caching
- GetLocation
- ProductQuery

Prerequisites

Complete the following tasks before running the examples:

Install TIBCO API Exchange Gateway

Install the TIBCO API Exchange Gateway software and complete the post installation tasks, as described in *TIBCO API Exchange Gateway Installation Guide*.

Verify the TIBCO API Exchange Gateway Server Status

Verify the installation of the TIBCO API Exchange Gateway server by following the steps listed in the "Checking TIBCO API Exchange Gateway Server Status" section of *TIBCO API Exchange Gateway Installation Guide*.

Install TIBCO Runtime Agent and TIBCO BusinessWorks

The examples require the installation of TIBCO Runtime Agent and TIBCO BusinessWorks products. Refer to the TIBCO API Exchange Gateway readme file at *TIBCO_HOME/release_notes* directory for the supported version of TIBCO Runtime Agent and TIBCO BusinessWorks products.

Examples

TIBCO API Exchange Gateway software ships with sample projects, such as the following examples:

GetLocation

GetLocation example is shipped with a GetLocation BusinessWorks project. The project contains the client operations and back-end services, such as GetLocationService to be executed. The GetLocation service demonstrates the routing capability of TIBCO API Exchange Gateway.

The gateway interacts with a mock-up back-end service that returns the coordinates of a device associated with a phone number. The gateway routes the request to a different back-end service depending on the input phone number in the request.

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

BookQuery

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

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

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

Caching

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

The main components of this example:

- ASG_CBA is a TIBCO BusinessWorks project that simulates an east side service. The service 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, or any other web service.
- A SOAP service, which is hosted on TIBCO API Exchange Gateway. This service type is NOOP which means that it returns a XML document with a received key and a translated value.

After the translated value is received from the service implemented in the ASG_CBA BusinessWorks project, this is plugged into a SOAP payload. This payload is used to call a SOAP service running on 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, it sends a TIBCO Rendezvous message to retrieve that value from the service in the ASG_CBA BusinessWorks project.

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

Configure an Endpoint Operation for TIBCO API Exchange Gateway

Create, configure and run a gateway project configuration.

Refer to the following high level steps to configure a service operation on the TIBCO API Exchange Gateway platform:



- Creating a New Configuration
- Configuring Partner Group
- Configuring Partner Data
- Configuring a Facade Operation
- Configure a Target Operation
- Configuring an Authorization Configuration

- Configuring Routing Configuration
- Testing the Gateway configuration

Creating a New Configuration

Steps to create a new gateway project configuration.


Procedure

1. Launch the GUI. See [Starting GUI](#) for details.
2. Expand the **Projects** node.
3. Click the **Add New Project Configuration**  icon. A new configuration with a default configuration name, `ASG_Config_uniqueNumId`, is created under **Projects** node.
4. Move the mouse pointer over the new project configuration, select the  icon to rename the configuration name as `ASG_Get_Start` and press Enter.

Configuring Partner Group

Steps to configure partner group.

Procedure

1. Click the **ASG_Get_Start** configuration.
2. Click the **Partner Groups** tab.
3. Click the **Add property (+)**  icon on the top menu bar.
4. Type the values for the fields as shown in the following table:


Partner Group Configuration

Parameter	Value
Group Name	supportASG
Email	support@tibco.com
Phone	0019202331999

Configuring Partner Data

Steps to configure partner data.

Procedure

1. Select the **ASG_Get_Start** configuration.
2. Click the **PARTNER** tab on the upper-right tab.
3. Click the **Partners** tab on the top menu bar.
4. Click the **Add property**  icon on the upper-right to create a new partner.
5. Type the values for the fields as shown in the following table:


Partner Data Configuration

Parameter	Value
Partner Name	tibcoASG
Partner Email	support@tibco.com
Partner Phone	0019202331999
Partner Group	supportASG (select from the drop-down list)

Configuring a Facade Operation

Steps to configure a facade operation.

Procedure

1. Select the **ASG_Get_Start** configuration.
2. Click the **ROUTING** tab on the upper-right tab.
3. Click the **Facade Operations** tab on the top menu bar.
4. Click the **Add property**  icon on the upper-right to create a new facade operation.
5. Type the values for the fields as shown in the following table:

Facade Operation Configuration

Parameter	Value
Operation Name	queryBookByAuthorBW
SOAP Action	"/GetBooksByAuthorEndpoint"
Operation URI	/ServerProcesses/GetBooksByAuthorEndpoint
Operation Service Name	MWC




Refer to the [list of special characters](#) that are supported and not supported in Facade Operation URI.

Configuring a Target Operation

Steps to configure a target operation.

Procedure

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


Target Operation Configuration

Parameter	Value
Operation Name	http.GetBooksByAuthor
Type	HTTP (select from the drop-down list.)
Timeout	30000
SOAP Action	"/GetBooksByAuthorEndpoint"
URI	/ServerProcesses/GetBooksByAuthorEndpoint
Host	127.0.0.1
Port	9696

Configuring an Authorization Configuration

Steps to configure a partner operation.

Procedure

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

Partner Authorization Configuration


Parameter	Value
Partner	tibcoASG (select from the drop-down list.)
Facade Operation	queryBookByAuthorBW (select from the drop-down list.)
Partner Timeout	5000

Configuring Routing Configuration

Steps to configure routing information for a request.

Procedure

1. Select the **ASG_Get_Start** configuration.

2. Click the **ROUTING** tab on the upper-right tab.
3. Click the **Routing** tab on the top menu bar.
4. Click the **Add property**  icon on the upper-right to create a new routing configuration.
5. Type in the following values:

Routing Configuration

Parameter	Value
Operation Name	queryBookByAuthorBW (select from the drop-down list.)
Routing Type	Target Operation (select from the drop-down list.)
Routing Key	default
Target Operation	http.GetBooksByAuthor (select from the drop-down list.)

Saving the Gateway Configuration

Save the gateway project configuration.

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

Testing the Gateway configuration

The following steps describe how to test the gateway configuration:

- Running Apache HTTP Server if not running
- Running Core Engine
- Testing the Configured Operation and Target operation

Running Apache HTTP Server if not running

See [Running the Apache HTTP Server](#) for details.

Running Core Engine

How to run Core Engine using a gateway configuration.

Procedure

1. Navigate to the TIBCO API Exchange Gateway installation as follows:

```
cd ASG_HOME/bin
```

2. Start the Core Engine for the **ASG_Get_Start** configuration as follows:

On the Windows platform, type the following command:

```
asg-engine -u asg-caching-core -a ASG_Get_Start
```

On the UNIX platform, type the following command:

```
./asg-engine -u asg-caching-core -a ASG_Get_Start
```

3. Verify that the Core Engine starts successfully without any errors.
4. In the Core Engine log file, verify that the configuration for operation, services and partner are loaded correctly. By default, the Core Engine log file is `asg-caching-core.log` which is located under `ASG_CONFIG_HOME\logs` directory.

Testing the Configured Operation and Target Operation

How to test a target operation.

Procedure

1. Launch TIBCO Designer.
2. Open the following project:
`ASG_HOME/examples/BookQuery/BookQuery`
3. Run the following server process:
`BooksInterface-service1`
4. Run the following client process:
`QueryByAuthorClient`
5. Verify that the process runs successfully without any errors.

Working with Studio

Studio provides the design-time environment for adding custom extensions to a project. It is an Eclipse-based user interface that 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.



Studio is supported only on Windows and Linux platforms.

Starting Studio

Steps to start Studio

Procedure

1. Navigate to the `ASG_HOME/studio/eclipse` directory.
2. Start TIBCO Business Studio.

On the 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
```



You can start the Studio by following the Windows menu **Start > All Programs > TIBCO > TIBCO ENV > TIBCO API Exchange 2.3 > Studio**.

On the LINUX platform, type the following command on a command prompt window:

```
cd ASG_HOME/studio/eclipse
studio
```

3. If you are prompted, select or create the Eclipse workspace directory where your project files are stored. If you select the option to use this workspace as default, you are not prompted again.
A Welcome screen displays when you run the Studio first time. Click the X next to **Welcome** to close the welcome screen.
4. Click **OK**.

Loading the Default ASG_DefaultImplementation Project

Procedure

1. Start the Studio. See [Starting Studio](#).
2. Select **File > Import > TIBCO BusinessEvents > Existing TIBCO BusinessEvents Studio Project**.
3. Click **Next**.
4. On **Existing TIBCO BusinessEvents Project Import Wizard**, select the values for the following fields:
 - a) **Existing project root directory**:
 1. Click **Browse**
 2. Select `ASG_HOME\projects\ASG_DefaultImplementation` project.
 3. Click **OK**.
 - b) Select the **Copy project into workspace** check box.
5. Click **Finish**.
6. In the Studio Explorer, on the upper left, 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 the Default ASG_DefaultImplementation Project

After you import the `ASG_DefaultImplementation` project in 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.

Adding or Edit a Resource in a Project

How to add or edit a resource in the project.

Procedure

1. In Studio Explorer, select the folder where you want to store the new resource and right-click to display the menu.
2. To add a new rulefunction, follow these steps:
 - a) Select the **RuleFunction** node, right-click and select **New > Rule Function**
 - b) Input the values in the Rule Function wizard, as appropriate.
 - c) Enter the code for the rule function in the editor.
 - d) Save the rulefunction.

3. To edit an existing rulefunction, follow these steps:
 - a) Select the rulefunction to be edited.
 - b) Double-click to open the rulefunction in an editor. Modify the function code as required.
 - c) Save the rulefunction.
4. 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 **Save**.

Validating a Project, Project Folder or Project Resource

How to validate a project, project folder, or project resource.

In Studio Explorer, follow one of the following steps:

Procedure

1. Right-click a project name, folder name, or a project resource name, and select **Validate Project**.
2. 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 are displayed in the **Problems** view.

Fixing Validation Errors

How to fix the validation errors using Studio.

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

Building the Default ASG_DefaultImplementation Project

How to build the default ASG_DefaultImplementation project.

You can build the EAR file of the project for testing before deployment.

Procedure

1. In the Studio Explorer, highlight the **ASG_DefaultImplementation** project. From the top menu, select **Project > Build Enterprise Archive**.
If you see a message prompting you to save all project resources, click **Yes**. This means that an unsaved resource editor is open.
2. In the Build Enterprise Archive dialog, complete the values according to the 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**. Click **OK** to build the archive.

Build Enterprise Archive Reference Parameters

Field	Description
Name	The name of the EAR configuration. (Not the EAR filename.) The default value is the project name.
Author	The person responsible for the EAR file. The default value is the currently logged-on user name.
Description	The optional description.
Archive Version	Specifies a version identifier. This increments on each build of the EAR. You can also manually enter a version identifier.
Generate Debug Info	Select this check box if you want to use the debugger. The default setting is selected.
Include all service level global variables	Select to include service level global variables.
File Location	The location where you want to store the EAR file. Browse to the directory to specify the location of EAR file and enter an EAR filename. For example, for the TIBCO API Exchange Gateway Core Engine, the ear file is set as <code>TIBCO_HOME\asg\2.3\bin\asg_core.ear</code> .
Delete Temporary Files	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, 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, clear the Delete Temporary Files check box, and specify where to store the Java files in the Compilation Directory field. The default setting is selected, that is, the temporary files are not saved.
Compilation Directory	If you clear the Delete Temporary Files check box, specify the directory where you want to save the Java files generated during the process of building the EAR file.

Debugging Project in Studio

How to debug the ASG_DefaultImplementation project.

You can set the breakpoints, stepping through the code, suspending launched programs, examining the contents of variables, providing rule input, and so on, using the Studio debugger.

The 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 Studio, follow these steps:

Procedure

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/2.0/bin/asg-engine.tra` to `TIBCO_HOME/be/5.1/bin/be-engine.tra`



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

3. Open the `ASG_CONFIG_HOME/asg/asg.properties` file and copy the entire 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, uncomment and set the following property to the project configuration folder.

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

For example,

- To debug a BookQuery project configuration, set the property as follows:

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

- To enable the generic pageflow, set the java property as follows:

```
java.property.be.engine.channel.pageflow.genericContext.enabled=true
```



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 Studio, if not already started. See [Starting Studio](#).
8. Import the `ASG_HOME/projects/ASG_DefaultImplementation` project in the Studio. See [Loading the Default ASG_DefaultImplementation Project](#).
9. In Studio Explorer, select the **ASG_DefaultImplementation** project. From the menu, select **Run > Debug Configurations**
10. For local debugging, On the Debug Configurations dialog, select and double-click the **TIBCO BusinessEvents Application** node on the left. Ensure that the **ASG_DefaultImplementation** project is selected.
 - a) Select the **Main** tab and configure values as explained in the "Debug Configuration Parameters Reference" table in step 10c.
 - b) Select the **Environment** tab and configure the environment variables as needed, to run or debug the project in Studio.

For example, edit the following variable to set the value as:

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

where PATH is the existing value of the PATH variable.



You can add new variables using the **Environment** tab. You can select and then edit the existing variables. You can append your edited variable to the existing environment variable, or replace the existing environment variable with the specified variable. 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.

- c) Select the **Classpath** tab and configure the class path for external libraries or custom functions, as needed.

For example, if the project uses Rendezvous or JMS channels, then you can add the Rendezvous or JMS libraries to the class path.



Debug Configuration Parameters Reference

Field	Notes
Name	A descriptive name. It appears in the drop-down list of configurations.
Main Tab	
Project	Browse to select the ASG_DefaultImplementation project. This appears by default as this project is currently selected in the Studio.
VM Arguments	Optional. Provide options and parameters using -V, -D, -X and so on. For example, <ul style="list-style-type: none"> To set global variables use: <code>-VVariable=value</code> To set properties for generic page flow, set java property as follows: <code>-Dbe.engine.channel.pageflow.genericContext.enabled=true</code>
CDD File	Browse to select the CDD file as <code>ASG_HOME\bin\asg_core.cdd</code> to be used for this debug configuration.
Processing Unit	Specifies the name of the processing unit (PU) whose values are used for this debug configuration. The drop-down list displays PUs available in the <code>asg_core.cdd</code> file. For example, set this value as <code>asg-caching-core</code> .
Working Directory	The location of the working directory for the Core Engine. This location is used to store temporary files and logs. Browse and select an existing directory. Path names that do not start with the root directory are assumed by the operating system to start from the working directory. For example, on the Windows platform, set the working directory as follows: <code>C:\temp\asgworkarea</code>
EAR File	Browse to select the EAR file to be used for this debug configuration. The EAR file must be generated with the Generate Debug Info option selected on the Project > Build Enterprise Archive . See Editing Validating and Building the Default ASG_DefaultImplementation Project to build the EAR file. Set this value as: <code>TIBCO_HOME\asg\2.3\bin\asg_core.ear</code>

- On the **Main** tab, click **Apply** to save the debug configuration changes.
- Click **Debug** to launch the debugger. Verify that the debugger is launched.

Setting the Debug Perspective within Studio

Procedure

1. Switch to the Debug perspective. Select **Window > Open Perspective** , or click **Open Perspective** (). Then select **Other > Debug** .
2. Click the down-arrow to the right of the debugger () button to display a drop-down list. Follow one of the following steps:
 - Select a debug configuration from the list.
 - Select **Debug Configurations**. From the Debug Configurations dialog, select a debug configuration and click **Debug**.
3. Verify that the debugger is launched.

Validation Tool (asg-validate)

Use validation tool to validate the configuration data for a gateway project.

The asg-validate tool checks for a complete and correct configuration set using the Config UI before the configuration can be used by the Core Engine at run time.

The asg-validate tool loads the data from all the configuration (cfg) files for a gateway configuration project into the memory and validates it against each other. The asg-validate also loads and compiles all the XSLT files for a gateway configuration project and therefore it checks if the XSLT files are valid.

The asg-validate tool ensures that the data in one configuration file that is dependent on the other configuration file is valid.

For example, you have defined a partner P_1 and assigned it to a partner group PG_1 for a myconfig project configuration. When you run asg-validate for myconfig project, it loads partner P_1 data into memory and finds that this is assigned to PG_1 partner group. Then it checks if the data of partner group PG_1 is also available in the memory of the Core Engine. If it does not find PG_1 partner group data, it throws an error indicating that partner group PG_1 is not defined in Partner Groups.

If all data such as partner, target operations, facade operations, throttles, and so on, is valid and correct for a project configuration, it displays a successful message as: The ASG configuration status is OK . Otherwise, it reports errors on the console.

You can find the output of the asg-validate tool in a log file. See [Log File for asg-validate](#).

Running asg-validate Using asg-tools

Utility to validate configuration data for a gateway project.

You can use the asg-tools utility to perform the validation for a gateway project configuration.

Procedure

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

```
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

Set the location of log file for asg-validate tool.

The output of the asg-validate tool is stored in a log file named `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 follows:

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

Limitations of asg-validate

Defines the limitations of asg-validate tool.

The asg-validate mainly validates the integrity of 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 Config UI uses the asg-validate tool to validate the configuration data entered on the UI. It is good practice to run the validation after you have entered the complete data for a project configuration set. This helps to find the errors at design time so that they can be fixed earlier and prevents errors at run time for the Core Engine.

Runtime Properties

Runtime properties for the Core Engine and the Central Logger component of the gateway.


The runtime properties of the Core Engine and the Central Logger component are defined in the `asg.properties` and `asg_cl.properties` files respectively.

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

- By using the Config UI:

To set the properties on the home page, select the **Gateway Engine Properties** from the drop-down



list next to the  **Gateway Engine Properties** icon.

- By using a text editor:

Directly edit the `asg.properties` and `asg_cl.properties` files using a text editor. The files are located in the `ASG_CONFIG_HOME` directory.

Runtime Properties of Core Engine

Explains the runtime properties of Core Engine.

The properties for the Core Engine are defined in the `asg.properties` file located in `ASG_CONFIG_HOME` directory. The following properties can be defined:

Core Engine Properties

Property Name	Description
<code>tibco.clientVar.ASG/HttpClient/useSynchHttpClient</code>	

Property Name	Description
	<p>A Boolean field used to send a request to the target side by running the http client in the same thread as the facade processing. Use the synchronous http client when a high load of short lived requests are expected</p> <p>If the value is set to true, the client sends a synchronous request for HTTP transport. If the value is set to false, an asynchronous request is sent for HTTP transport.</p> <p>The default is false.</p>
<code>tibco.clientVar.ASG/HttpClient/workers</code>	
	<p>The number of threads used for HTTP Client. This is relevant only if the value of the useSynchHttpClient is false (that is, when an asynchronous request is sent for HTTP transport).</p> <p>The default is 10.</p>
<code>tibco.clientVar.ASG/modRV/facade_request</code>	
	<p>Specifies the Rendezvous subject name which is used by the Core Engine to listen for requests from the Apache module.</p> <p>The default is <code>_LOCAL.asg.north.request</code>.</p>
<code>tibco.clientVar.ASG/modRV/facade_request_binary</code>	
	<p>Specifies the Rendezvous subject name which is used by the Core Engine to listen for binary requests from the Apache module.</p> <p>The default is <code>_LOCAL.asg.north.request_binary</code>.</p>
<code>tibco.clientVar.ASG/modRV/facade_response</code>	
	<p>Specifies the Rendezvous subject to send the response when the Apache module is configured not to use the auto-generated reply subject.</p> <p>If the Apache module is configured to use the reply subject, this is ignored.</p>
<code>tibco.clientVar.ASG/modRV/RvDaemon</code>	
	<p>Specifies the value of the Rendezvous daemon for the Core Engine to connect and listen for the requests from the Apache module.</p>
<code>tibco.clientVar.ASG/modRV/RvNetwork</code>	
	<p>Specifies the value of the Rendezvous network for the Core Engine to connect and listen for the requests from the Apache module.</p>
<code>tibco.clientVar.ASG/modRV/RvService</code>	
	<p>Specifies the value of the Rendezvous service for the Core Engine to connect and listen for the requests from the Apache module.</p>
<code>tibco.clientVar.ASG/GTM/RV/RvDaemon</code>	

Property Name	Description
	Specifies the value of the Rendezvous daemon for the Core Engine to connect to the Global Throttle Manager.
<code>tibco.clientVar.ASG/GTM/RV/RvNetwork</code>	
	Specifies the value of the Rendezvous network for the Core Engine to connect to the Global Throttle Manager.
<code>tibco.clientVar.ASG/GTM/RV/RvService</code>	
	Specifies the value of the Rendezvous service for the Core Engine to connect to the Global Throttle Manager.
<code>tibco.clientVar.ASG/CL/RV/RvDaemon</code>	
	Specifies the value of the Rendezvous daemon for the Core Engine to send messages to the Central Logger.
<code>tibco.clientVar.ASG/CL/RV/RvNetwork</code>	
	Specifies the value of the Rendezvous network for the Core Engine to send messages to the Central Logger.
<code>tibco.clientVar.ASG/CL/RV/RvService</code>	
	Specifies the value of the Rendezvous service for the Core Engine to send messages to the Central Logger.
<code>tibco.clientVar.Common/Connections/RV/SubjectPrefix</code>	
	Specifies the prefix for all Rendezvous subject names used between the Core Engine and the Central Logger, the Core Engine and the Global Throttle Manager components. The default is <code>TIBCO.ASG.INTERNAL</code> .
<code>tibco.clientVar.ASG/Deployments/AllowHotUpdate</code>	
	Set this field as <code>true</code> or <code>false</code> to enable or disable hot configuration updates. The default is <code>false</code> .
<code>tibco.clientVar.ASG/ConfigRoot</code>	
	<ul style="list-style-type: none"> Specifies the root directory for an entire configuration. If this property is set, the configuration value specified for this property takes precedence over the configuration specified by <code>asg-engine</code> command-line option such as <code>asg-engine.exe -a <Config project name></code>. <p>For example, the root directory for default configuration is: <code>C:/ProgramData/ASG200/tibco/cfgmgmt/asg/default</code>.</p>

Property Name	Description
<code>tibco.clientVar.EXAMPLES_HOME</code>	<p>Specifies the home directory for the examples shipped with TIBCO API Exchange Gateway.</p> <p>For example, the home directory for the examples directory is:</p> <p><code>ASG_HOME/examples</code></p> <p>.</p>
<code>tibco.clientVar.ASG/Response/DefaultContentType</code>	<p>Specifies the default content-type and format of error response content when the error message is returned from the TIBCO API Exchange Gateway for any incoming request. The possible values are as follows:</p> <ul style="list-style-type: none"> <code>application/json</code> <code>application/xml</code> <code>text/xml</code> <p>The default value is <code>application/json</code>.</p>
<code>tibco.clientVar.ASG/Logging/reportingEnabled</code>	<p>Specifies if reporting to the Central Logger is enabled or not. By default, the Core Engine does not record the transactions to Central Logger.</p> <p>The default is <code>false</code>.</p> <p>See Enabling Reporting to the Central Logger.</p>
<code>tibco.clientVar.ASG/Logging/interval</code>	<p>Specifies the time interval (in milliseconds) between the Core Engine and the Central Logger to record transactions.</p> <p>The default is <code>30000</code>.</p>
<code>tibco.clientVar.ASG/Logging/MinLogLevel</code>	<p>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 transaction.</p> <p>The default is <code>0</code>.</p>
<code>tibco.clientVar.ASG/Throttle/UpdateIntervalSec</code>	<p>Specifies the time interval (in seconds) for sending throttle updates to the Global Throttle Manager.</p> <p>The default is <code>10</code>.</p>
<code>tibco.clientVar.ASG/anonymous/PartnerName/Authenticated</code>	

Property Name	Description
	Specifies the default partner name for the unauthenticated requests. The default value is anon.
<code>tibco.clientVar.ASG/Endpoint/ESB/EnableTempQueueResponse</code>	
	Specifies the usage of temporary queue, instead of a dedicated response queue. The property enables every ESB response to a temporary queue. The default is false.
<code>tibco.clientVar.ASG/Endpoint/ESB0/requestQueue</code>	
	Specifies the queue name for an Enterprise Service Bus (ESB) channel (one) communication for the target operation request. The default queue name is <code>asg.out.request</code> .
<code>tibco.clientVar.ASG/Endpoint/ESB0/replyQueue</code>	
	Specifies the queue name for ESB channel (one) communication for the target operation response. The default queue name is <code>asg.out.request.reply.0.0</code> .
<code>tibco.clientVar.ASG/Endpoint/ESB1/requestQueue</code>	
	Specifies the queue name for an ESB channel (two) communication for the target operation request. The default queue name is <code>asg.out.request</code> .
<code>tibco.clientVar.ASG/Endpoint/ESB1/replyQueue</code>	
	Specifies the queue name for an ESB channel (two) communication for the target operation response. The default queue name is <code>asg.out.request.reply.0.1</code> .
<code>tibco.clientVar.ASG/Endpoint/ESB2/requestQueue</code>	
	Specifies the queue name for an ESB channel (three) communication for the target operation request. The default queue name is <code>asg.out.request</code> .
<code>tibco.clientVar.ASG/Endpoint/ESB2/replyQueue</code>	
	Specifies the queue name for an ESB channel (three) communication for the target operation response. The default queue name is <code>asg.out.request.reply.0.2</code> .
<code>tibco.clientVar.ASG/facade/ESB0/requestQueue</code>	

Property Name	Description
	<p>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).</p> <p>The default queue name is <code>asg.in.request</code>.</p>
<code>tibco.clientVar.ASG/facade/ESB0/replyQueue</code>	
	<p>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).</p> <p>The default queue name is <code>asg.in.request.reply.0</code></p>
<code>tibco.clientVar.ASG/facade/SOAPJMS/requestQueue</code>	
	<p>Specifies the queue name to be used for the facade requests sent by the client when the inbound transport is SOAP JMS.</p> <p>The default queue name is <code>asg.soap.in.request</code>.</p>
<code>tibco.clientVar.ASG/facade/SOAPJMS/replyQueue</code>	
	<p>Specifies the queue name to be used for the facade responses received for the client requests when the inbound transport is SOAP JMS.</p> <p>The default queue name is <code>asg.soap.in.request.reply.0</code>.</p>
<code>tibco.clientVar.ASG/Endpoint/SOAPJMS/DefaultTargetRequestQueue</code>	
	<p>Specifies the queue name to be used for the requests sent to target operations when the inbound transport is SOAP JMS.</p> <p>The default queue name is <code>asg.soap.forward</code>.</p>
<code>tibco.clientVar.ASG/Endpoint/SOAPJMS/TargetResponseQueue</code>	
	<p>Specifies the queue name to be used for the target responses received from the target operations when the inbound transport is SOAP JMS.</p> <p>The default queue name is <code>asg.soap.forward.reply.0</code></p>
EMS Server Connections (Facade ESB Channel)	
<code>tibco.clientVar.Common/Connections/JMS/FacadeEsbConnection0/JMSProviderURL</code>	
	<p>Specifies the connection URL for the EMS Server used for facade operation requests from ESB communication domain. ESB communication uses JMS transport with XML.</p> <p>The default is <code>tcp://localhost:7222</code>.</p>
<code>tibco.clientVar.Common/Connections/JMS/FacadeEsbConnection0/JNDIContextURL</code>	
	<p>Specifies the URL to the JNDI service provider used for facade operation requests with ESB communication domain.</p> <p>The default is <code>tibjmsnaming://localhost:7222</code>.</p>

Property Name	Description
<code>tibco.clientVar.Common/Connections/JMS/FacadeEsbConnection0/TopicConnectionFactoryName</code>	
	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. The default is TopicConnectionFactory.
<code>tibco.clientVar.Common/Connections/JMS/FacadeEsbConnection0/QueueConnectionFactoryName</code>	
	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. The default is QueueConnectionFactory.
<code>tibco.clientVar.Common/Connections/JMS/FacadeEsbConnection0/JNDIUsername</code>	
	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. The default is admin.
<code>tibco.clientVar.Common/Connections/JMS/FacadeEsbConnection0/JNDIPassword</code>	
	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.
<code>tibco.clientVar.Common/Connections/JMS/FacadeEsbConnection0/JMSUsername</code>	
	Specifies the user name for logging into the EMS server in the ESB communication domain at the facade side. The default is admin.
<code>tibco.clientVar.Common/Connections/JMS/FacadeEsbConnection0/JMSPassword</code>	
	Specifies the password for logging into the EMS server in the ESB communication domain at the facade side.
EMS Server Connections for ESB Channel1 at Target Side	
<code>tibco.clientVar.Common/Connections/JMS/TargetEsbConnection0/JMSProviderURL</code>	
	Specifies the connection URL for the EMS Server used for target operation requests sent to back-end services in ESB communication domain. ESB communication uses JMS transport with XML. The default is <code>tcp://localhost:7222</code> .
<code>tibco.clientVar.Common/Connections/JMS/TargetEsbConnection0/JNDIContextURL</code>	

Property Name	Description
	<p>Specifies the URL to the JNDI service provider used for target operation requests sent to back-end services in ESB communication domain.</p> <p>The default value is tibjmsnaming://localhost:7222</p>
<code>tibco.clientVar.Common/Connections/JMS/TargetEsbConnection0/TopicConnectionFactoryName</code>	
	<p>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.</p> <p>The default value is TopicConnectionFactory .</p>
<code>tibco.clientVar.Common/Connections/JMS/TargetEsbConnection0/QueueConnectionFactoryName</code>	
	<p>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.</p> <p>The default value is QueueConnectionFactory .</p>
<code>tibco.clientVar.Common/Connections/JMS/TargetEsbConnection0/JNDIUsername</code>	
	<p>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.</p> <p>The default value is admin.</p>
<code>tibco.clientVar.Common/Connections/JMS/TargetEsbConnection0/JNDIPassword</code>	
	<p>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.</p>
<code>tibco.clientVar.Common/Connections/JMS/TargetEsbConnection0/JMSUsername</code>	
	<p>Specifies the user name for logging into the EMS server in the ESB communication domain at the target operation side.</p> <p>The default value is admin.</p>
<code>tibco.clientVar.Common/Connections/JMS/TargetEsbConnection0/JMSPassword</code>	
	<p>Specifies the password for logging into the EMS server in the ESB communication domain at the target operation side.</p>
EMS Server Connections for ESB Channel2 at Target Side	
<code>tibco.clientVar.Common/Connections/JMS/TargetEsbConnection1/JMSProviderURL</code>	

Property Name	Description
	<p>Specifies the connection URL for the EMS Server used for target operation requests sent to back-end services in ESB communication domain. ESB communication uses JMS transport with XML.</p> <p>The default is <code>tcp://localhost:7222</code> .</p>
<code>tibco.clientVar.Common/Connections/JMS/TargetEsbConnection1/JNDIContextURL</code>	
	<p>Specifies the URL to the JNDI service provider used for target operation requests sent to back-end services in ESB communication domain.</p> <p>The default value is <code>tibjmsnaming://localhost:7222</code>.</p>
<code>tibco.clientVar.Common/Connections/JMS/TargetEsbConnection1/TopicConnectionFactoryName</code>	
	<p>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.</p> <p>The default value is <code>TopicConnectionFactory</code>.</p>
<code>tibco.clientVar.Common/Connections/JMS/TargetEsbConnection1/QueueConnectionFactoryName</code>	
	<p>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.</p> <p>The default value is <code>QueueConnectionFactory</code>.</p>
<code>tibco.clientVar.Common/Connections/JMS/TargetEsbConnection1/JNDIUsername</code>	
	<p>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.</p> <p>The default value is <code>admin</code></p>
<code>tibco.clientVar.Common/Connections/JMS/TargetEsbConnection1/JNDIPassword</code>	
	<p>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.</p>
<code>tibco.clientVar.Common/Connections/JMS/TargetEsbConnection1/JMSUsername</code>	
	<p>Specifies the user name for logging into the EMS server in the ESB communication domain at the target operation side.</p> <p>The default value is <code>admin</code>.</p>
<code>tibco.clientVar.Common/Connections/JMS/TargetEsbConnection1/JMSPassword</code>	
	<p>Specifies the password for logging into the EMS server in the ESB communication domain at the target operation side.</p>

Property Name	Description
EMS Server Connections for ESB Channel3 at Target Side	
<code>tibco.clientVar.Common/Connections/JMS/TargetEsbConnection2/JMSProviderURL</code>	
	<p>Specifies the connection URL for the EMS Server used for target operation requests sent to back-end services in ESB communication domain. ESB communication uses JMS transport with XML.</p> <p>The default value is <code>tcp://localhost:7222</code>.</p>
<code>tibco.clientVar.Common/Connections/JMS/TargetEsbConnection2/JNDIContextURL</code>	
	<p>Specifies the URL to the JNDI service provider used for target operation requests sent to back end services in ESB communication domain.</p> <p>The default value is <code>tibjmsnaming://localhost:7222</code>.</p>
<code>tibco.clientVar.Common/Connections/JMS/TargetEsbConnection2/TopicConnectionFactoryName</code>	
	<p>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.</p> <p>The default value is <code>TopicConnectionFactory</code>.</p>
<code>tibco.clientVar.Common/Connections/JMS/TargetEsbConnection2/QueueConnectionFactoryName</code>	
	<p>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.</p> <p>The default value is <code>QueueConnectionFactory</code>.</p>
<code>tibco.clientVar.Common/Connections/JMS/TargetEsbConnection2/JNDIUsername</code>	
	<p>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.</p> <p>The default value is <code>admin</code>.</p>
<code>tibco.clientVar.Common/Connections/JMS/TargetEsbConnection2/JNDIPassword</code>	
	<p>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.</p>
<code>tibco.clientVar.Common/Connections/JMS/TargetEsbConnection2/JMSUsername</code>	
	<p>Specifies the user name for logging into the EMS server in the ESB communication domain at the target operation side.</p> <p>The default value is <code>admin</code>.</p>

Property Name	Description
<code>tibco.clientVar.Common/Connections/JMS/TargetEsbConnection2/JMSPassword</code>	
	Specifies the password for logging into the EMS server in the ESB communication domain at the target operation side.
Connection Parameters for SOAP JMS (Facade)	
<code>tibco.clientVar.Common/Connections/JMS/SOAPConnection_Facade/JMSProviderURL</code>	
	Specifies the connection URL for the EMS Server used for client requests when SOAP JMS transport is used at the facade side. The default value is <code>tcp://localhost:7222</code> .
<code>tibco.clientVar.Common/Connections/JMS/SOAPConnection_Facade/JNDIContextURL</code>	
	Specifies the URL to the JNDI service provider used for client requests when SOAP JMS transport is used at the facade side. The default value is <code>tibjmsnaming://localhost:7222</code> .
<code>tibco.clientVar.Common/Connections/JMS/SOAPConnection_Facade/TopicConnectionFactoryName</code>	
	Specifies the name of TopicConnectionFactory object stored in JNDI. This object is used to create a topic connection with JMS application at facade side. The default value is <code>TopicConnectionFactory</code> .
<code>tibco.clientVar.Common/Connections/JMS/SOAPConnection_Facade/QueueConnectionFactoryName</code>	
	Specifies the name of QueueConnectionFactory object stored in JNDI. This object is used to create a queue connection with JMS application at facade side. The default value is <code>QueueConnectionFactory</code> .
<code>tibco.clientVar.Common/Connections/JMS/SOAPConnection_Facade/JNDIUsername</code>	
	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. The default value is <code>admin</code> .
<code>tibco.clientVar.Common/Connections/JMS/SOAPConnection_Facade/JNDIPassword</code>	
	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.
<code>tibco.clientVar.Common/Connections/JMS/SOAPConnection_Facade/JMSUsername</code>	

Property Name	Description
	<p>Specifies the user name for logging into the EMS server used to send client requests at facade side.</p> <p>The default value is admin.</p>
<code>tibco.clientVar.Common/Connections/JMS/SOAPConnection_Facade/JMSPassword</code>	
	<p>Specifies the password for logging into the EMS server used to send client requests at facade side.</p>
Connection Parameters for SOAP JMS (Target)	
<code>tibco.clientVar.Common/Connections/JMS/SOAPConnection_Target/JMSProviderURL</code>	
	<p>Specifies the connection URL for the EMS Server when SOAP JMS transport is used at the target operation side.</p> <p>The default value is <code>tcp://localhost:7222</code>.</p>
<code>tibco.clientVar.Common/Connections/JMS/SOAPConnection_Target/JNDIContextURL</code>	
	<p>Specifies the URL to the JNDI service provider when SOAP JMS transport is used at the target operation side.</p> <p>The default value is <code>tibjmsnaming://localhost:7222</code>.</p>
<code>tibco.clientVar.Common/Connections/JMS/SOAPConnection_Target/TopicConnectionFactoryName</code>	
	<p>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.</p> <p>The default value is TopicConnectionFactory.</p>
<code>tibco.clientVar.Common/Connections/JMS/SOAPConnection_Target/QueueConnectionFactoryName</code>	
	<p>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.</p> <p>The default value is QueueConnectionFactory .</p>
<code>tibco.clientVar.Common/Connections/JMS/SOAPConnection_Target/JNDIUsername</code>	
	<p>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.</p> <p>The default value is admin.</p>
<code>tibco.clientVar.Common/Connections/JMS/SOAPConnection_Target/JNDIPassword</code>	
	<p>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.</p>

Property Name	Description
<code>tibco.clientVar.Common/Connections/JMS/SOAPConnection_Target/JMSUsername</code>	
	Specifies the user name for logging into the EMS server when SOAP JMS transport is used at the target operation side. The default value is admin.
<code>tibco.clientVar.Common/Connections/JMS/SOAPConnection_Target/JMSPassword</code>	
	Specifies the password for logging into the EMS server when SOAP JMS transport is used at the target operation side.
Connection Parameters for HTTP Channel(Facade)	
<code>tibco.clientVar.DefaultImplementation/Connections/HTTP/FacadeHTTPConnection/Host</code>	
	Specifies the host where the Core Engine runs and accepts HTTP requests from the client. The default value is localhost.
<code>tibco.clientVar.DefaultImplementation/Connections/HTTP/FacadeHTTPConnection/Port</code>	
	Specifies the port through which the Core Engine accepts HTTP requests from the client. The default value is 9222.
<code>tibco.clientVar.DefaultImplementation/Connections/HTTP/FacadeHTTPConnection/ConnectionTimeout</code>	
	Specifies the timeout value for HTTP native transport when the target service takes more than one minute to send the response to client.
<code>tibco.clientVar.DefaultImplementation/Connections/HTTP/FacadeHTTPConnection/DebugFolder</code>	
	Specifies where the log file should be generated. Default is the currently running folder/logs.
<code>tibco.clientVar.DefaultImplementation/Connections/HTTP/FacadeHTTPConnection/DebugPattern</code>	
	Specifies the log pattern. The default log pattern is used.
Connection Parameters for HTTPS Channel(Facade)	
<code>tibco.clientVar.DefaultImplementation/Connections/HTTP/FacadeHTTPSSLConnection/Port</code>	
	Specifies the port through which the Core Engine accepts SSL enabled HTTP requests from client. The default value is 9233.

Property Name	Description
<code>tibco.clientVar.DefaultImplementation/Connections/HTTP/FacadeHTTPSSLConnection/useSSL</code>	
	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.
<code>tibco.clientVar.DefaultImplementation/Connections/HTTP/FacadeHTTPSSLConnection/Identity</code>	
	Specifies an identity resource which is used by FacadeHTTPSSLConnection HTTP shared resource to provide SSL properties.
<code>tibco.clientVar.DefaultImplementation/Connections/HTTP/FacadeHTTPSSLConnection/IdentityFileType</code>	
	<ul style="list-style-type: none"> Specifies the type of identity resource. The possible values are as follows: <ul style="list-style-type: none"> <code>url</code> <code>certPlusKeyURL</code> Set the IdentityFileType property to url for Identity File type. Set the IdentityFileType property to certPlusKeyURL for Certificate/Private Key type. If the IdentityFileType property is set to url, set the IdentityType, IdentityURL, and IdentityFilePassword properties. If the IdentityFileType property is set to certPlusKeyURL, set the CertificateURL, KeyURL and IdentityFilePassword properties.
<code>tibco.clientVar.DefaultImplementation/Connections/HTTP/FacadeHTTPSSLConnection/IdentityType</code>	
	<p>Specifies the type of the file if the IdentityFileType property is set to url. The supported values are as follows:</p> <ul style="list-style-type: none"> <code>JCEKS</code> <code>JKS</code> <code>PEM</code> <code>PKCS12</code>
<code>tibco.clientVar.DefaultImplementation/Connections/HTTP/FacadeHTTPSSLConnection/IdentityURL</code>	
	<p>Specifies the URL to the identity file if the IdentityFileType property is set to url.</p> <p>For example, C:\keystore.jks</p>
<code>tibco.clientVar.DefaultImplementation/Connections/HTTP/FacadeHTTPSSLConnection/IdentityFilePassword</code>	

Property Name	Description
	<p>Specifies the password of the identity file or private key.</p> <ul style="list-style-type: none"> if the <code>IdentityFileType</code> property is set to <code>url</code>, this refers to the password for the identity file. if the <code>IdentityFileType</code> property is set to <code>certPlusKeyURL</code>, this refers the password for the private key.
<code>tibco.clientVar.DefaultImplementation/Connections/HTTP/FacadeHTTPSSLConnection/CertificateURL</code>	
	<p>Specifies the URL to the certificate file if the <code>IdentityFileType</code> property is set to <code>certPlusKeyURL</code>.</p> <p>For example, <code>C:\mydomain.csr</code></p>
<code>tibco.clientVar.DefaultImplementation/Connections/HTTP/FacadeHTTPSSLConnection/KeyURL</code>	
	<p>Specifies the URL to the private key in certificate file if the <code>IdentityFileType</code> property is set to <code>certPlusKeyURL</code>.</p> <p>For example, <code>C:\keystore.jks</code></p>
<code>tibco.clientVar.DefaultImplementation/Connections/HTTP/FacadeHTTPSSLConnection/RequiresClientAuthentication</code>	
	<p>Indicates a boolean flag to enable or disable mutual SSL authentication for HTTPS transport between the client and the gateway.</p> <p>When this field is set to <code>true</code>, the Trusted Certificates Folder becomes enabled so that you can specify a location containing the list of trusted certificate authorities.</p>
<code>tibco.clientVar.DefaultImplementation/Connections/HTTP/FacadeHTTPSSLConnection/TruststorePassword</code>	
	<p>Specifies the password to access the certificate stored in the folder defined by <code>TrustedCertificateFolder</code> property.</p>
<code>tibco.clientVar.DefaultImplementation/Connections/HTTP/FacadeHTTPSSLConnection/TrustedCertificateFolder</code>	
	<p>Required when the <code>RequiresClientAuthentication</code> property is set to <code>true</code>.</p> <p>Specifies a folder in the project containing one or more certificates from trusted certificate authorities, which is required for mutual SSL authentication. This folder is checked when a client connects to ensure that the client is trusted.</p>
<code>tibco.clientVar.DefaultImplementation/Connections/HTTP/FacadeHTTPSSLConnection/SSLServerProtocols</code>	

Property Name	Description
	<p>Enables the SSLv3 support for the northbound request received through the native HTTP channel. The possible values are as follows :</p> <ul style="list-style-type: none"> • TLSv1 • TLSv1.1 • TLSv1.2
<code>tibco.clientVar.DefaultImplementation/Connections/HTTP/FacadeHTTPSSLConnection/ConnectionTimeout</code>	
	Specifies the timeout value for HTTPS native transport when the target service takes more than one minute to send the response to client.
<code>tibco.clientVar.DefaultImplementation/Connections/HTTP/FacadeHTTPSSLConnection/DebugFolder</code>	
	Specifies where the log file should be generated. Default is the currently running folder/logs.
<code>tibco.clientVar.DefaultImplementation/Connections/HTTP/FacadeHTTPSSLConnection/DebugPattern</code>	
	Specifies the log pattern. The default log pattern is used.
OAuth Properties	
<code>tibco.clientVar.DefaultImplementation/Connections/HTTP/OAuthWebappsConnection/Host</code>	
	Specifies the host IP address of the TIBCO API Exchange Gateway OAuth Server.
<code>tibco.clientVar.DefaultImplementation/Connections/HTTP/OAuthWebappsConnection/Port</code>	
	<p>Specifies the non-SSL port number of the TIBCO API Exchange Gateway OAuth Server.</p> <p>The default value is 9322.</p>
<code>tibco.clientVar.DefaultImplementation/Connections/HTTP/OAuthWebappsConnection/ConnectionTimeout</code>	
	Specifies the timeout value for OAuth HTTP transport when the target service takes more than one minute to send the response to client.
<code>tibco.clientVar.DefaultImplementation/Connections/HTTP/OAuthWebappsConnection/Webapps</code>	
	<ul style="list-style-type: none"> • Specifies the location of the OAuth web application. • Change this only if you want to add custom login page or access grant page to the OAuth server.

Property Name	Description
<code>tibco.clientVar.DefaultImplementation/Connections/HTTP/OAuthWebappsSSLConnection/Port</code>	
	<p>Specifies the SSL port number of the TIBCO API Exchange Gateway OAuth Server.</p> <p>The default value is 9333.</p>
<code>tibco.clientVar.DefaultImplementation/Connections/HTTP/OAuthWebappsSSLConnection/useSSL</code>	
	<p>A Boolean field that indicates if SSL should be enabled for accepting HTTPS requests for OAuth APIs and servlets. If set to <code>true</code> SSL is enabled to accept the requests using HTTPS transport for the OAuth server.</p> <p>The default value is <code>true</code>.</p>
<code>tibco.clientVar.DefaultImplementation/Connections/HTTP/OAuthWebappsSSLConnection/Identity</code>	
	<p>Specifies an identity resource which is used by OAuthWebappsSSLConnection HTTP shared resource to provide SSL properties for OAuth servlets and APIs.</p> <p>The default value is <code>/DefaultImplementation/SharedResources/HTTP/OAuthIdentityResource.id</code>.</p>
<code>tibco.clientVar.DefaultImplementation/Connections/HTTP/OAuthWebappsSSLConnection/IdentityFileType</code>	
	<ul style="list-style-type: none"> • Specifies the type of identity resource set for OAuth server. <p>The possible values are as follows:</p> <ul style="list-style-type: none"> • <code>url</code>. • <code>certPlusKeyURL</code>. • Set the <code>IdentityFileType</code> property to <code>url</code> for Identity File type. • Set the <code>IdentityFileType</code> property to <code>certPlusKeyURL</code> for Certificate/Private Key type. • If the <code>IdentityFileType</code> property is set to <code>url</code>, set the <code>IdentityType</code>, <code>IdentityURL</code>, and <code>IdentityFilePassword</code> properties. • If the <code>IdentityFileType</code> property is set to <code>certPlusKeyURL</code>, set the <code>CertificateURL</code>, <code>KeyURL</code> and <code>IdentityFilePassword</code> properties.
<code>tibco.clientVar.DefaultImplementation/Connections/HTTP/OAuthWebappsSSLConnection/IdentityType</code>	

Property Name	Description
	<p>Specifies the type of the file if the <code>IdentityFileType</code> property is set to <code>url</code>. The supported values are as follows:</p> <ul style="list-style-type: none"> • JCEKS • JKS • PEM • PKCS12
<code>tibco.clientVar.DefaultImplementation/Connections/HTTP/OAuthWebappsSSLConnection/IdentityURL</code>	
	<p>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, <code>C:\keystore.jks</code></p>
<code>tibco.clientVar.DefaultImplementation/Connections/HTTP/OAuthWebappsSSLConnection/IdentityFilePassword</code>	
	<p>Specifies the password of the identity file or private key used for SSL connection to the OAuth server.</p> <ul style="list-style-type: none"> • If the <code>IdentityFileType</code> property is set to <code>url</code>, this refers to the password for the identity file. • If the <code>IdentityFileType</code> property is set to <code>certPlusKeyURL</code>, this refers the password for the private key.
<code>tibco.clientVar.DefaultImplementation/Connections/HTTP/OAuthWebappsSSLConnection/CertificateURL</code>	
	<p>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.</p>
<code>tibco.clientVar.DefaultImplementation/Connections/HTTP/OAuthWebappsSSLConnection/KeyURL</code>	
	<p>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.</p>
<code>tibco.clientVar.DefaultImplementation/Connections/HTTP/OAuthWebappsSSLConnection/RequiresClientAuthentication</code>	
	<p>Indicates a Boolean flag to enable or disable mutual SSL authentication for HTTPS transport used for OAuth server requests from the requestor. When this field is set to <code>true</code>, the Trusted Certificates Folder becomes enabled so that you can specify a location containing the list of trusted certificate authorities.</p>
<code>tibco.clientVar.DefaultImplementation/Connections/HTTP/OAuthWebappsSSLConnection/TruststorePassword</code>	


Property Name	Description
	Specifies the password to access the certificate stored in the folder defined by the <code>TrustedCertificateFolder</code> property.
<code>tibco.clientVar.DefaultImplementation/Connections/HTTP/OAuthWebappsSSLConnection/TrustedCertificateFolder</code>	
	Required when the <code>RequiresClientAuthentication</code> property is set to true. Specifies a folder in the project containing one or more certificates from trusted certificate authorities, which is required for mutual SSL authentication. This folder is checked when a client connects to ensure that the client is trusted.
<code>tibco.clientVar.DefaultImplementation/Connections/HTTP/OAuthWebappsSSLConnection/SSLServerProtocols</code>	
	<p>Enables the SSLv3 support for the northbound request received through the OAuth HTTPS channel. The possible values are as follows :</p> <ul style="list-style-type: none"> • TLSv1 • TLSv1.1 • TLSv1.2
<code>tibco.clientVar.DefaultImplementation/Connections/HTTP/OAuthWebappsSSLConnection/ConnectionTimeout</code>	
	Specifies the timeout value for OAuth HTTPS transport when the target service takes more than one minute to send the response to client.
Enable JMS Transport for Central Logger	
<code>tibco.clientVar.ASG/Logging/transport</code>	
	Specifies the transport to be used between the Core Engine and the Central Logger component to log the transactions.
<code>tibco.clientVar.Common/Connections/JMS/CL_JMSConnection/JMSProviderURL</code>	
	<p>Specifies the URL to connect to the Enterprise Message Service (EMS) or a JMS server.</p> <p>Example: <code>tcp://localhost:7222</code> .</p>
<code>tibco.clientVar.Common/Connections/JMS/CL_JMSConnection/JNDIContextURL</code>	
	<p>Specifies a JNDI connection URL to look up a JMS server.</p> <p>Example:</p> <p><code>tibjmsnaming://localhost:7222.</code></p>
<code>tibco.clientVar.Common/Connections/JMS/CL_JMSConnection/TopicConnectionFactoryName</code>	

Property Name	Description
	<p>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.</p> <p>The default value is TopicConnectionFactory.</p>
<code>tibco.clientVar.Common/Connections/JMS/CL_JMSConnection/QueueConnectionFactoryName</code>	
	<p>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.</p> <p>The default value is QueueConnectionFactory .</p>
<code>tibco.clientVar.Common/Connections/JMS/CL_JMSConnection/JNDIUsername</code>	
	<p>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.</p> <p>For example, admin.</p>
<code>tibco.clientVar.Common/Connections/JMS/CL_JMSConnection/JNDIPassword</code>	
	<p>Specifies the password for logging into the JNDI server. If the JNDI provider does not require access control, this field can be empty.</p>
<code>tibco.clientVar.Common/Connections/JMS/CL_JMSConnection/JMSUsername</code>	
	<p>Specifies the user name to use to authenticate to the JMS server.</p>
<code>tibco.clientVar.Common/Connections/JMS/CL_JMSConnection/JMSPassword</code>	
	<p>Specifies the password to use to authenticate to the JMS server.</p>
<code>tibco.clientVar.Common/Connections/JMS/CL_JMSConnection/TransactionReportDestinationName</code>	
	<p>Specifies the name of the JMS destination to which the transaction reports are sent to the Central Logger by the Core Engine.</p> <p>For example, <code>asg.cl.transaction.queue</code>.</p>
<code>tibco.clientVar.Common/Connections/JMS/CL_JMSConnection/TransactionReportDestinationType</code>	
	<p>Specifies the type of the JMS destination to which the transaction reports are sent to the Central Logger by the Core Engine.</p> <p>The possible values are <code>queue</code> and <code>topic</code>.</p> <p>The default value is <code>queue</code>.</p> <p>The Central Logger always listens on a queue. If the value of destination type set to <code>topic</code>, the JMS administrator must configure a bridge between the topic and the queue.</p>
<code>tibco.clientVar.ConfigApi.ConfigurationBackupProperty</code>	

Property Name	Description
	Specifies the maximum number of latest revisions of the configuration projects to be kept.
SSL JMS Transport Properties for Central Logger	
<code>tibco.clientVar.Common/Connections/JMS/CL_JMSConnection/useSSL</code>	
	Specifies if SSL is enabled for the Central Logger when using the JMS transport. Set this to <code>true</code> to enable SSL for the JMS transport.
<code>tibco.clientVar.Common/Connections/JMS/CL_JMSConnection/TrustedCertificatesFolder</code>	
	If you use the SSL enabled JMS transport for Central Logger, this field specifies a location of the trusted certificates on this machine. The trusted certificates are a collection of certificates from servers with which you establish connections. If the server with which the connection is going to be established, presents a certificate that does not match one of your trusted certificates, the connection is refused. This prevents connections to unauthorized servers. Import the trusted certificates into a folder before you select the folder in this field.
<code>tibco.clientVar.Common/Connections/JMS/CL_JMSConnection/Identity</code>	
	Specifies an identity resource used to provide the SSL properties for JMS transport. The default value is <code>/Common/SharedResources/JMS/CL_JMSConnIdentityResource.id</code> .
<code>tibco.clientVar.Common/Connections/JMS/CL_JMSConnection/TrustStorePassword</code>	
	Specifies the password to access the certificate stored in the folder defined by the Trusted Certificate Folder field. The certificate is required for SSL connection.
<code>tibco.clientVar.Common/Connections/JMS/CL_JMSConnection/trace</code>	
	Specifies whether SSL tracing should be enabled during the SSL connection. If the value is set to <code>true</code> , the SSL connection messages are logged and sent to the console.
<code>tibco.clientVar.Common/Connections/JMS/CL_JMSConnection/debugTrace</code>	
	Specifies whether SSL debug tracing should be enabled during the SSL connection. Debug tracing provides more detailed messages than standard tracing.
<code>tibco.clientVar.Common/Connections/JMS/CL_JMSConnection/verifyHostName</code>	
	Specifies whether the host you are connecting to is the expected host. The host name in the digital certificate of the host is compared against the value in the Expected Host Name field. If the host name does not match the expected host name, the connection is refused.
<code>tibco.clientVar.Common/Connections/JMS/CL_JMSConnection/ExpectedHostName</code>	


Property Name	Description
	Specifies the name of the host you are expecting to connect to. This field is only relevant if the Verify Host Name field is set to <code>true</code> . If the name of the host in the digital certificate of host does not match the value specified in this field, the connection is refused.
<code>tibco.clientVar.Common/Connections/JMS/CL_JMSConnection/strongCipherSuitesOnly</code>	
	When this field is set to <code>true</code> , this specifies the minimum strength of the cipher suites that can be specified with the bw.plugin.security.strongcipher.minstrength custom engine property. See <i>TIBCO ActiveMatrix BusinessWorks Administration</i> for more information about this property. The default value of the property disables cipher suites with an effective key length below 128 bits. When this field is set to <code>false</code> , only cipher suites with an effective key length of up to 128 bits can be used.
<code>tibco.clientVar.Common/Connections/JMS/CL_JMSConnection/IdentityType</code>	
	<p>Speicifies the type of identity resource for SSL connection. The possible values are:</p> <ul style="list-style-type: none"> • <code>certPlusKeyURL</code> • <code>url</code> • <code>usernamePassword</code>
Properties for Identity Type=usernamePassword Use this option if you want to use a user name and password for authentication instead of a certificate.	
<code>tibco.clientVar.Common/Connections/JMS/CL_JMSConnection/IdentityUsername</code>	
	Specifies the name of the user for this identity.
<code>tibco.clientVar.Common/Connections/JMS/CL_JMSConnection/IdentityPassword</code>	
	Specifies the password for the user for this identity
Properties for Identity Type=url Use this option if the certificate includes the private key information in the same file.	
<code>tibco.clientVar.Common/Connections/JMS/CL_JMSConnection/IdentityFileType</code>	

Property Name	Description
	<p>Specifies the type of certificate file if Identity Type is set to url.</p> <p>The accepted values are as follows:</p> <ul style="list-style-type: none"> • Entrust • JCEKS • JKS • PEM • PKCS12
<code>tibco.clientVar.Common/Connections/JMS/CL_JMSConnection/IdentityURL</code>	
	<p>Specifies the location of the certificate (which includes the private key) if Identity Type is set to url.</p> <p>For example, C:\asgserver.pfx</p>
<code>tibco.clientVar.Common/Connections/JMS/CL_JMSConnection/KeyPassword</code>	
	Specifies the password for the certificate if Identity Type is of url type.
Properties for Identity Type=certPlusKeyURL Use this option if the private key and the certificate are in two separate files.	
<code>tibco.clientVar.Common/Connections/JMS/CL_JMSConnection/KeyPassword</code>	
	Specifies the password for the certificate if Identity Type is of certPlusKeyURL type.
<code>tibco.clientVar.Common/Connections/JMS/CL_JMSConnection/CertificateURL</code>	
	Specifies the URL to the certificate file if Identity Type is of certPlusKeyURL type.
<code>tibco.clientVar.Common/Connections/JMS/CL_JMSConnection/KeyURL</code>	
	Specifies the URL to the private key in the certificate file if Identity Type is of certPlusKeyURL type.
OAuth Server Related Settings	
Properties For OAuth Data Space	
<code>tibco.clientVar.oauth.dataspace.metaspace.name</code>	
	<ul style="list-style-type: none"> • Specifies the metaspace name used by the OAuth server. • The default value is ASG-OAuth-Tokens.
<code>tibco.clientVar.oauth.dataspace.local.discovery</code>	

Property Name	Description
	<p>Specifies the discovery URL for this OAuth instance of the metaspace discovers the current metaspace members.</p> <p>For example, <code>tcp://machine1_IP_Address:6300;machine2_IP_Address:6300</code></p>
<code>tibco.clientVar.oauth.dataspace.local.listen</code>	
	<p>Specifies the listening URL for this OAuth instance of the metaspace.</p> <p>For example, <code>tcp://machine1_IP_Address:6300</code></p>
<code>tibco.clientVar.oauth.dataspace.remote.discovery</code>	
	<p>Specifies the remote discovery URL for this OAuth instance of the metaspace discovers the current metaspace members.</p>
<code>tibco.clientVar.oauth.dataspace.remote.listen</code>	
	<p>Specifies the remote listening URL for this OAuth instance of the metaspace.</p>
<code>tibco.clientVar.oauth.dataspace.load.batch.size</code>	
	<p>Specifies the maximum number of entries to return when querying data such as access token.</p>
<code>tibco.clientVar.oauth.access.token.retention.period</code>	
	<p>Specifies the expiration time (in minutes) for an access token. The default value is 60 minutes. When the access token passes expiration time as specified by this property, it is no longer valid but still remains in the database. The access token is removed from the database based on the value specified by the <code>tibco.clientVar.oauth.access.token.retention.time</code> property.</p> <p>Note: In a multi-instance TIBCO API Exchange environment, if you change the TTL value on a gateway instance, shut down all instances that connect to that metaspace and then restart the instances.</p>
<code>tibco.clientVar.oauth.access.token.retention.time</code>	
	<p>Specifies the retention period (in minutes) for an access token. The default value is 1440 minutes (1 day) . When the access token passes retention period as specified by this property, the token is removed from the database. By default, the access token is removed from the database after 1 day.</p> <div>  <p>The value of <code>tibco.clientVar.oauth.access.token.retention.period</code> property must be less than the value specified by the <code>tibco.clientVar.oauth.access.token.retention.time</code> property.</p> </div>
<code>tibco.clientVar.oauth.dataspace.replication.policy</code>	

Property Name	Description
	<p>Specifies the OAuth access token replication policy when more than one OAuth servers are configured in a cluster. The possible policy options are as follows:</p> <ul style="list-style-type: none"> • <code>sync</code>. If you set this property to <code>sync</code> means that as the tokens are added to OAuth servers, they are replicated immediately to all seeders in the cluster. • <code>async</code>. If you set this property to <code>async</code> means that as the tokens are added to OAuth servers, it does not guarantee that the tokens are replicated immediately. <p>The default value is <code>async</code>.</p>
<code>tibco.clientVar.oauth.dataspace.replication.count</code>	
	<p>Specifies the number of seeders that are used to replicate token. If you have <code>n</code> number of OAuth servers, set this property to <code>n-1</code> to replicate the token to all servers. The default value is 0.</p> <p>For example, if you set this property to 1 means that the token is replicated to one additional seeder.</p>
<code>tibco.clientVar.oauth.cache.token.in.session</code>	
	Set this property to <code>false</code> to turn off caching.
Properties For OAuth Persister	
<code>tibco.clientVar.oauth.dataspace.persister.store</code>	
	<ul style="list-style-type: none"> • Defines the type of persistence store. The possible values are: <ul style="list-style-type: none"> – <code>InMemory</code> – <code>Database</code> <p>If the <code>Database</code> is set, you must define the properties for database server connection.</p>
<code>tibco.clientVar.oauth.access.token.include.attributes=true</code>	
	<p>Specifies if the response from token validation API should return the owner's attributes.</p> <p>The default is <code>false</code>.</p>
Properties For OAuth Server Persister Store of Database Type	
<code>tibco.clientVar.oauth.dataspace.persister.jdbc.driver</code>	
	Specifies the database JDBC driver when the database is used as OAuth persistence store.
<code>tibco.clientVar.oauth.dataspace.persister.jdbc.url</code>	

Property Name	Description
	Specifies the JDBC URL for the database server when the database is used as OAuth persistence store.
<code>tibco.clientVar.oauth.dataspace.persister.jdbc.username</code>	
	Specifies the user to connect to the database server when the database is used as OAuth persistence store.
<code>tibco.clientVar.oauth.dataspace.persister.jdbc.password</code>	
	Specifies the password of the user to connect to the database server when the database is used as OAuth persistence store.
Properties For OAuth Adapters	
<code>tibco.clientVar.oauth.adapter.resource.home</code>	
	<p>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 <i>ASG_HOME</i>.</p> <p>For example, if the value is specified as <i>/examples/Adapters/resources</i>, the custom adapter looks for the resources such as properties file in the directory <i>ASG_HOME/examples/Adapters/resources</i>.</p>
<code>tibco.clientVar.oauth.owner.adapter.class</code>	
	<ul style="list-style-type: none"> Specifies the class that provides the Owner Adapter interface. This adapter is used to authenticate the resource owner and provide the login and access grant pages. <p>For example, for file-based owner adapter interface, value specified as: <code>com.tibco.asg.oauth.identity.provider.file.OwnerAdapterService</code></p>
<code>tibco.clientVar.oauth.client.adapter.class</code>	
	Specifies the class that provides the Client Adapter interface.
<code>tibco.clientVar.oauth.scope.adapter.class</code>	
	<ul style="list-style-type: none"> Specifies the class that provides the Scope Adapter interface. This adapter is used to retrieve the scope description and the scope form a specific resource for a given owner. <p>For example, for file-based scope adapter interface, value specified as: <code>com.tibco.asg.oauth.identity.provider.file.ScopeAdapterService</code></p>
Advanced Properties	
<code>tibco.clientVar.ASG/Headers/SoapAction/RemoveQuotes</code>	

Property Name	Description
	<p>Enables the Core Engine to remove quotes from the incoming SOAPAction header of the facade request.</p> <p>Set this property to <code>true</code> to remove single or double quotes from SOAPAction header of the request. If the property is set to <code>false</code> or has no value, the single or double quotes from SOAPAction header are not removed, which is the default behavior.</p>
<code>tibco.clientVar.MessageEncoding</code>	
	<p>Specifies the encoding for the payloads in the request sent to TIBCO API Exchange Gateway through the Apache server using the HTTP transport. The default value is <code>ISO-8859</code>.</p> <p>Set this property to <code>UTF-8</code> to enable the Core Engine to handle non-ASCII characters in the request and response payloads of the request sent to TIBCO API Exchange Gateway using the RV channel through the Apache server.</p>
<code>tibco.clientVar.ASG/RV/Headers/RequiresLowerCase</code>	
	<p>Enables the Core Engine to convert all the header names in lower case for a request when a HTTP request is sent to TIBCO API Exchange Gateway from the Apache server.</p>
<code>tibco.clientVar.ASG/Request/Headers/KeepPassthroughHeaders</code>	
	<p>Enables the core engine to forward the HTTP headers of northbound incoming request to the southbound request the when pass-through mapping is configured.</p> <p>Set this property to <code>true</code> to copy the HTTP headers of northbound request to southbound request when the pass-through mapping is configured for forward mapper of the target operation. Do not configure any values for the Headers To Forward field of target operation when setting this property to <code>true</code>.</p> <div>  <p>According to HTTP/1.1 RFC, HTTP headers are case-insensitive. The native HTTP channel of API Exchange uses Apache Tomcat which enforces this behavior by converting all headers to lower case. We recommend that application developers must not depend on case sensitive headers in their application logic.</p> </div>
<code>tibco.clientVar.ASG/Operation/RequiresQueryDecoding</code>	
	<p>A Boolean property to indicate if the special characters in the query string of HTTP url for the facade request are decoded or not.</p> <ul style="list-style-type: none"> Set the value of this property to <code>false</code> to pass-through the special characters in the query string of HTTP url. The Core Engine does not remove any special characters in the query string of HTTP url. Set the value of this property to <code>true</code> to decode the special characters in the query string of HTTP url.
<code>com.tibco.asg.runtime.http_client.default_uri_format</code>	

Property Name	Description
	<p>A boolean property to indicate if the URI encoding is enabled before the http request is sent to the target operation or not.</p> <ul style="list-style-type: none"> Set the value of this property to <code>true</code> to encode the standard URI before the http request is sent to the target operation. Set the value of this property to <code>false</code> to pass-through the special characters in the standard URI before the http request is sent to the target operation.
<code>tibco.clientVar.ASG/Request/CorrelationHeaderName</code>	
	Specifies the HTTP Header for correlation ID. The default value is <code>X-Request-ID</code> .
ICAP server properties	
<code>tibco.clientVar.ASG/ICAP/Host/URL</code>	
	The IP address of the ICAP host server.
<code>tibco.clientVar.ASG/ICAP/Port</code>	
	The port where the ICAP host server runs. The default value is 1344.
<code>tibco.clientVar.ASG/ICAP/service</code>	
	Specifies the service name of ICAP server. For example, <code>McAfeeService</code> .
<code>tibco.clientVar.ASG/ICAP/useSSL=false</code>	
	Enables SSL communication for ICAP server. By default, the SSL communication is not enabled. Set the value to <code>true</code> to enable the SSL communication with ICAP server.
Properties for SSL communication with ICAP Server	
<code>tibco.clientVar.ASG/ICAP/keystoreFile</code>	
	Specifies the full path of the keystore file if SSL is enabled.
<code>tibco.clientVar.ASG/ICAP/keystoreType</code>	
	Specifies the type of keystore for SSL. For example, <code>JKS</code> . The supported formats are <code>JKS</code> , <code>PKCS12</code> .
<code>tibco.clientVar.ASG/ICAP/keystorePassphrase</code>	
	Specifies the password of the keystore file.
<code>tibco.clientVar.ASG/ICAP/binaryContentTypeList</code>	

Property Name	Description
	<p>Specifies a comma separated list of content type to be scanned by antivirus server. By default, the supported binary content types are <code>application/octet-stream</code> and <code>multipart/*</code>.</p> <p>For example, to send the <code>image/jpeg</code> and <code>application/pdf</code> content types as binary types for scanning, use the values as follows:</p> <pre>tibco.clientVar.ASG/ICAP/binaryContentTypeList=image/jpeg,application/pdf</pre>

Runtime Properties of Central Logger

The properties for the Central Logger are defined in the `asg_cl.properties` file located in `ASG_CONFIG_HOME` directory. The following properties can be defined, if the default values do not serve your purpose:

Central Logger Properties

Property Name	Description
<code>tibco.clientVar.CL/Logging/fileFilter</code>	
	<p>Specifies the lists of facade operation as a pipe (' ') separated string. The transaction logs of these facade operations are logged to the files instead of database.</p> <p>For example:</p> <pre>ping test addConfiguration</pre>
<code>tibco.clientVar.CL/Logging/files/directory</code>	
	<p>Specifies the directory name to store the log file used by the Core Engine to record the transactions of facade operations.</p> <p>The default value is <code>ASG_HOME/bin/logs</code>.</p>
<code>tibco.clientVar.CL/Logging/files/transactions</code>	
	<p>Specifies the name of the log file used by the Core Engine to record the transactions data. This is used only for the transactions of facade operations which are filtered by the <code>tibco.clientVar.CL/Logging/fileFilter</code> property.</p> <p>The default value is <code>trans_log.txt</code>.</p>
<code>tibco.clientVar.CL/Logging/files/maxcount</code>	
	<p>Specifies the maximum number of log files for the Core Engine to keep on roll over for the transactions log file.</p> <p>The default value is 3.</p>
<code>tibco.clientVar.CL/Logging/files/maxsize</code>	

Property Name	Description
	<p>Specifies the maximum size (in bytes) of the log file for writing the transactions data at which the Core Engine rolls over to the next log file.</p> <p>The default value is 5000000.</p>
<code>tibco.clientVar.ASG/Logging/tsFormat=yyyyMMdd HH:mm:ss.SSSS</code>	
	<p>Specifies the format of the log's timestamp value.</p>
<code>tibco.clientVar.Common/Connections/RV/SubjectPrefix</code>	
	<p>Specifies the prefix for all Rendezvous subject names used between the Core Engine and Central Logger.</p> <p>The default value is <code>TIBCO.ASG.INTERNAL</code>.</p>
RV Session Connection Parameters	
<code>tibco.clientVar.ASG/CL/RV/RvService</code>	
	<p>Specifies the service parameter for Rendezvous used between the Core Engine and Central Logger communication.</p> <p>The default value is 7500.</p>
<code>tibco.clientVar.ASG/CL/RV/RvDaemon</code>	
	<p>Specifies the daemon parameter for Rendezvous used between the Core Engine and Central Logger communication.</p> <p>The default value is <code>tcp:7500</code>.</p>
<code>tibco.clientVar.ASG/CL/RV/RvNetwork</code>	
	<p>Specifies the network parameter for Rendezvous used between the Core Engine and Central Logger communication.</p>
Database Connection Parameters	
<code>be.dbconcepts.dburi</code>	
	<code>/CentralLogger/SharedResources/Logging Database.sharedjdbc</code>
<code>be.dbconcepts.connection.retry.count</code>	
	<p>Specifies the number of attempts to be made for the Central Logger to connect to the database server.</p>
<code>be.dbconcepts.connection.check.interval</code>	
	<p>Specifies the time interval (in seconds) for the Central Logger to connect to the database server.</p> <p>The default value is 5.</p>

Property Name	Description
<code>tibco.clientVar.CL/Database/Username</code>	
	Specifies the username for the Central Logger to connect to the database server. The default value is <code>asguser</code> .
<code>tibco.clientVar.CL/Database/Password</code>	
	Specifies the password for the Central Logger to connect to the database server.
<code>tibco.clientVar.CL/Database/Driver</code>	
	Specifies the database driver for the database server used by the Central Logger. For example: For oracle database, the value is defined as: <code>oracle.jdbc.OracleDriver</code> . For MS SQL server, the value is defined as: <code>com.microsoft.sqlserver.jdbc.SQLServerDriver</code> .
<code>tibco.clientVar.CL/Database/Url</code>	
	Specifies the database server connection URL for the Central Logger. For example: For oracle database, the value is defined as: <code>jdbc:oracle:thin:@localhost:1521:asgstat</code> . For MS SQL server, the value is defined as: <code>sqlserver://localhost:1433;databaseName=asgstat</code> .
<code>tibco.clientVar.CL/Database/Schema</code>	
	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: <code>asgstat</code> For MS SQL server, the value is defined as: <code>dbo</code>
Enable JMS Transport for Central Logger Properties	
<code>tibco.clientVar.ASG/Logging/transport</code>	
	Specifies the transport to be used between the Core Engine and Central Logger component to log the transactions. Valid values are RV or JMS.
<code>tibco.clientVar.Common/Connections/JMS/CL_JMSConnection/JMSProviderURL</code>	
	Specifies the URL to connect to the Enterprise Message Service (EMS) or a JMS server. Example: <code>tcp://localhost:7222</code>

Property Name	Description
<code>tibco.clientVar.Common/Connections/JMS/CL_JMSConnection/JNDIContextURL</code>	
	<p>Specifies a JNDI connection URL to look up a JMS server.</p> <p>Example:</p> <p><code>tibjmsnaming://localhost:7222</code></p>
<code>tibco.clientVar.Common/Connections/JMS/CL_JMSConnection/TopicConnectionFactoryName</code>	
	<p>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.</p> <p>The default value is TopicConnectionFactory.</p>
<code>tibco.clientVar.Common/Connections/JMS/CL_JMSConnection/QueueConnectionFactoryName</code>	
	<p>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.</p> <p>The default value is QueueConnectionFactory.</p>
<code>tibco.clientVar.Common/Connections/JMS/CL_JMSConnection/JNDIUsername</code>	
	<p>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.</p> <p>Example, admin.</p>
<code>tibco.clientVar.Common/Connections/JMS/CL_JMSConnection/JNDIPassword</code>	
	<p>Specifies the password for logging into the JNDI server. If the JNDI provider does not require access control, this field can be empty.</p>
<code>tibco.clientVar.Common/Connections/JMS/CL_JMSConnection/JMSUsername</code>	
	<p>Specifies the user name to use to authenticate to the JMS server.</p>
<code>tibco.clientVar.Common/Connections/JMS/CL_JMSConnection/JMSPassword</code>	
	<p>Specifies the password to use to authenticate to the JMS server.</p>
<code>tibco.clientVar.Common/Connections/JMS/CL_JMSConnection/TransactionReportDestinationName</code>	
	<p>Specifies the name of the JMS destination where the transaction reports are sent to the Central Logger by the Core Engine.</p> <p>For example, <code>asg.cl.transaction.queue</code></p>
<code>tibco.clientVar.Common/Connections/JMS/CL_JMSConnection/TransactionReportDestinationType</code>	

Property Name	Description
	<p>Specifies the type of the JMS destination where the transaction reports are sent to the Central Logger by the Core Engine.</p> <p>The possible values are <code>queue</code> or <code>topic</code>.</p> <p>The default value is <code>queue</code>.</p> <p>The Central Logger always listens on a queue. If the value of destination type is set to <code>topic</code>, the JMS administrator must configure a bridge between the topic and the queue.</p>
SSL JMS Transport Properties for Central Logger	
<code>tibco.clientVar.Common/Connections/JMS/CL_JMSConnection/useSSL</code>	
	<p>Specifies if SSL is enabled for the Central Logger when using the JMS transport. Set this to <code>true</code> to enable SSL for the JMS transport.</p>
<code>tibco.clientVar.Common/Connections/JMS/CL_JMSConnection/TrustedCertificatesFolder</code>	
	<p>If you use the SSL enabled JMS transport for Central Logger, this field specifies a location of the trusted certificates on this machine. The trusted certificates are a collection of certificates from servers with which you establish connections. If the server with which the connection is going to be established, presents a certificate that does not match one of your trusted certificates, the connection is refused. This prevents connections to unauthorized servers. Import the trusted certificates into a folder before you select the folder in this field.</p>
<code>tibco.clientVar.Common/Connections/JMS/CL_JMSConnection/Identity</code>	
	<p>Specifies an identity resource used to provide the SSL properties for JMS transport. The default value is: <code>/Common/SharedResources/JMS/CL_JMSConnIdentityResource.id</code></p>
<code>tibco.clientVar.Common/Connections/JMS/CL_JMSConnection/TrustStorePassword</code>	
	<p>Specifies the password to access the certificate stored in the folder defined by the Trusted Certificate Folder field. The certificate is required for SSL connection.</p>
<code>tibco.clientVar.Common/Connections/JMS/CL_JMSConnection/trace</code>	
	<p>Specifies whether SSL tracing should be enabled during the SSL connection. If the value is set to <code>true</code>, the SSL connection messages are logged and sent to the console.</p>
<code>tibco.clientVar.Common/Connections/JMS/CL_JMSConnection/debugTrace</code>	
	<p>Specifies whether SSL debug tracing should be enabled during the SSL connection. Debug tracing provides more detailed messages than standard tracing.</p>
<code>tibco.clientVar.Common/Connections/JMS/CL_JMSConnection/verifyHostName</code>	

Property Name	Description
	Specifies whether the host you are connecting to is the expected host. The host name in the digital certificate of the host is compared against the value in the Expected Host Name field. If the host name does not match the expected host name, the connection is refused.
<code>tibco.clientVar.Common/Connections/JMS/CL_JMSConnection/ExpectedHostName</code>	
	Specifies the name of the host you are expecting to connect to. This field is only relevant if the Verify Host Name field is set to <code>true</code> . If the name of the host in the digital certificate of host does not match the value specified in this field, the connection is refused.
<code>tibco.clientVar.Common/Connections/JMS/CL_JMSConnection/strongCipherSuitesOnly</code>	
	When this field is set to <code>true</code> , this specifies the minimum strength of the cipher suites that can be specified with the bw.plugin.security.strongcipher.minstrength custom engine property. See <i>TIBCO ActiveMatrix BusinessWorks Administration</i> for more information about this property. The default value of the property disables cipher suites with an effective key length below 128 bits. When this field is set to <code>false</code> , only cipher suites with an effective key length of up to 128 bits can be used.
<code>tibco.clientVar.Common/Connections/JMS/CL_JMSConnection/IdentityType</code>	
	Specifies the type of identity resource for SSL connection. The possible values are: <ul style="list-style-type: none"> <code>certPlusKeyURL</code> <code>url</code> <code>usernamePassword</code>
Properties for Identity Type=usernamePassword	
Use this option if you want to use a user name and password for authentication instead of a certificate.	
<code>tibco.clientVar.Common/Connections/JMS/CL_JMSConnection/IdentityUsername</code>	
	Specifies the name of the user for this identity.
<code>tibco.clientVar.Common/Connections/JMS/CL_JMSConnection/IdentityPassword</code>	
	Specifies the password for the user for this identity
Properties for Identity Type=url	
Use this option if the certificate includes the private key information in the same file.	
<code>tibco.clientVar.Common/Connections/JMS/CL_JMSConnection/IdentityFileType</code>	

Property Name	Description
	<p>Specifies the type of certificate file if Identity Type is set to url.</p> <p>The accepted values are as follows:</p> <ul style="list-style-type: none"> • Entrust • JCEKS • JKS • PEM • PKCS12
<code>tibco.clientVar.Common/Connections/JMS/CL_JMSConnection/IdentityURL</code>	
	<p>Specifies the location of the certificate (which includes the private key) if Identity Type is set to url.</p> <p>For example, <code>C:\asgserver.pfx</code></p>
<code>tibco.clientVar.Common/Connections/JMS/CL_JMSConnection/KeyPassword</code>	
	Specifies the password for the certificate if Identity Type is of url type.
Properties for Identity Type=certPlusKeyURL Use this option if the private key and the certificate are in two separate files.	
<code>tibco.clientVar.Common/Connections/JMS/CL_JMSConnection/KeyPassword</code>	
	Specifies the password for the certificate if Identity Type is of certPlusKeyURL type.
<code>tibco.clientVar.Common/Connections/JMS/CL_JMSConnection/CertificateURL</code>	
	Specifies the URL to the certificate file if Identity Type is of certPlusKeyURL type.
<code>tibco.clientVar.Common/Connections/JMS/CL_JMSConnection/KeyURL</code>	
	Specifies the URL to the private key in the certificate file if Identity Type is of certPlusKeyURL type.

Building an EAR File at the Command Line

Procedure

1. Backup the EAR file. For example, if you generate the `asg_core.ear` file in the `ASG_HOME/bin` directory, take a backup of this file.
2. Navigate to `TIBCO_HOME/be/5.2/studio/bin`.
3. Open the `studio-tools.tra` file for editing.
4. Append the following parameters to the `java.extended.properties` property:
`-DTIBCO.BE.function.catalog.http.servlet.getRequestHeaders=true`

For example, the property is modified as follows:

```
java.extended.properties=-server -Xms1024m -Xmx1024m -javaagent:%BE_HOME%/lib/
cep-base.jar -XX:MaxPermSize=256m -Xbootclasspath/p:%TIB_JAVA_HOME%/lib/tools.jar
-DTIBCO.BE.function.catalog.http.servlet.getRequestHeaders=true
```

5. Save changes to the `studio-tools.tra` file.
6. Open a command prompt.
7. Execute a command with the following format (all on one line) at a command prompt:

```
studio-tools -core buildEar [-h] [-x] [-lc] [-o outputEarFile>] -p
studioProjectDir [-pl projectLibrariesFilePath] [-cp extendedClasspath]
```

For example:

```
studio-tools -core buildEar -x
-o C:\tibco\asg\2.3\bin\asg_core.ear
-p C:\tibco\asg\2.3\projects\ASG_DefaultImplementation
```

The following table provides detailed information about the options:

TIBCO BusinessEvents StudioTools Options for Building an EAR File

Option	Description
-core buildEar	Within the core category of operations, specifies the <code>buildEar</code> operation for building EAR files.
-h	Optional. Displays help.
-x	Optional. Overwrites the specified output file if it exists.
-lc	Optional. Specifies that the file-based legacy compiler must be used to build the EAR file. By default, the EAR files are built in memory.
-o	Optional. Specifies the filename for the output EAR file. If not specified the EAR file is the same as the final (leaf) directory name in the <i>projectDir</i> path.
-p	Absolute path to the TIBCO BusinessEvents Studio project directory. The EAR file is built using this project.
-pl	Optional. Specifies list of project library file paths to be used, separated by a path separator.
-cp	Optional. Specifies the extended classpath to be used.



When building an EAR file in memory for a large project, the JVM may run out of PermGenSpace and/or heap space. In such cases, edit the *TIBCO_HOME/be/5.2/studio/eclipse/studio.ini* and *TIBCO_HOME/be/5.2/studio/bin/studio-tools.tra* file to set appropriate values for the JVM settings. By default the heap size is set to -XX:MaxPermSize=256m.

Core Engine Configuration

This section explains the processing units, logging configuration, and runtime properties of TIBCO API Exchange Gateway.

Core Engine

The Core Engine provides the main functionality of TIBCO API Exchange Gateway at run time.

Starting Core Engine

To start the Core Engine, use the following command format. See [Table 11, Core Engine Command-Line Options](#) for the various command-line options.

```
ASG_HOME/bin/asg-engine [-h] [--propFile startup property file] [--propVar
varName=value
] [-p custom property file] [-n engine name] [-d][-a asg config folder] [-c CDD
file] [-u processing unit ID] [EAR file]
```

Core Engine Command-Line Options

Option	Description
-h	Displays this help.
--propFile	<p>Specifies the location of the startup property file (TRA file).</p> <p>When you execute asg-engine, by default, it looks for a property file of the same name (asg-engine.tra) in the directory where you execute the command. This property file provides the startup values and other parameters to the engine executable.</p> <p>To explicitly specify a path and file name of the startup property file, use the --propFile parameter. For example, to use asg_startup.tra file as the startup property file for the Core Engine, specify the --propFile parameter as follows:</p> <pre>asg-engine --propFile c:\tibco\mylocation\asg_startup.tra</pre>
--propVar	<p>Used to provide a value for a specified variable. This value overrides any other design-time value. The format is propVar <i>varName=value</i> . For example to specify the value of the %jmx_port% variable used in TRA files to configure a JMX connection, use this: --propVar jmx_port=XXXX, where XXXX is the port number.</p>
-p	Specifies the path and file name of a custom property file to be passed to asg-engine.
-n	<p>Provides a name for the Core Engine. The name provided here is used in the console and log files.</p> <p>The default value is the processing unit name.</p>
-d	Starts the debugger service on the Core Engine for remote debugging.

Option	Description
<code>-a</code>	Specifies the location of a TIBCO API Exchange Gateway configuration directory.
<code>-c</code>	Specifies the path and file name for the Cluster Deployment Descriptor (CDD) file to be used. The default is default.cdd.
<code>-u</code>	Specifies the processing unit name to be used for this engine. This processing unit name must exist in the CDD file referred to the <code>-c</code> option. The following processing unit names are available. See Processing Units of Core Engine for processing units details. <ul style="list-style-type: none"> • asg-core: Starts the Core Engine. • asg-caching-core: Starts the Core Engine in Cache Agent enabled mode. • asg-gtm: Starts the Global Throttle Manager. • asg-cache: Starts the Cache Agent. • asg-cache-cleanup: Starts the Cache Cleanup Agent. • asg-cl: Starts the Central Logger. The default processing unit is asg-caching-core.
<i>EAR filename and path</i>	Specifies the path and file name for the EAR file to be used. If you do not specify the EAR file name and no <code>tibco.repourl</code> property is set, the EAR file defaults to <code>ASG_HOME/bin/asg_core.ear</code> for all processing units of TIBCO API Exchange Gateway. If you do not specify the EAR file name and <code>tibco.repourl</code> property is set, the Core Engine uses the <code>tibco.repourl</code> property as the EAR file path and name. To use this property, add it to the <code>asg-engine.tra</code> file. If you deploy the Core Engine using TIBCO Administrator, the property is added to the generated TRA file automatically.

Processing Units of Core Engine

In the **Processing Units** tab of the CDD file (See [CDD File and Processing Units](#)), you can define which agents to include in the processing unit and which logging configuration to use.

Processing units are configured in the CDD files as follows:

CDD File and Processing Units

CDD File Name	Processing Unit Names
ASG_HOME/ bin/asg- core.cdd	asg-core, asg-caching-core, asg-gtm, asg-cache, asg-cache-cleanup
ASG_HOME/ bin/ asg_cl.cdd	asg-cl
ASG_HOME/ bin/ asg_validation. cdd	asg-validate

By default, the Core Engine supports the following processing units:

- **asg-core**
The main processing unit of TIBCO API Exchange Gateway is **asg-core**. The **asg-core** provides the critical gateway functions at run time. This requires a separate Cache Agent running to provide the caching functionality at run time and function properly.
- **asg-caching-core**
asg-caching-core is a processing unit of TIBCO API Exchange Gateway that provides the gateway functions at run time including the caching functionality. This agent is a 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 the Global Throttle Manager at run time.
- **asg-cache**
asg-cache is a processing unit of TIBCO API Exchange Gateway that provides the caching functionality to the Core Engine.
- **asg-cache-cleanup**
asg-cache-cleanup is a processing unit of TIBCO API Exchange Gateway that starts the Cache Cleanup Agent 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 the Central Logger component at run time.
- **asg-validate**
asg-validate is a processing unit of TIBCO API Exchange Gateway that validates for the correct configuration to be used at run time.

Configure Log Files Settings

The Core Engine logs the messages in the files. The configuration for the 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. For example, use **ASG_HOME/bin/asg_cl.cdd** for the **asg-cl** processing unit and **ASG_HOME/bin/**

asg_core.cdd for rest of the run time processing units. See [CDD File and Processing Units](#) to find the CDD file for a processing unit.

For a reference to the settings, see [File Properties Settings for a Log Configuration in the CDD Collections Tab](#).

File Properties Settings for a Log Configuration in the CDD Collections Tab

Property	Notes
Enable	Select the Enable check box to enable log files to be written. Configure the settings in this section to specify details. If this check box is cleared, all other properties in this section are ignored.
Directory	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 asg-engine.exe executable).
Name	The 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.
Max number	The number of log files to keep. When the value set in the Max size field is reached, a new log file is created for the next log entries. Files are created up to the value set in the Max number field. The oldest file is deleted when a new file is added after this value is reached. The default is value is 10.
Max size	The maximum size of one log file in bytes. The default is 10000000.
Append	If selected, new entries are added to the end of the file. If not selected, the contents of the file are flushed each time the engine starts.

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 the CDD Collections Tab](#).

If you do not type 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 name of the processing unit 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 select if a log file is flushed when the Core Engine starts, or when entries are appended. See [File Properties Settings for a Log Configuration in the CDD Collections Tab](#).

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

Setting the Log File Configuration Settings in a CDD File

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

Procedure

1. Copy the `asg_core.cdd` File to `ASG_DefaultImplementation` Project

Copy the `asg_core.cdd` file from the Core Engine directory to the `ASG_DefaultImplementation` project directory:

- a) Navigate to the `ASG_HOME/bin` directory.
- b) Copy the `asg_core.cdd` file to `ASG_HOME/projects/ASG_DefaultImplementation` folder.

2. Import the `ASG_DefaultImplementation` Project

- a) Start the TIBCO API Exchange Gateway Studio. See [Starting Studio](#).
- b) Import the `ASG_DefaultImplementation` project. See [Loading the Default ASG_DefaultImplementation Project](#).

3. Edit the `asg_core.cdd` File

Set the log file configuration settings in the `asg_core.cdd` file.

- a) In the Studio Explorer, expand the `ASG_DefaultImplementation` project node. Verify that you see the `asg_core.cdd` file.
- b) Double-click the `asg_core.cdd` file.
- c) Select and expand **Collections** tab.
- d) Select **Log Configurations > logConfig**.
- e) Go to **Configuration > Files** section.
- f) Edit the properties for log files configuration as needed. See [File Properties Settings for a Log Configuration in the CDD Collections Tab](#).
- g) Save the project.

4. Copy the `asg_core.cdd` File to Engine directory

Copy the modified `asg_core.cdd` file from the `ASG_DefaultImplementation` project directory to the Core Engine directory as follows:

- a) Back up the original `asg_core.cdd` file in the `ASG_HOME/bin` directory.
- b) 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, the modified `asg_core.cdd` file exists in the workspace directory. Make sure to copy the `asg_core.cdd` file from the workspace directory to the `ASG_HOME/bin` directory.

Logging Levels of Core Engine

You can choose the level of logging for the Core 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 Core Engine.

The following logging levels are supported:

Logging Levels of Core Engine

Level	Description
4	Specifies no logging for the Core Engine. The highest possible rank. This logging level filters out all the logging messages (turns logging off).
3	Specifies Error level of logging. Logs the run time error messages that might cause the Core Engine to stop running.
2	Specifies WARNING level of logging. Logs the potentially harmful runtime messages (warnings).
1	Specifies INFORMATION level of logging. Logs the runtime informational messages of general interest (Information).
0	Specifies Debug level of logging. The lowest possible rank. This logging level logs the detailed runtime messages. Use this level to identify and debug the issues.



The **Levels** property in the **CDD Collections Tab** should be at least set to 1 (INFORMATION) for the log levels of the Core Engine to work.

How to Set the Logging Level for the Core Engine

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

```
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`. `ASG_CONFIG_HOME` is defined during installation time.

For example, on the Windows platform, the value of `ASG_CONFIG_HOME` is `C:\ProgramData\TIBCOASG\tibco\cfgmgmt`.

Apache Module for TIBCO API Exchange Gateway

The Apache module terminates the HTTP transports.

The module first translates the inbound HTTPS requests to Rendezvous messages, then forwards the Rendezvous messages to the facade component of the gateway operational layer. The Apache module, also translates the response messages from the service providers and forwards each reply to the appropriate requester.

The Apache module of TIBCO API Exchange Gateway deploys and runs within an Apache HTTP server. The module uses the Rendezvous API to communicate with the Core Engine.

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

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



- Using the Apache module is optional. This module is required only when sending requests to TIBCO API Exchange Gateway through the Apache server.
- API key authorization is not supported for Apache module.
- OAuth authorization is not supported for Apache module.

It is good practice to use the Apache HTTP Server in the following scenarios:

- DMZ deployment
- Multiple HTTP endpoints (in case you have internal and external HTTP/HTTPS endpoints)

The requests can be sent directly to TIBCO API Exchange Gateway using the HTTP channel of the gateway. Refer to the *ASG_HOME/examples/BookQueryBE* example to send the requests directly over the HTTP channel.

You can use the HTTP channel of the gateway directly for better performance when there are less components to manage.

Installing Apache HTTP Server

The Apache module requires installation of 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

The Apache module requires installation of Apache HTTP Server with SSL to process the HTTPS requests.

You can install the Apache HTTP server from the Apache website <http://httpd.apache.org>. Refer to the *TIBCO API Exchange Gateway* readme file for Apache server version requirements, and download the supported version. See the "Appendix" in the *TIBCO API Exchange Gateway Installation* guide.

Configuring Apache HTTP Server Using HTTP Transport

This section explains the configuration required to set up the Apache HTTP server using the HTTP transport.

Follow these steps to configure the Apache HTTP server installation for using the Apache module of TIBCO API Exchange Gateway.

Procedure

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` with the directory location where TIBCO API Exchange Gateway product is installed.

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

```
Listen 8080
```



The default listening port is 80. Change the value of the listening port, if required.

The samples shipped with TIBCO API Exchange Gateway use the port value of 8080. Change the value of the listening port to run the samples.

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

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

On UNIX: After TIBCO API Exchange Gateway product is installed, ensure that the following environment variables are set correctly. Depending on the type of shell, you might have to use different commands to set these variables.

- `RV_HOME`: verify that this variable is set to the directory where TIBCO Rendezvous 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:

```
export LD_LIBRARY_PATH=$RV_HOME/lib:$RV_HOME/lib/64:$LD_LIBRARY_PATH
```

On the Windows Platform

Set the variables on the Windows platform.

After TIBCO API Exchange Gateway product is installed, verify that the `PATH` system variable includes `RV_HOME/bin`, where `RV_HOME` specifies the directory where the TIBCO Rendezvous product is installed.

Configuration On the UNIX Platform

Set the variables on the UNIX platform.

After TIBCO API Exchange Gateway product is installed, ensure that the following environment variables are set correctly. Depending on the type of shell, you might have to use different commands to set these variables.

- `RV_HOME`: verify that this variable is set to the directory where TIBCO Rendezvous 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:

```
export LD_LIBRARY_PATH=$RV_HOME/lib:$RV_HOME/lib/64:$LD_LIBRARY_PATH
```

Running the Apache HTTP Server

This section lists the basic commands to run the Apache server.

Refer to the Apache HTTP server documentation for details.

On the Windows Platform

Run the Apache HTTP Server on the Windows platform as follows:

Procedure

1. Navigate to the directory:

```
APACHE_HOME/bin
```
2. Run the following command:

```
httpd.exe
```

Running On the UNIX Platform

Run the Apache HTTP Server on the UNIX platform as follows:

Procedure

1. Navigate to the directory:

```
APACHE_HOME/bin
```
2. Run the following command:

```
./apachectl start
```

Secure Communications

TIBCO API Exchange Gateway supports transport for secure communications between the client requester and the Apache HTTP server.

The transport uses the Secure Sockets Layer (SSL) protocol for exchanging transactions in a secured way. The SSL protocol uses signed digital certificates from a certificate authority (CA) for authentication.

Mutual SSL Authentication

When the client sends a request using HTTPs transport, TIBCO API Exchange Gateway supports the authentication of the client based on the digital certificates. This is known as two-way (mutual) SSL authentication.

Mutual SSL authentication is also referred as client authentication, as with client authentication the client presents its certificate to the server after the server authenticates itself to the client.

TIBCO API Exchange Gateway uses X.509 digital certificates for mutual SSL authentication and to authorize client requests. In this case, authorization of the request is 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.

TIBCO 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 TIBCO 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 Core Engine.

Perform the following high-level steps for mutual SSL authentication.

Generate Keys and Certificates

See [Generate Private Keys And Public Certificates with OpenSSL](#) to generate the private keys and public certificates.

Configure SSL on Apache HTTP Server

You must configure SSL on Apache HTTP server for secure communications.

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

Enabling the 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 the 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 a uniquely correct end point for the communication. 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 the Apache module and the OpenSSL library.

For the configuration steps, see [Configuring Mutual SSL on Apache HTTP Server](#).

Configure Client Authentication with Digital Certificates on Apache HTTP Server

You must configure client authentication on Apache HTTP server for mutual SSL communications.

The Apache HTTP server and clients can communicate over an encrypted connection using the SSL communication. This 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. This 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, the clients must 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. The possible values are as follows:

- `none`: indicates that no client certificate is required at all.
- `optional`: indicates that the 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 TIBCO API Exchange Gateway.

- `optional_no_ca`: indicates that the 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 Certificate Authority (CA) within this depth, it declares the certificate invalid. The depth actually is the maximum number of intermediate certificate issuers, that is, the maximum number of CA certificates that is followed while verifying the client certificate. For example, Depth 0 (zero) means that 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 previously mentioned), or signed by a trusted CA. The default value is 1.

- `SSLCACertificatePath`

The `SSLCACertificatePath` directive specifies the path to a directory containing certificate authority's digital certificate files. Each digital certificate has a separate file. However, when you use this `SSLCACertificatePath` directive, the Apache HTTP server expects that 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. The Apache server expects this for efficiency reasons. When you have a large number of CA certificates, it becomes inefficient 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.



Configuring the `SSLCACertificateFile` directive is easy. As the number of trusted certificate authorities increases, 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, use the `SSLCACertificatePath` directive for best results.

See [Configuring Client Authentication with Digital Certificates on Apache HTTP Server](#) for configuration steps.

Configure Client Certificate Identification Details On Apache HTTP Server

After setting up the client authentication configuration on the Apache HTTP server, configure the identity details of the authenticated client on the Apache HTTP server.

The identity details of the authenticated client can be forwarded as custom HTTP headers to the Core Engine. The Core Engine matches the client identification details from the HTTP headers with the identification details configured on the Config UI.



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 are identified as being sent by the anonymous user.

TIBCO API Exchange Gateway retrieves the client's identity from the two custom HTTP header fields `CAissuer` and `SerialNumber`. The `CAissuer` field contains the distinguished name of the certificate authority that issued the client certificate. The distinguished name provides the unique 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 Config UI. 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, 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, you do not need to update the TIBCO API Exchange Gateway partner configuration in case you use the subject distinguished name.

The 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 the 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 the request processing steps. This includes the issuer distinguished name, the certificate serial number, and the subject distinguished name.

- Enable `mod_headers` module

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

- Set `RequestHeader` directives

Set the `RequestHeader` directives that add specific `CAissuer` and `SerialNumber` HTTP headers to the incoming request. The header values are populated with the values retrieved from the SSL environment variables including the issuer distinguished name, the certificate serial number, or the subject distinguished name.

See [Configure Client Certificate Identification Details On Apache HTTP Server](#) for configuration steps.

Register Partners on Config UI

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

See [Registering Partners On Config UI](#) for the procedure on configuration.

SSL Communications Configuration

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

Before you can set up the configuration for SSL, you should install the Apache server with SSL enabled. See [Mutual SSL Authentication Configuration](#).

Configuring One-Way SSL Authentication

Procedure

1. Ensure that the `mod_ssl` module is available in the Apache HTTP server installation.
2. Enable the `mod_ssl` module as follows:
 - 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 for the specified directives (if not already set), as follows:


```

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
```
 - c) Save the changes made to the `APACHE_HOME/conf/extra/httpd-ssl.conf` file.
4. Import the CA certificate as specified in the `SSLCACertificateFile` directive of the Apache Server configuration.
5. Verify that the SSL configuration is working.
 - a) Open a web browser window.
 - b) Enter the following URL to verify the connection to the Apache server.


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

 For example,


```
http://<machine-name>:8443
```
6. Verify that the connection to the Apache server is successful.

Mutual SSL Authentication Configuration

To enable authentication and X.509-based authorization, you must configure both the Apache HTTP server and the Module for the Apache HTTP Server in TIBCO API Exchange Gateway.

Before you start configuring mutual authentication and authorization, see [Prerequisites for Mutual SSL Setup](#).

Prerequisites for Mutual SSL Setup

- Apache HTTP server with `mod_ssl` module. Refer to the *TIBCO API Exchange Gateway* readme for the Apache server version information. Verify that you have set it up as specified in [Installing Apache HTTP Server](#).
- 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 is 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 subject distinguished name (or optionally the certificate's serial number) of each certificate that clients use to identify themselves to the Apache HTTP server.

Configuring Mutual SSL on Apache HTTP Server

This section explains the steps to enable mutual SSL on Apache HTTP server. Enabling the mutual SSL requires Apache HTTP server with `mod_ssl` module.

To use the `mod_ssl` module with Apache HTTP server, ensure that the following tasks are completed:

Prerequisites

- OpenSSL is installed on the Apache server's host computer.
- An RSA private key in PEM format is available to be used by the Apache HTTP server.
- A digital certificate in PEM format is available 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](#).

To configure mutual SSL on the Apache HTTP server, follow these steps:

Procedure

1. Ensure that the `mod_ssl` module is available and enabled on the Apache HTTP server installation. To enable the `mod_ssl` module, follow these 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.

2. Open the `APACHE_HOME/conf/extra/httpd-ssl.conf` file for editing.
 - a) Set the values for the specified directives (if not already set), as follows:

```
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 following are the example values:

```
SSLCertificateFile "C:\apache2\conf\server.crt"
SSLCertificateKeyFile "C:\apache2\conf\server.key"
SSLCACertificateFile "C:\apache2\certs\myrootca.crt"
```



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

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

```
Listen listening_port_value
```

- The default port for SSL/TLS requests on the Apache HTTP server side is 443. The regular Apache server listens on the port 80 so there is no conflict between a regular Apache listening on port 80 and an SSL/TLS enabled Apache listening on port 443. Both HTTP and SSL/TLS enabled can run with the same Apache server instance, usually by defining separate virtual hosts listening on port 80 and port 443 to separate the virtual servers.

- You can access the machine using the `http://<machine-name>:443/./` when the default port as 443 is used. If the port is changed to 8443, the access link is: `http://<machine-name>:8443/./`.



- Ensure that the firewall is open to *listening_port_value* specified in the Listen directive.

- c) Ensure that the global SSL configuration directives are defined as follows:

```
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 follows and 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"
```



- Ensure that you have copied the private key and the server's digital certificate in the directories specified in the SSLCertificateFile and SSLCertificateKeyFile directives.

3. Save the changes made to the `APACHE_HOME/conf/extra/httpd-ssl.conf` file.
4. Import the CA certificate as specified in the SSLCACertificateFile directive of the Apache Server configuration.
5. Verify that the SSL configuration is working.
 - a) Open a web browser window.
 - b) Enter the following URL to verify connection to Apache HTTP server.

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

For example,

```
http://<machine-name>:8443
```

- c) Verify that the connection to the Apache HTTP server is successful.



When you invoke a secure connection to the Apache server for the first time using HTTP or HTTPS transport, the browser displays a message indicating that the server presented is an untrusted certificate. 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

Configure the client authentication with digital certificates on the Apache HTTP server.

Procedure

1. Open the `APACHE_HOME/conf/extra/httpd-ssl.conf` file in a text editor.
2. Ensure that the following SSL directives are defined:

SSL Directives

Parameter	Value
SSLCACertificatePath	<p>Location of the directory containing the separate files for each certificate authority's digital certificate.</p> <p>For example, <code>/etc/apache2/ssl.crt</code></p> <p>Set either of the SSLCACertificatePath or SSLCACertificateFile directive, not both.</p>
SSLCACertificateFile	<p>Name and location of a single certificate file that contains all CA certificates.</p> <p>For example, <code>/etc/apache2/ssl.crt/cacert-bundle.pem</code></p> <p>Set one of the SSLCACertificatePath or SSLCACertificateFile directive, not both.</p>
SSLVerifyClient	require
SSLVerifyDepth	1

3. Save the changes and close the file.
4. Restart the Apache HTTP server, if already running.
5. Test the configuration changes by importing a client certificate into the web browser. To do this, 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.
6. Use one of the following browsers:
 - Firefox
 - Internet Explorer

Using Firefox

Import a client digital certificate using Firefox.

Procedure

1. Open the web browser.
2. Navigate to **Tools > Options** on the browser menu.
3. Select **Advanced** tab in the new window.
4. Select **Security** tab in the new dialog.
5. Click **View Certificates**.
6. Click **Import** and follow the wizard to import the file.

Using Internet Explorer

Import a client digital certificate using Internet Explorer.

Procedure

1. Open the web browser.
2. Navigate to **Tools > Internet Options** on the browser menu.
3. Select **Content** tab in the new dialog window.
4. Go to **Certificates** section and click **Certificates** tab.
5. Click **Import** and follow the wizard to import the file.

Testing the Imported Certificate

You must test the imported certificate for client authentication.

Procedure

1. After you have imported the PKCS12 file, open a browser window.
2. Enter the following URL to verify secure connection to the Apache server.

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

For example,

```
http://<machine-name>:8443
```

3. Verify that the connection to the Apache server is successful.

Forwarding Client Certificate Identification Details on Apache HTTP Server to Core Engine

You must configure Apache HTTP server to forward client certificate identification details to the Core Engine.

To configure the setup so that the Apache HTTP server forwards the client identification details to the Core Engine,

Procedure

1. Open the `ASG_HOME/modules/http_server/apache/mod_ASG.conf` file in a text editor.
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 follows:

```
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 Core Engine, if not already running. See [Starting Core Engine](#).
7. Test the configuration changes to see that only requests from clients that authenticate themselves with a client certificate are forwarded to the Core Engine. As no partners are configured yet on the Config UI with the credentials specified in the certificate, the incoming request fails the identification with this configuration.

8. To test this configuration setup, enter the following URL to submit a ping operation request:

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

If you have configured everything on the Apache HTTP server but have not registered the partner in the TIBCO API Exchange Gateway yet, you should receive the response from TIBCO API Exchange Gateway on the web browser as follows:

```
<asg:Error> <asg:ErrorCode> 2001 </asg:ErrorCode>
<asg:ErrorMessage> Partner null not identified </asg:ErrorMessage>
</asg:Error>
```

Registering Partners Using the Config UI

Register the partner using the Config UI.

Register the partner with the identity information as follows:

Procedure

1. Start the Config UI. See [Starting GUI](#) for details.
2. Select the gateway project configuration.
3. Click the **PARTNER** tab.
4. Add a new partner. See [Using Partners tab](#) for details.
5. Set the following fields for the new partner:

Parameter	Description
Partner Serial Number	<p>Specifies the client's identity that the Apache HTTP server forwards 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.</p> <p>For example, the value can be defined as follows:</p> <p>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</p>
Partner Issuer CA	<p>Specifies the realm in which the client's identity is valid. This is always the issuer distinguished name as used from the digital certificate for that partner.</p> <p>For example, the value can be defined as follows:</p> <p>Partner Issuer CA: /C=US/ST=California/O=TIBCO Software Inc./OU=ActiveMatrix Service Gateway/CN=TIBCO ASG Certificate Authority/emailAddress=admin@tibasg.co.pd</p>



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.

6. Click the **Partner Operations** tab.
7. Define a ping operation (internal_ping) for a new partner. See [Facade Access](#) for details.
8. Save the configuration.
9. Start the Core Engine. See [Starting Core Engine](#)

10. Test the configuration.

- a) Open a web browser window.
- b) Enter the following URL to submit a ping operation request:

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

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

Configure the Apache Server for Basic HTTP Authentication

To support basic HTTP authentication for a request, you must configure the Apache HTTP server.

Basic HTTP authentication requires the client to provide a user name and password when it sends the request to the 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 basic authentication by the client, it is good practice that you use one-way SSL for secure communication between the Apache HTTP server and the requester. Configure the following for secured communication:

- Configure the Apache HTTP server for one-way SSL. See [Configuring One-Way SSL Authentication](#). If you use the HTTP transport, do not set up the SSL configuration on Apache HTTP server.
- Configure the Apache HTTP server for Basic Authentication. See [Enabling Basic Authentication on Apache HTTP Server](#) for the configuration.
- Configure the Core Engine for Basic Authentication. See [Configuring TIBCO API Exchange Gateway for Basic Authentication](#).

Configuring Apache HTTP Server for Basic Authentication

This section explains the steps required to configure the Apache HTTP server for basic authentication.

Enabling Basic Authentication on Apache HTTP Server

Enable the basic authentication on Apache HTTP server.

Procedure

1. Open a terminal window.
2. Navigate to `ASG_HOME/modules/http_server/apache` directory.
3. Edit the `mod_ASG.conf` file.
4. Search for the following section in the file:

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

5. Insert the configuration for a new location above the old location configuration as follows:

```
<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 follows:

```
<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 user access with basic authentication.

7. Save the `mod_ASG.conf` file.

Creating a Password File for the Apache HTTP Server

The Apache HTTP server requires a password file.

Procedure

1. Open a terminal window.
2. Navigate to `APACHE_HOME`.
3. Create a **htpasswd** subdirectory 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 follows:

For example, on the 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 follows:

For example, on the Windows platform:

```
htpasswd APACHE_HOME\htpasswd\htpasswords asgpartner02
New Password:asgpartner02
Enter New Password:asgpartner02
```

Reloading the Configuration File for the Apache HTTP Server

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

On the Windows Platform

Procedure

1. Open a terminal window.
2. Navigate to the `APACHE_HOME/bin` directory.

3. Stop the Apache server, if already running.
4. Run the following command to start the Apache server: `httpd.exe`

On the UNIX Platform

Procedure

1. Open a terminal window.
2. Navigate to the `APACHE_HOME/bin` directory.
3. Run the following command to restart the Apache server:

```
./apachectl restart
```

Configuring a Client (Requester) for Basic Authentication (Example Use Case)

Configuration details to use TIBCO Business works as a client 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. Customize the changes accordingly if you are using a different client to send requests to TIBCO API Exchange Gateway.



Ensure that the Uniform resource identifier (URI) used by the client to send the request (for example, TIBCO BusinessWorks as a client) matches the URI used by the receiving gateway (for example, TIBCO API Exchange Gateway facade). This example uses `"/asg/ba/"` as the URI.

Configuring the Endpoint URL for Transport

In this example, the client side HTTP URL contains `"/asg/ba/"` string in the endpoint URL to access the server running the Core Engine.

For example, refer to **BookQuery** project shipped with TIBCO API Exchange Gateway at the `ASG_HOME/examples/BookQuery/BookQuery` location as follows:

Procedure

1. Open the `ASG_HOME/examples/BookQuery/BookQuery` project using TIBCO Designer.
2. Navigate to **BookQuery > Client** process.
3. Double-click **QueryByTitleClient** to open the process.
4. Click **SOAPRequestReply** activity to open it.
5. Click **Transport Details** tab. To use the basic authentication, change the endpoint URL as follows:
 - From:


```
http://127.0.0.1:9696/ServerProcesses/GetBooksByTitleEndpoint
```
 - To:


```
http://127.0.0.1:9696/asg/ba/ServerProcesses/GetBooksByTitleEndpoint
```
6. Save the changes to the configuration.

Creating an Identity Resource

You must create an identity as set on the Apache HTTP server configuration. See [Creating a Password File for the Apache HTTP Server](#).

For example, you can create an identity as follows for the **BookQuery** project:

Procedure

1. Select **Client Process** node.
2. Navigate to **Resources > Add Resources > General > Identity**.
3. Input the values for the following fields:
 - Name: a string value (For example, MyIdentity)
 - Type: select Username/Password from the drop-down list.
 - Username: *username* (*username* must match the username created at the Apache server. For example, asgpartner01. See [Creating a Password File for the Apache HTTP Server](#)).
 - Password: *password* (*password* must match the password created at the Apache server. For example, asgpartner01. See [Creating a Password File for the Apache HTTP Server](#)).
4. Save the changes to the configuration.

Configuring Identity For Transport

Configure the identity (username and password) for the HTTP transport.

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

Procedure

1. Double-click **QueryByTitleClient** to open the process.
2. Click **SOAPRequestReply** activity to open it.
3. Click **Transport Details** tab.
 - a) Select the **Use Basic Authentication** check box.
 - b) Set the **Identity** field as follows:
 - Click **Browse** next to the **Identity** field.
 - Select the Identity resource created as explained [Creating an Identity Resource](#).
 - Click **OK** to select the identity resource.
 - c) Click **Apply** to save the changes.
4. Save changes to the configuration.

Configuring TIBCO API Exchange Gateway for Basic Authentication

Procedure

1. Start Config UI.
See [Starting GUI](#) to launch the Config UI.
2. Add a New Facade Operation.

- a) Create a new configuration or select an existing configuration, as applicable.
 - b) Click the **ROUTING > Facade Operations** tab.
 - c) Add a new operation or select an existing operation. See [Facade Operations](#).
 - d) The Operation URI field contains "/asg/ba/" string in the endpoint as follows for the facade operation:
 - e) For example, /asg/ba/ServerProcesses/GetBooksByTitleEndpoint
 - f) Note that the client uses the "/asg/ba" in the transport URL to send the request to the Core Engine.
 - g) Save the changes to the configuration.
3. Add a New Partner Group.
 - a) Create a new configuration or select an existing configuration, as applicable.
 - b) Add a new partner group. See [Partner Groups](#).
 - c) Save the changes to the configuration.
 4. Add a New Partner.
 - a) Create a new configuration or select an existing configuration, as applicable.
 - b) Add a new partner as setup for the client. See [Using Partners tab](#).
 - See [Creating an Identity Resource](#) for the username. The partner name must match the username of the identity resource.

For example, set up the partner name as follows:

 - Partner Name: asgpartner01
 - c) Save changes to the configuration.
 5. Add a New Partner Group.

Create or add a new partner group for the partner. See [Partner Groups](#).
 6. 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.

 - a) Create a new configuration or select an existing configuration, as applicable.
 - b) Add a new facade access to authorize the partner to access the operation. See [Facade Access](#).
 - c) Save changes to the configuration.
 7. Add Routing.

You must add the routing data for the partner and operation created in [Add a New Partner](#) and [Add a New Facade Operation](#) sections so that the Core Engine can route the facade request to the appropriate target operation. See [Routing](#).

Configure Apache Module for RVRD Setup through a Firewall (DMZ)

Deployment overview of Apache module in the DMZ setup and other gateway components in a secure network.

By default, the Core Engine uses the TIBCO Rendezvous daemon (rvd) 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 the system, you can protect the system against the threat of malicious communications and provide stronger security.

When the services are exposed to an unsecured 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 de-militarized 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 explains 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](#).

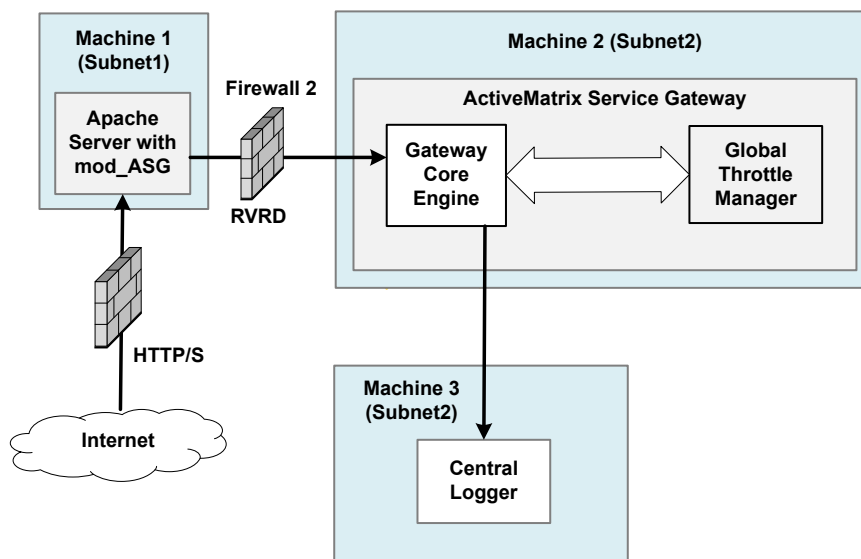
Rendezvous transport can be configured so that all connections between the Core Engine (asg_core) and the Apache HTTP server are instantiated from the internal secure zone into the DMZ (that is, the server running asg_core).

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



If you want to deploy TIBCO API Exchange Gateway in the DMZ setup between the firewalls, configure RVRD when the machines are in different subnets of the network.

Apache HTTP Server in DMZ and Other Components in Secure Network



Setting up TIBCO API Exchange Gateway in a DMZ Environment

Procedure

1. Install TIBCO Rendezvous on 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 the *TIBCO_HOME* directory for the supported version of the Apache HTTP server.
3. Install TIBCO API Exchange Gateway software on Machine 2.
4. Configure RVRD between Machine 1 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 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 follows:
 - a) Navigate to the TIBCO API Exchange Gateway installation directory on Machine 2.
 - b) Browse to the *ASG_HOME/modules/http_server/apache* directory.

- c) Copy the `mod_ASG.conf` file from the Machine 2 and place 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](#).
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 Core Engine Communication](#).
8. If you want to change the default values for the Rendezvous session connection parameters for the 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 Core Engine to Central Logger](#). 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](#).
9. To change the default values for the Rendezvous session connection parameters for the Core Engine and Global Throttle Manager, set or edit the parameters in the `ASG_CONFIG_HOME/asg.properties` file as described in the table, [Rendezvous Parameters for Core Engine And Global Throttle Manager Communication](#).
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 on the Config UI. See [Runtime Properties](#) for details.

Configure Apache HTTP Server as Reverse Proxy

Using Apache HTTP Server as Reverse Proxy.

TIBCO API Exchange Gateway supports the Apache HTTP server in reverse proxy mode. You can place the Apache HTTP server in front of the inner firewall accepting the requests from the clients, and forward those requests to TIBCO API Exchange Gateway residing behind the firewall.

The Apache HTTP server can be used as reverse proxy for the following purposes:

- Provide users access to a server that is behind a firewall.
- Balance the load among multiple instances of the TIBCO API Exchange Gateway.
- Bring multiple instances of TIBCO API Exchange Gateway servers into the same URL space.

The Apache HTTP server in reverse proxy mode can be used for both SSL and non-SSL communication.

Directives

List of directives for Apache HTTP server to be used as reverse proxy.

The following directives must be enabled to use Apache HTTP server as the reverse proxy:

- ProxyPass
- ProxyPassReverse
- SSLCertificateFile (for SSL communication only)
- SSLCertificateKeyFile (for SSL communication only)

Refer to Apache HTTP server documentation for the description of directives.

Setting up Reverse Proxy Server for Non-SSL Communication

Procedure

1. Navigate to the following directory of Apache HTTP server installation:

```
APACHE_HOME/conf
```

2. Open the `httpd.conf` file for editing.
3. Uncomment the following lines:

```
LoadModule proxy_module modules/mod_proxy.so
LoadModule proxy_ajp_module modules/mod_proxy_ajp.so
LoadModule proxy_http_module modules/mod_proxy_http.so
```

4. Add the following contents:

```
<VirtualHost *:ListenPort>
ProxyPreserveHost On
# Servers to proxy the connection, or;
# List of application servers:
# Usage:
# ProxyPass / http://[IP Addr.]:[port]/
# ProxyPassReverse / http://[IP Addr.]:[port]/
# Example:
ProxyPass / http://APIExchangeGateway_Host:HTTP_PORT/
ProxyPassReverse / http://APIExchangeGateway_Host:HTTP_PORT/
ServerName localhost
</VirtualHost>
```

where,

- *ListenPort* is the port on which Apache HTTP server runs. The *ListenPort* must match the value specified by Listen directive.
- *APIExchangeGateway_Host* is the machine on which TIBCO API Exchange Gateway runs.
- *HTTP_PORT* is the HTTP port for TIBCO API Exchange Gateway.



To forward a specific operation request, include the request URI in the ProxyPass and ProxyPassReverse directives. For example, define the directives for ping operation URI, as follows:

```
ProxyPass /ping http://APIExchangeGateway_Host:HTTP_PORT/ping
ProxyPassReverse /ping http://APIExchangeGateway_Host:HTTP_PORT/ping
```

5. Save the changes to the file.

Setting up Reverse Proxy Server for SSL Communication

Procedure

1. Navigate to the following directory of Apache HTTP server installation:

```
APACHE_HOME/conf
```

2. Open the `httpd.conf` file for editing.
3. Uncomment the following line:

```
LoadModule ssl_module modules/mod_ssl.so
```

4. Add the following line to include the SSL configuration file:

```
Include conf/extra/httpd-ssl.conf
```

5. Save changes to the `httpd.conf` file.
6. Navigate to the following directory of Apache HTTP server installation:

`APACHE_HOME/conf/extra`

7. Open the `httpd-ssl.conf` file for editing.
8. Add the following contents:

```
<VirtualHost *:ListenSSLPort>
SSLEngine On
ProxyPreserveHost On
# Set the path to SSL certificate
# Usage: SSLCertificateFile /path/to/example.crt
# Usage: SSLCertificateKeyFile /path/to/example.key
# Example:
    SSLCertificateFile Name_of_Server_public_certificate
    SSLCertificateKeyFile Name_of_Server_private_key
# Servers to proxy the connection, or;
# List of application servers:
# Usage:
# ProxyPass / http://[IP Addr.]:[port]/
# ProxyPassReverse / http://[IP Addr.]:[port]/
# Example:
    ProxyPass / http://APIExchangeGateway_Host:HTTP_PORT/
    ProxyPassReverse / http://APIExchangeGateway_Host:HTTP_PORT/
</VirtualHost>
```

where,

- *ListenSSLPort* is the SSL transport port on which Apache HTTP server runs. The *ListenSSLPort* must match the value specified by Listen directive.
 - *APIExchangeGateway_Host* is the machine on which TIBCO API Exchange Gateway runs.
 - *HTTP_PORT* is the HTTP port for TIBCO API Exchange Gateway.
 - *Name_of_Server_public_certificate* is the full path to the public certificate. For example, "c:\apache2\conf\server.crt".
 - *Name_of_Server_private_key* is the full path to the private key. For example, "C:\apache2\conf\server.key".
9. Save the changes to the file.
 10. Configure the client certificate authentication as described in the [Configuring Client Authentication with Digital Certificates on Apache HTTP Server](#) section.



If you want to use the Apache HTTP server in reverse proxy mode to forward the client requests to multiple instances of the Core Engines, use a load balancer. Refer to the [High Availability Deployment Of Runtime Components](#) chapter.

Transport Communication

Overview of various transports for inter-component communications.

This section explains the configuration settings required for various transports supported by TIBCO API Exchange Gateway at the facade and target side.

Facade Operation Requests

Overview of transports for the facade operation.

For the facade operation requests, TIBCO API Exchange Gateway supports the following transports:

HTTP

The Core Engine has the native HTTP Channel, which can receive requests out of the box. The native HTTP Channel can be used in conjunction with Apache HTTP server as described in the [Configure Apache HTTP Server as Reverse Proxy](#) section. The native HTTP Channel is the most straight forward configuration option for handling the HTTP requests.

Rendezvous (RV)

The HTTP requests can be sent to the Apache HTTP server, which communicates with the Core Engine using the RV module. The RV communication between the Apache RV module and the Core Engine can be configured to initiate the connection from Core Engine to the RV module to provide securer network topology. The RV transport used internally between RV module and the Core Engine is not exposed to the external HTTP client. Usage of RV module requires each instance of the Core Engine to be pairs with a dedicated Apache server.

JMS

The client can send either SOAP or non-SOAP requests over the JMS transport. Use the JMS transport to send the requests to the Core Engine when the reliable messaging is important.

Central Logger

Overview of the transports for Central Logger.

The Central Logger component of the TIBCO API Exchange Gateway supports the following transports:

Rendezvous (RV)

When the RV transport is used for the Central Logger, the Core Engine sends the log reports to the Central Logger using RV channel. The log reports are lost if the Central Logger is not running.

JMS

When the JMS transport is used for the Central Logger, the Core Engine sends the log reports to the Central Logger using JMS channel. The log reports are retained if the Central Logger is not running.

AS

When the AS transport is used for the Central Logger, the Core Engine sends the log reports to the Central Logger using ActiveSpaces channel.

Global Throttle Manager

Overview of the transports for Global Throttle Manager.

The Global Throttle Manager component of the TIBCO API Exchange Gateway supports the following transports:

Rendezvous (RV)

When the RV transport is used for the Global Throttle Manager, the Core Engine communicates with the Global Throttle Manager using RV channel.

AS

When the AS transport is used for the Global Throttle Manager, the Core Engine communicates with the Global Throttle Manager using AS Channel. When AS transport is configured for Global Throttle Manager, the Central Logger must use the AS transport.

The following table summarizes the supported transports for the components of TIBCO API Exchange Gateway:

Transports For Intercomponent Communication

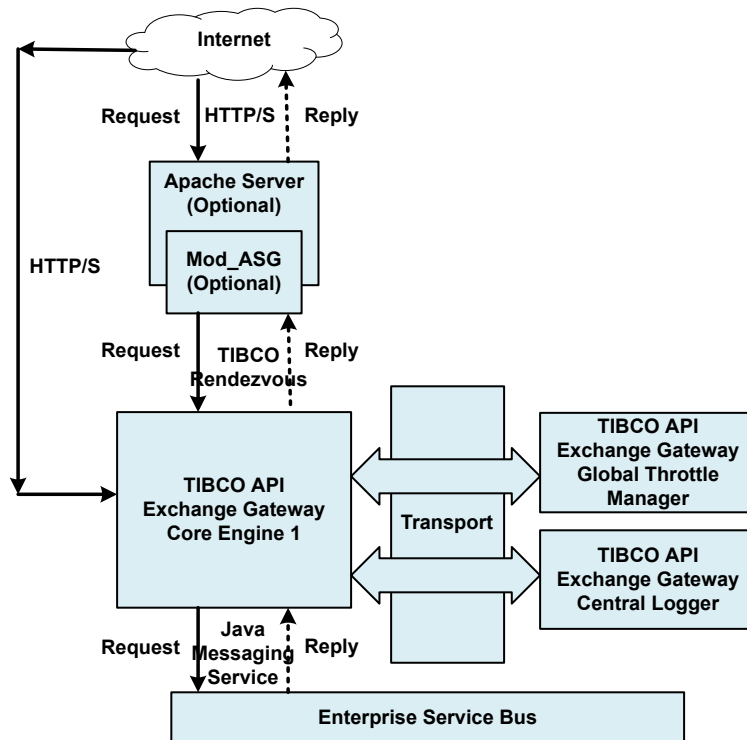
Transport	Engine	CL	GTM
HTTP	X		
JMS	X	X	
RV	X	X	X
AS		X	X

Rendezvous Transport Communication

Rendezvous is the internal transport used for communication between the various components of the Core Engine.

The following figure shows the communication for the various components using Rendezvous transport.

Rendezvous Transport Communication



Enabling Rendezvous Communication for TIBCO API Exchange Gateway

By default, Rendezvous channel is disabled. You must enable the RV communication if you want to use RV transport for TIBCO API Exchange Gateway.

Pre-requisites:

Install TIBCO Rendezvous.

Backup Files

To enable the RV transport communication, edit the CDD, TRA, and properties files. It is good practice to back up these files before making any changes to them.

Procedure

- Back up the following files in the `ASG_HOME/bin` directory and `ASG_CONFIG_HOME` directory:

In `ASG_HOME/bin`,

- `asg_core.cdd`
- `asg_cl.cdd`
- `asg_engine.tra`

In `ASG_CONFIG_HOME`,

- `asg.properties`
- `asg_cl.properties`

For example, to back up the `asg.properties` file on the Windows platform, type the following command:

```
copy asg.properties asg.properties.backup
```

Editing asg_core.cdd File

You must edit the `asg_core.cdd` file to enable RV channel and disable the AS transport.

Procedure

1. Navigate to the `ASG_HOME/bin` directory.
2. Open the `asg_core.cdd` file in an editor.
3. Search the `be.channel.deactivate` property:

```
<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,/Common/Channel/
CentralLoggerJMS,/Common/Channel/Channel,/DefaultImplementation/Channels/
OAuthWebappsHTTPPSChannel,/DefaultImplementation/Channels/
FacadeHTTPSSLChannel,/ASG/Channels/modRV_Channel,/ASG/Channels/
RvCacheableChannel,/ASG/Channels/RvMappingChannel,/Common/Channel/
CentralLoggerRV"/>
```

4. If you have not made a backup of the `be.channel.deactivate` property, copy the `be.channel.deactivate` property to the `be.channel.deactivate.backup` property, as follows:

```
<property name="be.channel.deactivate.backup" 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,/Common/Channel/CentralLoggerJMS,/Common/Channel/Channel,/
DefaultImplementation/Channels/OAuthWebappsHTTPPSChannel,/DefaultImplementation/
Channels/FacadeHTTPSSLChannel,/ASG/Channels/modRV_Channel,/ASG/Channels/
RvCacheableChannel,/ASG/Channels/RvMappingChannel,/Common/Channel/
CentralLoggerRV"/>
```

5. Remove the following channels from the value of the `be.channel.deactivate` property:

- `/Common/Channel/Channel`
- `/ASG/Channels/modRV_Channel`
- `/ASG/Channels/RvCacheableChannel`
- `/ASG/Channels/RvMappingChannel`
- `/Common/Channel/CentralLoggerRV`

6. Add the following channel to the value of `be.channel.deactivate` property:

```
/Common/Channel/AS
```

7. Save changes to the `asg_core.cdd` file.

Editing asg_cl.cdd File

You must edit the `asg_cl.cdd` file to enable RV channel and disable the AS transport.

Procedure

1. Navigate to the `ASG_HOME/bin` directory.
2. Open the `asg_cl.cdd` file in an editor.
3. Search the `be.channel.deactivate` property:

```
<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,/ASG/Channels/SOAPJMSChannel_North,/ASG/Channels/
SOAPJMSChannel_South,/Common/Channel/CentralLoggerJMS,/DefaultImplementation/
Channels/OAuthWebappsChannel,/DefaultImplementation/Channels/
OAuthWebappsHTTPChannel,/ASG/Channels/RvMappingChannel,/ASG/Channels/
RvCacheableChannel,/ASG/Channels/modRV_Channel,/Common/Channel/Channel,/Common/
Channel/CentralLoggerRV"/>
```

4. If you have not made a backup of the `be.channel.deactivate` property, copy the `be.channel.deactivate` property to the `be.channel.deactivate.backup` property, as follows:

```
<property name="be.channel.deactivate.backup" value="/DefaultImplementation/
Channels/SouthboundEsb0Channel,/DefaultImplementation/Channels/
SouthboundEsb1Channel,/DefaultImplementation/Channels/SouthboundEsb2Channel,/
DefaultImplementation/Channels/North_ESBChannel,/DefaultImplementation/Channels/
North_HTTPChannel,/ASG/Channels/SOAPJMSChannel,/ASG/Channels/
SOAPJMSChannel_North,/ASG/Channels/SOAPJMSChannel_South,/Common/Channel/
CentralLoggerJMS,/DefaultImplementation/Channels/OAuthWebappsChannel,/
DefaultImplementation/Channels/OAuthWebappsHTTPChannel,/ASG/Channels/
RvMappingChannel,/ASG/Channels/RvCacheableChannel,/ASG/Channels/modRV_Channel,/
Common/Channel/Channel,/Common/Channel/CentralLoggerRV"/>
```

5. Remove the following channel from the value of the `be.channel.deactivate` property:

```
/Common/Channel/CentralLoggerRV
```

6. Add the following channel to the value of the `be.channel.deactivate` property:

```
/Common/Channel/AS
```

7. Save changes to the `asg_cl.cdd` file.

Setting `tibco.env.RV_HOME` Property in TRA Files

You must set the `tibco.env.RV_HOME` property in the following TRA file:

- `asg-engine.tra`

Procedure

1. Navigate to the `ASG_HOME/bin` directory.
2. Using a text editor, open the `asg-engine.tra` file for editing.
3. Set the `tibco.env.RV_HOME` property to the location of TIBCO Rendezvous installation home.

For example, on the Windows platform, the property is set to `d:/tibco/tibrv/8.4`.

4. Set the `tibco.env.CUSTOM_EXT_PREPEND_CP` property to append the `%PSP%RV_HOME%/lib/tibrvj.jar` file.

For example, on the Windows platform, the property is set as follows:

```
tibco.env.CUSTOM_EXT_PREPEND_CP=%ASG_HOME%/lib%PSP%ASG_HOME%/lib/ext/hotfix%PSP%
ASG_HOME%/lib/ext/tibco%PSP%ASG_HOME%/lib/ext/tpcl%PSP%RV_HOME%/lib/tibrvj.jar
```

5. Save changes to the `asg-engine.tra` file.

Editing Properties Files

To enable Rendezvous communication for the gateway, set the transport to RV in the properties files for the Core Engine and Central Logger.

Procedure

1. Navigate to the `ASG_CONFIG_HOME` directory.

2. Using an editor, open the `asg.properties` file for editing.
3. Search the following property:

```
tibco.clientVar.ASG/transport=AS
```

4. Edit the value of following property from AS to RV:

```
tibco.clientVar.ASG/transport=RV
```

5. Save changes to the `asg.properties` file.
6. Using an editor, open the `asg_cl.properties` file for editing.
7. Search the following property:

```
tibco.clientVar.ASG/Logging/transport
```

8. Edit the value of following property from AS to RV (remove comment if it exists):

```
tibco.clientVar.ASG/Logging/transport=RV
```

9. Save changes to the `asg_cl.properties` file.

Configuration Setup

This section explains the configuration setup required for Rendezvous communication.

To ensure that Rendezvous is only used as an internal transport with the Core Engine, perform 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 follows:

- RvDaemon tcp:7500
- RvNetwork
- RvService 7500

If you want to change the 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` files. See [Setting Rendezvous Transport Properties](#).



If the Gateway components are deployed on the machines within the same subnet, the `rvd` daemon is used. In case the components are deployed on the machines within different subnets, it requires the `rverd` daemon setup. See *TIBCO Rendezvous Administration Guide* for the configuration setup and starting the Rendezvous daemons.

Setting Rendezvous Transport Properties

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

Rendezvous Session Connection Parameters for Apache Module

List of Rendezvous session connection parameters for Apache module.

The parameters used for Apache module to connect to the Rendezvous daemon 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 actions:

Procedure

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:

Rendezvous Session Connection Parameters for Apache Module

Parameter	Description
<code>AsgService</code>	Specifies the Rendezvous daemon service parameter. The default value is 7500.
<code>AsgNetwork</code>	Specifies the Rendezvous daemon network parameter. The default value is 10.
<code>AsgDaemon</code>	Specifies the Rendezvous daemon parameter. This parameter is set as the TCP port. The default value is: <code>tcp:7500</code> .
<code>AsgSubject</code>	<ul style="list-style-type: none"> • Specifies the subject to which the Rendezvous daemon sends the client request. The default value is: <code>_LOCAL.asg.north.request</code>. You can change the default value, as required. • This value should match the value set in the <code>tibco.clientVar.ASG/modRV/north_request</code> parameter of the <code>ASG_CONFIG_HOME/asg.properties</code> file.



For the communication between the Apache module and the Core Engine, the Rendezvous session connection parameters in the `ASG_CONFIG_HOME/asg.properties` file and `ASG_HOME/modules/http_server/apache/mod_ASG.conf` file must match.

4. Save the changes to the file.

Rendezvous Session Connection Parameters for Core Engine

List of Rendezvous session connection parameters for the Core Engine.

This section lists the properties required for the 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.

To set or edit the properties, follow these steps:

Procedure

1. Navigate to the `ASG_CONFIG_HOME` directory.
2. Open `asg.properties` file in a text editor.
3. Set the following properties:

Rendezvous Session Parameters for Apache Module and Core Engine Communication

Parameter	Description
<code>tibco.clientVar.ASG/modRV/facade_request</code>	<p>Specifies the subject name used by the Rendezvous daemon to listen to the requests from Apache module. The Core Engine listens to the requests on the same subject.</p> <ul style="list-style-type: none"> • This property value must match the subject value specified in the Apache module configuration file (<code>mod_ASG.conf</code>) for Apache server. The default value is: <code>_LOCAL.asg.north.request</code>
<code>tibco.clientVar.ASG/modRV/RvDaemon</code>	<p>Specifies the value of Rendezvous daemon for the Core Engine to connect and listen for the requests from the Apache module. The default value is: <code>tcp:7500</code></p>
<code>tibco.clientVar.ASG/modRV/RvNetwork</code>	<p>Specifies the value of the Rendezvous network for the Core Engine to connect and listen for the requests from the Apache module.</p> <p>This property value must match the network value specified in the Apache module configuration file for Apache server.</p>
<code>tibco.clientVar.ASG/modRV/RvService</code>	<p>Specifies the value of the Rendezvous service for the Core Engine to connect and listen for the requests from the Apache module. The default value is: <code>7500</code></p>

4. Save the changes to the file.

Rendezvous Session Connection Parameters For Core Engine and Central Logger Communication

List of Rendezvous session connection parameters for Core Engine and Central Logger Communication.

This section lists the properties required for Rendezvous communication between the 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 Core Engine to Central Logger

The Core Engine uses the Rendezvous daemon to send messages to the Central Logger. Set the properties for the Core Engine to communicate with the Rendezvous daemon for the Central Logger in the `ASG_CONFIG_HOME/asg.properties` file as follows:

Rendezvous Session Parameters for Core Engine to Communicate with Central Logger

Parameter	Description
<code>tibco.clientVar.Common/Connections/RV/SubjectPrefix</code>	Specifies the prefix for the Rendezvous subject names used by the Core Engine to communicate with the other components. The default value is: <code>TIBCO.ASG.INTERNAL</code>
<code>tibco.clientVar.ASG/CL/RV/RvDaemon</code>	Specifies the value of the Rendezvous daemon for the Core Engine to send messages to the Central Logger. The default value is: <code>tcp:7500</code> . By default, this property is not defined in the <code>ASG_CONFIG_HOME/asg.properties</code> file. You must explicitly add this.
<code>tibco.clientVar.ASG/CL/RV/RvNetwork</code>	Specifies the value of the Rendezvous network for the Core Engine to send messages to the Central Logger. By default, this property is not defined in the <code>ASG_CONFIG_HOME/asg.properties</code> file. You must explicitly add this.
<code>tibco.clientVar.ASG/CL/RV/RvService</code>	Specifies the value of the Rendezvous service for the Core Engine to send messages to the Central Logger. The default value is: <code>7500</code> . By default, this property is not defined in the <code>ASG_CONFIG_HOME/asg.properties</code> 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 Core Engine. Set the properties for the Central Logger to communicate with the Rendezvous daemon in the `ASG_CONFIG_HOME/asg_cl.properties` file as follows:

Rendezvous Session Connection Parameters for Central Logger

Parameter	Description
<code>tibco.clientVar.Common/Connections/RV/SubjectPrefix</code>	Specifies the prefix for the Rendezvous subject names used by the Core Engine to communicate with the Central Logger and the Global Throttle Manager. The default value is: <code>TIBCO.ASG.INTERNAL</code>
<code>tibco.clientVar.ASG/CL/RV/RvDaemon</code>	Specifies the value of the Rendezvous daemon for the Central Logger to listen to the messages from the Core Engine. The default value is: <code>tcp:7500</code> . By default, this property is not defined in the <code>ASG_CONFIG_HOME/asg.properties</code> file. You must explicitly add this.
<code>tibco.clientVar.ASG/CL/RV/RvNetwork</code>	Specifies the value of the Rendezvous network for the Central Logger to listen to the messages from the Core Engine. By default, this property is not defined in the <code>ASG_CONFIG_HOME/asg.properties</code> file. You must explicitly add this.
<code>tibco.clientVar.ASG/CL/RV/RvService</code>	Specifies the value of the Rendezvous service for the Central Logger to listen to the messages from the Core Engine. The default value is: <code>7500</code> . By default, this property is not defined in the <code>ASG_CONFIG_HOME/asg.properties</code> file. You must explicitly add this.



For the communication between the 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 Core Engine and Global Throttle Manager Communication

List of Rendezvous session connection parameters for the Core Engine and the Global Throttle Manager.

The properties must be defined in the `ASG_CONFIG_HOME/asg.properties` file.

Rendezvous Parameters for Core Engine and Global Throttle Manager Communication

Parameter	Description
<code>tibco.clientVar.Common/Connections/RV/SubjectPrefix</code>	Specifies the prefix for the Rendezvous subject names used by the Core Engine to communicate with other components. The default value is: <code>TIBCO.ASG.INTERNAL</code>
<code>tibco.clientVar.ASG/GTM/RV/RvDaemon</code>	Specifies the value of the Rendezvous daemon for the Core Engine to connect to the Global Throttle Manager. The default value is: <code>tcp:7500</code> . By default, this property is not defined in the <code>ASG_CONFIG_HOME/asg.properties</code> file. You must explicitly add this.
<code>tibco.clientVar.ASG/GTM/RV/RvNetwork</code>	Specifies the value of the Rendezvous network for the Core Engine to connect to the Global Throttle Manager. By default, this property is not defined in the <code>ASG_CONFIG_HOME/asg.properties</code> file. You must explicitly add this.
<code>tibco.clientVar.ASG/GTM/RV/RvService</code>	Specifies the value of the Rendezvous service for the Core Engine to connect to the Global Throttle Manager. The default value is: <code>7500</code> . By default, this property is not defined in the <code>ASG_CONFIG_HOME/asg.properties</code> file. You must explicitly add this.

Secure Deployments with TIBCO Rendezvous

Overview of secure communication using the Rendezvous secure daemons.

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

For the DMZ (De-Militarized Zone) setup, the Apache server runs on the machine outside the firewall and the Core Engine runs on a machine inside the firewall. The following options are available to run the Apache server:

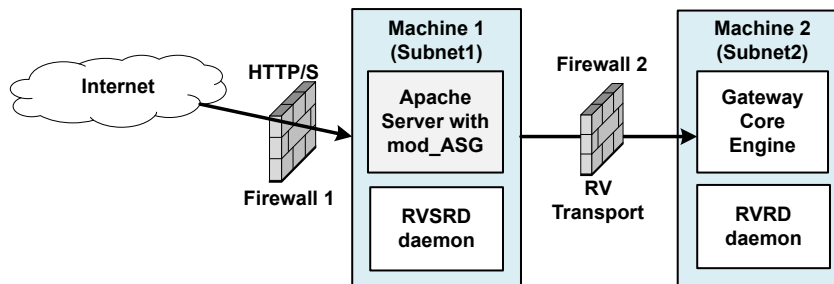
- Option 1: Run the Apache server on the same machine where secure `rvsrd` (or `rvsd`) daemon runs.
- Option 2: Run the Apache server and secure `rvsrd` (or `rvsd`) daemon on different machines. In this case, the Apache server acts a client to connect to `rvsrd` (or `rvsd`) daemons. If the Apache server and `rvsd` daemon are running on the machines in the same subnet, `rvrd` configuration is not required to connect from the Apache server to `rvsd`. However, if the Apache server and the `rvsrd` daemon are

running on machines in different subnets, you must configure routing daemon between the two machines (one with Apache server machine running `rverd` and the other machine running `rvsrd`).

The Core Engine runs on a machine in a secure network inside the firewall which has `rverd` running. Configure `rvsrd` (running on a machine where the Apache server runs in case of Option 1) and `rverd` (running on a machine where the Core Engine is running) as neighbors. See *TIBCO Rendezvous Administration Guide* for configuration setup details of the Rendezvous daemons.

This chapter mainly explains the configuration details required for the Apache module to connect to `rvsrd` (or `rvsd`) daemons.

Secure Deployment with Rendezvous



The figure above illustrates an example deployment of the Apache module and the Core Engine in a DMZ setup where the Apache module communicates with the secure Rendezvous daemons. In this deployment, the Apache server and Rendezvous secure daemon (`rvsrd`) runs on the same machine, Machine 1. The deployment consists of the following components:

Components of Machine 1

- The Apache server. Refer to `ASG_HOME/readme` file for the supported version of the Apache server.
- The TIBCO API Exchange Gateway Apache Module. See [Setting up and Configuring Apache Module](#) for details.
- The TIBCO Rendezvous secure daemon (`rvsrd`). Refer to the `ASG_HOME/readme` file for the supported Rendezvous version.

Components of Machine 2

- The Core Engine.
- The TIBCO Rendezvous daemon (`rverd`). Refer to the `ASG_HOME/readme` file for the supported Rendezvous version.



If Machine 1 and Machine 2 are on the same subnet, you can use `rvsd` on Machine 1 and `rvd` on Machine 2.

Configuration for Secure Rendezvous Daemon

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

Configuration Tips

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

- Ensure that the subject name configured for Local Area Network (LAN) during the `rvsrd` daemon configuration on Machine 1 matches the subject name configured for LAN for the `rverd` daemon

configuration on Machine 2. The authorized subject names used in the `rvsrd` configuration on Machine 1 must be the same as the subject name used for the `rvrd` configuration on Machine 2.

- Ensure that the authorized subject names used in the configuration of `rvsrd` on Machine 1 matches 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 the subject name set using the `tibco.clientVar.ASG/modRV/north_request` property in the Core Engine properties file. The properties file is located as the `ASG_CONFIG_HOME/asg.properties` file.
- Ensure that the subject name configured using the `AsgSubject` parameter in the Apache module configuration file (`mod_ASG.conf`) on Machine 1 matches the subject name set using the `tibco.clientVar.ASG/modRV/north_request` property in the Core Engine properties file. The properties file is located as the `ASG_CONFIG_HOME/asg.properties` file.
- The listen port used to start the `rvsrd` daemon on Machine 1 must be different from the listen port used to start the `rvrd` daemon on Machine 2.

For example,

- Start the `rvsrd` daemon on Machine 1 as follows:

```
rvsrd -store rvsrd.store -http 3500 -listen 7500
```

- Start the `rvrd` daemon on Machine 2 as follows:

```
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 Core Engine properties (`asg.properties`) file.

For example,

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

```
AsgNetwork ;239.1.1.11
```

In the Core Engine properties (`asg.properties`) file on Machine 2, set the network property value as follows:

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

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

Setup and Configure Rendezvous Daemons

This section explains the guidelines to 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](#) diagram.
- Configure the secure Rendezvous daemon (`rvsrd` or `rvsd`) on the machine that is outside the firewall. This is shown as Machine 1 in the [Secure Deployment with Rendezvous](#) diagram. See *TIBCO Rendezvous Administration* for detailed instructions to configure `rvsrd` or `rvsd`, as required.
- Configure the Rendezvous daemon (`rvrd` or `rva`) on the machine that is inside the inner security zone. This is shown as Machine 2 in the [Secure Deployment with Rendezvous](#) diagram. See *TIBCO Rendezvous Administration* for detailed instructions to configure `rvrd` or `rva`, as required.
- For `rvsrd` or `rvsd` configuration, see [Configuration Tips](#).

Configuration Setup for Apache Module and TIBCO API Exchange Gateway

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

Install Apache Server

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

Install TIBCO API Exchange Gateway

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

Setting up and Configuring Apache Module

This section explains the configuration to set up the Apache module.

Setting up Apache Module on Machine 1

To set up the Apache module on the machine where Apache server runs (Machine 1), follow these steps:

Procedure

1. Copy the following files from the TIBCO API Exchange Gateway 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. Save these files locally on Machine 1 where the Apache server runs. You may place these files in a location accessed by the 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.

Configuring Apache Module on Machine 1

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

Procedure

1. Open the `mod_ASG.conf` file for editing.
2. Set the parameters as described in the following table:

Apache Module Properties

Property	Description
AsgService	<p>Specifies the service parameter configured for <code>rvsrd</code> on Machine 1. For example, 1111.</p> <p>This parameter value must be configured different from the value specified for <code>rvrd</code> setup on Machine 2.</p>
AsgNetwork	<p>Specifies the network parameter set during the configuration of secure Rendezvous daemon (<code>rvsrd</code>).</p> <p>This parameter is a random multicast IP address used to broadcast messages to the machines in that multicast group.</p> <p>For example, ;239.1.1.11</p> <ul style="list-style-type: none"> This network parameter value must match the network value set during the configuration of <code>rvrd</code> setup on Machine 2.
AsgDaemon	<p>Specifies the daemon value set during the configuration of secure Rendezvous daemon (<code>rvsrd</code>). For example,</p> <p><code>ssl:ASGRVSecure:7500</code></p> <p>You must specify the <code>ssl</code> prefix before the machine name, otherwise the connection fails.</p>
AsgSubject	<p>Specifies the subject name used to send the message to the secure Rendezvous daemon (<code>rvsrd</code>).</p>
AsgSecureDaemon	<p>A Boolean property to enable or disable the secure Rendezvous daemon connection for the Apache module.</p> <p>The secure Rendezvous daemon can run on the same or different machines where the Apache server is running.</p> <p>Possible values are <code>On</code> and <code>Off</code>.</p> <p>Set this value to <code>On</code> to enable the Apache module to connect to the secure Rendezvous daemon (<code>rvsrd</code>).</p>
AsgSecureDaemonCert	<ul style="list-style-type: none"> Specifies the path to the public certificate of the secure Rendezvous daemon (<code>rvsrd</code>). This public certificate is configured during the <code>rvsrd</code> setup. <p>For example,</p> <p><code>C:\tibcoasg\tibrv\8.3\certs\cert2.pem</code></p> <ul style="list-style-type: none"> Required.

Property	Description
AsgSecureDaemonUsername	<ul style="list-style-type: none"> Specifies the username used in rvsrcd configuration. <p>If AsgSecureDaemonUsername is set, the Apache module uses the username and password to connect to the rvsrcd daemon.</p> <p>If AsgSecureDaemonUsername is not set, AsgSecureDaemonKey parameter must be set. See AsgSecureDaemonKey.</p> <ul style="list-style-type: none"> Optional.
AsgSecureDaemonPassword	<ul style="list-style-type: none"> Specifies the password used by the client in rvsrcd configuration. The password is required when connecting to the rvsrcd daemon either using the username or the client certificate. You can specify an obfuscated password for this parameter. The obfuscated password is generated using the asg-password-obfuscator utility located in the <code>ASG_HOME/bin</code> directory. Required.
AsgSecureDaemonKey	<ul style="list-style-type: none"> Specifies the path to the user certificate of secure Rendezvous daemon (rvsrcd). This user certificate is configured for a user in the rvsrcd setup. The certificate should be in text (PEM) format. <p>The Apache module connects to the secure Rendezvous daemon (rvsrcd) using the user certificate specified by this parameter.</p> <ul style="list-style-type: none"> If this parameter is not set, the Apache module connects to the secure Rendezvous daemon (rvsrcd) using the username and password specified by AsgSecureDaemonUsername and AsgSecureDaemonPassword parameters. Optional.

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 the AsgSecureDaemonPassword parameter of the `mod_ASG.conf` file of Apache module. See [asg-password-obfuscator Utility](#) for usage details.

Sample Properties For Apache Module

List of sample properties for Apache module.

The following is the list of properties with example values for the Apache module set in the `mod_ASG.conf` file. Refer to [Apache Module Properties](#) 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"

Configuring the Core Engine Properties

List of Core Engine Properties for Rendezvous communication with Apache module.

You must set the following properties for the Core Engine to receive the requests from the Apache module.

To set or edit the properties, follow these steps:

Procedure

1. Navigate to the `ASG_CONFIG_HOME` directory.
2. Open the `asg.properties` file in a text editor.
3. Set or edit the properties as described in the [Core Engine Properties](#) table:

Core Engine Properties

Property	Description
<code>tibco.clientVar.ASG/modRV/facade_request</code>	<ul style="list-style-type: none"> • Specifies the subject name used by the Rendezvous daemon to listen the requests from the Apache module. The Core Engine listens to the requests on the same subject. <ul style="list-style-type: none"> – This property value must match the subject value specified in the Apache module configuration file (<code>mod_ASG.conf</code>) for Apache server. – This property value must match the subject value specified for the <code>rvrd</code> configuration. • The default value is: <code>MachineName.asg.north.request</code>

Property	Description
<code>tibco.clientVar.ASG/modRV/RvDaemon</code>	<ul style="list-style-type: none"> Specifies the value of the Rendezvous daemon for the Core Engine to connect and listen for the requests from the Apache module. <ul style="list-style-type: none"> This property value should match the listen port value given for the command to start the <code>rvr</code> daemon on the machine where the Core Engine runs. For example, Set this property value to 7502 for the following command used to start the <code>rvr</code> daemon: <pre>rvr -store rvr.store -http 3500 -listen 7502</pre> This property value must be different from the listen port value given for the command to start the <code>rvs</code> daemon on the machine where the Apache server runs. The default value is: 7500.
<code>tibco.clientVar.ASG/modRV/RvNetwork</code>	<p>Specifies the value of the Rendezvous network for the Core Engine to connect and listen for the requests from the Apache module.</p> <p>This property value must match the network value specified in the Apache module configuration file for Apache server. See Setting up and Configuring Apache Module.</p> <p>For example, the value can be specified as: ;239.1.1.11</p>
<code>tibco.clientVar.ASG/modRV/RvService</code>	<p>Specifies the value of the Rendezvous service for the Core Engine to connect and listen for the requests from the Apache module.</p> <p>For example, 2222.</p>

4. Save the changes to the file.

Sample Properties for Core Engine

List of sample properties for Core Engine.

The following is the list of properties with example values for the Core Engine set in the `asg.properties` file. Refer to [Core Engine Properties](#) 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

asg-password-obfuscator Utility

This utility generates an obfuscated password.

TIBCO API Exchange Gateway provides the `asg-password-obfuscator` utility to generate an obfuscated password.

For example,

- An obfuscated password can be specified for the `tibco.env.ASG_ADMIN_PASSWORD` property of the Config UI. The Config UI uses this password to verify the credentials for the default authentication mechanism of the Config UI. See [Default Authentication](#).
- An obfuscated password is used by the Apache module (C module) to communicate with the Rendezvous daemon. The obfuscated password can be set for the `AsgSecureDaemonPassword` parameter in the Apache module configuration (`mod_ASG.conf`) file.

Usage

Usage: `asg-password-obfuscator <password>`

Example Output

```
C:\tibcoasg\asg\2.3\bin>asg-password-obfuscator 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):
[#_R9gvPGRME0hRIveQJJS9i9tAzshJUjfk]
```

If the obfuscated password generated by the `asg-password-obfuscator` utility contains special characters, you must escape the special characters before using the obfuscated password.

For example, to use the obfuscated password for `tibco.env.ASG_ADMIN_PASSWORD` property of the `ASG_HOME/bin/asg-configui.tra` file, escape the special characters if the obfuscated password contains the special characters. See the following examples:

- The `asg-password-obfuscator` utility generates `#_R9gvPGRME0hRIveQJJS9i9tAzshJUjfk` as the password. Use this password in the `ASG_HOME/bin/asg-configui.tra` file, as follows:

```
tibco.env.ASG_ADMIN_PASSWORD=\#\_R9gvPGRME0hRIveQJJS9i9tAzshJUjfk
```

- The `asg-password-obfuscator` utility generates `#!gJRT2BcdqcmvDRxKcJAcJxTRt3FHPK1AwOZlrvOiLE=` as the password. Use this password in the `ASG_HOME/bin/asg-configui.tra` file, as follows:

```
tibco.env.ASG_ADMIN_PASSWORD=\#\!gJRT2BcdqcmvDRxKcJAcJxTRt3FHPK1AwOZlrvOiLE\=
```

- The `asg-password-obfuscator` utility generates `#!gJRT2BcdqcmvDRxKcJAcJxTRt3FHPK1AwOZlrvOiLE=` as the password which contains the embedded special characters. Use this password in the `ASG_HOME/bin/asg-configui.tra` file, as follows:

```
tibco.env.ASG_ADMIN_PASSWORD=\#\!gJRT2BcdqcmvDRxKcJAc\!xTRt3FHPK1AwOZlrvOiLE\=
```



Enabling Facade HTTP Transport

To enable the native HTTP transport at the facade side of the TIBCO API Exchange Gateway, follow these steps:

Procedure

1. Start the Config UI.
2. Log in to the Config UI using your credentials.
3. On the home page of the Config UI, select **Gateway Engine Properties** from the drop-down list.
4. Click the **Transport** link to display the runtime properties.
5. Expand the **Facade Node**.
6. Set the **Port** field for HTTP request. The default is 9222.
7. Save the changes to the configuration.

Enable Facade HTTPS Transport

TIBCO API Exchange Gateway supports the native HTTPS channel at the facade side.

Setting SSL Properties

To enable the HTTPS channel at the facade side of the TIBCO API Exchange Gateway, set the SSL properties on the Config UI as follows:

Procedure

1. Start the Config UI.
2. Log in to the Config UI using your credentials.
3. On the home page of the Config UI, select the **Gateway Engine Properties** from the drop-down list.
4. Click the **Transport** link to display the runtime properties.
5. Set the SSL properties as explained in the [SSL](#) section of [Transport Properties](#) table.
6. Save the configuration changes to the project.



- The SSL properties can be set in the `ASG_CONFIG_HOME/asg.properties` file. Refer to the [Connection Parameters for HTTPS Channel\(Facade\)](#) properties.
- If the `tibco.clientVar.DefaultImplementation/Connections/HTTP/FacadeHTTPSSLConnection/IdentityFileType` property is set to `certPlusKeyURL`, download Tomcat and configure the TIBCO API Exchange Gateway to include the Tomcat native library. See [Download Tomcat Native Library](#) for details.

Download Tomcat Native Library

Perform the following steps to download and copy the required native library.

Downloading on Windows platform

Copy the library as follows:

Procedure

1. Download the `tomcat-native-1.1.31-win32-bin.zip` archive from the Apache Tomcat website.
For example, download the `tomcat-native-1.1.31-win32-bin.zip` archive file from the following URL to a temporary location, such as `C:\temp\apache_lib`.
`http://mirror.reverse.net/pub/apache/tomcat/tomcat-connectors/native/1.1.31/binaries/tomcat-native-1.1.31-win32-bin.zip`
2. Navigate to the temporary location directory, `C:\temp\apache_lib`, where the `tomcat-native-1.1.31-win32-bin.zip` archive is downloaded.
3. Extract the `tomcat-native-1.1.31-win32-bin.zip` archive in the same directory, `C:\temp\apache_lib`.
4. Copy the `tcnative-1.dll` library to the `TIBCO_HOME\tibcojre64\1.7.0\bin` directory.
 - a) For Windows 64-bit platform, the `tcnative-1.dll` library is found at the `C:\temp\apache_lib\tomcat-native-1.1.31-win32-bin\bin\x64` location.
 - b) For Windows 32-bit platform, the `tcnative-1.dll` library is found at the `C:\temp\apache_lib\tomcat-native-1.1.31-win32-bin\bin` location.

Downloading on Linux platform

Download, build, and copy the library.

Downloading and Building APR

Procedure

1. Install the Apache Portable Runtime (APR) and OpenSSL (if not already installed on the system).
2. For Debian-based Linux platform, execute the following command:
`apt-get install build-essential libapr1-dev libssl-dev`
3. For Redhat Linux, follow these steps:
 - a) Download the source (.tar.gz) file for APR from the following URL:
<http://apr.apache.org/download.cgi>
 - b) Execute the following command to extract the package:
`tar xvfz apr-1.5.0.tar.gz`
 - c) Execute the following commands at the command prompt in sequence:
`cd apr-1.5.0`
`./configure`
`make`
`make install`
 - d) Check the path shown at the end of `make install` command. The directory specified by this path should contain `apr-1-config`. The `apr-1-config` is required to build the Tomcat native library as mentioned in [Building Tomcat Native Library](#) section.

Building Tomcat Native Library

Procedure

1. Download the `tomcat-native.tar.gz` archive to a temporary location, such as `/home/user/tomnative`. The `tomcat-native.tar.gz` archive is located under `CATALINA_HOME/bin`.

2. Navigate to the temporary location directory where the `tomcat-native.tar.gz` archive is downloaded. For example, navigate to the `/home/user/tomnative` directory.
3. Extract the `tomcat-native.tar.gz` archive in the temporary location, such as `/home/user/tomnative` directory.
4. Change to the `/home/user/tomnative/tomcat-native-1.1.27-src/jni/native` directory.
5. Execute the following commands at the command prompt in sequence:
 - a) `./configure -with-apr=Full_APR_CONFIG_PATH -with-java-home=Full_Path_JDK -with-ssl=yes -prefix=Path_to_destination_directory`
 where,
 - `Path_to_destination_directory` is the destination directory where the libraries are created.
 - `Full_APR_CONFIG_PATH` is the full path to `apr-1-config`. For example, `/usr/bin/apr-1-config`.
 - `Full_Path_JDK` is the full path to JDK. For example, `/usr/tools/jdk_1.7.0_45`
 - b) `make`
 - c) `make install`
6. Verify that the `Path_to_destination_directory/lib` directory contains around five new files with extensions such as `.so`, `.la`, `.a` and so on after the successful installation.

Setting LD_LIBRARY_PATH

Set the `LD_LIBRARY_PATH` to the user's profile as follows:

Procedure

1. Open a command prompt window.
2. Set the `LD_LIBRARY_PATH` to the destination directory where the libraries are created, as follows

```
LD_LIBRARY_PATH = Path_to_destination_directory/lib:$LD_LIBRARY_PATH
export LD_LIBRARY_PATH
```

Setting Content-Type for Error Response

Using TIBCO API Exchange Gateway, you can set the Content-Type for the error message if the Content-Type is not set in the HTTP header of the incoming request.

When the error response is returned for any incoming request from TIBCO API Exchange Gateway, the content-type of the error response is set as follows:

- If the HTTP header of the incoming request message contains the Content-Type, TIBCO API Exchange Gateway sets the content type of the error response to the content type of the request message.
- If the HTTP header of the incoming request message does not contain the Content-Type, TIBCO API Exchange Gateway sets the content type of the error response using the `tibco.clientVar.ASG/Response/DefaultContentType` property. The property is set in the `ASG_CONFIG_HOME/asn.properties` file. See [tibco.clientVar.ASG/Response/DefaultContentType](#).



If the `tibco.clientVar.ASG/Response/DefaultContentType` property is not set, TIBCO API Exchange Gateway sets the Content-Type of the error response message to `application/json`.

To set the Content-Type for the error response from TIBCO API Exchange Gateway for any incoming request, follow these steps:

Procedure

1. Navigate to the `ASG_CONFIG_HOME` directory.
2. Edit the `asg.properties` file in a text editor.
3. Set the following property :

```
tibco.clientVar.ASG/Response/DefaultContentType
```

For example, you can set the property to `application/xml` as follows:

```
tibco.clientVar.ASG/Response/DefaultContentType=application/xml
```



- If the `tibco.clientVar.ASG/Response/DefaultContentType` property is set to `application/XML` or `text/XML`, the error response is returned in XML format to the requester.
- If the `tibco.clientVar.ASG/Response/DefaultContentType` property is set to `application/JSON`, the error response is returned in JSON format to the requester.

Endpoint Ports

Default ports for components of TIBCO API Exchange Gateway.

The following table lists default ports for various endpoints used in TIBCO API Exchange Gateway. You can change the default value of ports by editing the listed properties

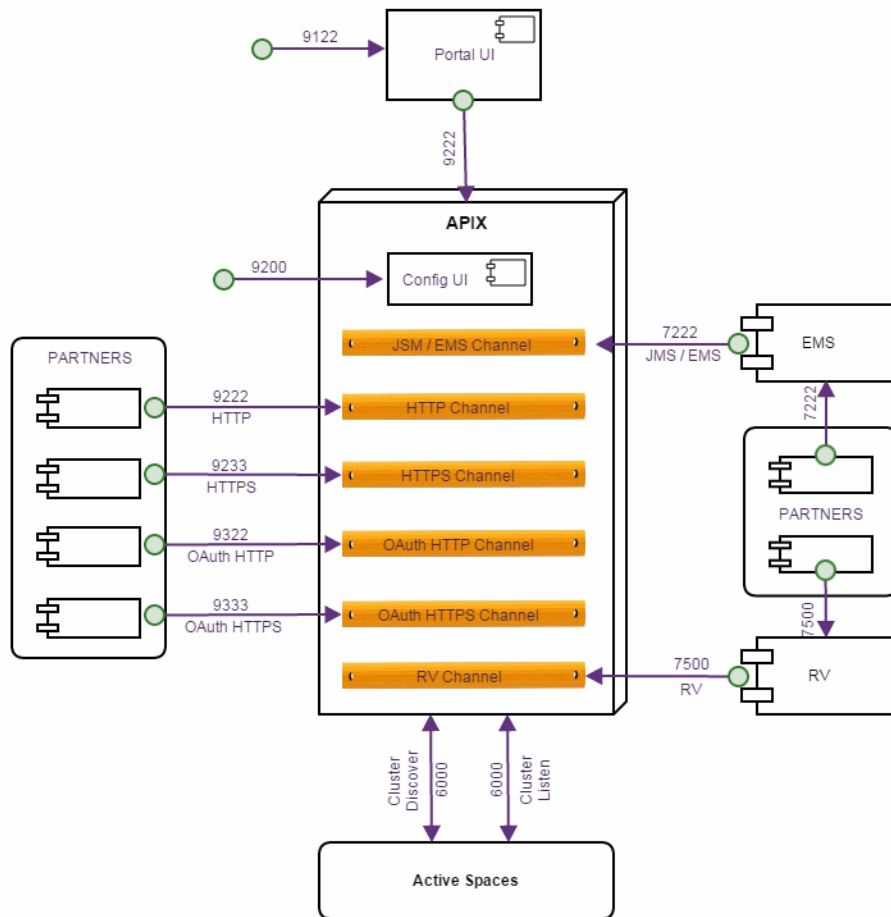
Endpoint Ports

Component/ Endpoint	Default Port Value	Property Name
ASG_CONFIG_HOME/asg_properties file:		
RV	7500	<code>tibco.clientVar.ASG/modRV/RvDaemon</code>
		<code>tibco.clientVar.ASG/modRV/RvService</code>
		<code>tibco.clientVar.ASG/GTM/RV/RvDaemon</code>
		<code>tibco.clientVar.ASG/GTM/RV/RvService</code>
		<code>tibco.clientVar.ASG/CL/RV/RvDaemon</code>
		<code>tibco.clientVar.ASG/CL/RV/RvService</code>

Component/ Endpoint	Default Port Value	Property Name
JMS or EMS	7222	<p>Specified by JMSProviderURL or JNDIContextURL property.</p> <p>For example,</p> <p>Set the port of ESB channel 1 of target service using the following property:</p> <ul style="list-style-type: none"> <code>tibco.clientVar.Common/Connections/JMS/TargetEsbConnection0/JMSProviderURL</code> <code>tibco.clientVar.Common/Connections/JMS/TargetEsbConnection0/JNDIContextURL</code> <p>Set the port of ESB channel 2 of target service using the following property:</p> <ul style="list-style-type: none"> <code>tibco.clientVar.Common/Connections/JMS/TargetEsbConnection1/JMSProviderURL</code> <code>tibco.clientVar.Common/Connections/JMS/TargetEsbConnection1/JNDIContextURL</code> <p>Set the port of ESB channel 3 of target service using the following property:</p> <ul style="list-style-type: none"> <code>tibco.clientVar.Common/Connections/JMS/TargetEsbConnection2/JMSProviderURL</code> <code>tibco.clientVar.Common/Connections/JMS/TargetEsbConnection2/JNDIContextURL</code> <p>Set the port of facade ESB channel using the following property:</p> <ul style="list-style-type: none"> <code>tibco.clientVar.Common/Connections/JMS/FacadeEsbConnection0/JMSProviderURL</code> <p>Set the port of SOAP JMS transport of facade using the following property:</p> <ul style="list-style-type: none"> <code>tibco.clientVar.Common/Connections/JMS/SOAPConnection_Facade/JMSProviderURL</code> <p>Set the port of SOAP JMS transport of target service using the following property:</p> <ul style="list-style-type: none"> <code>tibco.clientVar.Common/Connections/JMS/SOAPConnection_Target/JMSProviderURL</code>
Facade HTTP	9222	<code>tibco.clientVar.DefaultImplementation/Connections/HTTP/FacadeHTTPConnection/Port</code>
Facade HTTPS	9233	<code>tibco.clientVar.DefaultImplementation/Connections/HTTP/FacadeHTTPSSLConnection/Port</code>
OAuth HTTP	9322	<code>tibco.clientVar.DefaultImplementation/Connections/HTTP/OAuthWebappsConnection/Port</code>
OAuth HTTPS	9333	<code>tibco.clientVar.DefaultImplementation/Connections/HTTP/OAuthWebappsSSLConnection/Port</code>

Component/ Endpoint	Default Port Value	Property Name
OAuth Data Space	6300	<p>To set the port in the URL for OAuth data space, use the following properties:</p> <ul style="list-style-type: none"> tibco.clientVar.oauth.dataspace.local.discovery=tcp://localhost:6300 tibco.clientVar.oauth.dataspace.local.listen=tcp://localhost:6300
ASG_HOME/bin/asg-configui.tra file:		
Config UI	9200	tibco.env.ASG_PORT
<ul style="list-style-type: none"> ASG_HOME/bin/asg_core.cdd file: 		
Active Spaces- GTM	6001	<ul style="list-style-type: none"> be.engine.cluster.as.discover.url be.engine.cluster.as.listen.url be.mm.cluster.as.listen.url
Active Spaces- Cluster	6000	<ul style="list-style-type: none"> <discovery-url>tcp://IP_Address_Of_Machine:6000/</discovery-url> <listen-url>tcp://IP_Address_Of_Machine:6000-*/</listen-url>

The following figure illustrates the components and endpoints communication with TIBCO API Exchange Gateway using the default ports.

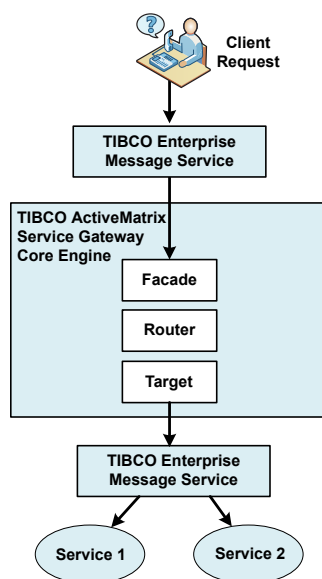


JMS Transport Communication

Overview of JMS transport communication.

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

You can use the JMS transport as SOAP/JMS or JMS/XML as ESB channels.



SOAP JMS Transport

The SOAP JMS channel of the 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 Core Engine to act as both the SOAP JMS server and SOAP JMS client. You can use this transport at both facade and target sides.

When the SOAP JMS is used for communication, the queue names for facade and target are specified as the global variables in the *ASG_CONFIG_HOME/asg.properties* file.



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

- Using the Config UI. See [Set Runtime Properties](#) for details.
- Editing the *ASG_CONFIG_HOME/asg.properties* file using a text editor.

Gateway as SOAP JMS Server

TIBCO API Exchange Gateway 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 place the request. By default, the queue name for the incoming request is `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 `asg.soap.in.request.reply.0`.

When a client sends a request using the SOAP JMS transport, it sets the `JMSReplyDestination` header field on the request message. The Core Engine uses the destination name as specified in the `JMSReplyDestination` header field to send the response. If the `JMSReplyDestination` header field of the request message is not set by the client, the Core Engine uses the queue name to send the response as specified in the *ASG_CONFIG_HOME/asg.properties* file.

The 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 the client. If the `JMSCorrelationId` header field value of the client request message is empty, the 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 the facade side.
- The request queue must exist on the JMS server at the facade side when the 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

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

Request Destination at Target Side

When the 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 the **Services** tab configuration of the Config UI 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 Config UI is empty.



The Config UI gives an error if the value of the **Destination Name** field of the **ROUTING > Target operations** tab on the Config UI is empty. You can set this value as empty in the TargetOperation.cfg directly located under the `ASG_CONFIG_HOME/ASG_Config_name` directory, where `ASG_Config_name` is the gateway configuration project.

Response Destination at Target Side

When the SOAP JMS transport is used at the southbound (target) side, the 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 Core Engine sets the `JMSReplyDestination` as the value specified in the reply queue name to store the responses from the target operations.

When the 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 the target side.
- The response queue should exist on the JMS server at the target side when the Core Engine is started. If the Core Engine cannot find them on the JMS server, it throws an exception. When the Core Engine is started and the SOAP JMS channel is enabled, the Core Engine connects to the JMS server and starts listening to the configured queue during the engine startup.

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

Configuring SOAP JMS Transport

Configuration setup to use SOAP JMS transport.

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

Enabling SOAP JMS Channels

By default, SOAP JMS channels are disabled. To enable the SOAP JMS channels, follow these steps:

Procedure

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,/ASG/Channels/
modAS_Channel,/Common/Channel/PSMChannel,/Common/Channel/CentralLoggerJMS"/>
```

3. If you have not taken the backup of the **be.channel.deactivate** property, copy the **be.channel.deactivate** property to **be.channel.deactivate.backup**, as follows:

```
<property name="be.channel.deactivate.backup" 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,/ASG/Channels/modAS_Channel,/Common/Channel/PSMChannel,/
Common/Channel/CentralLoggerJMS"/>
```

4. To enable the SOAPJMS channels, remove the /ASG/Channels/SOAPJMSChannel_North and /ASG/Channels/SOAPJMSChannel_South from the value of the **be.channel.deactivate** property. The modified value of the property is as follows:

```
<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/modAS_Channel,/Common/Channel/PSMChannel,/Common/Channel/
CentralLoggerJMS"/>
```

5. Save the changes to the file.

Edit asg.properties File for JMS Server

Edit the JMS server properties for SOAP JMS transport.

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

JMS Server Connection Parameters

List of properties for JMS server connection at facade and target side.

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

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

JMS Server Connection Parameters(Facade Side)

Property Name	Default Value
tibco.clientVar.Common/Connections/JMS/SOAPConnection_Facade/JMSProviderURL	tcp://localhost:7222
tibco.clientVar.Common/Connections/JMS/SOAPConnection_Facade/JNDIContextURL	tibjmsnaming://localhost:7222
tibco.clientVar.Common/Connections/JMS/SOAPConnection_Facade/TopicConnectionFactoryName	TopicConnectionFactory
tibco.clientVar.Common/Connections/JMS/SOAPConnection_Facade/QueueConnectionFactoryName	QueueConnectionFacto ry
tibco.clientVar.Common/Connections/JMS/SOAPConnection_Facade/JNDIUsername	admin
tibco.clientVar.Common/Connections/JMS/SOAPConnection_Facade/JNDIPassword	

Property Name	Default Value
tibco.clientVar.Common/Connections/JMS/SOAPConnection_Facade/JMSUsername	admin
tibco.clientVar.Common/Connections/JMS/SOAPConnection_Facade/JMSPassword	

JMS Server Connection Parameters at Target Side

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

JMS Server Connection Parameters (Target Side)

Property Name	Default Value
tibco.clientVar.Common/Connections/JMS/SOAPConnection_Target/JMSProviderURL	tcp://localhost:7222
tibco.clientVar.Common/Connections/JMS/SOAPConnection_Target/JNDIContextURL	tibjmsnaming://localhost:7222
tibco.clientVar.Common/Connections/JMS/SOAPConnection_Target/TopicConnectionFactoryName	TopicConnectionFactory
tibco.clientVar.Common/Connections/JMS/SOAPConnection_Target/QueueConnectionFactoryName	QueueConnectionFactory
tibco.clientVar.Common/Connections/JMS/SOAPConnection_Target/JNDIUsername	admin
tibco.clientVar.Common/Connections/JMS/SOAPConnection_Target/JNDIPassword	
tibco.clientVar.Common/Connections/JMS/SOAPConnection_Target/JMSUsername	admin
tibco.clientVar.Common/Connections/JMS/SOAPConnection_Target/JMSPassword	

JMS Server Queue Names

Queue Names for JMS server at facade and target side.

By default, the 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. Edit the default values, if needed.

Facade Queue Names (SOAPJMS)

Global Variable	Default Value
tibco.clientVar.ASG/facade/SOAPJMS/requestQueue	asg.soap.in.request
tibco.clientVar.ASG/facade/SOAPJMS/replyQueue	asg.soap.in.request.reply .0

Target Queue Names for SOAPJMS Transport

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

Target Queue Names (SOAPJMS)

Global Variable	Default Value
tibco.clientVar.ASG/Endpoint/SOAPJMS/DefaultTargetRequestQueue	asg.soap.forward
tibco.clientVar.ASG/Endpoint/SOAPJMS/TargetResponseQueue	asg.soap.forward.reply.0



Ensure that all queue names are created on the JMS server before the Core Engine is started. See [Create queues on EMS Server](#).

Create queues on EMS Server

By default, the TIBCO API Exchange Gateway 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, 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 change the default values as mentioned in [Facade Queue Names for SOAPJMS Transport](#) and [Target Queue Names for SOAPJMS Transport](#), create the queue names defined by the global variables.
- You can edit the global variable values as follows:
 - By using the Config UI. See [Set Runtime Properties](#).
 - By editing the `ASG_CONFIG_HOME/asg.properties` file using a text editor.

Create Users On JMS Server

Ask the JMS server administrator to create a normal user without administrator privileges. TIBCO API Exchange Gateway uses this user to connect to the JMS server when the JMS transport is used at the 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](#).

Setting JMS Jars in ClassPath

To include the JMS jar files in the classpath of the Core Engine, do any of the following:

- Copy JMS jars
Manually copy the JMS jar files in the following location:
ASG_HOME/lib/ext/tpcl
- Edit the *asg-engine.tra* file to set the *EMS_HOME* variable.
Set the *tibco.env.EMS_HOME* property as defined in the *ASG_HOME/bin/asg-engine.tra* file. Set this to the TIBCO Enterprise Message Service installation home.
For example,
tibco.env.EMS_HOME=c:/tibco/ems/5.1

Config UI Configuration

This section describes the configuration required to process the SOAP JMS requests using the Config UI.

Configuring New Partner

Configure a new partner for SOAP JMS transport.

Define a new partner to process the requests from this partner using the SOAP JMS transport.

Follow these steps to define a new partner:

Procedure

1. Start the gateway configuration interface. See [Starting GUI](#).
2. Select an existing project configuration or add a new project configuration. See [Manage a Gateway Project Configuration](#).
3. Define a new partner. See [Using Partners tab](#) to add the partner data.

The following table displays a sample value for the required field:

Partner Sample Configuration

Parameter	Sample Value
Partner Name	anon_JMS



Define the partner name **anon_JMS** must be defined by the following property in the `ASG_CONFIG_HOME/asg.properties` file: `tibco.clientVar.ASG/anonymous/PartnerName/Authenticated=anon_JMS`

4. Save the changes.

Configuring Facade Operation

Configure a new facade operation for SOAP JMS transport.

Define a new operation to process the request from the client using the SOAP JMS transport. See [Facade Operations](#).

The following are sample values for the required fields:

Facade Operation Sample Configuration

Parameter	Sample Value
Operation Name	GetBooks1
SOAP Action	"/QueryBooksByAuthor"
Operation URI	
Status Code on Error	500



The operation and SOAP Action must match the client request.

Configuring Partner Operation

Configure a partner operation for SOAP JMS transport.

Define a partner operation for a partner. See [Facade Access](#).

The following table displays sample values for the required fields:

Partner Operation Sample Configuration

Parameter	Sample Value
Partner	anon_JMS
Partner Operation	GetBooks1
Partner Timeout	20000
Forward Mapping	Pass-Through
Reverse Mapping	Pass-Through

Configuring Target Operation

Configure a new target operation for SOAP JMS transport.

Define a new service configuration to forward the client requests to a back-end service using the SOAP JMS transport. See [Adding a New Target Operation](#) to add a new service.

The following table displays sample values for the required fields:

Target Operation Sample Configuration

Parameter	Sample Value
Operation Name	getByAuthorService
Type	SOAP JMS
Timeout	30000
SOAP Action	"/QueryBooksByAuthor"
Destination Name	sample
Destination Type	Queue



Save the configuration changes.

SSL Support for JMS Transport

Overview of SSL connection properties for JMS transport.

TIBCO API Exchange Gateway supports SSL for SOAP JMS transport and ESB channels at the facade and target sides. The SSL configuration parameters for JMS transport are configured using the **Show SSL Properties** tab on the Config UI.

Go to the **Show SSL Properties** tab as follows:

Procedure

1. On the home page of the Config UI, select the **Gateway Engine Properties** in the drop-down list.
2. Click the **Transport** link.
3. Expand the required JMS transport node. For example, to set the SSL properties for ESB channel at the facade side, expand the **JMS Facade ESB Connection** node.
4. Click the **Show SSL Properties** tab.

See [Show SSLProperties](#) to configure the SSL connection parameters for JMS transport.



SSL is supported only when using TIBCO Enterprise Message Service. If you specify Identity Type as certpluskeyurl, the Identity File type must be set to use TIBCO security vendor as entrust61. Also, place enttoolkit.jar in the `ASG_HOME/lib/ext/tpcl` directory.

Set JMS Message Delivery and Acknowledgment Mode

Overview of delivery and acknowledgment modes for the JMS messages.

This section explains the delivery and acknowledgment modes for the JMS messages as processed by the TIBCO API Exchange Gateway. You can set the delivery and acknowledgment modes for a message sent over the JMS channel.

Refer to the *TIBCO Enterprise Message Service User's Guide* for details on the JMS message delivery and acknowledgment modes.

JMS Message Delivery Modes

Overview of delivery modes for JMS message.

The delivery mode for a JMS message is specified by the sender and instructs the server concerning persistent storage for the message. The JMSDeliveryMode message header field defines the delivery mode for the message.

JMS supports PERSISTENT and NON_PERSISTENT delivery modes for both topic and queue. TIBCO Enterprise Message Service extends these delivery modes to include a RELIABLE delivery mode.

To set the delivery mode for the JMS message, add the following configuration property in the `ASG_CONFIG_HOME/asg.properties` file:

```
tibco.clientVar.ASG/SharedResources/JMS/deliveryMode=delivery_mode_number
```

where *delivery_mode_number* is one of the numbers that represent a delivery mode, as shown in [JMS Message Delivery Modes](#) table.

For example, if you want to set the delivery mode as PERSISTENT, add the property as follows:

```
tibco.clientVar.ASG/SharedResources/JMS/deliveryMode=2
```

JMS Message Delivery Modes

No.	Mode	Description
1	NON_PERSISTENT	
		The message is not persisted on the disk or database by the server, so you lose the in-transit message when the server is restarted.
2	PERSISTENT	
		<ul style="list-style-type: none"> Ensures the delivery of messages to the destination on the server in almost all circumstances. This is applicable when a producer sends a PERSISTENT message when waiting for the server to reply with a confirmation. The message is persisted on the disk by the server. This is the default value.
22	RELIABLE	

No.	Mode	Description
		<ul style="list-style-type: none"> • TIBCO Proprietary • This value is an extension of the JMS standard delivery modes. • Defines the reliable delivery mode and is used only in TIBCO Enterprise Message Service. • When this delivery mode is used, it offers increased performance of the message producers.



- JMS providers such as IBM WebSphere MQ support the standard delivery modes (1 interpreted as NON_PERSISTENT and 2 interpreted as PERSISTENT), but, do not support the delivery mode 22 interpreted as RELIABLE.
- TIBCO Enterprise Message Service supports the standard delivery modes (1 interpreted as NON_PERSISTENT, 2 interpreted as PERSISTENT) and the extended mode 22 interpreted as RELIABLE.
- Persistent messaging is usually slower than non-persistent delivery.

JMS Message Acknowledgment Mode

Overview of acknowledgment modes for JMS message.

Set the acknowledgment mode for the JMS message by adding a configuration property in the `ASG_CONFIG_HOME/asg.properties` file as follows:

```
tibco.clientVar.ASG/SharedResources/JMS/ackMode=acknowledgement_mode_number
```

where *acknowledgement_mode_number* is one of the numbers that represent an acknowledgment mode, as shown in [JMS Message Acknowledgement Modes](#) table.

For example, if you want to set the acknowledgement mode as CLIENT_ACKNOWLEDGE, add the property as follows:

```
tibco.clientVar.ASG/SharedResources/JMS/ackMode=2
```

JMS Message Acknowledgment Modes

No.	Mode	Description
1	AUTO_ACKNOWLEDGE	
		Specifies that the session is to automatically acknowledge consumer receipt of messages when message processing is complete.
2	CLIENT_ACKNOWLEDGE	
		Specifies that the consumer is to acknowledge all messages delivered in this session. With this acknowledgment mode, the client acknowledges a consumed message by calling the message's acknowledge method.
3	DUPS_OK_ACKNOWLEDGE	

No.	Mode	Description
		<p>Specifies that the session is to "lazily" acknowledge the delivery of messages to the consumer. "Lazy" means that the consumer can delay the acknowledgment of messages to the server until a convenient time; meanwhile the server might redeliver messages. This mode reduces the session overhead. However, if JMS fails, the consumer can receive duplicate messages.</p>
22	NO_ACKNOWLEDGE (TIBCO Proprietary)	<ul style="list-style-type: none"> TIBCO Enterprise Message Service extension to JMS acknowledge modes. Suppresses the acknowledgment of received messages. After the server sends a message to the client, all information regarding that message for that consumer is eliminated from the server. Therefore, there is no need for the client application to send an acknowledgment to the server about the received message. Not sending acknowledgments reduces message traffic and saves time for the receiver, therefore allowing better utilization of system resources. <p>Note: Sessions created in NO_ACKNOWLEDGE receipt mode cannot be used to create durable subscribers.</p> <p>Note: Also, queue receivers on a queue that is routed from another server are not permitted to specify NO_ACKNOWLEDGE mode.</p>
23	EXPLICIT_CLIENT_ACKNOWLEDGE (TIBCO Proprietary)	<ul style="list-style-type: none"> TIBCO Enterprise Message Service extension to JMS acknowledge modes. This is the default. EXPLICIT_CLIENT_ACKNOWLEDGE is like CLIENT_ACKNOWLEDGE except it acknowledges only the individual message, rather than all messages received so far on the session. One example of when EXPLICIT_CLIENT_ACKNOWLEDGE is used when receiving messages and putting the information in a database. If the database insert operation is slow, use multiple application threads all doing simultaneous inserts. As each thread finishes its insert, it can use EXPLICIT_CLIENT_ACKNOWLEDGE to acknowledge only the message that it is currently working on.
24	EXPLICIT_CLIENT_DUPS_OK_ACKNOWLEDGE (TIBCO Proprietary)	<ul style="list-style-type: none"> TIBCO Enterprise Message Service extension to JMS acknowledge modes. EXPLICIT_CLIENT_DUPS_OK_ACKNOWLEDGE mode is similar to TIBEMS-DUPS-OK-ACKNOWLEDGE except it "lazily" acknowledges only the individual message, rather than all messages received so far on the session.



- The standard acknowledgement modes (1-AUTO_ACKNOWLEDGE, 2-CLIENT_ACKNOWLEDGE, 3-DUPS_OK_ACKNOWLEDGE) are supported by all the JMS providers such as IBM WebSphere MQ.
- The extended acknowledgment modes (22-NO_ACKNOWLEDGE, 23-EXPLICIT_CLIENT_ACKNOWLEDGE, 24-EXPLICIT_CLIENT_DUPS_OK_ACKNOWLEDGE) are provided as an extension only by TIBCO Enterprise Message Service. These modes are not supported by other JMS providers such as IBM WebSphere MQ.
- If you are using TIBCO Enterprise Message Service for JMS transport, use the extended acknowledgment modes (22-NO_ACKNOWLEDGE, 23-EXPLICIT_CLIENT_ACKNOWLEDGE, 24-EXPLICIT_CLIENT_DUPS_OK_ACKNOWLEDGE) for performance and tuning purposes.

Non-Standard JMS Headers

You can set the JMS properties using the non-standard header fields in the request message. The HTTP header fields such as Host, Content-Type, and Content-Length of the request message can be set as the JMS properties. TIBCO API Exchange Gateway forwards the request message containing the JMS properties to the target operation. The header fields can be set using a transformation (XSLT) file.

Setting up JMS Properties

Set up the JMS properties using the following steps:

Procedure

1. Create an XSLT File. Refer to Example XSLT.
2. Copy the XSLT file to the *ASG_CONFIG_HOME/ASG_Project/xslt/internal* directory.
3. Start the Config UI.
4. Create a new mapping as follows:
 - a) Select the gateway configuration project.
 - b) Click the **MAPPING > Mapping** tab.
 - c) Click the **Add Property** icon to create a new mapping.
 - d) Under **New Mapping**, add the parameters as follows:
 - **Mapping Name:** Enter a name for the mapping.
 - **Type:** select **XLST** from the drop-down list.
 - **Existing Files:** select the newly created XLST file from the drop-down list.
 - e) Click the **Save** icon to save the changes.
5. Upload the XSLT file for the facade operation as follows:
 - a) Click the **ROUTING** tab.
 - b) Click the **Facade Operations** tab.
 - c) Select the facade operation for which the headers are to be set.
 - d) Click on **Request Transform** field. Select the newly added mapping from the drop-down list.
6. Click the **Save** icon to save the changes.

Example XSLT

Refer to the following XSLT:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet
  version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:map="http://www.tibco.com/asg/mapping"
  xmlns:soap12="http://www.w3.org/2003/05/soap-envelope"
  xmlns:soap11="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:form="http://www.tibco.com/asg/functions/form"
  xmlns:c="http://www.tibco.com/schemas/asg/context"
  xmlns:h="http://www.tibco.com/asg/protocols/http"
  xmlns:k="http://www.tibco.com/asg/protocols/jms"
  xmlns:f="http://www.tibco.com/asg/content-types/form"
  xmlns:codecs="http://www.tibco.com/asg/functions/codecs"
  exclude-result-prefixes="xsl soap11 c h form codecs"
>
<xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes" omit-xml-
declaration="no"/>
<xsl:variable name="cnRequestHref">
  <xsl:value-of select="/transformation/cnRequest/@href"/>
</xsl:variable>
<xsl:variable name="context">
  <c:context>
    <xsl:for-each select="/transformation/context">
      <xsl:copy-of select="document(@href)/c:context/*"/>
    </xsl:for-each>
  </c:context>
</xsl:variable>
<xsl:variable name="recdRequest">
  <xsl:copy-of select="$context/c:context/c:entry[@key='asg:jmsRequest']/
k:request"/>
</xsl:variable>
<xsl:variable name="reqBody">
<xsl:choose xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <xsl:when test="count(document($cnRequestHref)/soap:Envelope/
soap:Body)=1">
    <xsl:copy-of select="document($cnRequestHref)"/>
  </xsl:when>
  <xsl:otherwise>
    <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/
envelope/" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://
www.w3.org/2001/XMLSchema-instance">
      <soapenv:Header>
      </soapenv:Header>
      <soapenv:Body>
        <xsl:copy-of select="document($cnRequestHref)"/>
      </soapenv:Body>
    </soapenv:Envelope>
  </xsl:otherwise>
</xsl:choose>
</xsl:variable>
<xsl:template match="/">
  <map:mapping-result>
    <map:failOnError>false</map:failOnError>
    <map:context>
      <c:context>
        <c:entry key="asg:jmsRequest">
          <k:override-header name="httpheadertest">identity</k:override-header>
        </c:entry>
      </c:context>
    </map:context>
    <map:payload-xml>
      <xsl:copy-of select="$reqBody"/>
    </map:payload-xml>
  </map:mapping-result>
</xsl:template>
</xsl:stylesheet>
```

ESB Channel

Overview of ESB channel.

TIBCO API Exchange Gateway 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.

Enabling ESB Channels

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

To enable the JMS channels, follow these steps:

Procedure

1. Open the *ASG_HOME/bin/asg_core.cdd* file for editing in a text editor.

2. Search the **be.channel.deactivate** 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,/ASG/Channels/
modAS_Channel,/Common/Channel/PSMChannel,/Common/Channel/CentralLoggerJMS"/>
```

3. If you have not taken the backup of the **be.channel.deactivate** property, copy the **be.channel.deactivate** property to **be.channel.deactivate.backup**, as follows:

```
<property name="be.channel.deactivate.backup" 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,/ASG/Channels/modAS_Channel,/Common/Channel/PSMChannel,/
Common/Channel/CentralLoggerJMS"/>
```

4. Enable the ESB channels as follows:

- a) To enable the northbound ESB channel, remove */DefaultImplementation/Channels/North_ESBChannel* from the value of the **be.channel.deactivate** property. The modified value of the property is as follows:

```
<property name="be.channel.deactivate" value="/DefaultImplementation/Channels/
SouthboundEsb0Channel,/DefaultImplementation/Channels/SouthboundEsb1Channel,/
DefaultImplementation/Channels/SouthboundEsb2Channel,/DefaultImplementation/
Channels/North_HTTPChannel,/ASG/Channels/SOAPJMSChannel_North,/ASG/Channels/
SOAPJMSChannel_South,/ASG/Channels/modAS_Channel,/Common/Channel/PSMChannel,/
Common/Channel/CentralLoggerJMS"/>
```

- b) To enable the southbound ESB channels, remove the */DefaultImplementation/Channels/SouthboundEsb0Channel,/DefaultImplementation/Channels/SouthboundEsb1Channel,/DefaultImplementation/Channels/SouthboundEsb2Channel* from the value of the **be.channel.deactivate** property. The modified value of the property is as follows:

```
<property name="be.channel.deactivate" value="/DefaultImplementation/Channels/
North_ESBChannel,/DefaultImplementation/Channels/North_HTTPChannel,/ASG/
```

```
Channels/SOAPJMSChannel_North,/ASG/Channels/SOAPJMSChannel_South,/ASG/Channels/
modAS_Channel,/Common/Channel/PSMChannel,/Common/Channel/CentralLoggerJMS"/>
```

5. Save the changes to the file.

Edit asg.properties File for ESB Channel Properties

The `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

List of properties for ESB channel at facade and target side.

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

JMS Server Connection Parameters at Facade Side

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

JMS Server Connection Parameters (facade)

Property Name	Default Value
tibco.clientVar.Common/Connections/JMS/FacadeEsbConnection0/JMSProviderURL	tcp://localhost:7222
tibco.clientVar.Common/Connections/JMS/FacadeEsbConnection0/JNDIContextURL	tibjmsnaming://localhost:7222
tibco.clientVar.Common/Connections/JMS/FacadeEsbConnection0/TopicConnectionFactoryName	TopicConnectionFactory
tibco.clientVar.Common/Connections/JMS/FacadeEsbConnection0/QueueConnectionFactoryName	QueueConnectionFactory
tibco.clientVar.Common/Connections/JMS/FacadeEsbConnection0/JNDIUsername	admin
tibco.clientVar.Common/Connections/JMS/FacadeEsbConnection0/JNDIPassword	
tibco.clientVar.Common/Connections/JMS/FacadeEsbConnection0/JMSUsername	admin
tibco.clientVar.Common/Connections/JMS/FacadeEsbConnection0/JMSPassword	

JMS Server Connection Parameters At Target Side

The 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 per your JMS server settings.

JMS Server Connection Parameters (Target)

Property Name	Default Value
tibco.clientVar.Common/Connections/JMS/TargetEsbConnection0/JMSProviderURL	tcp://localhost:7222
tibco.clientVar.Common/Connections/JMS/SouthboundEsbConnection0/JNDIContextURL	tibjmsnaming://localhost:7222
tibco.clientVar.Common/Connections/JMS/TargetEsbConnection0/TopicConnectionFactoryName	TopicConnectionFactory
tibco.clientVar.Common/Connections/JMS/TargetEsbConnection0/QueueConnectionFactoryName	QueueConnectionFactory
tibco.clientVar.Common/Connections/JMS/TargetEsbConnection0/JNDIUsername	admin
tibco.clientVar.Common/Connections/JMS/TargetEsbConnection0/JNDIPassword	
tibco.clientVar.Common/Connections/JMS/TargetEsbConnection0/JMSUsername	admin
tibco.clientVar.Common/Connections/JMS/TargetEsbConnection0/JMSPassword	
tibco.clientVar.Common/Connections/JMS/TargetEsbConnection1/JMSProviderURL	tcp://localhost:7222
tibco.clientVar.Common/Connections/JMS/TargetEsbConnection1/JNDIContextURL	tibjmsnaming://localhost:7222
tibco.clientVar.Common/Connections/JMS/TargetEsbConnection1/TopicConnectionFactoryName	TopicConnectionFactory
tibco.clientVar.Common/Connections/JMS/TargetEsbConnection1/QueueConnectionFactoryName	QueueConnectionFactory
tibco.clientVar.Common/Connections/JMS/TargetEsbConnection1/JNDIUsername	admin
tibco.clientVar.Common/Connections/JMS/TargetEsbConnection1/JNDIPassword	
tibco.clientVar.Common/Connections/JMS/TargetEsbConnection1/JMSUsername	admin
tibco.clientVar.Common/Connections/JMS/TargetEsbConnection1/JMSPassword	
tibco.clientVar.Common/Connections/JMS/TargetEsbConnection2/JMSProviderURL	tcp://localhost:7222

Property Name	Default Value
tibco.clientVar.Common/Connections/JMS/TargetEsbConnection2/JNDIContextURL	tibjmsnaming://localhost:7222
tibco.clientVar.Common/Connections/JMS/TargetEsbConnection2/TopicConnectionFactoryName	TopicConnectionFactory
tibco.clientVar.Common/Connections/JMS/TargetEsbConnection2/QueueConnectionFactoryName	QueueConnectionFactory
tibco.clientVar.Common/Connections/JMS/TargetEsbConnection2/JNDIUsername	admin
tibco.clientVar.Common/Connections/JMS/TargetEsbConnection2/JNDIPassword	
tibco.clientVar.Common/Connections/JMS/TargetEsbConnection2/JMSUsername	admin
tibco.clientVar.Common/Connections/JMS/TargetEsbConnection2/JMSPassword	

Facade Queue Names For ESB Transport

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

Facade Queue Names (ESB)

Global Variable	Default Value
tibco.clientVar.ASG/facade/ESB0/requestQueue	asg.in.request
tibco.clientVar.ASG/facade/ESB0/replyQueue	asg.in.request.reply.0

Target Queue Names For ESB Transport

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

Target Queue Names (ESB)

Global Variable	Default Value
tibco.clientVar.ASG/Endpoint/ESB0/requestQueue	asg.out.request
tibco.clientVar.ASG/Endpoint/ESB0/replyQueue	asg.out.request.reply.0.0
tibco.clientVar.ASG/Endpoint/ESB1/requestQueue	asg.out.request
tibco.clientVar.ASG/Endpoint/ESB1/replyQueue	asg.out.request.reply.0.1

Global Variable	Default Value
tibco.clientVar.ASG/Endpoint/ESB2/requestQueue	asg.out.request
tibco.clientVar.ASG/Endpoint/ESB2/replyQueue	asg.out.request.reply.0.2

Create queues on EMS Server

By default, TIBCO API Exchange Gateway 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](#) and [Target Queue Names For ESB Transport](#), you should create the queue names defined by the global variables.
- You can edit the global variables values as follows:
 - Using the Config UI. See [Set Runtime Properties](#) for details.
 - Editing the `ASG_CONFIG_HOME/asg.properties` file using a text editor.

Create Users On EMS Server

Ask the EMS server administrator to create a normal user without administrator privileges. TIBCO API Exchange Gateway 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](#).

Config UI

TIBCO API Exchange Gateway provides the graphical user interface (GUI). Using the Config UI, you can add a new gateway configuration for partners, facade operations, target operations, routing, throttles, mapping, schemas, and other configuration data required for various functions of TIBCO API Exchange Gateway.

Using the Config UI, you can manage the entire 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, and routing which is required by the Core Engine at run time.

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:\tibcoASGConfig\tibco\cfgmgmt\asg\default`

A project configuration folder contains the following types of files:

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

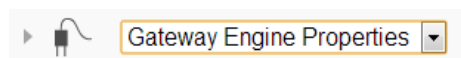


When the TIBCO API Exchange Gateway configuration server is started on a machine for the first time after the product installation, the ASGConfig.war archive is extracted under the `ASG_HOME/webapp` directory. You can notice few errors if you kill the configuration server before the complete extraction of the archive file.

The home page of the Config UI has the following main sections:

- Global Properties

Select `Gateway Engine Properties` from the drop-down list to set the run time properties for the Core Engine as follows:



See [Set Runtime Properties](#).

- Projects

You can manage a project configuration. See [Manage a Gateway Project Configuration](#).

- Advanced Settings

You can add the settings for an environment.

Starting GUI

How to start the Config UI.

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

Procedure

1. Open a window.
2. Navigate to the *ASG_HOME/bin* directory.
3. Start the Config UI as follows:

On the Windows platform, type the following command on a command prompt:

```
asg-configui.exe
```

On the UNIX platform, type the following command on a terminal window:

```
./asg-configui
```

4. Open a browser window and enter the following URL:

```
http://localhost:9200/ConfigUI
```



To access the Config UI using the HTTPs transport, see [Accessing Config UI through HTTPs Transport](#).

5. Enter the login information as follows:

```
username:admin
```

```
password:admin
```

6. Ensure that the user logs on the Config UI successfully.
7. Ensure that the following default project configurations are displayed under **Projects** panel:
 - default
 - GetLocation
 - BookQuery
 - BookQueryWSS
 - Caching
 - BookQueryBE
 - BookQueryBEWSS
 - APIExchange
 - BookQuerySecurity
 - ProductQuery

Accessing Config UI through HTTPS Transport

Procedure

1. Stop the GUI server (if running).
2. Open a command prompt window.
3. Navigate to the *ASG_HOME/bin* directory.
4. Make a copy of the *asg-configui.tra* file.
5. Using a text editor, edit the *asg-configui.tra* file. To enable SSL, set the following properties:

```
java.property.com.tibco.asg.designtime.configui.launcher.secure=true
java.property.com.tibco.asg.designtime.configui.launcher.SSLEnabled=true
```



- To enable SSL, both properties should be set to `true`.
- To disable SSL, both properties should be set to `false`.
- TIBCO API Exchange Gateway supports one-way and two-way (mutual) SSL to access Config UI using HTTPs transport.

6. Set the keystore configuration properties as explained in the [SSL Properties for HTTPs Transport](#) table.
7. Save changes to the file.

SSL Properties for Config UI

List of SSL properties for Config UI.

The following table explains the SSL properties to specify the keystore configuration for using the HTTPs transport:

SSL Properties for HTTPs Transport

Property	Description
<code>java.property.com.tibco.asg.designtime.configui.launcher.secure</code>	To enable SSL connection, set this property to <code>true</code> .
<code>java.property.com.tibco.asg.designtime.configui.launcher.SSLEnabled</code>	Enables the SSL communication. Set this property to <code>true</code> to use HTTPs transport. The default value is <code>false</code> . If this property is set to <code>true</code> , it requires the scheme and secure properties to be set correctly for HTTPs transport.
<code>java.property.com.tibco.asg.designtime.configui.launcher.scheme</code>	Set this property to the name of the protocol for SSL. Set this to <code>https</code> .
<code>java.property.com.tibco.asg.designtime.configui.launcher.keyAlias</code>	The alias used to specify the server certificate in the keystore. If not specified the first key read in the keystore is used.
<code>java.property.com.tibco.asg.designtime.configui.launcher.keystoreFile</code>	The path to the keystore file. The keystore file contains the server certificates. The JKS format keystore type is supported. For example, <code>C:/tibco/keystores/config.jks</code> By default, the path name is the <code>.keystore</code> file. The file is stored in the home directory of the user who is running Tomcat server.
<code>java.property.com.tibco.asg.designtime.configui.launcher.keystoreType</code>	Specifies the keystore type of the private credentials. Supported formats are JKS, PKCS12. For example, JKS.

Property	Description
<code>java.property.com.tibco.asg.designtime.configui.launcher.keystorePass</code>	The password used to access the server key from the specified keystore file.
<code>java.property.com.tibco.asg.designtime.configui.launcher.sslProtocol</code>	<p>The version of the SSL protocol to use. If not specified, the default value is TLS.</p> <p>The following SSL protocols are supported:</p> <p>SSL,SSLv3,TLSv1,TLSv1.1, TLSv1.2</p>
<code>java.property.com.tibco.asg.designtime.configui.launcher.server</code>	<p>Overrides the server header information for the http response.</p> <p>If this property is set, the value for this attribute overrides the default header of Tomcat or any server set by a web application. For example, Apache.</p> <p>If this property is not set, any value specified by the application is used. If the application does not specify a value, Apache-Coyote/1.1 is used by default.</p>
Mutual SSL Properties Set the following properties only for two-way (mutual) SSL authentication.	
<code>java.property.com.tibco.asg.designtime.configui.launcher.clientAuth</code>	<p>Specifies a boolean flag to enable or disable the mutual(two-way) SSL authentication for HTTPS transport between the client and the Config UI.</p> <p>When this field is set to true, set the <code>java.property.com.tibco.asg.designtime.configui.launcher.truststoreFile</code> property to specify a file containing the list of trusted certificate authorities.</p>
<code>java.property.com.tibco.asg.designtime.configui.launcher.truststoreFile</code>	<p>Specifies a file containing one or more certificates from trusted certificate authorities, which is required for mutual SSL authentication. You must set this property when the <code>java.property.com.tibco.asg.designtime.configui.launcher.clientAuth</code> property is set to true.</p>
<code>java.property.com.tibco.asg.designtime.configui.launcher.truststorePass</code>	<p>Specifies the password to access the certificate file defined by the <code>java.property.com.tibco.asg.designtime.configui.launcher.truststoreFile</code> property.</p>

Property	Description
java.property.com.tibco.asg.designtime.configui.i.launcher.truststoreType	Specifies the type of trusted store file. Supported formats are JKS, PKCS12.

Changing Login Host and Port Information

You can change the login information for Config UI.

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 follows:

```
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 these steps:

Procedure

1. Open a terminal window.
2. Navigate to the *ASG_HOME/bin* directory.
3. Make a copy of the **asg-configui.tra** file.
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 the **asg-configui** executable.

For example,

On the Windows, run the *asg-configui.exe*.

On the UNIX, type the following command:

```
./asg-configui
```



For production systems, it is good practice to deploy the *ASGConfig.war* file on a secure application server running in the production environment. The war file is located under the *ASG_HOME/webapp* directory.

Authentication Process for Config UI

Introduction to authentication process for Config UI.

Using the login screen of the Config UI, you can enter username and password for authentication. TIBCO API Exchange Gateway supports the authentication of users on the LDAP server or in a file. This functionality enables you to do the following:

- Create multiple users to login to the Config UI.
- Define the timeout value for an active session of the user.

- The option to logout after the user logs to the Config UI.

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

Authentication Properties

List of Authentication properties used by Config UI.

The following table explains the properties to enable the LDAP or FILE-based user authentication for the Config UI.

Authentication Properties

Property	Description
asg-configui.tra File Properties	
<code>java.property.com.tibco.asg.designtime.configui.ASGAuthProcess</code>	<p>Specifies the type of the authentication process required to login to the Config UI.</p> <p>The possible values are: LDAP or FILE.</p> <ul style="list-style-type: none"> • 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 <code>tibco.env.ASG_ADMIN_USERNAME</code> and <code>tibco.env.ASG_ADMIN_PASSWORD</code> properties of <code>asg-configui.tra</code> file. See Default Authentication for details. • The value of this property is required if you want to authenticate the user on a LDAP server or in a FILE.

Property	Description
<code>java.property.com.tibco.asg.designtime.configui.ASGPropFile</code>	<p>Specifies the path to the property file for the authentication process.</p> <ul style="list-style-type: none"> Value of this property is required if the <code>java.property.com.tibco.asg.designtime.configui.ASGAuthProcess</code> 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: <code>ASG_HOME/bin/ldapSearch.properties</code> where <code>ASG_HOME</code> is set to the directory where TIBCO 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: <code>ASG_HOME/mm/config/users.pwd</code> where <code>ASG_HOME</code> is set to the directory where the TIBCO API Exchange Gateway is installed. <p>See Authentication Property Files for details.</p>
<code>java.property.com.tibco.asg.designtime.configui.ASGSessionTimeout</code>	<p>Specifies the timeout value (in minutes) for a session.</p> <p>Set as Integer value. The session does not time out if the value is set to a negative value.</p> <p>Required.</p>
web.xml file properties	

Property	Description
asgAuthProcess	Same as the <code>java.property.com.tibco.asg.designtime.configui.ASGAuthProcess</code> property defined in <code>asg-configui.tra</code> file. See java.property.com.tibco.asg.designtime.configui.ASGAuthProcess .
asgAuthPropFile	Same as <code>java.property.com.tibco.asg.designtime.configui.ASGPropFile</code> property. See java.property.com.tibco.asg.designtime.configui.ASGPropFile .
session-timeout	Same as <code>java.property.com.tibco.asg.designtime.configui.ASGSessionTimeOut</code> . See java.property.com.tibco.asg.designtime.configui.ASGSessionTimeOut .

Configuration Setup for Authentication Process

This section explains the configuration setup required for the authentication process when the user logs in to the Config UI.

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

Enable LDAP server authentication for Config UI.

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

- To configure the `asg-configui.tra` file, see [Configuring asg-configui.tra File](#).
- To configure the `web.xml` file, see [Configuring web.xml File](#).

Configuring asg-configui.tra File

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

Procedure

1. Navigate to the `ASG_HOME` directory.
2. Open the `asg-configui.tra` file for editing.
3. Set the following properties. See [Authentication Properties](#) table for the description of properties.

```
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, the authentication property file and session timeout in the `web.xml` file, follow these steps:

Procedure

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](#) table for the description of properties.
 - a) Set the authentication process type to LDAP as follows:

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

- b) Set the property file for the LDAP authentication as follows:

```
<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 Config UI, as follows:

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

4. Save the changes to the file.

File-Based Authentication

Enable File-based server authentication for Config UI.

This method authenticates a user against the user data stored in a file-based repository. Do not use this method for production purposes.

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 the `asg-configui` file or `web.xml` file.

- To configure `asg-configui.tra` file, see [Configuring asg-configui.tra File\(FILE\)](#).
- To configure the `web.xml` file, see [Configuring web.xml File\(FILE\)](#)

Configuring asg-configui.tra File (FILE)

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

Procedure

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

3. Set the following properties. See [Authentication Properties](#) table for the description of properties.

```
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 (FILE)

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

Procedure

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](#) table for the description of properties.
 - a) Set the authentication process type to FILE as follows:

```
<init-param>
<param-name>asgAuthProcess</param-name>
<param-value>FILE</param-value>
</init-param>
```

- b) Set the property file for the FILE-based authentication as follows:

```
<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 Config UI as follows:

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

4. Save the changes to the file.



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

Authentication Property Files

List of property files used by Config UI for authentication.

Based on the authentication type, 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 an LDAP search property file. See [LDAP Server Property File for LDAP Server Authentication](#).

If you are using the FILE-based authentication, define a user credentials file. The user credentials file is a text file containing the usernames and passwords required to login to the Config UI. See [User Credentials File for File-Based Authentication](#).

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 the user. Define a property file for the LDAP server authentication.

Sample File

The sample file is located in the `ASG_HOME/bin/ldapSearch.properties`.

[LDAP Server Properties Sample File](#): shows the example properties to be defined in the property file for the LDAP server authentication.

LDAP Server Properties Sample File:

```
#LDAP search properties using trinity
com.tibco.trinity.runtime.core.provider.authn.ldap.initialCtxFactory=com.sun.jndi.ldap.LdapCtxFactory
com.tibco.trinity.runtime.core.provider.authn.ldap.serverURL=ldap://trinity.na.tibco.com:1389
com.tibco.trinity.runtime.core.provider.authn.ldap.searchTimeout=10000
com.tibco.trinity.runtime.core.provider.authn.ldap.userSearchExpression=(&(uid={0})(objectclass=*))
com.tibco.trinity.runtime.core.provider.authn.ldap.userDNTemplate=uid={0},ou=people,dc=example,dc=com
com.tibco.trinity.runtime.core.provider.authn.ldap.userAttributeUsersName=uid
com.tibco.trinity.runtime.core.provider.authn.ldap.groupSearchBaseDN=ou=groups,dc=example,dc=com
com.tibco.trinity.runtime.core.provider.authn.ldap.userSearchBaseDN=ou=people,dc=example,dc=com
com.tibco.trinity.runtime.core.provider.authn.ldap.groupSearchScopeSubtree=true
com.tibco.trinity.runtime.core.provider.authn.ldap.groupSearchExpression=uniquemember={0}
com.tibco.trinity.runtime.core.provider.authn.ldap.groupAttributeGroupName=cn
com.tibco.trinity.runtime.core.provider.authn.ldap.userIndicatesGroups=false
com.tibco.trinity.runtime.core.provider.authn.ldap.userAttributesExtra=mail givenname
com.tibco.trinity.runtime.core.provider.authn.ldap.followReferrals=true
```

User Credentials File for File-Based Authentication

In the file-based authentication, the credentials of the user are stored in a file. By default, the file is `users.pwd` file, which contains a list of user names, passwords, and roles. This file is referred as the password file. The user names, passwords, and roles are separated by colon (:) character and each pair must be present on a separate line. The user names are used to login the Config UI for file-based authentication.

The Config UI does not use the role for authentication process. The role is required only by the Management and Monitoring server. It is mandatory to define the role in the `users.pwd` file. Use `GUI_USER` as the role in the file when defining the user to log in the Config UI for file-based authentication.



The password in the `users.pwd` file must be generated using MD5 (Message-Digest 5) hashing algorithm. For example, refer to <http://www.md5hasher.net> to generate the password using MD5 algorithm.

Sample File

The sample file is found as follows:

`ASG_HOME/mm/config/users.pwd`

[Sample Property File for File Authentication](#) shows the example file for file-based authentication.

Sample Property File for File Authentication

```
jdoe:A31405D272B94E5D12E9A52A665D3BFE:MM_ADMINISTRATOR;
mm_user:11b2016b63c99ef7ab6d6d716be7b78e:MM_USER;
admin:21232f297a57a5a743894a0e4a801fc3:MM_ADMINISTRATOR;
```

Default Authentication

By default, TIBCO API Exchange Gateway does not authenticate the user from a file or an LDAP server to login to the Config UI.

When the authentication process property (`asgAuthProcess`) is blank, the Config UI 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
```

Perform the following, if you do not want the user authentication on an LDAP server or in a file.

Procedure

1. Open the *ASG_HOME/bin/asg-configui.tra* file for editing.
2. Set the following property to blank as follows:
`java.property.com.tibco.asg.designtime.configui.ASGAuthProcess=`
 (You can also specify the `asgAuthProcess` parameter as blank in the `web.xml` file.)
3. Set the following parameters to specify the username and password:

```
tibco.env.ASG_ADMIN_USERNAME
```

(Example, admin)

```
tibco.env.ASG_ADMIN_PASSWORD
```

(Example, admin 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 plain text or in TIBCO obfuscated form. The password can be obfuscated using the `asg-password-obfuscator` utility in the *ASG_HOME/bin* directory.



If the obfuscated password contains the special characters, you must escape the special characters of the obfuscated password.

For example,

```
tibco.env.ASG_ADMIN_PASSWORD=\#\!3nmDCAG\oN0vffUw\+H0DauShgrSDXpF1
```

See [asg-password-obfuscator Utility](#).

- When the authentication process property (`asgAuthProcess`) is blank, the Config UI allows only one user for the login as specified by the `tibco.env.ASG_ADMIN_USERNAME` parameter. If you want multiple users to login to the Config UI, you must use the LDAP or FILE based authentication.

Enable Debug Logging for Config UI

This section explains the steps to enable the debug level logging for the Config UI.

Creating Properties File

You must create a properties file for debug level logging.

Define a `log4j.properties` properties file as follows:

Procedure

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

Logging to stdout

Properties to send logging messages to the stdout.

To send the logging messages to the stdout, define the following properties in the `log4j.properties` file. To set the properties, complete the following steps:

Procedure

1. Open the `log4j.properties` file in a text editor.

2. Add the following properties:

```
log4j.rootLogger = DEBUG, Console, file
log4j.appender.Console=org.apache.log4j.ConsoleAppender
log4j.appender.Console.layout=org.apache.log4j.PatternLayout
log4j.appender.Console.layout.conversionPattern=%m%n
```

3. Save the changes to the file.

Logging to a File

Properties to send logging messages to a file.

To redirect the logging messages to a file instead of sending them to stdout, complete the following steps:

Procedure

1. Edit the `log4j.properties` file in a text editor.

2. Add the following properties:

```
log4j.rootLogger = DEBUG, Console, file
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 the changes to the file.

Using Properties File in the TRA File

Define properties file in the TRA file.

Use the `log4j.properties` in the TRA file of the Config UI after you have added the properties. To add the `log4j.properties` file in the TRA file, follow these steps:

Procedure

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 follows:

```
java.extended.properties=-server -Xmx256m -Dlog4j.debug -
Dlog4j.configuration=log4j.properties
```

4. Save the changes to the file.



You must restart the gateway configuration server to see the messages. See [Starting GUI](#).

Configuring Directory for Log Files

Property to set log files directory.

Procedure

1. Navigate to the `ASG_HOME/bin` directory.

2. Open the asg-configui.tra file for editing.
3. Edit the following property as follows:


```
java.property.com.tibco.asg.designtime.configui.logDir=path_to_log_directory
```
4. Save the changes to the file.

Manage a Gateway Project Configuration







Config UI is used to manage a project configuration of the gateway.

Using the Config UI, you can 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, facade operations, target operations, facade access, mappings, routing, and so on.

All the configuration files related to the partner data, facade access, facade operations, target operations, mappings, throttles, and so on, 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 the name `ASG_Get_Start`, the configuration server creates an `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, and so on.

To manage any project configuration, move the mouse pointer over the configuration and then select any of the options listed in the table below.

Manage Project Configuration

Action	Icon	Description
Add a new project configuration		You can add a new configuration for the gateway project.
Duplicate an existing configuration		You can copy an existing project configuration to a new project configuration.
Rename a project configuration		You can rename the project.
Validate an existing configuration		You can validate an existing configuration.
Publish a project configuration		You can publish the configuration changes to the Core Engine at run time.
Delete an existing configuration		You can delete an existing configuration.

Publish Project Configuration

The Publish project configuration enables you to send the configuration changes of a project to memory so that they are available to the Core Engine at run time without restarting the Core Engine.

You can publish and update the following changes at run time:

- Publish Configuration

Using the Publish Configuration option, you can publish any configuration changes saved in the configuration files to an active Core Engine instance. After you make any changes to the configuration of a project, you do not need to start the Core Engine instance, as the Core Engine instance picks all the changes as soon as you publish the configuration.

- Update Log Level

Using the Update Log Level option, you can dynamically change the levels of logging of the Core Engine at run time. For example, you can update the logging level from INFO to DEBUG at run time for the Core Engine.

- Update Central Log Level

Using the Update Central Log Level option, you can dynamically change the levels of logging of the Central Logger component at run time.

Publishing Configuration

You can publish the configuration changes using the Config UI.

Procedure

1. Start the Config UI. See [Starting GUI](#).
2. Move the mouse pointer on the gateway configuration project and select **Publish Project**

Configuration  icon.

3. In the dialog box wizard, set the values for the fields as follows:
 - Type: from the drop-down list, select **Publish Configuration**.
 - Gateway URL: Enter the URL of the Core Engine instance where the configuration changes are to be published. For example, `http://ASGServer:PortName`
where *ASGServer* is the server machine and *PortName* is the port number for a running Core Engine instance.
4. Click **Submit**.



- The Core Engine instance must be running with the same project configuration for which you published the configuration.
- The configuration changes are published only to a single engine instance locally. To update the configuration changes to multiple engine instances running remotely in a cluster environment, see [Updating Project Configuration](#).
- If you change the username and password to login to the Config UI, make sure to add the username and password in the `ASG_HOME/mm/config/users.pwd` file. See [User Credentials File for File-Based Authentication](#).

Change Log Level Settings

Using the Config UI, you can change the log level settings for the Core Engine and the Central Logger component.


See the following topics:

- [Changing Log Level Settings for Core Engine](#).
- [Changing Log Level Settings for Central Logger](#).

Changing Log Level Settings for Core Engine

To change the log level settings for the Core Engine, follow these steps:


Procedure

1. Start the Config UI. See [Starting GUI](#).
2. Move the mouse pointer on your gateway configuration project and select **Publish Project Configuration**  icon.
3. On the dialog box wizard, set the values for the fields as follows:
 - Type: from the drop-down list, select **Update Log Level**.
 - Gateway URL: Enter the URL of the Core Engine instance where the log level settings are to be applied. For example,
`http://ASGServerMachine:PortName`
 where *ASGServerMachine* is the server machine and *PortName* is the port number for a running the Core Engine instance.
 - Log Level: from the drop-down list, select the level of logging. For example, **DEBUG**.
 See [Logging Levels of Core Engine](#) for types of supported logging levels.
4. Click **Submit**.

Changing Log Level Settings for Central Logger

To change the log level settings for the Central Logger, follow these steps:

Procedure

1. Start the Config UI. See [Starting GUI](#).
2. Move the mouse pointer on the gateway configuration project and select **Publish Project Configuration**  icon.
3. On the dialog box wizard, set the values for the fields as follows:
 - Type: from the drop-down list, select **Update Central Logger Log Level**.
 - Gateway URL: Enter the URL of the Core Engine instance where the log level settings are to be applied. For example,
`http://ASGServerMachine:PortName`
 where *ASGServerMachine* is the server machine and *PortName* is the port number for a running Core Engine instance.
 - Log Level: Enter the level of logging from the drop-down list. For example, **Detail Logging ON**. See [Central Logger Log Level](#).
4. Click **Submit**.




- The Update log level settings for the Core Engine and the Central Logger are applicable only at run time.
- When the Core Engine is restarted, the original log level settings for the Core Engine and the Central Logger component are picked up from the `ASG_CONFIG_HOME/asg.properties` and `ASG_CONFIG_HOME/asg_cl.properties` files respectively.
 - For the Core Engine, the log level setting is defined by the `tibco.clientVar.ASG/Logging/MinLogLevel` property in the `ASG_CONFIG_HOME/asg.properties` file.
 - For the Central Logger component, the log level setting is defined by the `tibco.clientVar.ASG/Logging/clLogLevel` property in the `ASG_CONFIG_HOME/asg_cl.properties` file.

Updating Project Configuration

Using the Update project configuration, you can update the configuration changes to the multiple Core Engine instances running in a cluster with a project configuration without restarting any of the Core Engine instances. When you update the configuration changes to the multiple Core Engine instances, the same project configuration is used by all the instances at run time.

To update a project configuration, follow these steps:

Procedure

1. Start the Config UI. See [Starting GUI](#).
2. On the home page, go to **Advanced Settings** section. Select the environment and cluster for which you want to update the configuration.
3. Navigate to the **Projects** section to select the project configuration. Move the mouse pointer on the gateway configuration project and select the **Update Project Configuration**  icon.
4. In the dialog box wizard, set the values for the fields as follows:
 - Type: from the drop-down list, select **Cluster**.
5. Click **Submit**.



- If you make any configuration changes in a gateway project, the changes are applied to all the Core Engine instances running in a cluster with that gateway project.
- If you select **Gateway** type for **Update Project Configuration**, you can update the configuration for a single Core Engine instance running on a remote machine.

Validate Configuration

The Validate Configuration icon  on the Config UI 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](#).

Project Configuration

A project configuration consists of all information required by the Core Engine.

Config UI groups the configuration for any project into following categories:

- **MAPPING**

Using the **MAPPING** tab, you can enter the configuration data required by the Core Engine for mapping of the request and response documents. See [MAPPING](#) for details.

- **SECURITY**

Using the **SECURITY** tab, you can configure the data related to the security of the facade and target operations such as configuring policies, keystores, and so on. See [SECURITY](#) for details.

- **MONITORING**

Using the **MONITORING** tab, you can enter the data related to the monitors to enforce the throttle policy, KPI groups required for the reporting purposes. See [MONITORING](#) for details.

- **ROUTING**

Using the **ROUTING** tab, you can enter the configuration data for facade and target operations. You can configure the routing data required by the Core Engine to route any facade request to a target operation. See [ROUTING](#) for details.

- **PARTNER**

Using the **PARTNER** tab, you can enter the configuration data for the partner, partner groups, and facade operation authorization data for the partners. See [PARTNER](#) for details.

MAPPING

The MAPPING tab of the Config UI provides the configuration parameters for the following areas:


Mapping

Mapping Configuration Parameters

Using the **Mapping** tab, you can register the transformation (XSLT) files with the Core Engine.

To add a new mapping, follow these steps:

Procedure

1. Click the **Mapping** tab.
2. Click the **Add property**  icon to create a new mapping.
3. Enter the following mapping configuration parameters:

Mapping Configuration Parameters

Parameter	Description
Mapping Configuration	
Type	<p>Select the type of mapping from the drop-down list.</p> <p>The possible values are:</p> <ul style="list-style-type: none"> • RV • XSLT
RV Type	


Parameter	Description
Subject	Specifies the RV subject to send the mapping request to.
Transformation Name	Specifies the name of the transformation to perform on the message.
XSLT Type	
New File	The location of the transformation file.
Existing Files	The existing transformation file.
Response Type	<p>Specifies the response type. Select one of the possible values as follows:</p> <ul style="list-style-type: none"> • Payload • Full <p>See Transformations (XSLT Mapping)</p>

Schemas

Using the **Schemas** tab, you can 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 these steps:

Procedure

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

Schemas Configuration Parameters

Parameter	Description
Schema Key	Specifies the unique ID of the XSD schema file.
New XSD File	Specifies the location of the XSD file.
Existing XSD File	Specifies the existing XSD file.

Error Maps

Using the **ErrorMaps** tab, you can define all the error messages supported by TIBCO API Exchange Gateway.

To add a new errormap, follow these steps:

Procedure

1. Click the **ErrorMaps** tab.
2. Click the **Add property**  icon to create a new error map.
3. Enter the details for Error Maps configuration, as follows:

ErrorMaps Configuration Parameters

Parameter	Description
Error Id	The External ID of the ErrorMap. Must be globally unique.
Status	Specifies the status of the transaction reported by the gateway error message.
Component	Specifies the component of the gateway for this error message.
Error Description	Specifies the description of the error.
Category	Specifies the category of the error. The allowed values are: <ul style="list-style-type: none"> • <code>ServiceException(SVC:)</code> <code>PolicyException(POL:)</code>
Fault Code	Specifies the SOAP fault code. Usually, the value is Client.
Fault String	Specifies the SOAP fault string. For example, SVC0002
Fault Actor	Specifies the SOAP Fault Actor.
Message Id	Specifies the message ID or any client specific code. For example, SVC0001
Text	Specifies the error text with locations for variables to be embedded indicated by "%1", "%2", and so on.
Variables	A list of strings separated by a comma (,) that the gateway client uses for token replacement in the text.

SECURITY

The **SECURITY** tab of the Config UI provides the configuration parameters.

WSS

The **WSS** tab on the Config UI allows you to register the WSS resources with the gateway.

To add a new WSS resource, follow these steps:

Procedure

1. Click the **WSS** tab.

2. Click the **Add property**  icon to create a new WSS resource.
3. Enter the details for WSS resource parameters, as follows:

WSS Resource Configuration Parameters


Parameter	Description
WSS Name	The unique name which identifies a WSS configuration.
Type	The type of WSS configuration. Select the type from the drop-down list. The possible values are: <ul style="list-style-type: none"> • WSS • Subject Identity • Trust Identity
New Property File	A new property file which defines the WSS resources configuration. See Define the WSS Configuration Properties file .
Existing Property Files	Select an existing WSS resources configuration property file. The file must exist in the <code>ASG_CONFIG_HOME/ASG_Project_Configuration/wss</code> directory.

KeyStores

Using the **KeyStores** tab on the Config UI, you can upload the keystore configuration.

To add a new keystore file, follow these steps:

Procedure

1. Click the **KeyStores** tab.
2. Click the **Add property**  icon to upload a new keystore file.
3. Enter the details for the keystores file, as follows:

KeyStore Configuration Parameters

Parameter	Description
New KeyStores File	Specifies the security certificate files such as .jks type files.
Existing KeyStores Files	Specifies the existing security certificate files. The files exist in the <code>ASG_CONFIG_HOME/ASG_Project/security/keystore</code>



For more information on self-signed keystores, refer to:

<http://docs.oracle.com/javase/7/docs/technotes/tools/windows/keytool.html>


For an example of how to create self-signed keystores, refer to the readme included in the `ASG_CONFIG_HOME/ASG_Project/security/keystore` directory.

Policy Mapping

Using the **Policy Mapping** tab on the Config UI, you can upload a policy file.

To upload a new policy file, follow these steps:

Procedure

1. Click the **Policy Mapping** tab.
2. Click the **Add property**  icon to upload a new policy file.
3. Enter the details for the policy file, as follows:

Policy Mapping Configuration Parameters


Parameter	Description
Policy Name	Specifies the name for the policy.
Intent(Type)	Sets the type of the policy. For example, Authentication. See Security Policies for details.
Qualifier (SubType)	Sets the policy sub type. For example, UsernameToken. See Security Policies for details.
New Policy File	Specifies the policy definition file. Browse to choose a new policy file. See Security Policies for details.
Existing Policy File	Specifies an existing policy definition file. The policy file must exist in the gateway <code>ASG_CONFIG_HOME/configuration/policy</code> folder. For example, for the default configuration, the policy file must exist in the <code>ASG_CONFIG_HOME/default/policy</code> folder.

Policy Binding

Using the **Policy Binding** tab on the Config UI, you can associate a registered policy with one or more target operation endpoints.



To map a policy to a target operation, follow these steps:

Procedure

1. Click the **Policy Binding** tab.
2. Click the **Add property**  icon to map a policy file.

3. Enter the details for policy mapping configuration, as follows:

Policy Mapping Configuration Parameters

Parameter	Description
Policy	Specifies a name for the policy. The policy name must be configured under the Policy Mapping tab.
URI	Specifies the URI of the operation to which the policy is applied.  If the URI in facade operation is left blank, then URI field for security binding should use SOAP action instead of URI.
Facade Operation	Specifies the operation to which the policy is applied. The facade operation must be configured in the Facade Operation tab.
Target Operation	Specifies the target operation. The target operation must be configured in the Target Operation tab.
Binding	Specifies the binding component that the policy is applied to. This could be either a facade operation (service) or a target operation (reference).
Flow	Specifies the flow of the request or response. The possible values are: <ul style="list-style-type: none"> • in • out
Partner	Specifies the partner to which the policy is applied. This field can be left blank in case no partner needs to be applied.
Type	Specifies the type of the request.  Set this to SOAP for any SOAP request, or to http for any non-SOAP http request.


MONITORING

The **MONITORING** tab of the Config UI provides the configuration parameters.

Monitors

The **Monitors** tab enables you to define different types of throttles with different throttle metrics.

Procedure

1. Click the **Monitors** tab.
2. Click the **Add property**  icon to create a new monitor.
3. Enter the parameters for monitors, as follows:

Monitors Configuration Parameters

Parameter	Description
Throttle Parameters	
Monitor Name	The logical name for the monitor. This is the value used in the configuration to identify the monitor.
Monitor Type	<p>The type of monitor. The possible values for monitors are as follows:</p> <ul style="list-style-type: none"> Rate Quota High Water Mark Error
Interval	<p>The time interval during which the monitor is applied. It has different time units depending on the monitor type.</p> <p>For example,</p> <ul style="list-style-type: none"> Rate and Error throttle types specify the interval in seconds. Quota throttle type specify the interval in hours. High Water Mark throttle type does not have any time interval.
Max Limit	The number of requests allowed during the interval. This number should be a positive integer greater than zero.
Monitor Time Modifiers	<p>Specifies the time modifiers to apply to the monitor. Multiple time modifiers are allowed.</p> <p>This has the following parameters:</p> <ul style="list-style-type: none"> Max Count - Monitor limit to apply if the 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 the 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.

Parameter	Description
Monitor Count	<ul style="list-style-type: none"> Specifies the type of the counter used to increment the count of the used monitor. The possible values are: <ul style="list-style-type: none"> Request Count Input Payload Size Output Payload Size Transaction Payload Size The Monitor count is not applicable for Error and High Water Mark monitor types. <p>See Throttle Counter.</p>
Use Approximate Monitor	<p>Indicates if the maximum number of requests as specified by the <code>Max Limit</code> property is distributed by the number of running engine instances.</p> <p>If this value is set to <code>true</code>, the maximum number of requests is distributed by the number of active engines.</p> <p>If the value is set to <code>false</code>, the maximum number of requests is shared by the number of engines. You must set the Max Count Ratio parameter if the value of Use Approximate Monitor is set to <code>false</code>.</p>
Max Count Ratio	An integer value indicating the percentage of maximum number of requests allowed by instance of an engine locally.

KPI Groups

Using **KPI Groups** tab, you can 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 the following parameters:

Parameter	Description
KPI Groups Configuration	
KPI Group Name	<p>External ID of the KPI_Group. Set a unique name for this field. For example, <code>oneMinute</code>.</p> <p>Required.</p>
Interval (ms)	<p>Interval in milliseconds. The KPI data is written to the output file after this interval is expired.</p> <p>Required.</p>

Parameter	Description
Prefix	Specifies the logical name of the KPI. This value is stored in the KPI_FREQUENCY column of the ASG_KPI database table of the Central Logger component.

ROUTING

The **ROUTING** tab of the Config UI provides the configuration parameters.

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 Core Engine. Using the **Facade Operations** tab, you can add the list of operations supported by the TIBCO API Exchange Gateway.


Adding a New Facade Operation

To add a new facade operation, follow these steps:

Procedure

1. Click the **Facade Operations** tab.
2. Click the **Add property**  icon to create a new facade operation.
3. Enter the details for the operation, as follows:

Facade Operation Configuration Parameters

Parameter	Description
Operation Name	The logical operation name.
SOAP Action	The SOAP Action for this operation. This field is used to identify an operation to which an incoming request is applied to. This must be unique.
Operation URI	<div> The URI for this operation.  Refer to the list of special characters that are supported and not supported in Facade Operation URI. </div>
Operation Service Name	The logical service name (used for routing).
New ProcessBody Transform	The new XSLT transformation sheet file containing the rules to parse and validate the message. Optional.

Parameter	Description
ProcessBody Transform	The existing XSLT transformation sheet file containing the rules to parse and validate the message. Optional.
New FaultReport XSLT	The new XSLT transformation used to produce the fault message for the provided fault data.
Existing FaultReport XSLT	The existing XSLT transformation used to produce the fault message for the provided fault data. Optional.
Request Transform	The reference to forward the facade operation mapping. This mapping transforms from requestor API to a canonical request format. If no mapping reference needed, select Pass-Through from the drop-down list. Required.
Response Transform	The reference to reverse northbound mapping. This mapping transforms from a canonical request format to requestor API. If no mapping reference needed, select the value as Pass-Through from the drop-down list. Required.
Operation Method	The HTTP method used to separate REST requests which are made on the same URI but with different operations. Optional.
Operation Features	The list of keywords identifying the features required by the operation. The supported features are as follows: <ul style="list-style-type: none"> • Validation – The XSD validates northbound request and response. See Validation. • Shaping - See Traffic Shaping. • Proxy - Enables TIBCO API Exchange Gateway to act as Proxy Server. See Proxy Server. • CacheEnabled - Enables the caching of response messages. See Response Caching. • AntiVirusCheckEnabledOnRequest - Enables the antivirus scan on the request payload of a facade operation. • AntiVirusCheckEnabledOnResponse- Enables the antivirus scan on the response payload of a facade operation • Optional.
Status Code on Error	Specifies the HTTP status code to return if an error occurs in request processing. This is an optional field.

Parameter	Description
Enable WSS	This check box flag enables or disables an operation for WSS security.
Enable Caching	This check box flag enables or disables the response caching for an operation. See Response Caching Parameters .

Deleting an Operation

To delete an existing operation, click the red cross icon located at the top left corner of **Operation Name** field.

Characters Supported in Facade Operation URI

The following are the list of special characters that are supported and not supported by the TIBCO API Exchange Gateway HTTP Channel.

This is due to restrictions imposed by the dependencies of TIBCO API Exchange Gateway.

The list of supported and unsupported characters include:

Characters	REST/HTTP	SOAP/HTTP
Supported	<ul style="list-style-type: none"> • - (dash) • : (colon) • ~ (tilde) • (blank space) • . (dot) • " (quotation mark) • ' (Single quote) 	<ul style="list-style-type: none"> • . (dot) • ' (Single quote) • (blank space) • " (quotation mark)
Not Supported	<ul style="list-style-type: none"> • \$ (dollar sign) • ^ (caret) • & (ampersand) • // (double forward slash) • \ (backslash) 	<ul style="list-style-type: none"> • : (colon) • - (dash) • \$ (dollar sign) • ~ (tilde) • ^ (caret) • & (ampersand) • // (double forward slash) • \ (backslash)

Target Operations

A target operation is defined as a single type of request that the gateway instance sends to back-end systems. The backed-API defines the request structure, and the expected reply structure.

Using the **Target Operations** tab, you can configure target operation details.


TIBCO API Exchange Gateway supports the following types of target operations:

- ESB
- HTTP
- SOAPJMS
- HTTPs

Adding a New Target Operation

To add a new target operation, follow these steps:

Procedure


1. Click the **Target Operations** tab.
2. Click the **Add property**  icon to create a new facade operation.
3. Enter the configuration parameters for target operation, as follows:

Target Operation Configuration Parameters

Parameter	Description
Type: No Operation	Specifies an empty target operation. This does not accept any messages. This type is used for the operations that use the information retrieved during customer validation.
Operation Name	The name of the target operation.
Type	The type of transport to use when accessing the target operation. For example, No Operation.
Target Operation Group	The name of the Target Operation group. See Target Operation Groups to configure a target operation group.
Timeout	Timeout (in milliseconds) to use when accessing the target operation.
Request Transform	The mapping from request canonical form to the target operation API. The mapping details are defined in MAPPING > Mapping tab.
Response Transform	The mapping from back-end service API to response canonical form. The mapping details are defined in MAPPING > Mapping tab.
Monitor(s)	Throttle chain to be applied when invoking the back-end service. You can add one or more throttle names. The details of the throttles are defined in the MONITORING > Monitors tab.


Parameter	Description
Type: ESB When the JMS transport is used to invoke the target operation, configure the following parameters specific to JMS transport:	
ESB Channel	<p>The number of predefined ESB channels. By default, three ESB channels are supported, this field can have the values as 0, 1,2.</p> <p>You can add more ESB Channels by customizing ASG_DefaultImplementation project.</p>
ESB Service	<p>The name of the ESB service to call.</p> <p>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.</p>
Service Instance	<p>The identity of service instance to call.</p> <p>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.</p>
ESB Operation	<p>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 Operation.</p> <p>Note that this tuple determines SOAP Action used by ESB as: "/esb/service//operation"</p>
Destination Name	<p>The name of the queue or topic for the JMS channel used to override the default JMS destination. The default destination is specified by the global variables tibco.clientVar.ASG/Endpoint/ESB0/requestQueue, tibco.clientVar.ASG/Endpoint/ESB1/requestQueue, tibco.clientVar.ASG/Endpoint/ESB2/requestQueue in ASG_CONFIG_HOME/asg.properties file respectively.</p> <p>Optional.</p>
Destination Type	<p>The type of the destination for JMS channel.</p> <p>The default value is queue.</p> <p>Optional.</p>

Parameter	Description
Mode	<p>Specifies a mode for a back-end JMS service. The valid values are SYNC/ASYNC.</p> <p>For the 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.</p> <p>For the async mode, the gateway does not wait for the southbound response from the target operation after the southbound request is sent to the target operation. A default northbound response payload is created after the async request is sent to the target operation.</p>
Type: HTTP When the HTTP transport is used to invoke the target operation, configure the following parameters specific to HTTP transport:	
SOAP Action	The value of the SOAP Action as defined by the WSDL of the target operation API.
URI	Specifies the URI to use when invoking the target operation.
Host	The IP address or hostname of the target operation implementation when invoked over HTTP.
Port	The TCP port of the target operation implementation when invoked over HTTP.
Username	The username with BASIC authentication.
Password	The password with BASIC authentication.

Parameter	Description
Headers To Forward	<p>Using this field, you can copy the HTTP headers information from the northbound incoming request (facade operation) and forward it to the target operation at the southbound side. This field can contain the following characters:</p> <ul style="list-style-type: none"> • A comma separated list of named HTTP header names. You can specify any header name from the incoming HTTP request such as <code>content-type</code>, <code>soap-action</code>, and <code>content-length</code>. • An asterix (*) as the wildcard symbol to forward all HTTP headers from the facade service request to the target reference request. • An asterix (*) 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. • <code>apikey</code> as the keyword to forward api key. • <code>{query_string}</code> as the keyword to forward the query string. <p>For example,</p> <ul style="list-style-type: none"> • If the value of Headers To Forward field is specified as *, then all the headers are copied. • If the Headers To Forward field contains <code>"*, -SoapAction"</code>, any incoming SOAP Action header is removed from the incoming headers and the value set on the endpoint is ignored. <p>The default value is <code>*, -apikey</code> which copies all the HTTP headers except the <code>apikey</code> to the target operation request at the southbound side.</p> <div>  <p>According to HTTP/1.1 RFC, HTTP headers are case-insensitive. The native HTTP channel of API Exchange uses Apache Tomcat which enforces this behavior by converting all headers to lower case. We recommend that application developers must not depend on case sensitive headers in their application logic.</p> </div>

Parameter	Description
Method	<p>Specifies the method to be used as an HTTP method for sending a southbound request over the HTTP transport.</p> <p>The following methods are available for the HTTP transport:</p> <p>OPTIONS, GET, HEAD, POST, PUT,DELETE,TRACE,CONNECT</p> <p>The default value is POST</p>
Retry Count	The number of retries.
Retry Interval	The interval between the HTTP connection retries. A value of 0 indicates no retry.
Retry Timeout	The timeout value on each attempt of HTTP connection. This value is specified in milliseconds. A value of 0 indicates no timeout.
Communications Type	<p>Specifies a mode for a back-end HTTP service. The valid values are SYNC/ASync.</p> <p>For the 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.</p> <p>For the async mode, the gateway does not wait for the southbound response from the target operation after the southbound request is sent to the target operation. A default northbound response payload is created after the async request is sent to the target operation.</p>
Type: SOAPJMS When the SOAPJMS transport is used to invoke the back-end 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 JMSEExpiration header to set in the outgoing JMS message.
Destination Name	The name of the destination on JMS server.
Destination Type	The type of the destination (TOPIC/QUEUE).
Target Service	The name of the service to call.

Parameter	Description
Content Type	The value of JMSType header to set in the outgoing JMS message
Is Async	
Type: HTTPS 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 the target operation.
URI	The URI to use when invoking the target operation.
Host	The IP address or host name of the target operation implementation when invoked over the HTTPS.
Port	The TCP port of the target operation implementation when invoked over HTTPS.
Username	Username with BASIC authentication.
Password	Password with BASIC authentication.

Parameter	Description
Headers To Forward	<p>Using this field, you can copy the HTTP headers information from the northbound incoming request (facade operation) and forward it to the target operation at the southbound side. This field can contain the following characters:</p> <ul style="list-style-type: none"> • A comma separated list of named HTTP header names. You can specify any header name from the incoming HTTP request such as <code>content-type</code>, <code>soap-action</code>, and <code>content-length</code>. • An asterix (*) as the wildcard symbol to forward all HTTP headers from the facade service request to the target reference request. • An asterix (*) 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. • <code>apikey</code> as the keyword to forward api key. • <code>{query_string}</code> as the keyword to forward the query string. <p>For example,</p> <ul style="list-style-type: none"> • If the value of Headers To Forward field is specified as *, then all the headers are copied. • If the Headers To Forward field contains <code>"*, -SoapAction"</code>, any incoming SOAP Action header is removed from the incoming headers and the value set on the endpoint is ignored. <p>The default value is <code>*, -apikey</code> which copies all the HTTP headers except the <code>apikey</code> to the target operation request at the southbound side.</p> <div>  <p>According to HTTP/1.1 RFC, HTTP headers are case-insensitive. The native HTTP channel of API Exchange uses Apache Tomcat which enforces this behavior by converting all headers to lower case. We recommend that application developers must not depend on case sensitive headers in their application logic.</p> </div>

Parameter	Description
Method	<p>Specifies the method to be used as a HTTP method for sending a southbound request over the HTTPS transport.</p> <p>The following methods are available for the HTTPS transport:</p> <p>OPTIONS, GET, HEAD, POST, PUT,DELETE,TRACE,CONNECT</p> <p>The default value is POST</p>
Retry Count	The number of retries.
Retry Interval	The interval between the HTTPS connection retries. A value of 0 indicates no retry.
Retry Timeout	The timeout value on each attempt of the HTTP connection. This value is specified in milliseconds. A value of 0 indicates no timeout.
New Property File	Specifies the DSS properties file to use the HTTPs transport for target operation. See Define DSS Properties for Services .
Existing Property Files	Specifies an existing DSS property file from the drop-down list if the file exists in the wss directory of the project configuration.
Is Anonymous	<p>This field is a Boolean field and 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 <code>false</code>. If the Is Anonymous flag is set to <code>true</code>,the service does not require the authentication of the client.</p> <p>See Configure Secure Services with TIBCO API Exchange Gateway for details.</p>

Target Operation Groups

Using the **Target Operation Groups** tab, you can 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 the load balancing functionality. See [Configuring a Target Operation Group](#) for details.

To define a target operation group, configure the following fields:

Target Operation Groups Configuration Parameters

Parameter	Description
Service Groups Configuration	

Parameter	Description
Group Name	<ul style="list-style-type: none"> The user defined name of the target operation group. Required.
Description	The user defined description.
Type	<ul style="list-style-type: none"> Specifies the type of the target operation group. See Types of Target Operations Group for details. The supported types of target operation group are as follows: <ul style="list-style-type: none"> 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 Config UI. You can add the target operations by clicking the "+" icon sign, and selecting a target operation from the drop-down list.

Routing

Configuring Routing Parameters

Using the **Routing** tab, you can configure the routing information for the Core Engine. The routing provides the binding between a facade operation and a target operation.

Procedure

1. Click the **Routing** tab.
2. Enter the following parameters:

Routing Configuration Parameters

Parameter	Description
Operation Name	The name of the facade operation. This is defined in the Facade Operations tab.
Routing Type	<p>Specifies the type of routing which indicates if the request is routed to a target operation or target operation group.</p> <p>The possible values are as follows:</p> <ul style="list-style-type: none"> Target Operation Target Operation Group

Parameter	Description
Routing Key	Evaluated routing key for the given operation. See Routing Key .
Type: Target Operation When the routing type is specified as Target Operation , configure the following parameters specific to the target operation:	
Target Operation Version	The version number of the target operation.
Target Operation.	The name of the target operation. The target operation must be defined in the Target Operation tab.
Type: Target Operation Group When the routing type is specified as Target Operation Group , configure the following parameters specific to target operation group:	
Target Operation Group	The name of the target operation group used for load balancing of requests for target operations. See Target Operation Groups to configure a target operation group.

PARTNER


The **PARTNER** tab of the Config UI provides the configuration parameters.

Partners

Using the **Partners** tab, you can 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 these steps:

Procedure

1. Click the **Partner** tab.
2. Click Add property  icon to create a new partner.
3. Enter the details for the partner data, as follows:

Partner Data Configuration

Parameter	Description
Partner Name	The partner name. Required.
Partner Email	The contact email for the partner.
Partner Phone	The contact phone for the partner.


Parameter	Description
Partner Group	The name of the group of which the partner is a member. The partner group name is defined in the Partner Groups tab.
Partner Serial Number	Specifies the value used for partner identification. For example, for SSL mutual authentication, this specifies the serial number of the SSL certificate of the partner. See Partner Identification Fields for details.
Partner Issuer CA	Specifies the value used for partner identification. For example, for SSL mutual authentication, this specifies the issuing Certificate Authority of the SSL certificate of the partner. See Partner Identification Fields for details.
Enable Secondary ACL	If selected, this partner exists in the secondary ACLs list. If not selected, this partner does not exist in the secondary ACLs list.
Monitors	Defines the throttle chain which is applied to the partner. 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 Monitors tab.

Partner Groups

Using the **Partner Groups** tab, you can 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.

Adding a Throttle for a Partner Group

Procedure

1. Click the **Partner Groups** tab.
2. Click the **Add property**  icon to create a new partner.
3. Enter the details for partner group and monitor, as follows:

Partner Group Configuration

Parameter	Description
Group Name	The name of the partner group. Required.
Email	The contact email for the partner group.
Phone	The contact phone for the partner group.
Monitor	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 Monitors tab.


Facade Access

Use **Facade Access** tab to configure an operation for a specific partner.

Using the **Facade Access** tab, you can invoke a specific operation by a specific partner. This tab also defines any throttles that are applied for the operation from this partner.

Adding a New Facade Access

Procedure

1. Click the **Facade Access** tab.
2. Click the **Add property**  icon to create a new facade operation for a partner.
3. Enter the details for partner and operation data, as follows:

Facade Access Configuration

Parameter	Description
Partner	The partner name. Required.
Facade Operation	The name of the facade operation for which the partner is granted the access. This operation is configured in the Facade Operations tab.
Monitors	A list of the throttle names to be applied to any requests sent by this partner which invoke the operation. The monitors are applied in the order given in the chain.
Partner Timeout	The timeout in milliseconds to any incoming request from the partner to the selected operation.
Partner Secondary ACL Check	If selected, this partner is verified in the secondary ACL list when the partner invokes the operation.
Request Transform	The partner-specific mapping reference to call on an inbound message.
Response Transform	The partner-specific mapping reference to call on an outbound message.
Allowed Requestor IDs	The list of authorized partner's references when the partner is an aggregator.
API Key	Specifies the API Key used for partner identification when the request is sent from the TIBCO API Exchange Manager.
Preferred Routing Key	Enables the preferred routing based on the value specified in this field. See Preferred Routing .

Partner API Key

Using the **Partner API Key** tab, you can enter the API key and partner name required to access the API hosted by TIBCO API Exchange Gateway.

If you use TIBCO API Exchange Manager to send a request to access the API registered with TIBCO API Exchange Gateway, the fields under **Partner API Key** tab such as **API Key**, and **Partner** are populated.



Do not edit the API key and partner values for a request sent from TIBCO API Exchange Manager to TIBCO API Exchange Gateway.

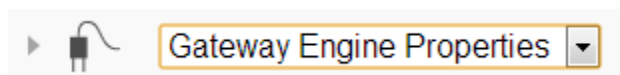
New Partner API Key

Parameter	Description
API Key	<p>An identifier assigned to an API that is registered with TIBCO API Exchange Gateway. The API key is used to access the API, and is passed as an HTTP header or as a URL parameter with the request.</p> <p>For example,</p> <pre>https://demoserver.booksore.com:9222/Books/BookOperations/Author/Vivek_Ranadive?API_Key=95-532d7700-44fe-9175-3a9d408a7286</pre>
Partner	Specifies the name of partner accessing the API.
Identify Partner	Specifies if the Client ID retrieved from the API key is used to identify the partner.

Set Runtime Properties

Use the Config UI to set the runtime properties of the Core Engine.

To set the runtime properties of the Core Engine, select the **Gateway Engine Properties** in the drop-down list next to icon on the home page of the Config UI.



Using the Config UI, the runtime properties can be set to the following links:

- General
- Monitoring
- Database
- Transport
- Security

On the home page of the Config UI, select the **Gateway Engine Properties** to display the runtime properties of the Core Engine. These properties are saved in the `ASG_CONFIG_HOME/asg.properties` and `ASG_CONFIG_HOME/asg_cl.properties` files respectively.

Setting General Properties

Procedure

1. Select the **Gateway Engine Properties** from the drop-down list.
2. Click the **General** link:

General Properties

Property	Description
Common	
Example Home	Specifies the home location for the examples directory shipped with TIBCO API Exchange Gateway. For example, the home directory for the examples location is as follows: <i>ASG_HOME/examples</i>
Allow Hot Update	Select this check box to set the value to true.If the check box is not selected, the value is set to false. The default is false. <ul style="list-style-type: none"> • If this check box is set to true, the hot configuration update is enabled. • If this check box is set to false, the hot configuration update is disabled.
Enable Default Operation	Select this to enable the default operation feature. See Pass-Through Gateway .

Setting Monitoring Properties

Procedure

1. Select the **Gateway Engine Properties** from the drop-down list.
2. Click the **Monitoring** link.

Monitoring Properties

Property	Description
Generic	

Property	Description
Use Synchronous HTTP Client	<p>A Boolean field used to send a request to the target side by running the http client in the same thread as the facade processing. It is good practice to use the synchronous http client when high load of short lived requests are expected.</p> <p>Select this check box to set the value to true. If the check box is not selected, the value is set to false. The default is false.</p> <ul style="list-style-type: none"> • If this check box is set to true, the client sends the synchronous request for the HTTP transport. • If this check box is set to false, the asynchronous request is sent for the HTTP transport.
Number of Threads	<p>The number of threads used for the HTTP Client. This is relevant only if the value of useSynchHttpClient is false (that is, when asynchronous request is sent for the HTTP transport).</p> <p>The default value is 10.</p>
ModRV North Request Subject Name	<p>Specifies the Rendezvous subject name, which is used by the Core Engine to listen for requests from the Apache module.</p> <p>The default value is _LOCAL.asg.north.request.</p>
RV Subject Prefix	<p>Specifies the prefix for all Rendezvous subject names used between the Core Engine and Central Logger, Core Engine and Global Throttle Manager components.</p> <p>The default value is TIBCO.ASG.INTERNAL.</p>
Throttle Update Interval (sec)	<p>Specifies the time interval (in seconds) for sending throttle updates to the Global Throttle Manager.</p> <p>The default value is 10.</p>
Request Binary	<p>The default value is _LOCAL.asg.north.request_binary.</p>
RV Daemon	<p>Specifies the value of Rendezvous daemon for the Core Engine to listen for the requests from the Apache module.</p> <p>The default value is tcp:7500.</p>
RV Network	<p>Specifies the value of the Rendezvous network for the Core Engine to listen for the requests from the Apache module.</p>
RV Service	<p>Specifies the value of the Rendezvous service for the Core Engine to connect and listen for the requests from the Apache module.</p> <p>The default value is 7500.</p>
GTM RV Daemon	<p>Specifies the value of the Rendezvous daemon for the Core Engine to connect to the Global Throttle Manager.</p> <p>The default value is tcp:7500.</p>

Property	Description
GTM RV Network	Specifies the value of the Rendezvous network for the Core Engine to connect to the Global Throttle Manager.
GTM RV Service	Specifies the value of the Rendezvous service for the Core Engine to connect to the Global Throttle Manager. The default value is 7500.
Logging	
Enable Reporting to Central Logger	Specifies if the reporting to the Central Logger is enabled or not. By default, the Core Engine does not record the transactions to the Central Logger. The default value is false. See Enabling Reporting to the Central Logger .
Logging Interval (ms)	Specifies the time interval (in milliseconds) between Core Engine and Central Logger to record transactions. The default value is 30000.
Minimum Log Level	Specifies the logging level for the Central Logger component. The possible values are as follows: <ul style="list-style-type: none"> DEBUG INFO WARN ERROR NO LOGGING The default value is DEBUG.
Central Logger Log Level	Specifies if the detail level logging is enabled for the Central Logger component. The possible values are as follows: <ul style="list-style-type: none"> Detail Logging ON Detail Logging OFF 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. The default value is Detail Logging OFF.
Central Logger : The Central Logger section is populated on the Config UI when Enable Reporting to Central Logger property in the Logging section is selected as true.	
File Filter Regular Expression	Specifies the lists of facade operation as a pipe (' ') separated string. The transaction logs of these facade operations are logged to the files instead of database. For example: ping test addConfiguration

Property	Description
Transactions Log File Directory	Specifies the directory name to store the log file used by the Core Engine to record the transactions data. The default value is <code>ASG_HOME/bin/logs</code> .
Transactions Log File	Specifies the name of the log file used by the Core Engine to record the transactions data. This is used only for the transactions of facade operations which are filtered by the <code>tibco.clientVar.CL/Logging/fileFilter</code> property. The default value is <code>trans_log.txt</code> .
Number of Log Files	Specifies the maximum number of log files for the Core Engine to keep on roll over for the transactions log file. The default value is 3.
Log File Max Size	Specifies the maximum size (in bytes) of the log file for writing the transactions data at which the Core Engine rolls over to the next log file. The default value is 5000000.
Timestamp Format	Specifies the format of the log's timestamp value.
RV Subject Prefix	Specifies the prefix for all Rendezvous subject names used between the Core Engine and Central Logger. The default value is <code>TIBCO.ASG.INTERNAL</code> .
RV Service	Specifies the service parameter for Rendezvous used between the Core Engine and Central Logger communication. The default value is 7500.
RV Daemon	Specifies the daemon parameter for Rendezvous used between the Core Engine and Central Logger communication. The default value is <code>tcp:7500</code> .
RV Network	Specifies the network parameter for Rendezvous used between the Core Engine and Central Logger communication.

Setting Database Properties

Procedure

1. Select the **Gateway Engine Properties** from the drop-down list.
2. Click the **Database** link.

Database Properties

Property	Description
Connection Check Interval	Specifies the time interval (in seconds) after which the Central Logger polls for the connection to the database server. The default value is 5.
Connection Retry Count	Specifies the number of attempts made by the Central Logger to connect to the database server if the database server goes down. The default value is -1.
DB Driver	Specifies the database driver for the database server used by the Central Logger.
DB URL	Specifies the database server connection URL for the Central Logger. For example, for the MS SQL server, the following value is used: <code>jdbc:sqlserver://10.107.174.56:1433;databaseName=asgstat</code>
DB Username	Specifies the username for the Central Logger to connect to the database server. The default value is <code>asguser</code>
DB Password	Specifies the password for the Central Logger to connect to the database server. The default value is obfuscated format of <code>asgpass</code> .
DB Schema	Specifies the database schema to be used by the Central Logger on the database server. The default value is <code>asgstat</code> .

Setting Transport Properties**Procedure**

1. Select the **Gateway Engine Properties** from the drop-down list.
2. Click the **Transport** link.

Transport Properties

Property	Description
Facade	
Port	Specifies the port through which the gateway accepts the HTTP requests from the client. The default value is 9222.

Property	Description
Active Spaces	
Transport Type	Type of the transport for Global Throttle Manager and Central Logger. Select AS from the drop-down list.
Metaspace Name	<p>The name of the ActiveSpaces Metaspace to connect to. A metaspace is a logical group of spaces—a cluster of hosts and processes that share the same metaspace name and set of discovery transport attributes. The hosts and processes in a metaspace work together by joining the same spaces.</p> <p>Refer to <i>TIBCO ActiveSpaces Developer's Guide</i> for details on Metaspaces.</p> <p>The default value for the metaspace name is GTM.</p>
Discovery URL	<p>Specifies the discovery URL to use to discover spaces in the cluster. The TCP discovery has the following format:</p> <p><code>tcp://ip1[:port1];ip2[:port2],...</code></p> <p>where any number of <code>ip[:port]</code> well-known addresses can be listed. If no port is specified, the default port number value of 50000 is assumed.</p> <p>For example, <code>tcp://127.0.0.1:13000</code>.</p> <p>Refer to <i>TIBCO ActiveSpaces Developer's Guide</i> for details on the discovery URLs.</p>
Listen URL	<p>Specifies a URL that is used for direct communication between the members of the metaspace.</p> <p>To use a listen URL, use a string of the form:</p> <p><code>tcp://[interface[:port]]</code></p> <p>This syntax specifies that the member should bind to the specified interface and the specified port when creating the TCP socket that will be used for direct communication between the members of the metaspace. If not specified, it will default to 0.0.0.0 (INADDR_ANY) for the interface and to the first available port starting from port 5000 and above.</p> <p>For example, <code>tcp://127.0.0.1:13000-*/</code></p>

Property	Description
Log Level	<p>The logging level of the messages when the ActiveSpaces channel is used.</p> <p>The default value is INFO.</p> <p>The possible values are as follows:</p> <p>INFO</p> <p>WARN</p> <p>ERROR</p> <p>FATAL</p> <p>FINE</p> <p>FINER</p> <p>FINEST</p> <p>NONE</p>
Log Directory	<p>The directory to store the log files. The log files contain the messages when the ActiveSpaces channel is used.</p> <p>The default value is <code>ASG_CONFIG_HOME/logs</code></p> <p>For example,</p> <p><code>C:/TIBCO_HOME/APIXCONFIG/tibco/cfgmgmt/asg/logs</code></p>
SSL	
Port	<p>Specifies the port through which the gateway accepts SSL enabled HTTP requests from client.</p> <p>The default value is 9233.</p>
Use SSL	<p>This is a boolean field which indicates if SSL should be enabled for accepting the HTTPS requests. If this is set to true, SSL is enabled to accept the requests using the HTTPS transport.</p>
Identity Resource	<p>Specifies an identity resource which is used by FacadeHTTPSSLConnection HTTP shared resource to provide the SSL properties.</p>

Property	Description
Identity File Type	<p>Specifies the type of identity resource.</p> <p>The possible values are as follows:</p> <ul style="list-style-type: none"> Identity File Certificate/Private Key If Identity File Type is of the Identity File type, enter the Identity Type, Identity URL, and Identity File Password parameters. If Identity File Type is of the Certificate/Private Key type, enter Certificate URL, Key URL, and Key Password parameters.
Identity Type	<p>Specifies the type of the keystore if the Identity File Type is of the Identity File type. The supported values are as follows:</p> <ul style="list-style-type: none"> JCEKS JKS PEM PKCS12
Identity URL	<p>Specifies the URL to the identity file if Identity File Type is of the Identity File type.</p> <p>For example, C:\asgserver.pfx</p>
Identity File Password	<p>Specifies the password for the identity file used for the SSL connection if Identity File Type is of the Identity File type.</p>
Certificate URL	<p>Specifies the URL to the certificate file if Identity File Type is of the Certificate/Private Key type.</p>
Key URL	<p>Specifies the URL to the private key in the certificate file if Identity File Type is of the Certificate/Private Key type.</p>
Key Password	<p>Specifies the password for the private key used for the SSL connection if Identity File Type is of the Certificate/Private Key type.</p>
Requires Client Authentication	<p>Indicates a Boolean flag to enable or disable mutual SSL authentication for HTTPs transport between the client and the gateway.</p> <p>When this field is set to true, the Trusted Certificates Folder becomes enabled so that you can specify a location containing the list of trusted certificate authorities.</p>

Property	Description
Truststore Password	Specifies the password to access the certificate stored in the folder defined by Trusted Certificate Folder field.
Trusted Certificate Folder	<p>Specifies a folder containing one or more certificates from trusted certificate authorities, which is required for mutual SSL authentication.</p> <p>Required when the RequiresClientAuthentication property is set to <code>true</code>.</p>
JMS Facade ESB Connection	
JMS Provider URL	<p>Specifies the connection URL for the EMS Server used for facade operation requests from the ESB communication domain. The ESB communication uses JMS transport with XML.</p> <p>The default value is <code>tcp://localhost:7222</code>.</p>
JNDI Context URL	<p>Specifies the URL to the JNDI service provider used for facade operation requests with ESB communication domain.</p> <p>The default value is <code>tibjmsnaming://localhost:7222</code></p>
Topic Connection Factory Name	<p>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.</p> <p>The default value is <code>TopicConnectionFactory</code>.</p>
Queue Connection Factory Name	<p>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.</p> <p>The default value is <code>QueueConnectionFactory</code>.</p>
JNDI Username	<p>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.</p> <p>The default value is <code>admin</code>.</p>
JNDI Password	<p>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.</p>

Property	Description
JMS Username	<p>Specifies the user name for logging into the EMS server in the ESB communication domain at the facade side.</p> <p>The default value is admin.</p>
JMS Password	<p>Specifies the password for logging into the EMS server in the ESB communication domain at the facade side.</p>
Request Queue	<p>Specifies the queue name for an ESB channel (one) communication for the target operation request.</p> <p>The default value is <code>asg.out.request</code>.</p>
Reply Queue	<p>Specifies the queue name for ESB channel (one) communication for the response from the target operation.</p> <p>The default value is <code>asg.out.request.reply.0.0</code>.</p>
JMS Target ESB Connection Primary	
JMS Provider URL	<p>Specifies the connection URL for the EMS Server used for facade operation requests from ESB communication domain. ESB communication uses JMS transport with XML.</p> <p>The default value is <code>tcp://localhost:7222</code>.</p>
JNDI Context URL	<p>Specifies the URL to the JNDI service provider used for facade operation requests with ESB communication domain.</p> <p>The default value is <code>tibjmsnaming://localhost:7222</code>.</p>
Topic Connection Factory Name	<p>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.</p> <p>The default value is <code>TopicConnectionFactory</code>.</p>
Queue Connection Factory Name	<p>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.</p> <p>The default value is <code>QueueConnectionFactory</code>.</p>
JNDI Username	<p>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.</p> <p>The default value is admin.</p>

Property	Description
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. The default value 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. The default value is <code>asg.out.request</code> .
Reply Queue	Specifies the queue name for the ESB channel (one) communication for the response from the target operation. The default value is <code>asg.out.request.reply.0.0</code> .
JMS Target ESB Connection Secondary	
JMS Provider URL	Specifies the connection URL for the EMS Server used for facade operation requests from the ESB communication domain. The ESB communication uses JMS transport with XML. The default value is <code>tcp://localhost:7222</code> .
JNDI Context URL	Specifies the URL to the JNDI service provider used for facade operation requests with ESB communication domain. The default value is <code>tibjmsnaming://localhost:7222</code> .
Topic Connection Factory Name	Specifies the name of <code>TopicConnectionFactory</code> object stored in JNDI. This object is used to create a topic connection with ESB services at the facade side. The default value is <code>TopicConnectionFactory</code> .
Queue Connection Factory Name	Specifies the name of <code>QueueConnectionFactory</code> object stored in JNDI. This object is used to create a queue connection with ESB services at the facade side. The default value is <code>QueueConnectionFactory</code> .

Property	Description
JNDI Username	<p>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.</p> <p>The default value is <code>admin</code>.</p>
JNDI Password	<p>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.</p>
JMS Username	<p>Specifies the user name for logging into the EMS server in the ESB communication domain at the facade side.</p> <p>The default value is <code>admin</code>.</p>
JMS Password	<p>Specifies the password for logging into the EMS server in the ESB communication domain at the facade side.</p>
Request Queue	<p>Specifies the queue name for an ESB channel (one) communication for the target operation request.</p> <p>The default value is <code>asg.out.request</code>.</p>
Reply Queue	<p>Specifies the queue name for ESB channel (one) communication for the response from the target operation.</p> <p>The default value is <code>asg.out.request.reply.0.0</code>.</p>
JMS Target ESB Connection Tertiary	
JMS Provider URL	<p>Specifies the connection URL for the EMS Server used for facade operation requests from the ESB communication domain. The ESB communication uses JMS transport with XML.</p> <p>The default value is <code>tcp://localhost:7222</code>.</p>
JNDI Context URL	<p>Specifies the URL to the JNDI service provider used for facade client requests with the ESB communication domain.</p> <p>The default value is <code>tibjmsnaming://localhost:7222</code>.</p>
Topic Connection Factory Name	<p>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.</p> <p>The default value is <code>TopicConnectionFactory</code>.</p>

Property	Description
Queue Connection Factory Name	<p>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.</p> <p>The default value is QueueConnectionFactory.</p>
JNDI Username	<p>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.</p> <p>The default value is admin.</p>
JNDI Password	<p>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.</p>
JMS Username	<p>Specifies the user name for logging into the EMS server in the ESB communication domain at the facade side.</p> <p>The default value is admin.</p>
JMS Password	<p>Specifies the password for logging into the EMS server in the ESB communication domain at the facade side.</p>
Request Queue	<p>Specifies the queue name for an ESB channel (one) communication for the target operation request.</p> <p>The default value is <code>asg.out.request</code>.</p>
Reply Queue	<p>Specifies the queue name for ESB channel (one) communication for the response from the target operation.</p> <p>The default value is <code>asg.out.request.reply.0.0</code>.</p>
JMS SOAP Connection North	
JMS Provider URL	<p>Specifies the connection URL for the EMS Server used for facade operation requests from the ESB communication domain. The ESB communication uses JMS transport with XML.</p> <p>The default value is <code>tcp://localhost:7222</code>.</p>
JNDI Context URL	<p>Specifies the URL to the JNDI service provider used for facade client requests with the ESB communication domain.</p> <p>The default value is <code>tibjmsnaming://localhost:7222</code>.</p>

Property	Description
Topic Connection Factory Name	<p>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.</p> <p>The default value is TopicConnectionFactory.</p>
Queue Connection Factory Name	<p>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.</p> <p>The default value is QueueConnectionFactory.</p>
JNDI Username	<p>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.</p> <p>The default value is admin.</p>
JNDI Password	<p>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.</p>
JMS Username	<p>Specifies the user name for logging into the EMS server in the ESB communication domain at the facade side.</p> <p>The default value is admin.</p>
JMS Password	<p>Specifies the password for logging into the EMS server in the ESB communication domain at the facade side.</p>
Request Queue	<p>Specifies the queue name for an ESB channel (one) communication for the target operation request.</p> <p>The default value is asg.out.request.</p>
Reply Queue	<p>Specifies the queue name for the ESB channel (one) communication for the response from the target operation.</p> <p>The default value is asg.out.request.reply.0.0.</p>
JMS SOAP Connection South	
JMS Provider URL	<p>Specifies the connection URL for the EMS Server used for facade operation requests from the ESB communication domain. The ESB communication uses JMS transport with XML.</p> <p>The default value is tcp://localhost:7222.</p>

Property	Description
JNDI Context URL	<p>Specifies the URL to the JNDI service provider used for facade client requests with the ESB communication domain.</p> <p>The default value is <code>tibjmsnaming://localhost:7222</code>.</p>
Topic Connection Factory Name	<p>Specifies the name of <code>TopicConnectionFactory</code> object stored in JNDI. This object is used to create a topic connection with ESB services at the facade side.</p> <p>The default value is <code>TopicConnectionFactory</code>.</p>
Queue Connection Factory Name	<p>Specifies the name of <code>QueueConnectionFactory</code> object stored in JNDI. This object is used to create a queue connection with ESB services at the facade side.</p> <p>The default value is <code>QueueConnectionFactory</code>.</p>
JNDI Username	<p>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.</p> <p>The default value is <code>admin</code>.</p>
JNDI Password	<p>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.</p>
JMS Username	<p>Specifies the user name for logging into the EMS server in the ESB communication domain at the facade side.</p> <p>The default value is <code>admin</code>.</p>
JMS Password	<p>Specifies the password for logging into the EMS server in the ESB communication domain at the facade side.</p>
Request Queue	<p>Specifies the queue name for an ESB channel (one) communication for the target operation request.</p> <p>The default value is <code>asg.out.request</code>.</p>
Reply Queue	<p>Specifies the queue name for an ESB channel (one) communication for the response from the target response.</p> <p>The default value is <code>asg.out.request.reply.0.0</code>.</p>
<p>Show SSLProperties</p> <p>Use the Show SSL Properties button to configure the SSL connection parameters for JMS transport at the facade and target side.</p>	

Property	Description
Use SSL	<p>A boolean field to indicate if SSL is enabled for accepting requests using the JMS transport.</p> <p>Set this to true to enable SSL for the JMS transport.</p>
Trusted Certificate Folder	<p>Specifies a location of the trusted certificates on this machine. The trusted certificates are a collection of certificates from servers with which you establish connections. If the server with which the connection is going to be established, presents a certificate that does not match one of your trusted certificates, the connection is refused. This prevents connections to unauthorized servers.</p> <p>Import the trusted certificates into a folder before you select the folder in this field.</p>
Identity Resource	<p>Specifies an identity resource used to provide the SSL properties for JMS transport.</p> <p>For example,</p> <pre>/DefaultImplementation/SharedResources/JMS/Target_ESB0ConnIdentityResource.id</pre>
TrustStorePassword	<p>Specifies the password to access the certificate stored in the folder defined by the Trusted Certificate Folder field.</p>
Trace	<p>Specifies whether SSL tracing should be enabled during the connection. If checked, the SSL connection messages are logged and sent to the console.</p>
Debug Trace	<p>Specifies whether SSL debug tracing should be enabled during the connection. Debug tracing provides more detailed messages than standard tracing.</p>
Verify Host Name	<p>Specifies whether the host you are connecting to is the expected host. The host name in the host's digital certificate is compared against the value in the Expected Host Name field. If the host name does not match the expected host name, the connection is refused.</p> <p>Note: The default context factories for TIBCO Enterprise Message Service automatically determine if host name verification is necessary. If you are using a custom implementation of the context factories, your custom implementation must explicitly set the verify host property to the correct value. For example:</p> <pre>com.tibco.tibjms.TibjmsSSL.setVerifyHost(false)</pre>

Property	Description
Expected Host Name	<p>Specifies the name of the host you are expecting to connect to. This field is only relevant if the Verify Host Name field is also selected.</p> <p>If the name of the host in the host's digital certificate does not match the value specified in this field, the connection is refused.</p> <p>This prevents hosts from attempting to impersonate the host you are connecting to.</p>
Strong Cipher Suites Only	<p>When selected, this field specifies that the minimum strength of the cipher suites used can be specified with the <code>bw.plugin.security.strongcipher.minstrength</code> custom engine property. See TIBCO ActiveMatrix BusinessWorks Administration for more information about this property. The default value of the property disables cipher suites with an effective key length below 128 bits.</p> <p>When this field is unchecked, only cipher suites with an effective key length of up to 128 bits can be used.</p>
Identity Type	<p>Specifies the type of identity resource. The possible values are:</p> <ul style="list-style-type: none"> • <code>certPlusKeyURL</code> • <code>url</code> • <code>usernamePassword</code>
Certificate/Private Key Identity Use this option if the private key and the certificate are in two separate files.	
Certificate URL	Specifies the URL to the certificate file if the Identity Type is of the certPlusKeyURL type.
Key URL	Specifies the URL to the private key in the certificate file if the Identity Type is of the certPlusKeyURL type.
Key Password	Specifies the password for the private key used for SSL connection if the Identity Type is of the certPlusKeyURL type.
Username/Password Use this option if you want to use a username and password for authentication instead of a certificate.	
Identity User Name	Name of the user for this identity.
Identity Password	Password for the user for this identity.

Property	Description
url	Use this option if the certificate includes the private key information in the same file.
Identity URL	Specifies the location of the certificate (which includes the private key) if Identity Type is of url type. For example, C:\asgserver.pfx
Identity File Type	Specifies the type of certificate file if Identity Type is of url type. Choose the certificate file type from the drop-down list: <ul style="list-style-type: none"> • Entrust • JCEKS • JKS • PEM • PKCS12
Key Password	Password for the certificate if Identity Type is of url type.

Security Properties

Procedure

1. Select the **Gateway Engine Properties** from the drop-down list.
2. Click the **Security** link:

Security Properties

Property	Description
Common	
Anonymous Partner Name	Specifies a default partner name for the unauthenticated requests.
OAuth	

Property	Description
Transport Scheme	<ul style="list-style-type: none"> Specifies the transport type used to connect to for TIBCO API Exchange Gateway OAuth server. <p>The possible values are:</p> <ul style="list-style-type: none"> – HTTP – HTTPS <ul style="list-style-type: none"> See OAuth HTTP Transport to set the OAuth HTTP transport settings. See OAuth WebApps SSL to set the OAuth HTTPS (SSL) transport settings.
OAuth HTTP Transport	
Port	<p>Specifies the non-SSL port number of the TIBCO API Exchange Gateway OAuth Server.</p> <p>The default value is 9322.</p>
OAuth WebApps SSL	
Properties are shown if the transport scheme is selected as HTTPS.	
Host	<p>Specifies the IP address of the TIBCO API Exchange Gateway OAuth Server.</p>
Port	<p>Specifies the SSL port number of the TIBCO API Exchange Gateway OAuth Server.</p> <p>The default value 9333.</p>
Use SSL	<p>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 the OAuth server.</p> <p>The default value is true.</p>
Identity Resource	<p>Specifies an identity resource used by OAuthWebappsSSLConnection HTTP shared resource to provide SSL properties for the OAuth servlets and APIs.</p> <p>The default value is</p> <pre>/DefaultImplementation/SharedResources/HTTP/OAuthIdentityResource.id</pre>

Property	Description
Identity File Type	<p>Specifies the type of identity resource used by OAuthWebappsSSLConnection HTTP shared resource.</p> <p>The possible values are as follows:</p> <ul style="list-style-type: none"> • Identity File • Certificate/Private Key <p>If Identity File Type is of type Identity File , enter the Identity Type, Identity URL, and Identity File Password parameters.</p> <p>If Identity File Type is of type Certificate/Private Key, enter the Certificate URL, Key URL, Key Password parameters.</p>
Identity Type	<p>Specifies the type of the keystore if the Identity File Type is of the Identity File type. The supported values are as follows:</p> <ul style="list-style-type: none"> • JCEKS • JKS • PEM • PKCS12
Identity URL	<p>Specifies the URL to the identity file used for the OAuth server SSL connection if the Identity File Type is of the Identity File type.</p> <p>For example, C:\keystore.jks</p>
Identity File Password	<p>Specifies the password for the identity file used for the OAuth server SSL connection if the Identity File Type is of the Identity File type.</p>
Certificate URL	<p>Specifies the URL to the certificate file used for the OAuth server SSL connection if Identity File Type is of the Certificate/Private Key type.</p>
Key URL	<p>Specifies the URL to the private key in certificate file used for the OAuth server SSL connection if Identity File Type is of the Certificate/Private Key type.</p>
Key Password	<p>Specifies the password for the private key used for the OAuth server SSL connection if Identity File Type is of the Certificate/Private Key type.</p>
Requires Client Authentication	<p>Indicates a Boolean flag to enable or disable mutual SSL authentication for HTTPs transport used for OAuth server requests from the requestor.</p> <p>When this field is set to true, the Trusted Certificates Folder becomes enabled so that you can specify a location containing the list of trusted certificate authorities.</p>

Property	Description
Truststore Password	Specifies the password to access the certificate stored in the folder defined by the Trusted Certificate Folder field.
Trusted Certificate Folder	<p>Specifies a folder containing one or more certificates from trusted certificate authorities, which is required for mutual SSL authentication.</p> <p>Required when the RequiresClientAuthentication property is set to true.</p>
Portal Engine Integration Properties	
Engine URL	<p>Specifies the URL of the portal engine if used as a client. For example, the engine URL can be specified as follows: <code>http://portal_host_name:9122</code></p> <p>where <i>portal_host_name</i> is the host machine running the portal engine.</p>
General	
WebApps Path	<ul style="list-style-type: none"> • Specifies the location of the OAuth web application. • Change web.xml file if you want to add custom login page or access grant page to the OAuth server. • Do not remove the content of this war file.
OAuth Data Space	
Access Token Retention Period	
	Specifies the expiration time (in minutes) for an access token. The default value is 60 minutes. When the access token passes expiration time as specified by this property, it is no longer valid but still remains in the database.
MetaSpace Name	
	<ul style="list-style-type: none"> • Specifies the metaspace name used by the OAuth server. • The default value is ASG-OAuth-Tokens
Local Discovery URL	
	<p>Specifies the discovery URL for this OAuth instance of the metaspace discovers the current metaspace members.</p> <p>For example, <code>tcp://machine1_IP_Address:6300;machine2_IP_Address:6300</code></p>
Local Listen URL	

Property	Description
	<p>Specifies the listening URL for this OAuth instance of the metaspace.</p> <p>For example, <code>tcp://machine1_IP_Address:6300</code></p>
Batch Size	
	<p>Specifies the maximum number of entries to return when querying the data such as access token details.</p>
Properties For OAuth Persister	
Persister Store	
	<ul style="list-style-type: none"> Defines the type of persistence store. The possible values are: <ul style="list-style-type: none"> 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 as Database.
Database Connection Properties for OAuth Persister as Database	
Set the properties if the OAuth Persister is selected as Database type.	
Driver	
	<p>Specifies the database JDBC driver when Database is used as OAuth persistence store.</p>
JDBC URL	
	<p>Specifies the JDBC url for the database server when Database is used as the OAuth persistence store.</p>
JDBC User Name	
	<p>Specifies the user name to connect to the database server when Database is used as the OAuth persistence store.</p>
JDBC Password	
	<p>Specifies the password of the user to connect to the database server when Database is used as the OAuth persistence store.</p>
Properties For OAuth Adapters	

Property	Description
Resource Path Name	<p>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 <i>ASG_HOME</i>.</p> <p>For example, if the value is specified as <i>/examples/OAuth/resources</i>, the custom adapter looks for the resources such as properties file in the <i>ASG_HOME/examples/OAuth/resources</i> directory.</p>
Owner Adapter	<ul style="list-style-type: none"> Specifies the class that provides the owner adapter interface. This adapter is used to authenticate the resource owner and provide the login and access grant pages. See the Owner Service Provider Interface for details. <p>For example, for the file based owner adapter interface, the value is specified as:</p> <pre>com.tibco.asg.oauth.identity.provider.file.OwnerAdapterService</pre> <ul style="list-style-type: none"> The jar file that contains this adapter implementation must be placed in a directory in the classpath set in the <i>ASG_HOME/bin/asg-engine.tra</i> file. See Deploying Custom Adapters for details.
Client Adapter	<ul style="list-style-type: none"> Specifies the class that provides the client adapter interface. This adapter is used to authenticate the client and to retrieve the client attributes. See Client Service Provider Interface for details. <p>For example, for the file based client adapter interface, the value is specified as:</p> <pre>com.tibco.asg.oauth.identity.provider.file.ClientAdapterService</pre> <ul style="list-style-type: none"> The jar file that contains this adapter implementation must be placed in a directory in the classpath set in the <i>ASG_HOME/bin/asg-engine.tra</i> file. See Deploying Custom Adapters for details.
Scope Adapter	

Property	Description
	<ul style="list-style-type: none"> Specifies the class that provides the Scope Adapter interface. This adapter is used to retrieve the scope description and the scope from a specific resource for a given owner. See Scope Service Provider Interface for details. <p>For example, for the file based scope adapter interface, value is specified as:</p> <pre>com.tibco.asg.oauth.identity.provider.file.ScopeAdapterService</pre> <ul style="list-style-type: none"> The jar file that contains this adapter implementation must be placed in a directory in the classpath set in the <i>ASG_HOME/bin/asg-engine.tra</i> file. See Deploying Custom Adapters for details.
Portal Engine Integration Properties	
Engine URL	
	Specifies the URL of the portal engine. For example, the engine URL can be specified as: <code>http://portal_host_name:9122</code>

Transaction Pipeline processing

This section describes the full cycle of transaction pipeline processing and the mapping and transformation capabilities of TIBCO API Exchange Gateway.

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 types of transaction pipeline processing are:

- 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 stages.

Processing Order

1. Transport Level Authentication

When the RV module of TIBCO API Exchange Gateway is used with Apache server for a partner to send a request using the HTTP transport, the first level of authentication is performed at the Apache server depending on 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](#) for details.

The 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 follows:

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

For example, No Authentication is defined as follows:

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

After the request is processed by Apache server for authentication, the request is passed to the Core Engine over Rendezvous transport.



- When the Apache server is used as reverse proxy in front of the native HTTP channel of the TIBCO API Exchange Gateway, the same types of authentication can be configured at the Apache Server using the Apache directives. See [Configure Apache HTTP Server as Reverse Proxy](#).
- When the native HTTP Channel is used, the HTTP channel can be configured to provide basic authentication and SSL(Mutual) authentication. See [Enable Facade HTTPS Transport](#) for the SSL authentication configuration details.

2. Operation Identification

When TIBCO API Exchange Gateway receives a request, the gateway identifies the operation based on URI and headers of the request. See [Operation Identification](#) for details.

3. Partner Identification

After the operation is identified, the partner is identified from the request context message. See [Partner Identification](#) and [Partner API key](#) for details.

4. Request Parsing (Optional)

After the operation and partner are identified from the incoming request, the next step is to parse the request. The parse step is an optional preprocessing of the facade request. See [Parsing Step](#) for details on the parsing step of request processing.

5. Authorization

The Core Engine checks if the identified partner is authorized to access the requested operation. The partner and the associated operation is configured in the **Facade Access** under **PARTNER** tab of the Config UI.

6. Request Message Validation

If the flag for request validation is enabled, the request message is validated for syntax against XSD for the incoming northbound request message.

7. Facade Throttling

You can enforce the commercial throttles for service level agreements for a partner request using the facade throttling. Facade Throttling is applied on the partner, partner group, and partner operations.

After the request reaches this stage in the processing pipeline, how often this partner can invoke the operation is checked.

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 the operation request message to the canonical request message. Whether or not the mapping is required for this request operation, it is configured using the **Request Transform** field on the **ROUTING > Facade Operations** tab or **PARTNER > Facade Access** tab.



If a request transform is configured at both places (Facade Operations and Facade Access), only the Facade Access transform is executed.

The transformation details are defined in the **MAPPING > Mappings** tab of the Config UI.

By default, if no mappings are defined, 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 the **ROUTING > Routing** tab of the Config UI, the Core Engine determines the target operation endpoint for the incoming request. See [Transaction Pipeline processing](#) for details.

10. Service Throttling

After the name of the southbound service endpoint is derived for the invoked operation, the service throttling policy is applied. Service throttles are technical throttles implemented to protect the overuse 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, it is configured using the **Request Transform** field on the **ROUTING > Target Operations** tab. The transformation details are defined in **MAPPING > Mappings** tab of the Config UI.

By default, if no mappings are defined, 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 Core 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:

Processing Order

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 the canonical response message. Whether the mapping is required for this request operation or not, it is configured using the **Response Transform Mapping** field on the **ROUTING > Target Operations** tab. The transformation details are defined in the **Mappings** tab of the Config UI.

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

Using the reverse northbound mapping, you can transform the canonical response message to the northbound response message. Whether or not the mapping is required for this request operation, it is configured using the **Request Transform** field on the **ROUTING > Facade Operations** tab or **PARTNER > Facade Access** tab.



If a request transform is configured at both places (Facade Operations and Facade Access), only the Facade Access transform is executed.

The transformation details are defined in the **Mappings** tab of the Config UI.

This mapping can be used for the censor response policy. Using the censor response policy, you can 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 Core 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.

Parsing Step

The parsing step can be used to pre-process a facade operation request.

The parse step is an optional preprocessing of the facade request used for both content validation and for identifying and normalizing control data. Using the parse step you can enhance the downstream processing by setting or overwriting 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 > Facade Operations** tab of the Config UI.

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
- Enriching the transaction audit trail logging information of the request
- Validating the request content with the ability to set an error code and error message when a request does not pass the content validation rules
- Setting a metric increment for content-based throttles
- Setting sticky key for load-balancing with StickyResourceAffinity routing algorithm type.

Set the Partner Identity for Request

Using transformation file to set partner identify 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 the request or the transport context of the 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 overrides any other standards-based identity associated with the request.

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 are stored with a value of incomingPartner for the key_type column.

Set the Routing Key for Request

Using transformation (XSLT) file to set 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, 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 [Transaction Pipeline processing](#).

Enrich the Audit Trail Log for Request

Using transformation (XSLT) file to enrich the audit trail log data.

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 is 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 a 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 database. The value of the key element is stored in the `KEY_VALUE` column.

The key element provides the following attributes:

- `@type`
- `@log`

@type attribute

When you set a `@type` attribute for the key element, the `@type` attribute is stored in the `KEY_TYPE` column in the `ASG_TRANSACTION_KEYS` table.

By using the `@type` attribute for the key element, users can search for specific audit trail logging records in the Central Logger database through 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 TIBCO API Exchange Gateway itself.

@log attribute

If you do not want to store the parameters 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` is set for the key elements and the key elements are 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`.

Logging Request Headers

Using transformation (SLT) file to store the header values from the request .

The BookQueryBE example ships the sample XSLT file to parse the request for queryBookByAuthor facade operation and store the header values into the database.

To store the header values using the `parse_headers.xsl`, follow these steps:

Procedure

1. Stop the Config UI server, if running.

2. Navigate to the *ASG_CONFIG_HOME/BookQueryBE/xslt* directory.
3. Copy the *parse_headers.xsl* file to *ASG_CONFIG_HOME/BookQueryBE/xslt/operations* directory.
4. Start the Config UI server.
5. Login to the Config UI using your credentials.
6. Click BookQueryBE project.
7. Select **ROUTING > Facade Operations** tab.
8. Select **queryBookByAuthor** operation.
9. Select *operations/parse_headers.xsl* from the drop-down list in the **ProcessBody Transform** field.
10. Save changes to the configuration.

Sample XSLT

Refer to the following sample XSLT to store the headers of the request message:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:f="http://www.tibco.com/asg/content-types/form"
xmlns:c="http://www.tibco.com/schemas/asg/context"
xmlns:h="http://www.tibco.com/asg/protocols/http"
xmlns:form="http://www.tibco.com/asg/functions/form" exclude-result-prefixes="xsl h
c f form">
<xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes" omit-xml-
declaration="no"/>
<xsl:variable name="context">
<xsl:for-each select="/transformation/context">
  <xsl:copy-of select="document(@href)"/>
</xsl:for-each>
</xsl:variable>
<xsl:variable name="httpRequest">
<xsl:copy-of select="$context/c:context/c:entry[@key='asg:httpRequest']/h:request"/>
</xsl:variable>
  <xsl:variable name="request-uri">
    <xsl:copy-of select="$httpRequest/h:request/h:request-uri"/>
  </xsl:variable>
  <xsl:template match="/">
    <output>
      <key type="requestURI" log="true">
        <xsl:copy-of select="normalize-space($request-uri)"/>
      </key>
      <key type="clientIP" log="true">
        <xsl:choose>
          <xsl:when test="$httpRequest/h:request/h:header[lower-case(@name)='x-
forw   arded-for']">
            <xsl:value-of select="$httpRequest/h:request/h:header[lower-case(@name)='x-
fo   rwarded-for']"/>
          </xsl:when>
          <xsl:otherwise>
            <xsl:value-of select="$httpRequest/h:request/h:client-ip"/>
          </xsl:otherwise>
        </xsl:choose>
      </key>
      <xsl:for-each select="$httpRequest/h:request/h:header">
        <xsl:variable name="hdr_name" select="@name"/>
        <xsl:if test="lower-case($hdr_name)='host'">
          <key type="host" log="true">
            <xsl:value-of select="."/>
          </key>
        </xsl:if>
        <xsl:if test="lower-case($hdr_name)='apikey'">
          <key type="apikey" log="true">
            <xsl:value-of select="."/>
          </key>
        </xsl:if>
        <xsl:if test="lower-case($hdr_name)='referer'">
          <key type="referer" log="true">
```

```

    <xsl:value-of select="."/>
  </key>
</xsl:if>
<xsl:if test="lower-case($hdr_name)='user-agent'">
  <key type="useragent" log="true">
    <xsl:value-of select="."/>
  </key>
</xsl:if>
</xsl:for-each>
</output>
</xsl:template>
</xsl:stylesheet>

```



You can use the key elements populated by the parse step XSLT transformation to pass parameters to custom policies, which can be developed leveraging the custom extension mechanism. The SampleStage custom extension in the ASGDefaultImplementation project uses the key elements.

Validate the Request Content

Using transformation (SLT) file to validate the request content.

You can use XPath expressions in the parse step XSLT transformation. Using the XPath expressions, you can apply content validation logic to the request payload message. When the request 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 a part. Then if an error XSLT is configured for the operation it is called, and any API-specific error document can be created.

Set Metric Increment for Content-Based Throttles

Using transformation(XSLT) file to set the metricContent for content-based throttles.

TIBCO API Exchange Gateway supports the notion of content-based throttles, by extending a throttle configuration with a monitor. You can define a content-based metric increment counter using the throttle configuration.

Using the content-based throttles, you can define custom throttle policies. The custom throttle policies enforce the API usage controls on business metric such as order amounts. When a throttle configuration has been extended with a monitor definition, it uses the parse step XSLT transformation at run time. The parse step XSLT transformation maps 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.

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 the 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](#) section.

Set Sticky Key for Load-Balancing with StickyResourceAffinity

Using transformation (XSLT) file to set sticky key.

TIBCO API Exchange Gateway supports multiple load-balancing policy types for target operations that are grouped within a target group. When a load-balancing policy with StickyResourceAffinity is configured for a target group, 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 needs 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](#).

Overriding HTTP Headers

Use southbound forward mapping to overwrite the HTTP headers of a facade operation request.

To overwrite the host, port, and URI and any other header fields, enable the southbound forward mapping for a target operation, as follows:

Procedure

1. Create the XSLT file, such as pass-through.xml. Refer to the [Overriding HTTP Headers](#) to override the fields in the request message.
2. Copy the XSLT file (pass-through.xml) to the `ASG_CONFIG_HOME/ASG_Project/xslt` directory.
3. Click on **MAPPING > Mapping** tab.
4. Click the **Add Property** icon.
5. Enter the parameters as follows:

Mapping Configuration to Upload XSLT File

Parameter	Description
Mapping Name	Enter a name for the mapping. For example, pass-through-map.
Type	Select XSLT from the drop-down list.
New File	Select the XSLT file. For example, pass-through.xml.
Existing Files	Select the XSLT file (pass-through.xml) from the drop-down list, if the XSLT is located in the <code>ASG_CONFIG_HOME/ASG_Project/xslt</code> directory.
Response Type	Select Full from the drop-down list.

6. Save the changes to your project.
7. Click the **PARTNER > Target Operations** tab.
8. Click the **Add Property** icon.
9. Configure forward mapping for the target operation as follows:
 - Select the mapping, such as pass-through-map in the **Request Transform** field.
10. Save the changes to your project.

Sample XSLT File

The following is an example XSLT to override the host, port, and URI :

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet
  version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:map="http://www.tibco.com/asg/mapping"
  xmlns:soap12="http://www.w3.org/2003/05/soap-envelope"
  xmlns:soap11="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:form="http://www.tibco.com/asg/functions/form"
  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/content-types/form"
  xmlns:codecs="http://www.tibco.com/asg/functions/codecs"
  exclude-result-prefixes="xsl soap11 c h form codecs">
  <xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes" omit-xml-
  declaration="no"/>
  <xsl:variable name="cnRequestHref">
    <xsl:value-of select="/transformation/cnRequest/@href"/>
  </xsl:variable>
  <xsl:variable name="context">
    <c:context>
      <xsl:for-each select="/transformation/context">
        <xsl:copy-of select="document(@href)/c:context/*"/>
      </xsl:for-each>
    </c:context>
  </xsl:variable>
  <xsl:variable name="recdRequest">
    <xsl:copy-of select="$context/c:context/c:entry[@key='asg:httpRequest']/
  h:request"/>
  </xsl:variable>
  <xsl:template match="/">
    <map:mapping-result>
      <map:context>
        <c:context>
          <xsl:for-each select="$recdRequest/h:request/h:header">
            <xsl:variable name="hdr_name" select="@name"/>
            <xsl:if test="lower-case($hdr_name)!='accept-encoding'">
              <xsl:element name="h:override-header">
                <xsl:attribute name="name"><xsl:value-of select="$hdr_name"/></
  xsl:attribute>
                <xsl:value-of select="."/>
              </xsl:element>
            </xsl:if>
          </xsl:for-each>
          <h:override-header name="Accept-Encoding">identity</h:override-header>
          <xsl:choose>
            <xsl:when test="string-length($recdRequest/h:request/h:query-string)>0">
              <h:override-URI><xsl:value-of select="concat($recdRequest/h:request/
  h:request-uri, '?', $recdRequest/h:request/h:query-string)"/></h:override-URI>
            </xsl:when>
            <xsl:otherwise>
              <h:override-URI><xsl:value-of select="$recdRequest/h:request/h:request-
  uri"/></h:override-URI>
            </xsl:otherwise>
          </xsl:choose>
          <h:override-method><xsl:value-of select="$recdRequest/h:request/
  h:method"/></h:override-method>
        </c:context>
      </map:context>
      <map:payload-xml><root><xsl:value-of select="$recdRequest/h:request/
  h:body"/></root></map:payload-xml>
    </map:mapping-result>
  </xsl:template>
</xsl:stylesheet>
```



For non-XML payloads, use the `isBinary` attribute. For example:

```
<map:payload isBinary="true"><xsl:value-of select="$recdRequest/h:request/h:body"/></map:payload>
```

Parsing XSLT Documents

Structure of XSLT document.

The XSLT transformation file used in the parse step of request processing has a predefined input document as well as a predefined output document.

Input Document

The input document used by the XSLT transformation file of the parse step has the same structure as the input document used by the other mapping and transformation steps in TIBCO API Exchange Gateway. The structure of this generic transformation and mapping input document in the [Mapping Schemas](#) section.

Output Document

The output document is unique for the XSLT transformation file of the parse step as it supports the specific behavior of TIBCO API Exchange Gateway.

The table [Elements and Attributes of Parsing Output Document](#) describes the elements and attributes of output document used for the parsing step. See [Parsing Output Document Schema](#):

Elements and Attributes of Parsing Output Document

XML Node	Type	Required	Description
/output	Sequence of Complex Type	Yes	Root element for the document.
/output/requester	String	No	This value is stored in the RequestorId attribute of the TIBCO API Exchange Gateway transaction object. This is currently not used.
/output/serviceInterface / Version	String	No	This value is stored in the ServiceInterfaceVersion attribute of the TIBCO API Exchange Gateway transaction object. This is currently not used.
/output/referenceId	String	No	The value in this field is 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.

XML Node	Type	Required	Description
/output/serviceId	String	No	This is currently not used.
/output/timestamp	String	No	This is currently not used.
/output/correlationId	String	No	This is currently not used.
/output/identityId	String	No	This is currently not used.
/output/opCoId	String	No	This value is stored in the OpCo attribute of the TIBCO API Exchange Gateway transaction object. This is currently not used.
/output/partnerId	String	No	This element is used to perform content-based identity mapping for the requester. It must contain the partner name as configured in the Config UI.
/output/routingKey	String	No	If routingKey element is present, its value is used to set the routing key for the request to determine which target operation or target group the request should be routed to.
/output/stickyRoutingKey	String	No	This element is used to set the sticky key for a request for a load-balancing policy with StickyResourceAffinity target operation type.
/output/monitor	Sequence of Complex Type	No	This is a node element for defining a content-based throttle monitor configuration.
/output/monitor/metricName	String	Yes	Specifies a throttle name for which a content-based counter increment needs to be set.
/output/monitor/metricIncrement	Integer	Yes	Specifies content-based counter increment value that is applied to the throttle as defined in the metricName element of the monitor.

XML Node	Type	Required	Description
/output/key	String	No	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 Central Logger component.
/output/key@type	String	No	Specifies the key used to store the contextual information in the Central Logger database. Its value is stored in the KEY_TYPE column of the ASG_TRANSACTION_KEYS table.
/output/key@log	Boolean	No	<ul style="list-style-type: none"> This is a Boolean field used to suppress the logging of a key in the Central Logger database. This attribute is useful when the key is used to pass parameters to custom policies that are not required to be logged. The 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	<ul style="list-style-type: none"> If the parse step returns an errorCode, 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 the Config UI and a FaultReport XSLT is defined on the Facade Operation tab, the mapping succeeds. Otherwise a default value of 1001 is 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.

XML Node	Type	Required	Description
/output/errorMessage	String	Yes	Specifies the detailed error description, which indicates that the validation check has been violated.

Parsing Output Document Schema

The following is the schema for the output document used for parsing step:

```
<?xml version="1.0" encoding="UTF-8" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://www.tibco.com/
schemas/asgMapping/Schema.xsd2" xmlns:tns="http://www.tibco.com/schemas/asg/
context" targetNamespace="http://www.tibco.com/schemas/asgMapping/Schema.xsd2"
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: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" />
          </xs:choice>
          <xs:element ref="key" minOccurs="0" maxOccurs="unbounded" />
          <xs:element ref="tns:context" />
        </xs:sequence>
        <xs:sequence>
          <xs:element ref="errorCode" />
          <xs:element ref="errorMessage" />
        </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>
        <xs:documentation>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.
    </xs:documentation>
    </xs:annotation>
</xs:element>
<xs:element name="routingKey" type="xs:string">
    <xs:annotation>
        <xs:documentation>If this element is present its 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.</xs:documentation>
    </xs:annotation>
</xs:element>
<xs:element name="stickyRoutingKey" type="xs:string">
    <xs:annotation>
        <xs:documentation>This element is used to set the sticky key for a
request for which a load-balancing policy with StickyResourceAffinity.</
xs:documentation>
    </xs:annotation>
</xs:element>
<xs:element name="monitor">
    <xs:annotation>
        <xs:documentation>Node element for defining content based throttle
monitor configuration.
    </xs:documentation>
    </xs:annotation>
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="metricName" />
            <xs:element ref="metricIncrement" />
        </xs:sequence>
    </xs:complexType>
</xs:element>
<xs:element name="metricName" type="xs:string">
    <xs:annotation>

```

```

        <xs:documentation>Throttle name for which a content based counter
increment needs to be set.</xs:documentation>
    </xs:annotation>
</xs:element>
<xs:element name="metricIncrement" type="xs:int">
    <xs:annotation>
        <xs:documentation>Content based counter increment value that will be
applied to the throttle as defined in the metricName element.</xs:documentation>
    </xs:annotation>
</xs:element>
<xs:element name="key">
    <xs:annotation>
        <xs:documentation>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:documentation>
    </xs:annotation>
    <xs:complexType>
        <xs:simpleContent>
            <xs:extension base="xs:string">
                <xs:attribute name="type" type="xs:string">
                    <xs:annotation>
                        <xs:documentation>The key used to store the contextual
information in the Central Logger database, Its value will be stored in the
KEY_TYPE column of the ASG_TRANSACTION_KEYS table.</xs:documentation>
                    </xs:annotation>
                </xs:attribute>
                <xs:attribute name="log" type="xs:boolean">
                    <xs:annotation>
                        <xs:documentation>Switch to suppress logging of a key
in the Central Logger database. The default value is True, including when you omit
the attribute, as such its only impact is when you set it to False.This is useful
when key is used to pass parameters to custom policies that don't need to be logged.
                    </xs:documentation>
                </xs:annotation>
            </xs:attribute>
        </xs:extension>
    </xs:simpleContent>
</xs:complexType>
</xs:element>
<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>
</xs:schema>

```



Overriding headers at parse step is not supported.

Mappings and Transformations

TIBCO API Exchange Gateway provides message transformations using the mappings. Using the mappings, you can 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 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 the following four points:

- Facade Request Handler to Router Boundary
After the request has been received by the 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 a 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

You can do the following actions using the mapping transformations:

- 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 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 as follows:

- Forward mapping - The transformations are done from request canonical form to back-end service API as per defined mapping
- Reverse mapping - The transformations are done from the back-end service API to the response canonical form as per defined mapping

The mapper of TIBCO API Exchange Gateway provides the six different versions of documents as follows:

- Northbound request message
- Canonical request message
- Southbound request message
- Southbound response message
- Canonical response message
- Nouthbound response message

TIBCO API Exchange Gateway supports mapping at both northbound and southbound sides.

Northbound Mapping

Using the northbound mapping, you can map as follows:

- 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 Config UI.

Southbound Mapping

Using the southbound mapping, you can map as follows:

- 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 Config UI.

Mapping Types

The following types of mapping are supported:

Rendezvous (RV) Mapping

Using the Rendezvous mapping, you can 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 Rendezvous messages that transform the given input messages to other messages. The Core Engine calls the mappers services whenever it transforms messages between the 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 \(XSLT Mapping\)](#) for details.

Mapping Configuration

Using the Config UI, you can add the mapping configuration for the 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 the gateway endpoint operation and the gateway service endpoint.

Rendezvous Mapping Type

Configuring Rendezvous Mapping Type.

Procedure

1. Start the GUI, if not already started. See [Starting GUI](#) for details.
2. Click the **MAPPING > Mapping** tab for a project.
3. Type the parameters, as follows:

RV Mapping Type Configuration

Parameter	Description
Mapping Name	The name of mapping
Type	RV (select from the drop-down list.)
Subject	Specifies the RV subject to send the mapping request to
Transformation Name	The transformation to perform on the message. If it is default, the transformation is determined by mpSubject. Other values force specific transformation.

XSLT Mapping Type

Configuring XSLT Mapping Type.

Procedure

1. Start the GUI, if not already started. See [Starting GUI](#) for details.
2. Click the **MAPPING > Mapping** tab for a project.
3. Type the parameters, as follows:

XSLT Mapping Type Configuration

Parameter	Description
Mapping Name	The name of mapping.
Type	XSLT (select from the drop-down list).
New File	The file name containing the transformations (XSLT file).
Existing Files	The file name containing the transformations (XSLT file).
Response Type	<p>The output type from the transformation. This can have the following values:</p> <ul style="list-style-type: none"> • Payload: the output document from the transformation contains just the payload. • Full: the output document from the transformation contains both the request context and the payload. <p><Details> (Payload Full)</p>

Assign to the Gateway Operation Endpoint

Selecting a mapping for a facade operation.

After the mappings are registered, they are assigned to the gateway operation endpoint for forward and reverse mappings to be performed for request operations. The transformation can be applied at only one level for a given facade operation. If applied at both levels, the facade operation

transformation takes precedence over facade access transformation. They are defined in the **Facade Operations and Facade Access** tab of the GUI. See [Facade Operations](#) and [Facade Access](#).

Assign to the Gateway Reference Endpoint

Selecting a mapping for a target operation.

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 [Adding a New Target Operation](#).

Transformations (XSLT Mapping)

Response type for transformations.

To define the XSLT mapping for facade or target request and response messages, files are defined in the **Mapping** tab of the Config UI. See [XSLT Mapping Type](#).

For the XSLT mapping, the **Response Type** indicates the output of the transformations, which can be as follows:

Payload

If the **Response Type** is specified as Payload, the output document of the 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 on 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 **Response Type** is specified as Full, you can access additional fields from the transaction object. The main field is the request context field which contains the HTTP headers information, URI, user name, and other authentication fields.

This provides the mapping and transformations of the full transaction (request) object. The transaction context contains the header information and the payload. After the transformation is done, the Core Engine processes the request as follows:

- 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.

You can do the following using the **Response Type** as Full transformation:

- Map and update the document with additional fields available in the request context (HTTP headers, JMS headers, and so on).
- Set error code for content validation.
- Create documents for enumeration orchestration.

The following is an example XSLT of a northbound request transformation:

```
<xsl:variable name="nbRequestHref">
  <xsl:value-of select="/transformation/nbRequest/@href" />
</xsl:variable>
<xsl:variable name="nbRequest">
  <xsl:copy-of select="document($nbRequestHref)/*" />
</xsl:variable>
<xsl:variable name="RequestParam">
  <xsl:value-of select="$nbRequest//book:Request" />
</xsl:variable>
```

```

</xsl:variable>
<xsl:template match="/">
  <map:mapping-result>
    <map:context>
      <c:context>
        <c:entry key="asg:httpRequest">
          <h:override-URI>
            <xsl:value-of select="concat('/Books/BookOperations/Title/',
$RequestParam)" />
          </h:override-URI>
        </c:entry>
        <c:entry key="http:override">
          <h:override-header name="accept">application/xml</h:override-
header>
          <h:override-header name="content-type">application/xml</
h:override-header>
        </c:entry>
      </c:context>
    </map:context>
    <map:payload-xml>
      <html/>
    </map:payload-xml>
  </map:mapping-result>
</xsl:template>

```

The template includes the following elements:

- nbRequestHref - This element represents northbound request href attribute
- nbRequest - This element represents the northbound request payload.
- RequestParam - This element represents request parameters

The following functionality is implemented using the Full **Response Type** of mapping transformation capability:

Set error codes for content validation

With the mapping transformation with **Response Type** as Full, you can 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 the 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 the 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, and is sent back to the original requester with the error message. See [Implementing Request Validation](#).

Validation

Using the validation feature, you can 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

Procedure

1. Set Operation Features for the Operation

In the **ROUTING > Facade Operations** tab of the Config UI, set the **Operation Features** parameter as follows:

Operation Features Validation

2. Configure Schema

In the **MAPPING > Schemas** tab of the Config UI, set the XSD file reference as follows:

New XSD File *location of the XSD file*

Implementing Request Validation

Validation of requests is supported for the following stages:

- Request Parsing

The request message can be validated at the request parsing stage. The following elements exist in the schema for the output message to be used by the Parse XSLT:

```
<errorCode>
<errorMessage>
```

If `errorCode` is set, 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, follow these steps:

- 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 Core 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 the Parsing step, you can only set the **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. Using the additional elements, **errorBody** and **errorDetails** elements you can 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 has to be executed.

Map the Protocol Headers in Request Context

You can transform the protocol headers within the request and response messages, which includes the transport related information.

Use the `responseType` as `Full` to access the request context field. The request context field contains the transport level information and is available for any transformations. The request context field is used for the following purposes:

- 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.

The following example shows how to convert the JMS message element to HTTP header:

```
<xsl:template match="/">
  <map:mapping-result>
    <map:context>
      <c:context>
        <c:entry key="asg:httpResponse">
          <h:response>
            <h:status-code>200</h:status-code>
            <h:header name="X-Powered-By">Servlet/2.5 JSP/2.1</h:header>
            <h:header name="Content-Type">
              <xsl:value-of select="concat($sbContentType, '; start-info=text/
xml ')" />
            </h:header>
            <h:body>
              <xsl:value-of select="$jmsResponse/k:body" />
            </h:body>
          </h:response>
        </c:entry>
      </c:context>
    </map:context>
  </map:mapping-result>
</xsl:template>
```

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 might 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.

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 Core 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. Enumeration orchestration modifies the routing of sub-requests to a different service at the southbound forward mapping boundary.

The following example shows the snippet of forward southbound mapping transformation for enumeration orchestration:

```
<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 rejoin the multiple response documents for each sub-request into a single response document.

Response Transformation

Use the following tag in your reverse northbound mapping to apply transformations to a response before sending it out.

Use reverse northbound mapping to apply transformations to a response before sending it out. As an example, you can add a status code by specifying the following element in the mapping file:

```
<h:status-code>...</h:status-code>
```

Mapping Schemas

This section lists the schemas for the transformation (XSLT) files.

Mapping Container

The input document to the XSLT file for XSLT transformations is described by the following schema(XSD):

```

<?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:schema>

```

```

</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, use the following XSLT snippet to map a value when the received request is SOAP :

```

<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>

```

Mapping XSLT Schema

Any type of transformation can be done using the following XSLT schema:

```

<?xml version="1.0" encoding="UTF-8" ?>
<xsd:schema xmlns="http://www.w3.org/1999/xhtml" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:tns="http://www.tibco.com/asg/mapping" targetNamespace="http://www.tibco.com/asg/mapping" elementFormDefault="qualified" attributeFormDefault="unqualified">
  <xsd:element name="mapping-result" type="tns:mapping-result" />
  <xsd:complexType name="mapping-result">
    <xsd:sequence>
      <xsd:element name="context" minOccurs="0">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:any processContents="lax" />
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:choice>
        <xsd:element name="payload-xml" minOccurs="0" maxOccurs="unbounded">
          <xsd:complexType>

```

```

        <xsd:sequence>
            <xsd:any processContents="lax" />
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
<xsd:element name="payload-text" minOccurs="0"
maxOccurs="unbounded" type="xsd:string" />
</xsd:choice>
</xsd:sequence>
</xsd:complexType>
</xsd:schema>

```

Context Document

The contents of the context document are described by the following XSD:

```

<?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:element name="entry">
        <xs:complexType>
            <xs:sequence>
                <xs:any namespace="##any" processContents="lax" />
            </xs:sequence>
            <xs:attribute name="key" type="xs:string" />
        </xs:complexType>
    </xs:element>
    <xs:simpleType name="key">
        <xs:restriction base="xs:string" />
    </xs:simpleType>
</xs:schema>

```

If the incoming request is received from an HTTP server, the contents of the request context document are described by the following XSD:

```

<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
targetNamespace="http://www.tibco.com/asg/protocols/http" xmlns:xs="http://
www.w3.org/2001/XMLSchema">
    <xs:element name="request">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="request-id" type="xs:short" minOccurs="0"/>
                <xs:element name="server-ip" type="xs:string" minOccurs="0"/>
                <xs:element name="server-port" type="xs:short" minOccurs="0"/>
                <xs:element name="client-ip" type="xs:string" minOccurs="0"/>
                <xs:element name="client-port" type="xs:byte" 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="query-string" type="xs:string" minOccurs="0"/>
                <xs:element name="header" maxOccurs="unbounded" minOccurs="0">
                    <xs:complexType>
                        <xs:simpleContent>
                            <xs:extension base="xs:string">
                                <xs:attribute name="name" type="xs:string" use="optional"/>
                            </xs:extension>
                        </xs:simpleContent>
                    </xs:complexType>
                </xs:element>
                <xs:element name="payloadSize" type="xs:short" minOccurs="0"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>

```

```
</xs:element>
</xs:schema>
```

The following is an HTTP Request Context sample:

```
<?xml version="1.0" encoding="utf-8"?>
<ctx:context xmlns:ctx="http://www.tibco.com/schemas/asg/context">
  <!--Context doc href: context.0.xml-->
  <c:entry xmlns:c="http://www.tibco.com/schemas/asg/context"
key="asg:httpRequest">
    <c:origin>i_http</c:origin>
    <h:request xmlns:h="http://www.tibco.com/asg/protocols/http">
      <h:request-id>2192</h:request-id>
      <h:server-ip>localhost</h:server-ip>
      <h:server-port>9222</h:server-port>
      <h:client-ip>127.0.0.1</h:client-ip>
      <h:client-port>0</h:client-port>
      <h:scheme>http</h:scheme>
      <h:method>POST</h:method>
      <h:request-uri>/TerminalLocationService/services/TerminalLocation</
h:request-uri>
      <h:protocol-version>HTTP/1.1</h:protocol-version>
      <h:query-string/>
      <h:header name="content-type">text/xml; charset="utf-8"</h:header>
      <h:header name="soapaction">"/getLocation"</h:header>
      <h:header name="connection">close</h:header>
      <h:header name="user-agent">Jakarta Commons-HttpClient/3.0.1</h:header>
      <h:header name="host">localhost:9222</h:header>
      <h:header name="content-length">489</h:header>
      <h:payloadSize>489</h:payloadSize>
    </h:request>
  </c:entry>
</ctx:context>
```

The contents of HTTP Response Context document are described by the following XSD:

```
<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
targetNamespace="http://www.tibco.com/asg/protocols/http" xmlns:xs="http://
www.w3.org/2001/XMLSchema">
  <xs:element name="response">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="status-code" type="xs:short" minOccurs="0"/>
        <xs:element name="all_headers" type="xs:string" minOccurs="0"/>
        <xs:element name="header" maxOccurs="unbounded" minOccurs="0">
          <xs:complexType>
            <xs:simpleContent>
              <xs:extension base="xs:string">
                <xs:attribute name="name" type="xs:string" use="optional"/>
              </xs:extension>
            </xs:simpleContent>
          </xs:complexType>
        </xs:element>
        <xs:element name="payloadSize" type="xs:short" minOccurs="0"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

The following is an HTTP Response Context sample:

```
<?xml version="1.0" encoding="utf-8"?>
<ctx:context xmlns:ctx="http://www.tibco.com/schemas/asg/context">
  <!--Context doc href: context.0.xml-->
  <c:entry xmlns:c="http://www.tibco.com/schemas/asg/context"
key="asg:sbHttpResponse">
    <h:response xmlns:h="http://www.tibco.com/asg/protocols/http">
      <h:status-code>200</h:status-code>
      <h:all_headers>Date: Fri, 29 Apr 2016 00:15:16 GMT
Content-Length: 486
Content-Type: text/xml; charset=utf-8
Server: Apache-Coyote/1.1

      </h:all_headers>
```

```

        <h:header name="Date">Fri, 29 Apr 2016 00:15:16 GMT</h:header>
        <h:header name="Content-Length">486</h:header>
        <h:header name="Content-Type">text/xml; charset=utf-8</h:header>
        <h:header name="Server">Apache-Coyote/1.1</h:header>
        <h:payloadSize>486</h:payloadSize>
    </h:response>
</c:entry>
</ctx:context>

```

If the incoming request is received from a JMS server, the contents of the request context document are described by the following XSD:

```

<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
targetNamespace="http://www.tibco.com/asg/protocols/jms" xmlns:xs="http://
www.w3.org/2001/XMLSchema">
  <xs:element name="request">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="request-id" type="xs:int" minOccurs="0"/>
        <xs:element name="JMSCorrelationID" type="xs:string" minOccurs="0"/>
        <xs:element name="JMSReplyTo" type="xs:string" minOccurs="0"/>
        <xs:element name="body" type="xs:string" minOccurs="0"/>
        <xs:element name="payloadSize" type="xs:short" minOccurs="0"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>

```

The following is a JMS Request Context sample:

```

<?xml version="1.0" encoding="utf-8"?>
<c:context xmlns:c="http://www.tibco.com/schemas/asg/context">
  <c:entry key="asg:jmsRequest">
    <j:request xmlns:j="http://www.tibco.com/asg/protocols/jms">
      <j:request-id>40709</j:request-id>
      <j:JMSCorrelationID>GetLocation_118009</j:JMSCorrelationID>
      <j:JMSReplyTo>asg.out.request.reply.0.0</j:JMSReplyTo>
      <j:body>40709req</j:body>
      <j:payloadSize>397</j:payloadSize>
    </j:request>
  </c:entry>
</c:context>

```

The contents of JMS Response Context document are described by the following XSD:

```

<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
targetNamespace="http://www.tibco.com/asg/protocols/jms" xmlns:xs="http://
www.w3.org/2001/XMLSchema">
  <xs:element name="message">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="JMSCorrelationID" type="xs:int" minOccurs="0"/>
        <xs:element name="Service" type="xs:string" minOccurs="0"/>
        <xs:element name="ServiceInstance" type="xs:string" minOccurs="0"/>
        <xs:element name="Operation" type="xs:string" minOccurs="0"/>
        <xs:element name="JMSExpiration" type="xs:byte" minOccurs="0"/>
        <xs:element name="body">
          <xs:complexType>
            <xs:simpleContent>
              <xs:extension base="xs:string">
                <xs:attribute name="size" type="xs:short"/>
                <xs:attribute name="isBinary" type="xs:string"/>
              </xs:extension>
            </xs:simpleContent>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>

```

The following is a JMS Response Context sample:

```
<?xml version="1.0" encoding="utf-8"?>
<c:context xmlns:c="http://www.tibco.com/schemas/asg/context">
  <c:entry key="asg:sbJMSResponse" xmlns:c="http://www.tibco.com/schemas/asg/
context">
    <k:message xmlns:k="http://www.tibco.com/asg/protocols/jms">
      <k:JMSCorrelationID>40724</k:JMSCorrelationID>
      <k:Service/>
      <k:ServiceInstance/>
      <k:Operation/>
      <k:JMSExpiration>0</k:JMSExpiration>
      <k:body size="371" isBinary="false"></k:body>
    </k:message>
  </c:entry>
</c:context>
```

JSON XML Transformation

Functions to transform JSON XML messages.

TIBCO API Exchange Gateway can handle non-XML request and response messages, and provide XSLT extension functions to handle non-XML message serializations.

Using the mapping feature of TIBCO API Exchange Gateway, you can transform the messages as follows:

- Request message from one message format to other format using the forward mapper. For example, use the northbound forward mapper to modify the JSON request to an XML request. See [BookQueryBE Example Request Message](#).
- Response message from one message format to other format using the reverse mapper. For example, use northbound reverse mapper to modify the XML response message format to a JSON response message format. See [BookQuery Example Response Message](#).

To support the JSON XML message transformations, TIBCO API Exchange Gateway provides the following options:

JSON Extension

The JSON extension is defined in the namespace `http://www.tibco.com/asg/functions/json`

XSLT Functions

TIBCO API Exchange Gateway provides the following two functions:

- `parse()`
The `parse()` function converts the JSON message format to XML message format.
- `render()`
The `render()` function converts the XML message format to JSON message format.



The XML message format has a 1-to-1 correspondence with the JSON structure.

JSON Schema

TIBCO API Exchange Gateway provides the following built-in schema for JSON message:

```
<?xml version="1.0" encoding="UTF-8" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:json="http://
www.tibco.com/asg/content-types/json" targetNamespace="http://www.tibco.com/asg/
content-types/json" elementFormDefault="qualified"
attributeFormDefault="unqualified">
  <xs:element name="json_xml">
    <xs:complexType>
      <xs:sequence>
```



```

        <xs:element ref="json:list" />
    </xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name="list">
    <xs:complexType>
        <xs:choice minOccurs="0" maxOccurs="unbounded">
            <xs:element ref="json:text" />
            <xs:element ref="json:double" />
            <xs:element ref="json:list" />
            <xs:element ref="json:dict" />
        </xs:choice>
        <xs:attribute name="key" type="xs:string" />
    </xs:complexType>
</xs:element>
<xs:element name="text">
    <xs:complexType>
        <xs:simpleContent>
            <xs:extension base="xs:string">
                <xs:attribute name="key" type="xs:string" />
            </xs:extension>
        </xs:simpleContent>
    </xs:complexType>
</xs:element>
<xs:element name="double">
    <xs:complexType>
        <xs:simpleContent>
            <xs:extension base="xs:string">
                <xs:attribute name="key" type="xs:string" />
            </xs:extension>
        </xs:simpleContent>
    </xs:complexType>
</xs:element>
<xs:element name="dict">
    <xs:complexType>
        <xs:choice minOccurs="0" maxOccurs="unbounded">
            <xs:element ref="json:text" />
            <xs:element ref="json:double" />
            <xs:element ref="json:list" />
        </xs:choice>
        <xs:attribute name="key" type="xs:string" />
    </xs:complexType>
</xs:element>
</xs:schema>

```

Converting XML Message to JSON Message

By default, TIBCO API Exchange Gateway sends the response message in XML format from the target operation to the client. If a client requests the response message in a JSON format, you can convert the XML message to a JSON message as follows:

Procedure

1. Create an XSLT File.

To create an XSLT file to render the data in JSON format, follow these high level steps:

- Add namespaces:

The render() function requires the following namespace: <http://www.tibco.com/asg/functions/json>

- Build JSON structure.
- Use the render() function:

See [Example XSLT to Convert BookQuery XML Response to JSON Response](#) for an example XSLT.

2. Upload the XSLT File.

Using the Config UI, upload the XSLT file in the northbound reverse mapper to render the XML response message in JSON message format as follows:

- a) Start the GUI server, if not already running.
- b) Log in to the Config UI using your credentials.
- c) Add a new project or select an existing project under **Projects**. For example, select BookQuery project.
- d) Add the mapping configuration as follows:
 1. Click the **MAPPING > Mapping** tab.
 2. Click the **Add property** to create a new mapping.
 3. Enter the parameters, as follows:
 - **Mapping Name:** XML_JSON_Mapping
 - **Type:** XSLT (select from the drop-down list.)
 - **New File:** Click Choose File and select the XSLT file.
 - **Response Type:** Full (select from the drop-down list.)
 - Save the changes to your configuration.
- e) Select the mapper for the facade operation, as follows:
 1. Click the **ROUTING > Facade Operations** tab.
 2. Select the facade operation.
 3. In the **Response Transform** field, select the **XML_JSON_Mapping** mapper, as created in the mapping configuration from the drop-down list.
 4. Save the changes to your configuration.

BookQuery Example Response Message

When the BookQuery example is run, TIBCO API Exchange Gateway sends the response message in XML format to the client, by default.

BookQuery Example XML Response

BookQuery Response message in XML format.

The response message in XML format is shown as follows:

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Body>
    <ns0:BookStore xmlns:ns0="http://www.example.com/xsd/books">
      <ns0:Book>
        <ns0:Title>The Power of Now</ns0:Title>
        <ns0:Author>Vivek Ranadive</ns0:Author>
        <ns0:Date>1999</ns0:Date>
        <ns0:ISBN>0-06-566778-9</ns0:ISBN>
        <ns0:Publisher>Tibco Software Inc</ns0:Publisher>
      </ns0:Book>
    </ns0:BookStore>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Transformed JSON Response for BookQuery Service:

```
{
  "BookStore": {
    "Book": [
      {
        "Publisher": "Tibco Software Inc",
        "ISBN": "0-06-566778-9",
        "Author": "Vivek Ranadive",
```

```

        "Title": "The Power of Now"
    }
  ]
}

```

Example XSLT to Convert BookQuery XML Response to JSON Response

Use the following XSLT to convert XML message format to JSON message format:

```

<xsl:stylesheet version="1.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform" xmlns:ns0="http://
www.example.com/xsd/books" xmlns:fn="http://www.tibco.com/asg/functions/json"
xmlns:json="http://www.tibco.com/asg/content-types/json" xmlns:SOAP-ENV="http://
schemas.xmlsoap.org/soap/envelope/" exclude-result-prefixes="xsl fn json ns0 SOAP-
ENV">
  <xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes" omit-
xmldeclaration="
yes" />
  <xsl:variable name="cnResponseHref">
    <xsl:value-of select="/transformation/cnResponse/@href" />
  </xsl:variable>
  <xsl:variable name="cnResponse">
    <xsl:copy-of select="document($cnResponseHref)/*" />
  </xsl:variable>
  <xsl:variable name="JsonXmlResponse">
    <json:json_xml>
      <json:dict>
        <json:dict key="BookStore">
          <json:list key="Book">
            <xsl:for-each select="$cnResponse/SOAP-ENV:Envelope/SOAP-
ENV:Body/ns0:BookStore/
ns0:Book">
              <json:dict key="Book">
                <json:text key="Title">
                  <xsl:value-of select="ns0:Title" />
                </json:text>
                <json:text key="ISBN">
                  <xsl:value-of select="ns0:ISBN" />
                </json:text>
                <json:text key="Author">
                  <xsl:value-of select="ns0:Author" />
                </json:text>
                <json:text key="Publisher">
                  <xsl:value-of select="ns0:Publisher" />
                </json:text>
              </json:dict>
            </xsl:for-each>
          </json:list>
        </json:dict>
      </json:dict>
    </json:json_xml>
  </xsl:variable>
  <xsl:template match="/">
    <xsl:value-of select="fn:render($JsonXmlResponse//json:json_xml/
json:dict)" />
  </xsl:template>
</xsl:stylesheet>

```

BookQuery Example JSON Transformed Response

JSON response message for BookQuery example.

The following is the transformed JSON response message from the XSLT defined as [Example XSLT to Convert BookQuery XML Response to JSON Response](#).

```

{
  "BookStore": {
    "Book": [
      {
        "Publisher": "Tibco Software Inc",

```

```

    "ISBN": "0-06-566778-9",
    "Author": "Vivek Ranadive",
    "Title": "The Power of Now"
  }
]
}
}

```

Converting JSON Message to XML Message

Using the northbound forward mapper, parse the JSON encoded string to generate an XML message. Usually, the JSON data is encoded using base64 format, therefore, the data must be converted from base64 to text. TIBCO API Exchange Gateway provides the `codecs:base64ToText()` function to convert the json data to text.

To parse a JSON encoded string in a north-side forward mapper, follow these steps:

Procedure

1. Create an XSLT File.

To create an XSLT file to parse the JSON encoded, perform the following steps:

- Add namespaces:

The `parse()` function requires the following namespaces: `http://www.tibco.com/asg/functions/json`

`http://www.tibco.com/asg/functions/codecs`

- Use the `parse()` function as follows:
 - Extract the base64 encoded request payload from the context document.
 - Use the `codecs:base64ToText()` function to convert the payload to text message.
 - Pass the text message to the `json:parse()` function.

See [Example XSLT to Convert BookQueryBE JSON Request to XML Request](#) for an example XSLT.

2. Upload the XSLT File.

Using the Config UI, upload the XSLT file in the northbound forward mapper to parse the JSON encoded request message in XML message format as follows:

- Start the GUI server, if not already running.
- Log in to the Config UI using your credentials.
- Add a new project or select an existing project under **Projects**. For example, select the BookQueryBE project.
- Add the mapping configuration as follows:

- Click the **MAPPING > Mapping** tab.
- Click the **Add property** to create a new mapping.
- Enter the parameters, as follows:
 - **Mapping Name:** JSON_XML_Mapping.
 - **Type:** XSLT (select from the drop-down list.).
 - **New File:** Click Choose File and select the XSLT file.
 - **Response Type:** Full (select from the drop-down list.)
 - Save the changes to your configuration.

- Select the mapper for the facade operation, as follows:

1. Click the **ROUTING > Facade Operations** tab.
2. Select the facade operation.
3. In the **Request Transform** field, select **JSON_XML_Mapping** mapper, as created in the mapping configuration from the drop-down list.
4. Save the changes to your configuration.

BookQueryBE Example Request Message

For example, when the BookQueryBE example is run, the client can send a request message in JSON format. This section describes how to convert the JSON data to an XML request using the `json:parse()` function for the BookQueryBE example.

BookQueryBE Example JSON Message

BookQueryBE response message in JSON format.

The client sends the following request message in JSON encoded string:

```
{
  "BookStore": {
    "Book": [
      {
        "Publisher": "Tibco Software Inc",
        "ISBN": "0-06-566778-9",
        "Author": "Vivek Ranadive",
        "Title": "The Power of Now"
      }
    ]
  }
}
```

Example XSLT to Convert BookQueryBE JSON Request to XML Request

Use the following XSLT to convert the JSON message to an XML message format:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:fn="http://www.w3.org/2005/xpath-functions"
  xmlns:j="http://www.tibco.com/asg/content-types/json"
  xmlns:f="http://www.tibco.com/asg/content-types/form"
  xmlns:m="http://www.tibco.com/asg/mapping"
  xmlns:c="http://www.tibco.com/schemas/asg/context"
  xmlns:h="http://www.tibco.com/asg/protocols/http"
  xmlns:v="http://tag-pg.vipnet.hr/pg/content-types/formdata"
  xmlns:form="http://www.tibco.com/asg/functions/form"
  xmlns:json="http://www.tibco.com/asg/functions/json"
  xmlns:codecs="http://www.tibco.com/asg/functions/codecs"
  exclude-result-prefixes="xsl soapenv fn h c j f v form json" >
  <xsl:variable name="context">
    <xsl:for-each select="/transformation/context">
      <xsl:copy-of select="document(@href)"/>
    </xsl:for-each>
  </xsl:variable>
  <xsl:variable name="httpRequest">
    <xsl:copy-of select="$context/c:context/c:entry[@key='asg:httpRequest']/"
    h:request"/>
  </xsl:variable>
  <xsl:template match="/">
    <xsl:if test="$httpRequest/h:request/h:body">
      <xsl:copy-of select="json:parse(codecs:base64ToText($httpRequest/h:request/
      h:body))"/>
    </xsl:if>
  </xsl:template>
</xsl:stylesheet>
```

BookQueryBE Example XML Transformed Request

JSON response message for BookQueryBE example.

The following is the transformed XML request message from the XSLT defined as [Example XSLT to Convert BookQueryBE JSON Request to XML Request](#).

```
<?xml version="1.0" encoding="UTF-8" ?>
<dict xmlns="http://www.tibco.com/asg/content-types/json">
  <dict key="BookStore">
    <list key="Book">
      <dict>
        <text key="Publisher">Tibco Software Inc</text>
        <text key="ISBN">0-06-566778-9</text>
        <text key="Author">Vivek Ranadive</text>
        <text key="Title">The Power of Now</text>
      </dict>
    </list>
  </dict>
</dict>
```

XSLT Functions for URL Encode and URL Decode

To encode or decode any special characters in the headers or the payload body of the request message, TIBCO API Exchange Gateway provides the following XSLT functions to be used in the XSLT file:

- decode()
- encode()

The namespace for the above functions is as follows:

<http://www.tibco.com/asg/functions/url>

Decode() Function

The decode function decodes any URL-encoded characters in the input string passed as an argument.

For example, if a string contains the %20 character , the decode () function decodes it to the space character.

In the following example, Single%20general%20admission is converted as Single general admission.

Sample XSLT (Decode)

The following is the sample XSLT for the decode function, which decodes any URL-encoded characters in the <payload> element of request message.

In the following example, the description field is defined as Single%20general%20admission%20theater%20ticket. The decode() function transforms the value of description field as Single general admission theater ticket.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<xsl:stylesheet version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:url="http://www.tibco.com/asg/content-types/url" xmlns:f="http://
www.tibco.com/asg/functions/url" exclude-result-prefixes="xsl f">
  <xsl:output indent="yes" method="xml" />
  <xsl:template match="/">
    <context>
      <entry name="request">
        <http>
          <payload>
            <xsl:copy-of select="f:decode(context/
entry[@name='request']/http/payload)" />
          </payload>
        </http>
      </entry>
```

```

    </context>
  </xsl:template>
</xsl:stylesheet>

```

Sample Input XML(Decode)

The following is the sample input XML for the decode function:

```

<?xml version="1.0" encoding="UTF-8" ?>
<context>
  <entry name="request">
    <http>

    <payload>endUserId=acr:Authorization&transactionOperationStatus=charged&description=
    Single%20general%20admission%20theater%20ticket&code=wac-11faf3e6-
    e440-4daa-824e-62d8ed83723e&referenceCode=REF-ASM600-239238&onBehalfOf=WAC
    %20Cinemas %20Inc&amp;purchaseCategoryCode=Ticket&channel=WAP
    </payload>
    </http>
  </entry>
</context>

```

Sample Output XML(Decode)

The following is the sample output XML from the decode function:

```

<?xml version="1.0" encoding="UTF-8"?>
<context xmlns:url="http://www.tibco.com/asg/content-types/url">
  <entry name="request">
    <http>
      <payload>
        <url xmlns="http://www.tibco.com/asg/content-types/
url">endUserId=acr:Authorization&transactionOperationStatus=charged&description=Sing
le general admission theater ticket&code=wac-11faf3e6-
e440-4daa-824e-62d8ed83723e&referenceCode=REF-ASM600-239238&onBehalfOf=WAC Cinemas
Inc&purchaseCategoryCode=Ticket&channel=WAP
</url>
      </payload>
    </http>
  </entry>
</context>

```

Encode() Function

The encode() function encodes the special characters in the input string to the URL-encoded characters.

For example, if a string contains the backspace character , the encode () function encodes it to the %08 character.

Sample XSLT (Encode)

The following is the sample XSLT for the encode function, which encodes the special characters in the <payload> element of request message to the URL- encoded characters.

In the following example, the description field is defined as Single general admission theater ticket. The encode() function transforms the value of description field as Single%20general%20admission %20theater%20ticket.

```

<?xml version="1.0" encoding="ISO-8859-1" ?>
<xsl:stylesheet version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:url="http://www.tibco.com/asg/content-types/url" xmlns:f="http://
www.tibco.com/asg/functions/url" exclude-result-prefixes="xsl f">
  <xsl:output indent="yes" method="xml" />
  <xsl:template match="/">
    <context>
      <entry name="request">
        <http>
          <payload>

```

```

                <xsl:copy-of select="f:encode(context/
entry[@name='request']/http/payload)" />
            </payload>
        </http>
    </entry>
</context>
</xsl:template>
</xsl:stylesheet>

```

Sample Input XML (Encode)

The following is the sample input XML for encode function:

```

<?xml version="1.0" encoding="UTF-8"?>
<context xmlns:url="http://www.tibco.com/asg/content-types/url">
    <entry name="request">
        <http>

<payload>endUserId=acr:Authorization&transactionOperationStatus=charged&description=
Single general admission theater ticket&code=wac-11faf3e6-
e440-4daa-824e-62d8ed83723e&referenceCode=REF-ASM600-239238&onBehalfOf=WAC Cinemas
Inc&purchaseCategoryCode=Ticket&channel=WAP
        </payload>
    </http>
</entry>
</context>

```

Sample Output XML (Encode)

The following is the sample output XML from the encode function:

```

<?xml version="1.0" encoding="UTF-8" ?>
<context>
    <entry name="request">
        <http>
            <payload>endUserId=acr:Authorization
%26amp;transactionOperationStatus=charged %26amp;description=Single%20general
%20admission%20theater%20ticket %26amp;code=wac-11faf3e6-e440-4daa-824e-62d8ed83723e
%26amp;referenceCode=REFASM600- 239238%26amp;onBehalfOf=WAC%20Cinemas%20Inc
%26amp;purchaseCategoryCode=Ticket %26amp;channel=WAP
            </payload>
        </http>
    </entry>
</context>

```

Using Encode() and Decode() Functions

Procedure

1. Create an XSLT File.

Create an XSLT file with `decode()` and `encode()` functions, as required. Refer to the following XSLT files to create an XSLT file:

- [Sample XSLT \(Decode\)](#)
- [Sample XSLT \(Encode\)](#)

2. Upload the XSLT File.

To upload the XSLT file for a facade operation, follow these steps:

- a) Start the Config UI, if not running.
- b) Log in to the Config UI using your credentials.
- c) Select the project under Projects.
- d) On the **ROUTING** tab, click **Routing** on the top menu.
Select the **Facade Operations** tab.

- e) In the **New ProcessBody Transform** field, upload the XSLT file.
- f) Save the changes to the project configuration.

XSLT Functions for Base64 Encode and Decode

To encode or decode any Base64 fields in an XSLT file, TIBCO API Exchange Gateway provides the following XSLT functions:

- `codecs:base64ToText()`
- `codecs:textToBase64()`

To use these functions, add the following namespace to the XSL style sheet: `xmlns:codecs="http://www.tibco.com/asg/functions/codecs"`

textToBase64()

The `codecs:textToBase64()` function encodes any ASCII (text) string in Base64 format.

base64ToText()

The `codecs:base64ToText()` function decodes any Base64 encoded string in ASCII (text) format.

Sample XSLT

```
<xsl:variable name="sbResponse">
<xsl:copy-of select="codecs:base64ToText($httpResponse/h:body)"/>
</xsl:variable>
```

Custom Java Functions

Using Java functions in transformations.

TIBCO API Exchange Gateway bundles the Saxon Professional Edition (PE) product libraries with the installation package to support the Java functions in the XSLT files.

To create your own Java functions and use the functions in the XSLT files, complete the following steps:

Procedure

1. Create the function in the Java program. The function must have static modifier.
2. Create the jar file for the Java class. Put the jar file in the `ASG_HOME/lib/ext/tpcl` directory.
3. Call the function using the package name as namespace in the XSLT file. See the following example:

Java Function

Sample Java Function

The HelloWorld java function is defined as follows:

```
package Test;
public class HelloWorld {
public static String getName(){
return "Hello";
}
}
```

XSLT File

Sample XSLT File

Use the HelloWorld function in an XSLT file as follows:

```
<routingKey>
<xsl:value-of select="nameUtils:getName()"
xmlns:nameUtils="java:Test.HelloWorld"/>
</routingKey>
```

Pass-Through Gateway

TIBCO API Exchange Gateway acts as a pass-through gateway for the REST, SOAP/HTTP, SOAP/JMS, and ESB facade operation requests, which contains the URI or SOAPAction that do not match the operation URI or SOAP Action of the facade operation configured in the gateway.

TIBCO API Exchange Gateway supports the pass-through functionality using the DefaultOperation facade operation. You must configure a DefaultOperation facade operation for your project using the Config UI. See [Configuring DefaultOperation Facade Operation](#).

To enable the TIBCO API Exchange Gateway as a pass-through gateway, complete these tasks:

Starting Config UI

Procedure

1. Start the Config UI. See [Starting GUI](#).
2. Log in to the Config UI using your credentials.

Enabling Default Operation

Enable the DefaultOperation feature as follows:

Procedure

1. Select the **Gateway Engine Properties** from the drop-down list.
2. Click the **General** link.
3. Expand the **Common** node.
4. Select the **Enable Default Operation** field.

To set the runtime property in the asg.properties file, follow these steps:

5. Navigate to the *ASG_CONIG_HOME* directory.
6. Edit the asg.properties file in a text editor.
7. Set the following property to true:
8. `tibco.clientVar.ASG/Operation/EnableDefaultOperation=true`
9. Save the changes to the file.

Configuring DefaultOperation Facade Operation

You can configure Default operation for any facade operation using Config UI.

By default, TIBCO API Exchange Gateway provides the DefaultOperation facade operation for the example projects shipped with the product. For example, BookQuery project has a DefaultOperation operation configured under **ROUTING > Facade Operations**.

To use the DefaultOperation feature for your project, configure a facade operation as follows:


Configuring DefaultOperation Facade Operation (REST)

For the REST requests, configure a DefaultOperation facade operation as follows:

Procedure

1. Click the **PARTNER > Facade Operations** tab.
2. Click the **Add Property** icon.
3. Add a new facade operation as follows:

DefaultOperation Facade Operation Parameters (REST)

Parameter	Value
Operation Name	Enter DefaultOperation as the name of the facade operation.
Operation URI	<p>The URI of the facade operation request. For example, /asg/defaultOperation</p> <div>  <p>The Operation URI supports regular expressions which are evaluated by Java.util.regex at runtime.</p> </div>
Operation Service Name	<p>Enter a logical service name. For example, Internal</p>

4. Save the changes to your configuration.

Configuring DefaultOperation Facade Operation (SOAP)

For the SOAP/HTTP, SOAP/JMS, and ESB requests, configure a DefaultOperation as follows:

Procedure

1. Click the **PARTNER > Facade Operations** tab.
2. Click the **Add Property** icon.
3. Add a new facade operation as follows:

DefaultOperation Facade Operation Parameters (SOAP)

Parameter	Value
Operation Name	Enter DefaultOperation as the name of the facade operation.
Operation URI	The URI of the facade operation request. For example, /asg/defaultOperation
SOAP Action	The SOAP action of the facade operation request. For example, "/GetDefaultOperation"
Operation Service Name	Enter a logical service name. For example, MWC

4. Save changes to the configuration.

Configuring Target Operation for DefaultOperation

A target operation must exist for DefaultOperation of a facade operation.

Configure a target operation (if not existing), where the DefaultOperation facade operation request is routed.

For example, configure a HTTP type target operation for the DefaultOperation facade operation as follows:

Procedure

1. Click the **ROUTING > Target Operations** tab.
2. Click the **Add Property** icon.
3. Add a new target operation as follows:

Target Operation Configuration

Parameter	Description
Target Operation Name	The name of the target operation. For example, http.DefaultTargetOperation
Type	The type of the target operation. For example, HTTP
URI	The URI of the target operation. For example, /asg/test
Host	The host name of the target operation.

Parameter	Description
Port	The TCP port of the target operation implementation when invoked over HTTP.

4. Save the changes to the configuration.

Configuring Routing Key for DefaultOperation

You must configure a routing key for the DefaultOperation of a facade operation.

Configure a routing key for the DefaultOperation as follows:

Procedure

1. Click the **ROUTING > Routing** tab.
2. Click the **Add Property** icon.
3. Add a routing key as follows:

Routing Key Configuration

Parameter	Description
Operation Name	Select DefaultOperation from the drop-down list.
Routing Type	Select Target Operation from the drop-down list.
Routing Key	Enter a routing key.
Target Operation	Select http.DefaultTargetOperation from the drop-down list.

4. Save changes to the configuration.

Configuring Facade Access for DefaultOperation

Configure the partner for DefaultOperation as follows:

Procedure

1. Click the **PARTNER > Facade Access** tab.
2. Click the **Add Property** icon.
3. Add the anon partner to access the DefaultOperation as follows:

Facade Access Configuration

Parameter	Description
Partner	Select anon from the drop-down list.

Parameter	Description
Facade Operation	Select DefaultOperation from the drop-down list.

4. Save changes to the configuration.

Pass-Through Special Characters in Query String

TIBCO API Exchange Gateway decodes the special characters specified in the parameter values of the query string of the HTTP URL for a facade request. To pass through the special characters in the parameter values of the query string, follow these steps:

Procedure

1. Navigate to the `ASG_CONFIG_HOME/bin` directory.
2. Edit the `asg.properties` file.
3. Set the following properties:

```
com.tibco.asg.runtime.http_client.default_uri_format=false
tibco.clientVar.ASG/Operation/RequiresQueryDecoding=false
```



Refer to [Runtime Properties of Core Engine](#) for the description of properties.

4. Save changes to the file.

Proxy Server

TIBCO API Exchange Gateway acts as a proxy server for facade operation requests and target operation responses. When you use the TIBCO API Exchange Gateway as proxy server, TIBCO API Exchange Gateway does not process or validates the incoming facade operation request. TIBCO API Exchange Gateway just forwards the facade operation request to the appropriate target operation. Similarly, for any response message from the target service, TIBCO API Exchange Gateway passes the response message to the client. For example, when an error response is returned from the target service, TIBCO API Exchange Gateway sends the error response to the client without modifying or customizing the error code and message. Using the proxy feature improves the performance of the request processing in TIBCO API Exchange Gateway.



When TIBCO API Exchange Gateway is used as a proxy server and the user has not explicitly configured the response caching for a facade operation on the Config UI, TIBCO API Exchange Gateway provides the response caching with the default caching parameters, as follows:

Cache Type: SimpleCache

Time To Live: 1 day (24 hours)

You can configure the headers of the request message to be forwarded to the target operation.

To use TIBCO API Exchange Gateway as a proxy server, enable the proxy functionality as follows:

Procedure

1. Start the Config UI, if not running.
2. Log in to the Config UI using your credentials.
3. Select the project under Projects.

4. On the **ROUTING** tab, click **Facade Operations** on the top menu.
5. Select an existing facade operation to be used by proxy server. If the facade operation does not exist, add a new facade operation.
6. Select **Operation Features** field. Enter Proxy.
7. Save changes to the configuration.

Configuring HTTP Headers

When a facade operation request is passed to TIBCO API Exchange Gateway in a proxy enabled mode, you can copy the headers information from the facade operation incoming request and forward it to the target operation.

Configure the **Headers To Forward** field to copy the headers of request message as follows:

Procedure

1. On the **ROUTING** tab, click **Target Operations** on the top menu.
2. Select an existing target operation for the operation request. If the target operation does not exist, add a new target operation for the operation request.
3. Select **Headers to Forward** field. Enter the value, as required. See the following values as an example:
 - Enter "*" to forward all the headers.
 - Enter "*,-SoapAction" forwards all the headers except SOAP Action header.
4. Save changes to the configuration.



According to [HTTP/1.1 RFC](#), HTTP headers are case-insensitive. The native HTTP channel of API Exchange uses Apache Tomcat which enforces this behavior by converting all headers to lower case. We recommend that application developers must not depend on case sensitive headers in their application logic.

Routing Overview

Overview of the routing functionality of TIBCO API Exchange Gateway

The routing capabilities of TIBCO API Exchange Gateway determine the target operation to process a facade request. TIBCO 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 Core Engine to balance the load of requests processing across target operations. See [Target Operation Group](#).

When the 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](#) for details.

A routing key is used to determine the target operation or a target operation group to process an incoming facade request. See [Routing Key](#) for the details.

TIBCO API Exchange Gateway provides options to derive a routing key.

Content Based Routing

When the routing key is generated from the content (data) of the request message, the routing is defined as content based routing. The generated routing key is used to route the client request to a target operation or target operation group.

Context Based Routing

When the routing key is derived from the HTTP or JMS header fields in the request context message, the routing is defined as context based routing. The generated routing key is used to route the client request to a target operation or target operation group.

Routing Key

Use a routing key to forward a facade operation request to a target operation endpoint.

A routing key is used by the Core Engine to select a target operation or target operation group for an incoming facade request. Therefore, the routing key is a 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.

TIBCO API Exchange Gateway uses routing key in one of the following ways:

- By default. The default routing key is provided.
- The preferred routing key.
- Deriving the routing key from a custom XSLT file.

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. To derive a routing key, define a transformation file in the parsing step of the request processing pipeline.

If no routing key is derived from the parsing of the request message and no preferred routing key is configured, default routing key is used.

The routing key can have the following values:

- default: default routing key is used:
 - if the **ProcessBody transform** field for a facade operation does not generate a routing key.
 - if preferred routing key is not configured for a facade operation.
 - if the routing key derived from custom XSLT or specified in **Preferred Routing Key** field for a facade operation is not configured in the **Routing** tab.
- *routingKeyValue*: indicates a specific value of the routing key derived from an XSLT file. The *routingKeyValue* is a value populated from the transformation (XSLT) file using the routingKey element tag.
- Preferred Routing Key: a specific value specified in the **Preferred Routing Key** field of **PARTNER > Facade Access** tab.

Routing Key using XSLT

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, follow these main steps:

- Derive a Routing Key
- Configure a Routing Key

To derive a routing key, define a XSLT file with routingKey element tag and upload this XSLT file in the **ProcessBody transform** field 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 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 Config UI. See [How to Derive and Configure Routing Key](#) for details.

How to Derive and Configure Routing Key

This section explains how to derive and configure a routing key.

Define a Transformation File

Define a transformation (XSLT) file that contains the routingKey element tag to populate the routing key.

See the following references:

- See [Routing Schema document](#) as a reference to the XSLT file schema for the routing key.
- See [Example XSLT File to Derive Routing Key](#).

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 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>

```

Example XSLT File to Derive Routing Key

Refer to the following transformation (XSLT) file shipped with the GetLocation example:

- `ASG_CONFIG_HOME/GetLocation/xslt/operations/parse_getLocation.xml`

The example illustrates that the routing key is populated based on the value of `opCoId`, which is derived as a substring of the address element in the request message.

```

<routingKey>
  <xsl:choose>
    <xsl:when test="$opCoId != ''"><xsl:value-of                select="$opCoId"/></xsl:when>
    <xsl:otherwise>undefined</xsl:otherwise>
  </xsl:choose>
</routingKey>

```

Navigating to the ROUTING Tab

Navigate to the Routing tab on the Config UI.

Procedure

1. Start the GUI server, if not already running. See [Starting GUI](#).
2. Log in to the Config UI using your credentials.
3. Add a new project or select an existing project under **Projects**.
4. Click the **ROUTING** tab on the upper right.

Uploading the Transformation (XSLT) File

To upload the XSLT transformation file for a facade operation request, follow these steps:

Procedure

1. Ensure that you are on the **ROUTING** tab of the Config UI. See [Clicking the ROUTING Tab](#).
2. Click the **Facade Operations** tab on the top menu.
3. Add a new facade operation or select an existing facade operation.
4. Upload the XSLT transformation file as follows:
 - a) Click the **Choose File** button in the **New ProcessBody Transform** field to select a XSLT transformation file.
 - b) Click **Open** to upload the XSLT file.



- Define the XSLT file to upload in the **New ProcessBody Transform** field. See [Define a Transformation File](#).
- If the XSLT file is located in the `ASG_CONFIG_HOME/ASGConfigName/xslt/` operations directory, select the XSLT file in the **ProcessBody Transform** field.
- If the **New ProcessBody Transform** or **ProcessBody Transform** field of the facade operation configuration contains no XSLT file or has an XSLT file with no `routingKey` element tag, the Core Engine uses the **Routing** configuration with the default routing key.

5. Save the changes to the configuration.

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 Config UI as follows:

- See [Routing Configuration for a Target Operation](#)
- See [Routing Configuration for a Target Operation Group](#)

Routing Configuration for a Target Operation

To configure the routing key for a target operation, follow these steps:

Procedure

1. Ensure that you are on the **ROUTING** tab of the Config UI.
2. Click the **Routing** tab on the top menu.
3. Enter the parameters defined as follows:

Routing Configuration for a Target Operation

Parameter	Description
Operation Name	<ul style="list-style-type: none"> • Specifies the name of the facade operation. The operation name must be defined in the Facade Operations tab of the Config UI. • Select a predefined facade operation from this drop-down list. • This is a required field.
Routing Type	<ul style="list-style-type: none"> • Determines whether the facade request is routed to a target operation or a target operation group containing target operations. • The possible values are: <ul style="list-style-type: none"> – Target Operation – Target Operation Group • Select Target Operation to route the facade request to a target operation. • This is a required field.

Parameter	Description
Routing Key	<ul style="list-style-type: none"> Specifies the evaluated routing key for the given operation. A routing key must be defined. See How to Derive and Configure Routing Key for details. The default value is default. This is a required field.
Target Operation Name	<ul style="list-style-type: none"> Specifies the name of the target operation. The target operation name must be defined in the Target Operations tab of the Config UI. Select a predefined target operation from this drop-down list. This is a required field.

4. Save changes to your configuration.

Routing Configuration for a Target Operation Group

To configure the routing key for a target operation group, follow these steps:

Procedure

1. Ensure that you are on the **ROUTING** tab of the Config UI.
2. Click the **Routing** tab on the top menu.
3. Enter the parameters defined as follows:

Routing Configuration for a Target Operation Group

Parameter	Description
Operation Name	<ul style="list-style-type: none"> Specifies the name of the facade operation. The operation name must be defined in the Facade Operations tab of the Config UI. Select a predefined facade operation from this drop-down list. This is a required field.
Routing Type	<ul style="list-style-type: none"> Determines whether the facade operation request is routed to a target operation or a target operation group containing the target operations. The possible values are: <ul style="list-style-type: none"> – Target Operation – Target Operation Group Select Target Operation Group to route the operation request to a target operation group. The target operation group is a group of the target operations. This is a required field.

Parameter	Description
Routing Key	<ul style="list-style-type: none"> Specifies the evaluated routing key for a given operation. A routing key must be defined. See How to Derive and Configure Routing Key for the details. The default value is default. This is a required field.
Target Operation Group	<ul style="list-style-type: none"> Specifies the name of the target operation group for the load balancing functionality. Select a predefined target operation group from this drop-down list. The target operation group must be defined in the Target Operation Groups tab of the Config UI. See Target Operation Group for the details. This is a required field.

4. Save the changes to your configuration.

Routing Use Case using XSLT

This section describes the GetLocation sample shipped with the TIBCO API Exchange Gateway product to illustrate the configuration steps required for routing. Refer to the GetLocation example in the *ASG_HOME/examples* directory.

Sample Name

GetLocation

Description

The GetLocation example illustrates how to use a routing key to route the facade request to a different target operation or target operation group. The routing key is derived from the telephone number specified in the address element of the request. The target operation or target operation group must be configured in the Config UI.

For example, If the value of the address element is specified as "tel:+498948956000", the opCoId derived using the substring function (substring(\$address,6,2)) from the address element is 49. The routing key is populated based on the opCoId as 49. See [Configuration](#) for the routing configuration.

Sample Location

ASG_CONFIG_HOME/GetLocation

Configuration

Routing configuration using XSLT file for GetLocation example.

Define a Transformation File

Refer to the [Sample XSLT File](#) sample file to define a transformation file. This XSLT file shows that the routing key is derived based on the value of the address element of the request message. The address element contains a telephone number. You can edit the file, as required.

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" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:loc="urn:oma:wsdl:pxprof:terminallocation:1.0:interface:local">
  <xsl:template match="/">
    <xsl:variable name="nbRequestHref">
      <xsl:value-of select="/transformation/nbRequest/@href" />
    </xsl:variable>
    <xsl:variable name="nbRequest">
      <xsl:copy-of select="document($nbRequestHref)/soap:Envelope/
soap:Body/*" />
    </xsl:variable>
    <output>
      <xsl:variable name="address">
        <xsl:value-of select="$nbRequest/loc:getLocation/loc:address" />
      </xsl:variable>
      <xsl:variable name="partner">
        <xsl:value-of select="$nbRequest/loc:getLocation/loc:requester" />
      </xsl:variable>
      <xsl:variable name="opCoId">
        <xsl:value-of select="substring($address,6,2)" />
      </xsl:variable>
      <requester>
        <xsl:value-of select="$partner" />
      </requester>
      <serviceInterfaceVersion></serviceInterfaceVersion>
      <referenceId></referenceId>
      <serviceId></serviceId>
      <timestamp></timestamp>
      <correlationId></correlationId>
      <identityId></identityId>
      <opCoId>
        <xsl:value-of select="$opCoId" />
      </opCoId>
      <partnerId>
        <xsl:value-of select="$partner" />
      </partnerId>
      <routingKey>
        <xsl:choose>
          <xsl:when test="$opCoId != ''">
            <xsl:value-of select="$opCoId" /></xsl:when>
            <xsl:otherwise>undefined</xsl:otherwise>
          </xsl:choose>
        </routingKey>
      <address><xsl:value-of select="$address"/></address>
    </output>
  </xsl:template>
</xsl:stylesheet>
```

Uploading the Transformation File

To upload the parse_getLocation.xsl transformation file, complete the following steps:

Procedure

1. Start the Config UI. See [Starting GUI](#).
2. Log in to the Config UI using your credentials.
3. Select **Getlocation** under Projects.

4. Click the **ROUTING** tab on upper left.
5. Click the **Facade Operations** tab.
6. Expand the **getLocationBW** operation.
7. Make sure that the operations/parse_getLocation.xml file is populated in the **ProcessBody Transform** field. If it is not selected, select it from the drop-down list.
8. Save any changes to the configuration.

Routing Configuration

Configure a routing key for the getLocationBW operation to route the request to the http.getLocation target operation, as follows:

Procedure

1. On **ROUTING** tab, click the **Routing** tab on the top menu.
2. Click the **Add Property** to add a new routing configuration.
3. Enter the fields as follows:

Routing Configuration for GetLocation Example

Parameter	Value
Operation Name	Select getLocationBW from the drop-down list.
Routing Type	Select Target Operation from the drop-down list.
Routing Key	Enter 49 as the routing key.
Target Operation	Select the target operation as http.getLocation from the drop-down list. The http.getLocation target operation is defined in the Target Operations tab.

4. Save any changes to the configuration.



The routing configuration for the GetLocation example demonstrates the getLocationBW operation. For the getLocationBW operation request, 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.

Preferred Routing

TIBCO API Exchange Gateway supports routing of the client request based on preferred routing key. To use preferred routing, specify a preferred routing key for a partner operation as a facade access.

TIBCO API Exchange Gateway processes a facade operation request from the partner for preferred routing, as follows:

- If the **Preferred Routing Key** is not blank, preferred routing is enabled and the Core Engine uses this value as the routing key. Make sure to configure a routing key which matches the preferred routing key.
- If the **Preferred Routing Key** is blank, the Core Engine does not enable preferred routing.

- To select the target operation for the facade operation request, configure the preferred routing key on the **Routing** tab of the Config UI.
 - If the preferred routing key is not configured on the **Routing** tab, the default routing key is used.
 - If the default routing key is not configured, an error such as Route not found for transaction ID *transaction_ID* is returned.

Use Case for Preferred Routing

Using plan type for preferred routing.

TIBCO API Exchange Gateway supports the plan type for preferred routing. When the plan type is passed from the TIBCO API Exchange Manager to TIBCO API Exchange Gateway, the Core Engine populates the plan type in the **Preferred Routing Key** field of the **Facade Access** tab on the Config UI of TIBCO API Exchange Gateway.

Refer to *TIBCO API Exchange Manager Administration* on how to configure a plan type.

Example

Illustrates the routing configuration for the Gold plan type as preferred routing.

If you selected the Gold plan type in TIBCO API Exchange Manager for the getLocationBW facade operation, use the following configuration to route the getLocationBW facade operation request to the http.getLocation target operation:

Procedure

1. Start the Config UI, if not running.
2. Log in to the Config UI using your credentials.
3. Select the project under Projects.
4. On the **ROUTING** tab, click **Routing** on the top menu.
5. Click **Add Property** to add a new routing configuration.
6. Enter information for the fields as follows:

Routing Configuration for Preferred Routing Key Type

Parameter	Value
Operation Name	Select getLocationBW from the drop-down list.
Routing Type	Select Target Operation from the drop-down list.
Routing Key	The routing key must match the value populated in the Preferred Routing Key field of the Facade Access tab as the routing key. For example, Gold.
Target Operation	Select the target operation as http.getLocation from the drop-down list. The http.getLocation target operation is defined in the Target Operations tab.

7. Save the changes to the configuration.



- If the preferred routing key is not configured as the routing key in the **Routing** tab of Config UI, the default routing key is used.
- You can change the value of the preferred routing key using the XSLT file. The XSLT file is uploaded in the **New ProcessBody Transform** field of the facade operation to derive the routing key. The routing key derived from the XSLT file takes the precedence over the routing key configured as preferred routing in the **Routing** tab of Config UI. See [Overriding Preferred Routing Key using XSLT](#) for details.

Overriding Preferred Routing Key using XSLT

To change the value of the preferred routing key using the XSLT file, follow these steps:

Procedure

1. Start the Config UI, if not running.
2. Log in to the Config UI using your credentials.
3. Select the project under Projects.
4. On the **ROUTING** tab, click **Routing** on the top menu.
5. Select the **Facade Operations** tab.
6. In the **New ProcessBody Transform** field, upload the XSLT file. The XSLT file generates the routing key using the `<routingKey>` tag. See [Example XSLT File](#).
7. Save the changes to the project configuration.

Example XSLT File

The following is an example of an XSLT file that sets the routing key as SILVER. Set any value for the routing key in the XSLT file to overwrite the **Preferred Routing Key** value specified in the **Facade Access** tab.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/
Transform"
  xmlns:fn="http://www.w3.org/2005/xpath-functions"
  xmlns:c="http://www.tibco.com/schemas/asg/context"
  xmlns:h="http://www.tibco.com/asg/protocols/http"
  exclude-result-prefixes="xsl fn h c">
  <xsl:output method="xml" version="1.0" encoding="UTF-8"
    indent="yes" omit-xml-declaration="no"/>
  <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:template match="/">
  <output>
  <routingKey>SILVER</routingKey>
  </output>
  </xsl:template>
</xsl:stylesheet>
```

Target Operation Group

A target operation group consists of target operations.

TIBCO API Exchange Gateway provides the load balancing functionality to distribute 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. TIBCO API Exchange Gateway supports the failover for a target operation group. Using the failover mechanism, the request is processed by an alternate target operation within the target operation group if the primary target operation is not 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.

Using TIBCO API Exchange Gateway you can group target operations and add multiple target operations in a target operation group. You can also create multiple target operation groups as needed. When any client request is routed to a target operation group containing multiple target operations, the Core Engine forwards the request to a target operation based on 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.



The property `tibco.clientVar.ASG/Routing/MinimumFailoverHTTPStatusCode` in the `asg.properties` file allows the failure response from the backend to be propagated to the client even when the **TargetOperations** are configured with **RoundRobinWithFailover** or **WeightedRoundRobinWithFailover**. This is the minimum Error Status code after which the load balancer will continue with failover. Normally the value should be 500, but in some use cases, this can go up to 550.

See [Types of Target Operations Group](#) for supported types of target operation groups.

Types of Target Operations Group

TIBCO API Exchange Gateway supports the following types of target operation groups to provide the load balancing of requests within target operations in a group.

- **LoadBalanced**

when the type of target operation group is defined as LoadBalanced, the Core Engine picks up a random target operation from the list of target operations grouped into a target operation group. See [LoadBalanced](#) 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 the following subtypes:

- RoundRobin
- RoundRobin with Failover

See [RoundRobin](#) for details.

- **Weighted RoundRobin**

when the type of target operation group is defined as Weighted RoundRobin, it means 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 the following subtypes:

- Weighted RoundRobin
- Weighted RoundRobin with Failover

See [Weighted RoundRobin](#) 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 the IP address of the client machine, XPath, and the machine domain name.

See [Sticky Resource Affinity](#) 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

Consider a target operation group SG1 containing three target operations as A, B, and C. The Core Engine routes the incoming facade operation requests (for example, Request 1-n) to the SG1 target operation group based on the routing key. The Core Engine selects the target operation within the SG1 target operation group based on the type of the target operation group, which are defined as follows:

LoadBalanced

If the target operation group type for the SG1 target operation group is defined as LoadBalanced, the Core Engine uses the LoadBalanced routing algorithm. For the LoadBalanced routing algorithm type, the Core Engine picks up a random target operation that is a part of this target operation group. The Core Engine distributes the requests evenly over a large number of requests.

When the Core Engine routes the client requests (Request 1-n) to SG1 target operation group with the 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 the target operation B.
- The second facade request Request 2 is forwarded to the target operation C.
- The third facade request Request 3 is forwarded to the target operation A.
- The fourth request Request 4 is forwarded to the target operation A.
- The subsequent facade requests are forwarded to the target operations at random.

RoundRobin

If the target operation group type for the SG1 target operation group is defined as RoundRobin, the Core Engine uses the RoundRobin routing algorithm. For the RoundRobin routing algorithm type, the Core Engine selects the target operation in a RoundRobin fashion. This means that the Core Engine rotates through the list of target operations one at a time to process the client requests.

Figure [RoundRobin Routing](#) illustrates the SG1 target operation group with RoundRobin type.

When the 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

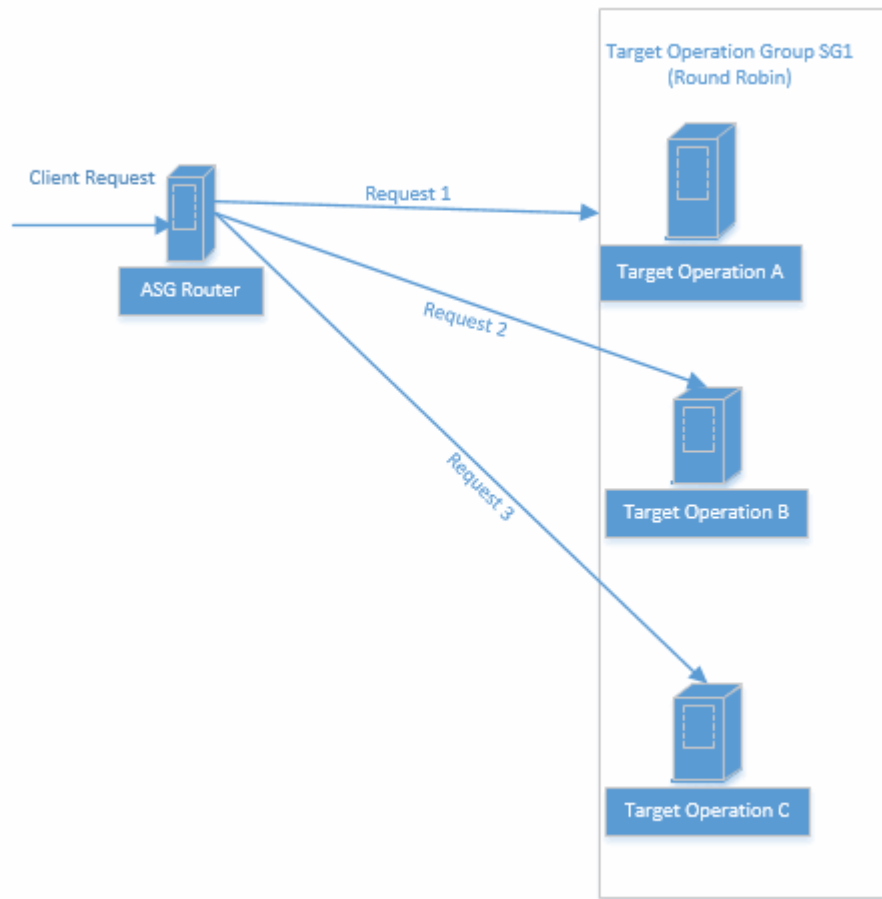
The Core Engine uses the following scheduling pattern for the RoundRobin algorithm type:

(Target Operation A, Target Operation B, Target Operation C, Target Operation A, Target Operation B, Target Operation C, Target Operation A, and so on.)

If the selected target operation is not running or times out to process the facade request, the Core Engine processes the request as follows:

- Retries to route the request to the same target operation as per the retry mechanism.
- The Core Engine does not route the facade request to an alternate target operation in the target operation group.
- Returns the fault message to the client for the facade request.

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 the Config UI.

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, 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 type as Weighted RoundRobin, the Core Engine selects the target operation within the target operations group based on the weight values.

Figure [Weighted RoundRobin](#) shows the SG1 target operation group containing three target operations (target operation A, target operation B, and 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, and Target operation C is assigned weight value as 3.

When the Core Engine routes any facade request to the SG1 target operation group, the 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 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 Core Engine uses the scheduling pattern as follows for the 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.)

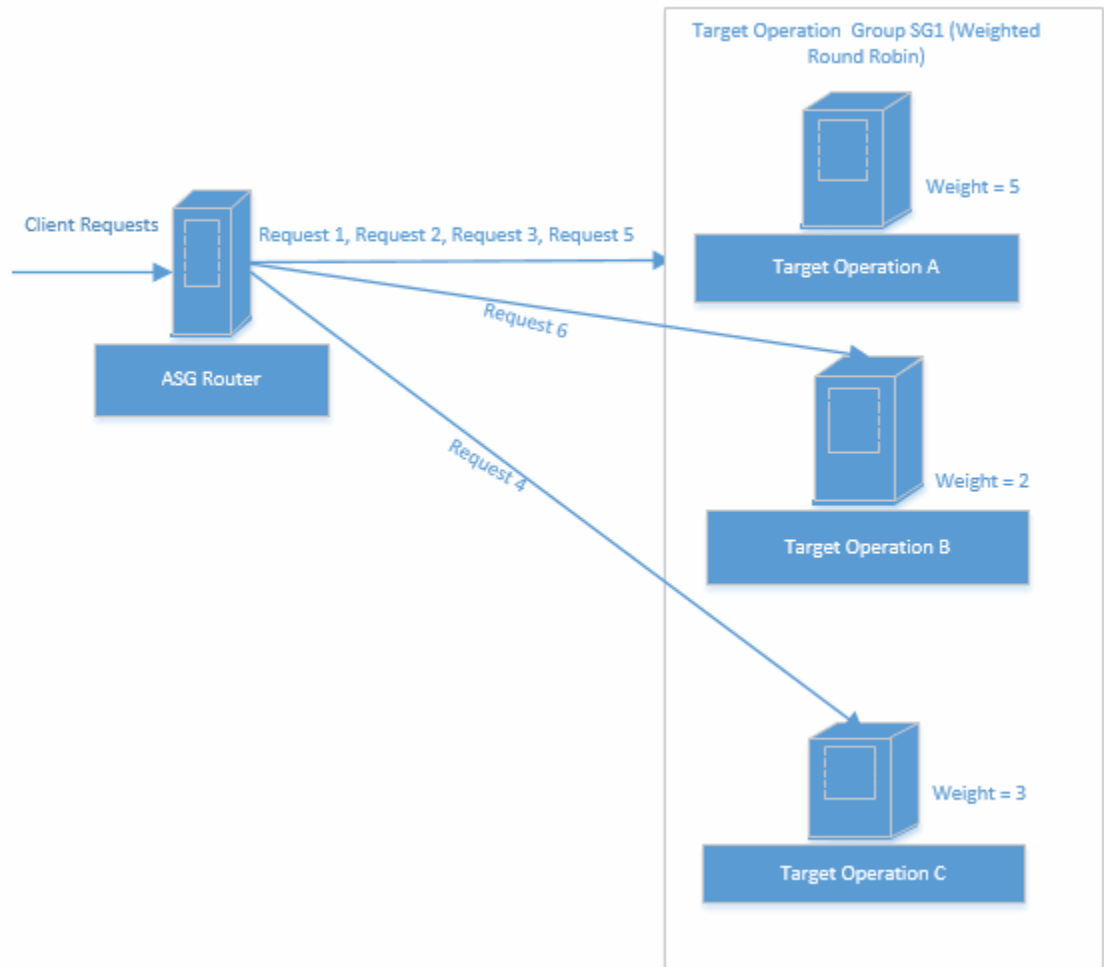
If the selected target operation is not running or times out to process the facade request, the Core Engine processes the request as follows:

- Retries to forward the request to the same target operation as per the retry mechanism.
- The Core Engine does not route the facade request to an alternate target operation in the target operations group.
- Returns the fault message to the client for the facade request.



When the target operations in a target operation group have the same weight values, the Core Engine selects the target operations in a RoundRobin fashion.

Weighted RoundRobin



RoundRobin with Failover

For a target operation group of RoundRobin with Failover type, the Core Engine selects the target operation as per the RoundRobin algorithm. See [RoundRobin](#) for details. After the target operation is selected, the Core Engine checks the health of the target operation using the HealthCheck reference functionality of TIBCO API Exchange Gateway. See [HealthCheck for Reference](#).

The Core Engine processes the client request as follows:

- The 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 the RoundRobin algorithm. This process continues until a target operation is found within the target operations group that is available to process the request. The facade request is routed to the available target operation for processing.

Weighted RoundRobin with Failover

For a target operation group of Weighted RoundRobin with Failover type, the Core Engine selects the target operation as per the assigned weight to the target operations. See [Weighted RoundRobin](#) for details. After the target operation is selected, the Core Engine checks the health of the target operation

using the HealthCheck functionality. See [HealthCheck for Reference](#) 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 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 that 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, the Core Engine distributes the client requests between the target operations in the target operations group based on the information such as the sticky key. When the Core Engine routes a client request to a target operation group of the 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 Core Engine balances the load of the target operations in the target operation group when it selects the available target operation from the group.

Define a sticky key to route the facade operation request to a specific target operation. See [Defining and Configuring Sticky Key](#) for details.

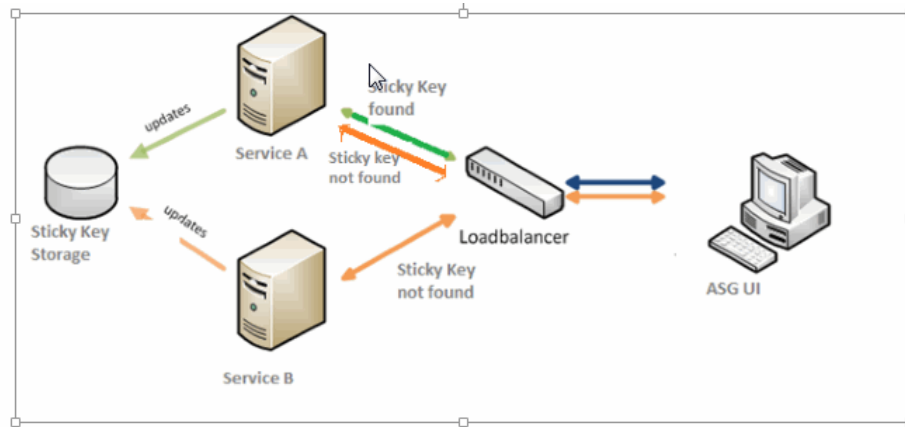
For the SG1 target operation group with Sticky Resource Affinity type, consider the following configuration:

- 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 Core Engine processes the client requests as follows:

- 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 Core Engine does not associate the SK4 sticky routing key with the request, but chooses target operation A as next target operation in the RoundRobin way. As target operation A has processed 10 requests, the 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.

StickyResourceAffinity Routing Algorithm



Defining and Configuring Sticky Key

To use the Sticky Resource Affinity routing algorithm, 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 Config UI. The Core Engine retrieves the sticky key from the transformation. See [StickyResourceAffinity Target Operation Group Configuration](#).

Target Operation Group Configuration

This section explains the configuration setup for a 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.

Configuring a Target Operation Group


To configure a target operation group, follow these steps:

Procedure

1. Ensure that you are on the **Routing** tab of the Config UI. See [Clicking the ROUTING Tab](#).
2. Click the **Target Operation Groups** tab on the top menu.
3. Enter the value for the following fields:

Target Operation Group Configuration

Parameter	Description
Group Name	<ul style="list-style-type: none"> Specifies the name of the target operation group. This can be any name defined by a user. This is a Required field.
Description	A short user description of the target operation group.

Parameter	Description
Type	<p>Specifies the type of target operation group. See Routing Algorithms for Target Operation Group for details.</p> <p>Select one of the possible values from the drop-down list:</p> <ul style="list-style-type: none"> • LoadBalanced • RoundRobin. See Configuring a RoundRobin Target Operation Group. • WeightedRoundRobin • RoundRobinWithFailover • WeightedRoundRobinWithFailover • StickyResourceAffinity.
Target Operations	<p>Specifies the list of target operations in the group. Add multiple target operations using the Add Target Operation() icon to a target operations group. The target operations must be configured under the Target Operations tab.</p>

4. Save the changes to the configuration.



If you configure a target operation group of the type RoundRobinWithFailover or WeightedRoundRobinWithFailover, you must define the HealthCheck configuration for each target operation in the group. See [HealthCheck Configuration for Target Operation](#) for configuration details.

Configuring a RoundRobin Target Operation Group


To configure a target operation group of RoundRobin type, select the type of target operation group as RoundRobin. Follow these steps:

Procedure

1. Click the **Target Operation Groups** tab.
2. Enter the value for the following fields:

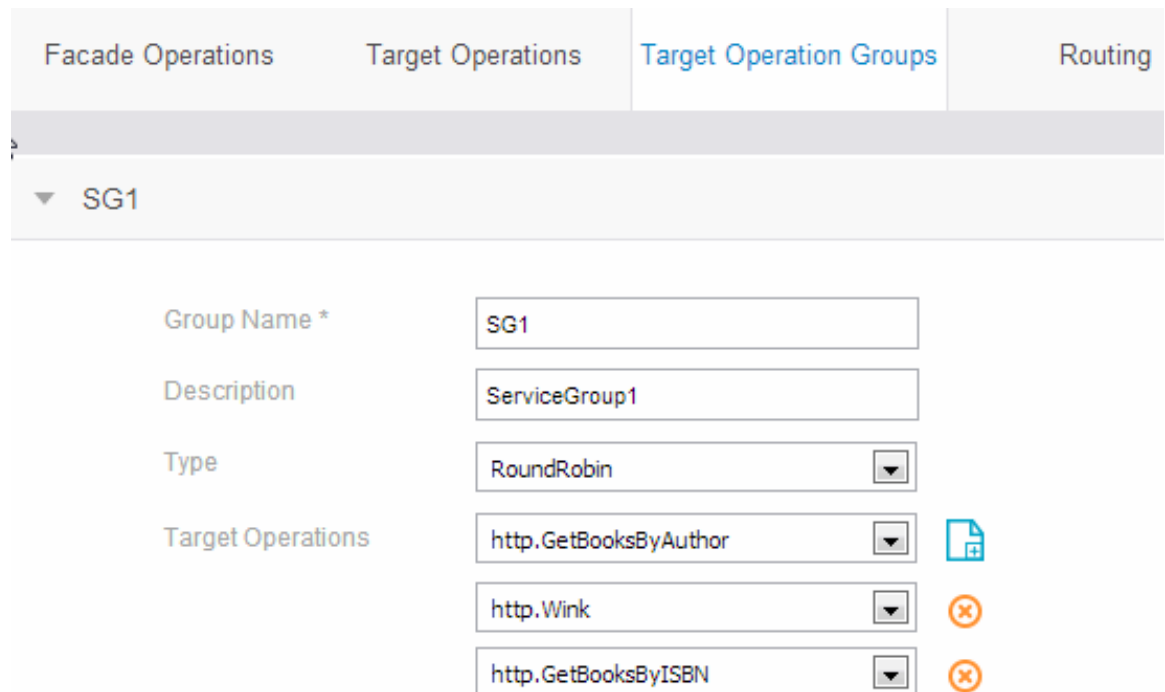
RoundRobin Target Operation Group Configuration

Parameter	Description
Group Name	<ul style="list-style-type: none"> • Specifies the name of the target operation group. This can be any name defined by a user. • This is a Required field.
Description	A short user description of the target operation group.
Type	Select RoundRobin from the drop-down list. See RoundRobin Target Operation Group Configuration .

Parameter	Description
Target Operations	Specifies the list of target operations in the group. Add multiple target operations using the Add Target Operation () icon to a target operations group. The target operations must be configured under the Target Operations group.

3. Save the changes to the configuration.
4. Figure [RoundRobin Target Operation Group Configuration](#) shows the RoundRobin Target Operation Group configuration.

RoundRobin Target Operation Group Configuration



Facade Operations Target Operations **Target Operation Groups** Routing

▼ SG1

Group Name * SG1

Description ServiceGroup1

Type RoundRobin

Target Operations

- http.GetBooksByAuthor
- http.Wink
- http.GetBooksByISBN

Configuring a WeightedRoundRobin Target Operation Group


To configure a target operation group of WeightedRoundRobin type, select the type of the target operation group as WeightedRoundRobin. Follow these steps:

Procedure

1. Click the **Target Operation Groups** tab.
2. Enter the value for the following fields:


WeightedRoundRobin Target Operation Group Configuration

Parameter	Description
Group Name	<ul style="list-style-type: none"> • Specifies the name of the target operation group. This can be any name defined by a user. • This is a Required field.

Parameter	Description
Description	A short user description of the target operation group.
Type	Select WeightedRoundRobin from the drop-down list. See WeightedRoundRobin Target Operation Group Configuration .
Target Operations	Specifies the list of target operations in the group. Add multiple target operations using the Add Target Operation () icon to a target operations group. The target operations must be configured under the Target Operations group.

3. Save the changes to the configuration.
4. Figure [WeightedRoundRobin Target Operation Group Configuration](#) shows the RoundRobin Target Operation Group configuration.

WeightedRoundRobin Target Operation Group Configuration



▼ SG1

Group Name *

Description

Type

Target Operations

<input type="text" value="http.GetBookResourcesByISBN"/>	<input type="text" value="2.0"/>	
<input type="text" value="http.GetBooksByTitle"/>	<input type="text" value="3.0"/>	
<input type="text" value="http.AddBookResources"/>	<input type="text" value="1.0"/>	

Configuring a RoundRobinWithFailOver Target Operation Group


To configure a target operation group of RoundRobinWithFailover type, select the type of the target operation group as RoundRobinWithFailover. Follow these steps:

Procedure

1. Click the **Target Operation Groups** tab.
2. Enter the value for the following fields:

RoundRobinWithFailOver Target Operation Group Configuration

Parameter	Description
Group Name	<ul style="list-style-type: none"> Specifies the name of the target operation group. This can be any name defined by a user. This is a required field.
Description	A short user description of the target operation group.

Parameter	Description
Type	Select RoundRobinWithFailover from the drop-down list. See RoundRobinWithFailOver Target Operation Group .
Target Operations	Specifies the list of target operations in the group. Add multiple target operations using the Add Target Operation () icon to a target operations group. The target operations must be configured under the Target Operations group.

3. Save the changes to the configuration.
4. Figure [RoundRobinWithFailOver Target Operation Group](#) shows the **RoundRobinWithFailover** Target Operation Group configuration:

RoundRobinWithFailOver Target Operation Group

▼ SG1

Group Name *

SG1


Description

RoundRobinWithFailoverTargetOperationG

Type

RoundRobinWithFailover ▼

Target Operations

http.AddBookResources ▼ 

▼ Health Check


Health Check Mode

Timer ▼

Check Interval (ms)*

Health Check Method

HTTP ▼

http.GetBooksByTitle ▼ 

▼ Health Check

Health Check Mode

Reset ▼

Reset Interval (ms)*


Configuring a WeightedRoundRobinWithFailOver Target Operation Group

To configure a target operation group of WeightedRoundRobinWithFailover type, select the type of the target operation group as WeightedRoundRobinWithFailover. Follow these steps:

Procedure

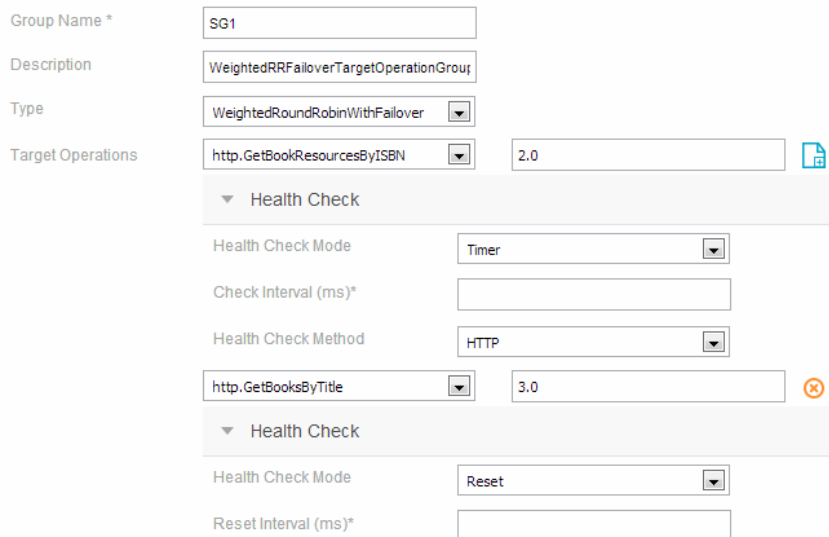
1. Click the **Target Operation Groups** tab.
2. Enter the value for the following fields:

WeightedRoundRobinWithFailOver Target Operation Group Configuration

Parameter	Description
Group Name	<ul style="list-style-type: none"> Specifies the name of the target operation group. This can be any name defined by a user. This is a Required field.
Description	A short user description of the target operation group.
Type	Select WeightedRoundRobinWithFailover from the drop-down list. See WeightedRoundRobinWithFailOver Target Operation Group .
Target Operations	Specifies the list of target operations in the group. Add multiple target operations using the Add Target Operation () icon to a target operations group. The target operations must be configured under the Target Operations group.

- Save the changes to the configuration.
- Figure [WeightedRoundRobinWithFailOver Target Operation Group](#) shows the RoundRobinWithFailover Target Operation Group configuration:


WeightedRoundRobinWithFailOver Target Operation Group



Group Name * SG1

Description WeightedRRFailoverTargetOperationGroup

Type WeightedRoundRobinWithFailover


Target Operations http.GetBookResourcesByISBN 2.0 

▼ Health Check

Health Check Mode Timer

Check Interval (ms)*

Health Check Method HTTP

http.GetBooksByTitle 3.0 

▼ Health Check

Health Check Mode Reset

Reset Interval (ms)*

StickyResourceAffinity Target Operation Group Configuration

This section explains the steps required to configure the target operation group of StickyResourceAffinity type.

Define Sticky Routing Key

The sticky routing key can be defined based on many parameters such as the IP address, machine domain name, and so on. The StickyKey is populated from the <stickyRoutingKey> element tag in the transformation (XSLT) file.

Refer to [Example XSLT File for StickyRoutingKey](#) to define an XSLT file for the sticky routing key.

Example XSLT File for StickyRoutingKey

The following example illustrates an XSLT file to derive a StickyRouting Key. Refer to the <stickyRoutingKey> element tag in the file defined as follows:

```
<?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($nbRequestHref)/soapenv:Envelope/soapenv:Body/*"/>
</xsl:variable>
<xsl:template match="/">
<output>
<xsl:variable name="username">
<xsl:value-of select="$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="$username = 'Vivek Ranadive'"><xsl:value-of select="$username"/></
xsl:when>
<xsl:when test="$username = 'Vivek Ranadive1'"><xsl:value-of select="$username"/></
xsl:when>
<xsl:when test="$username = 'Vivek Ranadive3'"><xsl:value-of select="$username"/></
xsl:when>
<xsl:otherwise>default</xsl:otherwise>
</xsl:choose>
</routingKey>
<stickyRoutingKey>
<xsl:choose>
<xsl:when test="$parsedQueryString != ''"><xsl:value-of
select="$parsedQueryString"/></xsl:when>
<xsl:otherwise>default</xsl:otherwise>
</xsl:choose>
</stickyRoutingKey>
</output>
</xsl:template>
</xsl:stylesheet>

<?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($nbRequestHref)/soapenv:Envelope/
soapenv:Body/*" />
  </xsl:variable>
  <xsl:template match="/">
    <output>
      <xsl:variable name="username">
        <xsl:value-of select="$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="$username = 'Vivek Ranadive'">
            <xsl:value-of select="$username" /></ xsl:when>
          <xsl:when test="$username = 'Vivek Ranadive1'">
            <xsl:value-of select="$username" /></ xsl:when>
          <xsl:when test="$username = 'Vivek Ranadive3'">
            <xsl:value-of select="$username" /></ xsl:when>
          <xsl:otherwise>default</xsl:otherwise>
        </xsl:choose>
      </routingKey>
      <stickyRoutingKey>
        <xsl:choose>
          <xsl:when test="$parsedQueryString != ''">
            <xsl:value-of select="$parsedQueryString" /></xsl:when>
          <xsl:otherwise>default</xsl:otherwise>
        </xsl:choose>
      </stickyRoutingKey>
    </output>
  </xsl:template>
</xsl:stylesheet>

```

Uploading Sticky Routing Key File

To upload the XSLT file containing the <stickyRoutingKey> element, follow these steps:

Procedure

1. Start the Config UI, if not running.
2. Log in to the Config UI using your credentials.
3. Add a new project or select an existing project under **Projects**.
4. Click the **ROUTING** tab on the right-hand side.
5. Click the **Facade Operations** tab on the top menu.
6. Add a new operation or select an existing operation.

7. Upload the XSLT file as follows:
 - a) Click **Choose File** in the **New ProcessBody Transform** field to select the XSLT transformation file containing stickyRoutingKey tag.
 - b) Click **Open** on the new dialog to upload the file.
 - c) If the XSLT file is located in the `ASG_CONFIG_HOME/ASGConfigName/xslt/operations` directory, choose the XSLT file in the **ProcessBody Transform** field.
8. Save the changes to the configuration.


Configuring StickyResourceAffinity Type Target Operation Group

To configure a target operation group of StickyResourceAffinity type, select the type of the target operation group as StickyResourceAffinity.

Procedure

1. Click the **Target Operation Groups** tab.
2. Enter the value for the following fields:

StickyResourceAffinity Target Operation Group Configuration

Parameter	Description
Group Name	(Required) Specifies the name of the target operation group.
Description	A short user description of the target operation group.
Type	Select StickyResourceAffinity from the drop-down list. See StickyResourceAffinity Target Operation Group Configuration for details.
Target Operations	Specifies the list of target operations in the group. Add multiple target operations using the Add Target Operation () icon to a target operations group. The target operations must be configured under the Target Operations group.

3. Save the changes to the configuration.

The following figure shows the StickyResourceAffinity Target Operation Group configuration:

StickyResourceAffinity Target Operation Group Configuration

Facade Operations	Target Operations	Target Operation Groups	Routing
<div> <div>▼</div> <div>SG1</div> </div>			
Group Name *	<input type="text" value="SG1"/>		
Description	<input type="text" value="ServiceGroup1"/>		
Type	<input type="text" value="StickyResourceAffinity"/>		
Target Operations	<div> <input type="text" value="http.GetBooksByAuthor"/> <input type="button" value="Add"/> </div> <div> <input type="text" value="http.GetBooksByISBN"/> <input type="button" value="Remove"/> </div>		

HealthCheck for Reference

TIBCO API Exchange Gateway provides HealthCheck for Reference functionality to monitor the availability of the target operations in a target operations group.

The Load Balancing Router component of the TIBCO API Exchange Gateway uses the HealthCheck for Reference functionality when it routes an operation request to a target operation group.

TIBCO API Exchange Gateway supports the HealthCheck for Reference functionality for the following types of target operation group:

- RoundRobinWithFailover
- WeightedRoundRobinWithFailover

When a facade operation request is routed to a target operation group of the RoundRobinWithFailover or WeightedRoundRobinWithFailover type, the Load Balancer Router of the gateway uses HealthCheck for Reference 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 that is running.

Using the HealthCheck for Reference functionality, the router component of the gateway intelligently picks the available target operation. This increases the throughput and reduces the latency of the request processing at runtime.



The HealthCheck for Reference functionality is available to the Gateway Load balancer Router component to monitor the health of target operations in a target operations group with failover routing algorithms only (RoundRobinWithFailover or WeightedRoundRobinWithFailover).

HealthCheck Modes for a Target Operation

HealthCheck 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 HealthCheck method for the timer mode. See [HealthCheck Methods for Timer Mode](#) for the details.

- Reset

You can configure the 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](#).

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 the HealthCheck configuration is selected for a target operation in the target operation group.

HealthCheck Methods for Timer Mode

Choose a method for the timer mode of HealthCheck. HealthCheck for Reference uses a specified method to check the health status of the target operation. The following types of HealthCheck methods are supported:

- HTTP

The Core Engine forms an HTTP based URL from the fields configured for the HTTP based target operation such as service-type, URI, host, and port. Once the URL is formed, the Core Engine invokes an HTTP connection to ping the target operation. Based on the response code returned from the HTTP connection, HealthCheck for Reference determines the health of the target operation. For example, if the returned response code indicates a success, the HealthCheck for Reference returns that the target operation is running. If the returned response code indicates a failure, the HealthCheck for Reference returns that the target operation is not running. The response code returned is checked to determine the health of the target operation.

- HTTPS

The 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. After the URL is formed, the Core Engine invokes a URL connection to ping the target operation. Based on the response code returned from the HTTP connection, HealthCheck for Reference determines the health of the target operation. For example, if the returned response code indicates a success, the HealthCheck for Reference returns that the target operation is running. If the returned response code indicates a failure, the HealthCheck for Reference returns that the target operation is not running.



For the HTTPS based target operation, configure the SSL properties for mutual SSL authentication. If the HTTPS based target operation is enabled for one-way authentication, SSL properties configuration is not required.

- HealthCheckURL

Using the HealthCheckURL method, you can specify the URL to invoke a HTTP connection. The Core Engine attempts to invoke the HTTP connection with the configured URL for this method. The 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 running. If the returned response code is a failure, the HealthCheck returns that the target operation is not running.



When HealthCheck method is HealthCheckURL or ContentVerification, for HTTPS based Load Balanced Target Groups, the Core Engine looks for Certificates under JRE's cacerts instead of Keystore configured for Target Operation.

- ContentVerification

The ContentVerification method uses the configured HealthCheck URL and sends an HTTP GET request to the URL. The HealthCheck for Reference 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 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.

Configure the following parameters for the TCPEcho method:

- Health Check URL
- Keyword

- TCPEcho

For TCPEcho Health Check to work, a TCP echo server needs to be running on the host where the target service runs and the echo server must listen on the port specified in the Health Check configuration. The TIBCO API Exchange Gateway engine will connect to this server, send a message and expects the echo server to send the same message back. Note that TCPEcho Health Check will only indicate whether the host on which the target service is running is up or not.

Configure the following parameters for the TCPEcho method:

- TCP Host
- TCP Port

- SampleRequest

The 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.

Configure the following parameters for the TCPEcho method:

- Health Check URL
- Content File

HealthCheck Configuration for Target Operation

This section explains the configuration set up to use the HealthCheck for Reference functionality for the target operations in a target operation group. Configure the HealthCheck 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, configure the ResetInterval parameter.

To configure the ResetInterval parameter for a target operation, follow these steps:

Procedure

1. Start the GUI server, if it is not already running.
2. Log in to the Config UI using your credentials.

3. Add a new project or select an existing project under **Projects**, as applicable.
4. Click the **Target Operation Groups** tab.
5. Select a target operation group and expand the node.
6. Select a Target Operation of RoundRobinWithFailover or WeightedRoundRobinWithFailover type for the **Type** parameter from the drop-down list.
7. Expand the **Health Check** node.
8. Select Reset for the **Health Check Mode** parameter from the drop-down list.
9. Type the value of the **Reset Interval** parameter. See [Reset Interval](#).

Configuration for Timer Based HealthCheck

After you enable the timer-based HealthCheck 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, follow these steps:

Procedure

1. Start the GUI server, if not already running.
2. Log in to the Config UI using your credentials.
3. Add a new project or select an existing project under **Projects**, as applicable.
4. Click the **Target Operation Groups** tab.
5. Select a Target Operation of RoundRobinWithFailover or WeightedRoundRobinWithFailover type for the **Type** parameter from the drop-down list.
6. Expand the **Health Check** node to enter the HealthCheck configuration parameters described as follows:

HealthCheck Configuration Parameters

Parameter	Description
Health Check Mode	<ul style="list-style-type: none"> • Specifies the mode for the HealthCheck functionality. • The possible values are: <ul style="list-style-type: none"> – Timer – Reset • Select Timer from the drop-down list.

Parameter	Description
Health Check Method	<ul style="list-style-type: none"> Specifies the method to be used by HealthCheck to check the availability of the target operation. The possible values are: <ul style="list-style-type: none"> HTTP HTTPS HealthCheckURL TCPEcho ContentVerification SampleRequest This is a Required field.
Check Interval	<p>Specifies a time interval (in milliseconds) which is used by HealthCheck as an expiration time to check the health status of the target operation.</p> <p>For example, if this value is specified as 10000, 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.</p>

7. Save the changes to your configuration.

Configuration for HTTP HealthCheck Method

Procedure

1. Configure the parameters as explained in [HealthCheck Configuration Parameters](#) table.
2. Select HTTP in the **Health Check Method** field from the drop-down list.

Configuration for HTTPS HealthCheck Method

Procedure

1. Configure the parameters as explained in [HealthCheck Configuration Parameters](#) table.
2. Select HTTPS in the **Health Check Method** field from the drop-down list.

Configuration for HealthCheckURL HealthCheck Method

If you want to use the HealthCheck method as HealthCheckURL, follow these steps:

Procedure

1. Configure the parameters as explained in [HealthCheck Configuration Parameters](#) table.
2. Select HealthCheckURL in the **Health Check Method** field from the drop-down list.
3. Enter the following parameter:

- **Health Check URL:** specifies the URL to invoke a HTTP connection. See [HealthCheckURL](#).

Configuration for TCPEcho HealthCheck Method

If you want to use the HealthCheck method as TCPEcho, follow these steps:

Procedure

1. Configure the parameters as explained in [HealthCheck Configuration Parameters](#) table.
2. Select TCPEcho in the **Health Check Method** field from the drop-down list.
3. Enter the parameters described as follows:

TCPEcho HealthCheckMethod Configuration Parameters

Parameter	Description
TCPHost	Specifies the host on which the TCP server is running.
TCPPort	Specifies the port on which the TCP server is running.

Configuration for ContentVerification HealthCheck Method

If you want to use the HealthCheck method as ContentVerification, follow these steps:

Procedure

1. Configure the parameters as explained in [HealthCheck Configuration Parameters](#) table.
2. Select ContentVerification in the **Health Check Method** field from the drop-down list.
3. Enter the parameters described as follows:

ContentVerification HealthCheckMethod Configuration Parameters

Parameter	Description
Health Check URL	Specifies the URL to which the heathcheck module sends the HTTP GET request.
Keyword	Specifies the keyword to be searched in the content returned in response to the HTTP GET request.

Configuration for SampleRequest HealthCheck Method

If you want to use the HealthCheck method as SampleRequest, follow these steps:

Procedure

1. Configure the parameters as explained in [HealthCheck Configuration Parameters](#) table.
2. Select SampleRequest in the **Health Check Method** field from the drop-down list.
3. Enter the parameters described as follows:

SampleRequest HealthCheckMethod Configuration Parameters

Parameter	Description
Health Check URL	Specifies the URL to which the HealthCheck for Reference sends the HTTP POST request.
Content File	Specifies a file from where the HealthCheck for Reference reads the data contents and sends it in HTTP POST request.
Existing Content File	Specifies an existing content file from the HealthCheck directory under <i>ASG_CONFIG_HOME</i> . The HealthCheck directory is created when the first file is uploaded. The HealthCheck for Reference reads the data contents from this content file and sends it in the HTTP POST request.

Throttles Overview

This section explains the throttle functionality of TIBCO API Exchange Gateway.

TIBCO API Exchange Gateway supports throttles to control the flow of requests from the client to the target operations. TIBCO API Exchange Gateway uses a throttle policy to determine if a client request should be passed on to a target operation or rejected. TIBCO API Exchange Gateway uses the throttles for the following operations:

- Protect the target operations from overuse
- Maintain the limit of requests load on a target operation
- Protect the target operation to be accessed by unauthorized partners
- Enforce the service level agreements at a partner level

Using throttles, you can 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 a time interval for the throttle.

The throttles define a condition for a throttle type and metric (entity). TIBCO API Exchange Gateway checks the condition for an incoming request before processing the request. For example, you can define a condition to allow only five client requests within 10 seconds to the target operation for a partner request.

A throttle policy is defined using the **Monitors** tab of the Config UI. After defining a throttle policy, the policy can be applied to a metric such as partner, partner group, partner operation, or target operation.

Facade Throttles

Facade throttles are designed for the partners, partner groups, and partner+operation. The Core Engine applies the facade throttle after it identifies the facade operation.

The supported throttle types can be applied at the following level:

- **Partner**
After defining a throttle policy, apply the policy at a partner level. The Core Engine checks the throttle condition on every request sent by the partner.
- **Partner Group**
After defining a throttle policy, apply the policy at a partner group level. The Core Engine checks the throttle condition on every request sent by the partners in a group.
- **Partner + Operation**
After defining a throttle policy, apply the policy by the combination of the partner and the operation. The 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. The 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, apply the policy for a target operation. The 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, the throttle is not applied.

Throttle Types

TIBCO API Exchange Gateway supports the following types of throttles:

- [Rate](#)
- [Quota](#)
- [High Water Mark](#)
- [Error](#)

Rate

The 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 the following:

- Request Count
- Payload Size. See [Payload Size Throttles](#).
- Monitor metric retrieved from the context of the input request message. See [Content Based Throttles](#).

The rate throttle is a technical throttle which can be applied to any metric. For example:

- Define a rate throttle to apply at a partner level. Using the throttle at a partner level, limit the number of requests from a partner.
- Define a rate throttle to apply at a target operation level. Using the throttle at a target operation level, protect the target operations from overuse.

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, if the throttle counter is set to Request Count. See [Throttle Counter](#) for the details.



Rate throttle is applied for the shorter intervals and is reset once the interval is reached.

For a rate throttle, define throttle interval (in seconds) and throttle max limit. The throttle max limit is reset as specified by Throttle UpdateInterval. See [Throttle UpdateInterval](#).

Example

Set the following throttle configuration on the Config UI to allow only five maximum requests to be processed by a target operation within 10 seconds. The following throttle configuration allows five requests every 10 seconds and rejects the other requests within 10 seconds.

```
Interval : 10
Max Limit: 5
Monitor Type: Rate Throttle
Monitor Counter: Request Count
```

Quota

Quota throttle is similar to the Rate throttle, but it uses a much larger count over much longer intervals (such as hours). Quota throttles are increased on the request. For a quota throttle, define throttle interval (in hour) and throttle max limit, if the throttle counter is set to Request Count.

Quota throttles are commercial throttles designed to prevent commercial overuse of the target operations, such as wholesale usage.

Example

Set the following throttle configuration for a target operation on the Config UI to allow only maximum 10 requests to be processed by a target operation within one hour. The following throttle configuration allows 10 requests to be processed within one hour and rejects the other requests within one hour.

```
Interval : 1
Max Limit: 10
Type: Quota Throttle
Monitor Counter: Request Count
```

Display Quota Usage Statistics

The quota usage statistics are displayed on the **Organizations** page of the Management Portal. See the “Configuring the API Exchange Engine and the Portal Engine” section in Chapter 2 of the *TIBCO API Exchange Getting Started* guide, “Deploying the Product Components” for configuration details.

Support Quota Notification

To support quota throttle notification, start each engine instance using the `-n EngineName` parameter for the `asg-engine` command:

For example, on the Windows platform, run the `asg-engine` as follows:

```
asg-engine.exe -n EngineName -u asg-gtm
```

where `EngineName` specifies a unique engine name.



If you are running multiple instances of the Core Engines, you must run the Global Throttle Manager as described in Chapter 16, “High Availability Deployment Of Runtime Components”.

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 requester.

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. Define a throttle max count for a high water mark throttle. Using the High Water Mark throttle, you can process a 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 five requests to a target operation A by five users concurrently, define the throttle configuration for a target operation on the Config UI as follows. The following throttle configuration allows only five 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

An Error throttle acts as a Rate Throttle in logic, but the Error throttle counts the number of error responses as opposed to the number of requests. The throttle count of an error throttle is increased on the error responses.

For an error throttle, define throttle interval (in seconds) and throttle max count. The throttle max count is reset as specified by throttle `updateInterval`. See [Throttle UpdateInterval](#).

Example

You can define an error throttle for a target operation when you want to block the requests after a certain number of error responses are returned from this target operation.

Set the following throttle configuration on the Config UI if you allow only 10 error responses from a target operation within 10 seconds. This means that 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. The Core Engine rejects the 11th request onwards. After 10 seconds, the throttle maximum 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
```

The following diagram shows how the throttles are implemented for a throttle type:

Throttle Types

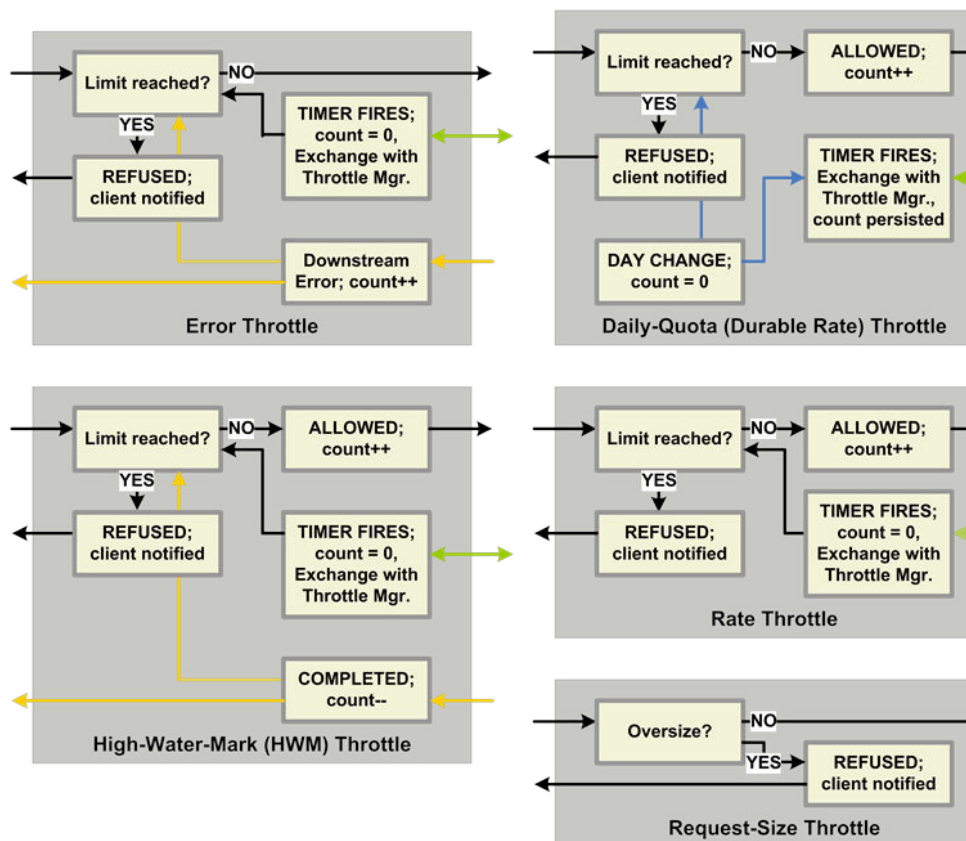


Figure 18: Some Example Throttles – Error, High-Water-Mark, Quota, Rate and Size

Monitor Time Modifiers

Using TIBCO API Exchange Gateway, you can design the throttles that work on the time modifiers. You can modify the throttle limits depending on the time of the day by applying time modifiers. Throttles time modifier specifies the time ranges when the time modifier can be active. You can specify multiple days and time ranges. For example, a throttle can relax or tighten its limit based on the time of day, such as 9a.m. - 1a.m.

The time modifiers of a throttle specify when any throttle can be overridden with the new values such as date and time range, days of the week, and the 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, edit a throttle configuration and add the day, date, and time modifiers. The parameters for the throttle time modifiers are explained in the following table:

Throttle Time Modifiers

Parameter	Description
Max Count	<ul style="list-style-type: none"> Specifies a maximum count that is active during this time. Refers to a throttle limit to apply if the throttle is in range. Required.
Start Date	<ul style="list-style-type: none"> Specifies the start date when the time modifier can be active. Optional.
End Date	<ul style="list-style-type: none"> Specifies the end date when the time modifier can be active. Optional.
Day of Week	<ul style="list-style-type: none"> Specifies the days of the week when the time modifier can be active. Optional.
Time Range	<ul style="list-style-type: none"> Specifies the time ranges when the time modifier can be active. Multiple ranges can be given. Optional.

- When you configure time modifiers for monitors, and if no day is selected, the time modifiers settings are applicable for all days of the week.
For example, if you configure throttle time modifiers for a rate throttle as follows, the settings work for all the days of the week.
 - Max Count: 10
 - Start Date: Current Date
 - End Date: Current Date
 - Time Range: 9 A.M. - 11 A.M.
- The date and time modifiers for monitors use the local time as set on the host machine where the Core Engine instance runs.
- The time zone for time modifiers should match the **Server Time Zone** of the Joomla! -based portal of TIBCO API Exchange. Refer to *Adapter Code for TIBCO® API Exchange and Joomla! User's Guide* to set the **Server Time Zone** under **Location Settings** of the system. The *Adapter Code for TIBCO® API Exchange and Joomla! User's Guide* available at the following location: <https://github.com/API-Exchange/JoomlaAdapter/wiki>.



Throttle Chaining

Using TIBCO API Exchange Gateway, you can assemble the 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 complex throttling needs. For example, the following throttle chains can be configured:

- Rate and High Water Mark: this throttle chain addresses the interface load.
- Quota: this throttle chain addresses the interface agreements.

The configuration of the throttles determines whether the request is passed on to be processed by the Core Engine or whether it is applied to any other throttle. Passing on the request to other throttles is called throttle chaining.

Throttle Counter

The throttle counter provides the type of counter used to increment the throttle count.

The possible values for the throttle counter are as follows:

- Request Count
The throttle count is incremented by 1 after each request is processed. To use this throttle counter, 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. The Input Payload Size throttle counter is used for Payload size throttle type. See [Payload Size Throttles](#).
- Output Payload Size
The throttle count is incremented by the size of response payload message (in bytes) after each request is processed. The Output Payload Size throttle counter is used for Payload size throttle type. See [Payload Size Throttles](#).
- 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. The Transaction Payload Size throttle counter is used for Payload size throttle type. See [Payload Size Throttles](#).

Throttle UpdateInterval

Throttle UpdateInterval is the time interval (in seconds) which is used by the 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 Core Engine refreshes the throttle count allowed for the next 10 seconds using the **Interval** and **Max Limit** parameters configured for the throttle Instance in the Config UI. For example, if the **Interval** is 60 seconds and **Max Limit** is 6, at most one request is allowed 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
```

Configuring Throttles

Using the Config UI, you can configure all types of the 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

List of parameters for throttle configuration.

The following table describes the parameters required to configure throttles. These parameters can be configured under the **Monitors** tab of a project configuration on the Config UI.

Throttle Configuration Parameters

Parameter	Description
Monitor Name	<ul style="list-style-type: none"> 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	<ul style="list-style-type: none"> Specifies the type of throttle. Select one of the supported values from the drop-down list: <ul style="list-style-type: none"> Rate Quota HWM Error Required.
Interval	<ul style="list-style-type: none"> Specifies a time period for which the maximum count of throttles are allowed.
Max Limit	<ul style="list-style-type: none"> Specifies the maximum number of throttle count allowed during the throttle interval period.
Monitor Counter	<ul style="list-style-type: none"> Specifies the type of the counter used to increment the throttle count. Not Applicable for the Error and High Water Mark throttle types. The possible values for the monitor counter are as follows: <ul style="list-style-type: none"> Request Count Input Payload Size Output Payload Size Transaction Payload Size The default value is Request Count. Request Count applies to all throttle types, except Payload size throttles. Input Payload Size, Output Payload Size, and Transaction Payload Size are applicable for only Rate and Quota throttle types.

Parameter	Description
Monitor Time Modifiers	Specifies the date, day, and time ranges for the throttle when the time modifiers are active. See Monitor Time Modifiers .
Use Approximate Monitor	Indicates if the maximum number of requests as specified by the <code>Max Limit</code> property is distributed by the number of running engine instances. If this value is set to <code>true</code> , the maximum number of requests is distributed by the number of active engines. If the value is set to <code>false</code> , the maximum number of requests is shared by the number of engines. You must set the Max Count Ratio parameter if the value of Use Approximate Monitor is set to <code>false</code> .
Max Count Ratio	An integer value indicating the percentage of maximum number of requests allowed by an Core Engine instance locally.

Creating a Throttle Policy Definition

This section explains how to configure a quota monitor T1 with a limit of five requests within 24 hours for service throttle metric.

Prerequisites

This example assumes that you have completed [Configure an Endpoint Operation for TIBCO API Exchange Gateway](#).

Modifying the Existing Configuration to Add Throttles

This section explains how to open the existing configuration.


Procedure

1. Start the GUI, if not already started. See [Starting GUI](#) for details.
2. Expand the **Projects** node, click the **ASG_Get_Start** configuration.

Defining a Quota Throttle

Add a new throttle as follows:

Procedure

1. Click the **MONITORING** tab.
2. Click the **Monitors** tab.
3. Click the **Add property**  icon to create a new monitor.
4. Specify the throttles details, as follows:

Quota Throttle Configuration

Parameter	Value
Monitor Name	T1
Monitor Type	Quota
Interval	24
Max Limit	5

Assigning a Throttle Policy to the Target Operation

To assign a Quota throttle to the target operation, follow these steps:

Procedure

1. Click the **ROUTING** tab on the upper right.
2. Click the **Target Operations** tab on the top menu.
3. Edit the http.GetBooksByAuthor target operation configuration.
4. Enter the details defined as follows:

Edit Target Operation Configuration

Parameter	Value
Operation Name	http.GetBooksByAuthor
Type	HTTP (select from the drop-down list.)
Monitor(s)	T1 (select from the drop-down list.)

Testing the Target Operation

Test the configured throttle policy.

Procedure

1. Launch the TIBCO Designer and open the following 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 a few more requests to the gateway.

6. Notice that after five successful requests, an error appears for the Quota Throttle violation.

Content Based Throttles

Overview of content based throttles.

Using TIBCO API Exchange Gateway allows you can create custom throttles based on the content information in the facade request.

For any throttle configuration specified in the Config UI, you can extend a throttle configuration to create a monitor. Using this monitor, you 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 a custom throttle monitor by defining a metric based on the request content of the message. The metric is defined by the following parameters:

- **metricName:** specifies the name of the throttle to be used from the Config UI configuration.
- **metricIncrement:** specifies the increment value for the throttle count. The metric increment value for the throttle (monitor) is defined, typically as an 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 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 that orders the books from a store. You can define the throttle in such a way 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 using the Config UI as follows:

Throttle Name: T1_BookOrder

Throttle Type: Rate

Throttle Interval: 10 seconds

Throttle Max Count: 20

For this scenario, the Core Engine allows a maximum of 20 requests within 10 seconds. You can extend the throttle policy by creating a monitor to allow a 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 an XPath formula based on the total number of book orders in the request message. In this case, you can define the **metricIncrement** to count the book orders in the request. So, 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.

Configure Content-Based Throttles

To configure the content based throttles, you need a XSLT file which typically contains an XPath formula. The XSLT file defines the metric definition to create the custom monitor. The 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, follow these steps:

Configuring Throttle

Define the throttle in the Config UI to create a custom throttle monitor.

Procedure

1. Start the GUI server, if not already running.
2. Log in to the Config UI using your credentials.
3. Add a new project or select an existing project under **Projects**.
4. Click the **MONITORS > Monitors** tab.
5. Define a T1_BookOrder throttle.

Throttle Configuration Parameters

Parameter	Description
Monitor Name	<ul style="list-style-type: none"> Specifies the name of the throttle. For example, T1_BookOrder. Required.
Monitor Type	<ul style="list-style-type: none"> Specifies the type of throttle. Select one of the possible values from the drop-down list: <ul style="list-style-type: none"> Rate Quota Required. <p>For example, Rate.</p>
Interval	<ul style="list-style-type: none"> Specifies a time period (in seconds) for which the maximum count of throttles is allowed. <p>For example, 10.</p>
Max Limit	<ul style="list-style-type: none"> Specifies the maximum number of throttle count allowed during throttle interval period. <p>For example, 20.</p>

6. Save the changes to the configuration.

Define XSLT File

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 follows:

```
<output>
  <xsl:variable name="childnodes">
    <xsl:value-of select="$nbRequest/loc:NewOrderReq/count(loc:OrderDt1)" />
  </xsl:variable>
  <monitor>
    <metricName>T1</metricName>
    <metricIncrement>
      <xsl:value-of select="$childnodes" />
    </metricIncrement>
  </monitor>
</output>
```

```
</monitor>
</output>
```

In the preceding 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 Config UI under **Monitors** tab.
- You can create multiple monitors in an XSLT file. Each monitor is defined using a separate `<monitor>` tag.
- If the XPath formula used in the `metricIncrement` field returns an invalid value, the content throttle increments the throttle count value by 1, which is the default value.

Example XML (Payload) Files

This section shows the sample XML files which shows payloads with 2 Orderdetails and 4 Orderdetails. Refer to [Example XSLT File](#) for the XSLT file for these payloads.

- Payload with 2 Orderdetails

```
<?xml version = "1.0" encoding = "UTF-8"?>
<NewOrderReq xmlns = "http://www.tibco.com/schemas/TAO/NewOrderSchema-v3.xsd">
  <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"?>
<NewOrderReq xmlns = "http://www.tibco.com/schemas/TAO/NewOrderSchema-v3.xsd">
  <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>
<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 an XSLT file. This XSLT file illustrates how you can use the number of OrderDtl element in a payload.

```

<?xml version="1.0" encoding="ISO-8859-1" ?>
<xsl:stylesheet version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/" xmlns:loc="http://
www.tibco.com/schemas/TAO/NewOrderSchema-v3.xsd">
  <xsl:template match="/">
    <xsl:variable name="nbRequestHref">
      <xsl:value-of select="/transformation/nbRequest/@href" />
    </xsl:variable>
    <xsl:variable name="nbRequest">
      <xsl:copy-of select="document($nbRequestHref)/soap:Envelope/
soap:Body/*" />
    </xsl:variable>
    <output>
      <xsl:variable name="childnodes">
        <xsl:value-of select="$nbRequest/loc:NewOrderReq/
count(loc:OrderDtl)" />
      </xsl:variable>
      <monitor>
        <metricName>T1</metricName>
        <metricIncrement>
          <xsl:value-of select="$childnodes" />
        </metricIncrement>
      </monitor>
    </output>
  </xsl:template>
</xsl:stylesheet>

```

Uploading XSLT File

After you define the XSLT file for a throttle monitor, upload the file in the **New ProcessBody XSLT** field on **facade Operations** tab in the Config UI.

To upload the XSLT file, follow these steps:

Procedure

1. Start the GUI server, if not already running.
2. Log in to the Config UI using your credentials.
3. Add a new configuration or select an existing configuration.
4. Click the **ROUTING** tab on the right hand side.
5. Click the **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 follows:
 - a) Click **Choose File** in the **New ProcessBody Transform** field to select the monitor XSLT transformation file.
 - b) Click **Open** to upload the XSLT file.
8. Save the changes to the configuration.

Payload Size Throttles

Overview of Payload size throttles.

Using TIBCO API Exchange Gateway, you can 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, 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 the payload size throttles, 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, 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 the request message, response message, or both request and response message.
- The payload size throttles are not supported for the Error and High Water Mark (HWM) throttle types.

Payload Size Throttle Types

Based on the throttle counter value, the payload size throttles can be any of the following subtypes:

- **Input Payload Size**
Specifies that the throttle count for a throttle is incremented by the size of the input message (that is, request message).
- **Output Payload Size**
Specifies that the throttle count for a throttle is incremented by the size of the output message (that is, response message).
- **Transaction Payload Size**
Specifies that the throttle count for a throttle is incremented by the sum of the size of input and output message (that is, request and response payload).

For example, you can define a payload size throttle T2 as follows:

```
Throttle Name: T2
Throttle Type : Rate
Throttle Interval : 10 seconds
Throttle Max Count: 10000
Throttle Counter: Input Payload Size
```

For this throttle T2 configuration, the 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 Core Engine performs the following actions:

- Checks the current throttle count, which is 0 bytes as this is the first request.
- Compares if the 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 the current throttle count (5000 bytes) is less than or equal to the throttle max count (10000 bytes).
- Forwards the request to the back-end service for processing as the throttle count is less than max count.

If the second request received after 3 seconds has a payload size of 5000 bytes, the Core Engine performs the following actions:

- Checks the current throttle count, which is 5000 bytes.
- Compares if the 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 the current throttle count (10000 bytes) is less than or equal to the throttle max count (10000 bytes).
- Forwards the request to the back-end service for processing as the throttle count(10000) is less than the max count (10000).

If the third request received after 2 seconds has a payload size of 1000 bytes, the Core Engine performs the following actions:

- Checks the current throttle count, which is 10000 bytes.
- Compares if the current throttle count (10000 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 15000 bytes.
- Checks if the current throttle count (15000 bytes) is less than or equal to the throttle max count (10000 bytes).
- Rejects the request for processing as the throttle count (15000) reached the max count (10000). The Core Engine sends an error back to the client.

Configuring Payload Size Throttles

The payload size throttles are configured using the Config UI. You can configure the throttles for the payload size of request, response, or both request and response messages.

To configure the payload size throttles, follow these steps:

Procedure

1. Start the GUI server, if not already running.
2. Log in to the Config UI using your credentials.
3. Add a new configuration or select an existing configuration.

4. Click the **MONITORING** tab on the right hand side.
5. Click the **Monitors** tab.
6. Specify the following parameters:

Payload Size Throttle

Parameter	Description
Monitor Name	<ul style="list-style-type: none"> Specifies the name of the throttle. For example, T2 Required.
Monitor Type	<ul style="list-style-type: none"> Specifies the type of throttle. Select one of the possible values from the drop-down list for the payload size throttle: <ul style="list-style-type: none"> Rate Quota Required. Example, Rate
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 the throttle interval period.
Monitor Counter	<p>Specifies the message type for which the payload size is calculated. Select one of the possible values from the drop-down list:</p> <ul style="list-style-type: none"> Input Payload Size Output Payload Size Transaction Payload Size

Configuring Monitor Counter

Configuring 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.

Configuring 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.

Configuring Payload Size Throttle for Request and Response Message

To configure the payload size throttle for both the request and response message, configure the **Throttle Counter** as Transaction Payload Size for a throttle configuration.



If an XSLT is configured to define content based throttle using metricName and metricIncrement for a throttle which also has payload based throttle configured, the Core Engine applies the content based throttle policy at run time.

Traffic Shaping

Shaping is a new throttle violation policy. TIBCO API Exchange Gateway is capable of shaping the requests traffic after reaching the throttle metric. By default, TIBCO 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 the rate throttle metric, the transaction is placed in 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 Core Engine continues to process these transactions. This means that all throttle chains are reevaluated when the transaction is released.

For the performance reasons, the Core Engine does not remove the transactions from the Shaping queue after the transactions are processed from the queue. The gateway uses a separate timer to compact the queue to truncate the queue. See [QueueCompactionInterval](#).

If the transactions in this queue time out, the Core Engine processes the request with normal error handling.



- Shaping is only valid for the rate throttles.
- The Shaping feature does not support the custom monitors.
- When the Shaping feature is enabled for a rate throttle and if the rate throttle is applied at the end of a throttle chain, the earlier throttles in the chain are not decremented.

Configuration

To enable Shaping for a facade operation, follow these steps:

Procedure

1. Start the Config UI.
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. In the **Operation Features** field, type Shaping.
6. Save the configuration changes.

QueueCompactionInterval

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.

Shared Throttles Overview

Use the shared throttles functionality to share the maximum throttle count for a throttle between the running engine instances.

TIBCO API Exchange Gateway does not require the Global Throttle Manager instance to distribute the maximum throttle count between the running engine instances. TIBCO API Exchange Gateway uses ActiveSpaces to provide the shared throttles functionality.



Configuration Setup for Shared Throttles

The shared throttles functionality uses ActiveSpaces to share the maximum throttle count between the gateway engine instances. You must set up the properties for ActiveSpaces metaspace configuration.

Configuring ActiveSpaces Metaspace Connection Properties

The shared throttles functionality uses ActiveSpaces metaspace to share the maximum throttle count for a throttle between the running Core Engine instances. You must setup the connection properties for ActiveSpaces metaspace.

To configure the ActiveSpaces connection properties, follow these steps:

Procedure

1. Navigate to the `ASG_CONFIG_HOME` directory.
2. Edit the `asg.properties` file in a text editor.
3. Set the following properties:

ActiveSpaces Metaspace Properties Configuration

Property	Description	Example
<code>tibco.clientVar.ASG/AS/MetaspaceName</code>	Name of the ActiveSpaces metaspace.	<code>APIXMS</code>
<code>tibco.clientVar.ASG/AS/DiscoveryUrl</code>	Discovery URL of the metaspace.	<code>tcp://192.168.0.10:13000</code>
<code>tibco.clientVar.ASG/AS/ListenUrl</code>	Listen URL of the metaspace.	<code>tcp://192.168.0.10:13000-*/</code>
<code>tibco.clientVar.ASG/AS/asLogLevel</code>	Log level of ActiveSpaces logs.	<code>0</code>
<code>tibco.clientVar.ASG/AS/asLogDir</code>	Log directory of ActiveSpaces logs.	<code>ASG_CONFIG_LOGS/logs</code>

4. Save changes to the file.

Enable Shared Throttles

To use ActiveSpaces as a metaspace to share the maximum throttle count for a throttle between the gateway engine instances, enable the shared throttle property as follows:

Prerequisites

ActiveSpaces metaspace connection properties must be configured.

Procedure

1. Navigate to the `ASG_CONFIG_HOME` directory.
2. Edit the `asg.properties` file in a text editor.
3. Set the following property to `true`, as follows:
`tibco.clientVar.ASG/ST/useSharedThrottle=true`
4. Save changes to the file.

Throttle Configuration Parameters

Use the Config UI to set the configuration parameters for a throttle.

Prerequisites

- ActiveSpaces metaspace connection parameters are set. See [Configuring ActiveSpaces Metaspace Connection Properties](#).
- Enable the shared throttles. See [Enable Shared Throttles](#).

Procedure

1. Start Config UI, if not running.
2. Login to the Config UI using your credentials.
3. Click on your project under **Projects**.
4. Click the **Monitors** tab.
5. Set the following parameters for a throttle:



Refer to [Configuration Parameters for Throttles](#) for the description of parameters.

- Use Approximate Monitor
 - Max Count Ratio
6. Save changes to the project configuration.

Example Use Case

TIBCO API Exchange Gateway shares the maximum throttle count of a rate throttle between the two active Core Engine instances.

TIBCO API Exchange Gateway requires the following configuration setup to share the throttle count for a T1 throttle between the two active Core Engine instances:

- Number of active Core Engine instances : 2
- Throttle Configuration:

- Throttle Name: T1
- Throttle Type: Rate
- Max Limit: 5
- Use Approximate Monitor: false
- Max Count Ratio:10
- Set the `useSharedThrottle` property to `true` in the `ASG_CONFIG_HOME/asg.properties` file.
- Start the two engine instances.



Do not start any Global Throttle Manager instance.

TIBCO API Exchange Gateway processes the requests between the active Core Engine instances as follows:

1. If you send two requests to the Core Engine1 instance, the value of throttle count in ActiveSpaces metaspace is set as follows:

- Current throttle count : 2
- Max Limit: 5

The Core Engine1 instance processes two requests successfully.

2. If you send three requests to the Core Engine2 instance, instance, the value of throttle count in ActiveSpaces metaspace is set as follows:

- Current throttle count : 5
- Max Limit: 5

The Core Engine1 instance processes three requests successfully.

3. If you send any additional request to the Core Engine1 or Core Engine2 instance, the value of throttle count in ActiveSpaces metaspace is set as follows:

- Current throttle count : 6
- Max Limit: 5

TIBCO API Exchange Gateway rejects this request as the value of current throttle count in metaspace is greater than maximum limit for the throttle.

Authentication and Authorization

Overview of user authentication and authorization functionality.

This section explains the following topics:

- User Authentication
- WS Security Services Authentication
- Configuring Web Services Security Authentication
- Configure Secure Services with TIBCO API Exchange Gateway
- Partner Authorization

User Authentication

TIBCO API Exchange Gateway Supports authentication at the following two levels:

- Transport Layer
This layer supports the authentication mechanism supported by the transport.
- Gateway Layer
This layer supports the WebServices Security (WSS) authentication.

Transport and Protocol Level Authentication

When the partner sends a request, the first level authentication is done at the transport layer. TIBCO API Exchange Gateway supports the following transports for authentication:

- HTTPS
TIBCO API Exchange Gateway uses the Apache HTTP Server for HTTP/HTTPs transport.
- JMS
TIBCO API Exchange Gateway uses TIBCO Enterprise Message Service for JMS transport.

Authentication at Apache HTTP Server

For the HTTP/HTTPs transport, the request is created with the header fields in the context message. This request context message becomes part of the RV message using the protocol termination functionality when forwarded to the gateway.

The following types of authentication are supported for HTTP/HTTPs transport:

- No Authentication: indicates that the request does not have any user credentials. In this case, the request is processed as an anonymous user.
- Basic Authentication: 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 the Apache server.

Authentication at TIBCO Enterprise Message Service

TIBCO API Exchange Gateway provides the authentication mechanism supported by TIBCO Enterprise Message Service for the JMS transport. The request is created with the JMS application header fields in

the context message for the JMS transport. The request context message is forwarded to the gateway using the protocol termination functionality.

WS Security Services Authentication

Overview of WSS authentication

TIBCO API Exchange Gateway supports the WebServices Security (WSS) authentication services for the northbound messages.



- The configuration mechanism for WS security policies on **Facade Operations** tab in TIBCO API Exchange Gateway 2.x is provided for the backward compatibility to use with TIBCO ActiveMatrix Service Gateway 1.2.0 product release. This configuration mechanism is deprecated in 2.x release of the software.
- WS Security is supported using the security policies in the TIBCO API Exchange Gateway 2.x release. Refer to [Security Policies](#) chapter for details on how to use security policies.

TIBCO API Exchange Gateway supports the 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 Core Engine can verify the signature of the sender of the request using the trust store as well as can decrypt it.
- Northbound Response Messages
The Core Engine can sign the response message using a private key to maintain integrity and 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 Gateway 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 tokens 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

The following table lists the types of service providers used by WSS configuration.

Types of Service Providers

Type	Description
LDAP	LDAP authentication service provider (LDAP ASP) provides the ability to authenticate a username and password against an LDAP server.
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 a 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	WSS 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 the 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.

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 the following properties:

```
com.tibco.trinity.runtime.core.provider.lookup=com.tibco.trinity.runtime.core.provider.authn.ldap
com.tibco.trinity.runtime.core.provider.authn.ldap.groupAttributeGroupsName=cn
com.tibco.trinity.runtime.core.provider.authn.ldap.groupAttributeUsersName=uniquemember
com.tibco.trinity.runtime.core.provider.authn.ldap.userAttributeUsersName=uid
com.tibco.trinity.runtime.core.provider.authn.ldap.userDNTemplate=uid={0},ou=people,dc=example,dc=com
com.tibco.trinity.runtime.core.provider.authn.ldap.groupSearchBaseDN=ou=groups,dc=example,dc=com
com.tibco.trinity.runtime.core.provider.authn.ldap.userSearchBaseDN=ou=people,dc=example,dc=com
com.tibco.trinity.runtime.core.provider.authn.ldap.searchTimeout=1000
com.tibco.trinity.runtime.core.provider.authn.ldap.userSearchExpression=(&(uid={0})(objectclass=*))
com.tibco.trinity.runtime.core.provider.authn.ldap.userAttributesExtra=mail givenname
com.tibco.trinity.runtime.core.provider.authn.ldap.serverURL=ldap://trinity.na.tibco.com:1389
com.tibco.trinity.runtime.core.provider.authn.ldap.initialCtxFactory=com.sun.jndi.ldap.LdapCtxFactory
com.tibco.trinity.runtime.core.provider.authn.ldap.groupSearchScopeSubtree=true
com.tibco.trinity.runtime.core.provider.authn.ldap.groupIndication=groupHasUsers
com.tibco.trinity.runtime.core.provider.authn.ldap.enableSAML11Assertion=true
com.tibco.trinity.runtime.core.provider.authn.ldap.groupSearchExpression=uniquemember={0}
com.tibco.trinity.runtime.core.provider.authn.ldap.groupAttributeSubgroupsName=uniquemember
com.tibco.trinity.runtime.core.provider.authn.ldap.samlValiditySeconds=5
```

Properties

Table [Properties for LDAP Authentication Service Provider](#) describes the properties for LDAP Authentication Service Provider.

Properties for LDAP Authentication Service Provider

Property	Description
<code>com.tibco.asg.intent.usernameToken</code>	
	<p>Boolean intent property indicates if the LDAP authentication method can be enforced on the request message or not. Possible values are true or false.</p> <p>If the value of this property set to true, the request message must contain a valid username token.</p>
<code>com.tibco.trinity.runtime.core.provider.authn.ldap.initialCtxFactory</code>	
	<p>Specifies the name of the JNDI Factory to use.</p> <p>The default value is <code>com.sun.jndi.ldap.LdapCtxFactory</code> (Sun's <code>LdapCtxFactory</code>).</p> <p>Optional.</p>
<code>com.tibco.trinity.runtime.core.provider.authn.ldap.serverURL</code>	

Property	Description
	<ul style="list-style-type: none"> Specifies the URL to connect to the LDAP directory server. TIBCO API Exchange Gateway supports list of multiple values separated by comma to configure LDAP server in a high availability and fault tolerant setup. The LDAP URL is defined as: <code>ldap://hostname1:port, ldap://hostname2:port</code> The LDAP SSL URL is defined as: <code>ldaps://hostname1:port, ldaps://hostname2:port,</code> Required.
<code>com.tibco.trinity.runtime.core.provider.authn.ldap.searchTimeOut</code>	
	<p>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.</p> <p>Optional.</p>
<code>com.tibco.trinity.runtime.core.provider.authn.ldap.userAttributeUsersName</code>	
	<p>The name of the attribute in the user object that represents the user's name. The value depends on which LDAP server is used. If you are using ActiveDirectory LDAP Server, set this value as CN. If SunOne or OpenLDAP LDAP Server is used, set this value as <i>uid</i>.</p> <p>Optional.</p>
<code>com.tibco.trinity.runtime.core.provider.authn.ldap.userAttributesExtra</code>	
	<p>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.</p> <p>For example, mail givenname</p> <p>Optional.</p>
<code>com.tibco.trinity.runtime.core.provider.authn.ldap.userSearchBaseDN</code>	
	<p>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 the administrator's credentials in search mode.</p> <p>For example, ou=people,ou=na,dc=example,dc=org</p> <p>Required in admin (search) mode.</p>
<code>com.tibco.trinity.runtime.core.provider.authn.ldap.userSearchExpression</code>	

Property	Description
	<p>Specifies the expression to be used for searching in admin mode against potential user objects. For example, the search expression is specified as: (&(uid={0})(objectClass=person)).</p> <p>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.</p> <p>This property is relevant only when credentialProvider property is set and the binding is done as an administrator; otherwise userDNTemplate is used.</p> <p>Required in admin (search) mode.</p>
<code>com.tibco.trinity.runtime.core.provider.authn.ldap.userDNTemplate</code>	
	<p>Specifies a template to be used when formatting user's DN before binding. It is used as an alternative to admin (search) mode.</p> <p>For example, uid={0},ou=employee,ou=tsi,o=tibco</p> <p>Required for bind mode (not in admin (search) mode).</p>
<code>com.tibco.trinity.runtime.core.provider.authn.ldap.userAttributeGroupsName</code>	
	<p>If you specified "LDAP user indicates groups" (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.</p>
<code>com.tibco.trinity.runtime.core.provider.authn.ldap.userAttributesExtraList</code>	
	<p>Same as userAttributesExtra property but this is specified in list form.</p> <p>Optional.</p>
<code>com.tibco.trinity.runtime.core.provider.authn.ldap.userSearchScopeSubtree</code>	
	<p>A Boolean property which determines if the entire sub-tree is searched or not. If a true value is specified, the entire sub-tree starting at the base DN is searched. Otherwise, the nodes one level below the base DN are searched.</p> <p>Optional.</p>
<code>com.tibco.trinity.runtime.core.provider.authn.ldap.groupSearchBaseDN</code>	
	<p>Specifies the base distinguished name (DN) where the searches for the groups begin. Supply the base DN that narrows the search to the smallest set of objects that includes all valid groups.</p> <p>For example, ou=groups,ou=na,dc=example,dc=org</p> <p>The default value is empty.</p> <p>Optional.</p>
<code>com.tibco.trinity.runtime.core.provider.authn.ldap.enableNestedGroupSearch</code>	

Property	Description
	<p>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.</p> <p>The default value is <code>false</code>.</p> <p>Optional.</p>
<code>com.tibco.trinity.runtime.core.provider.authn.ldap.groupSearchExpression</code>	
	<p>Specifies the expression to be used for searching against potential groups. For example, search expression is specified as: <code>(&(uid={0})(objectClass=person))</code>.</p> <p>In this string, the variable <code>{0}</code> 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 groups to find all groups containing the username.</p> <p>The values might be different for different LDAP server.</p> <p>For example, its defined as <code>uniquemember={0}</code> for SunOne, <code>cn={0}</code> for OpenLDAP, <code>member={0}</code> for Active Directory.</p> <p>Required.</p>
<code>com.tibco.trinity.runtime.core.provider.authn.ldap.groupSearchScopeSubtree</code>	
	<p>A Boolean property which determines if the entire sub-tree is searched or not. If a 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 are searched.</p> <p>Optional.</p>
<code>com.tibco.trinity.runtime.core.provider.authn.ldap.groupIndication</code>	

Property	Description
	<p>Specifies how the group memberships for users are found.</p> <p>The default value is noGroupInfo.</p> <p>Optional.</p> <p>The possible values are as follows:</p> <ul style="list-style-type: none"> • 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 DN's 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 are not handled.
<code>com.tibco.trinity.runtime.core.provider.authn.ldap.groupAttributeGroupsName</code>	
	<p>Depends on value of groupIndication. Required if the groupIndication property has groupHasUsers value.</p> <ul style="list-style-type: none"> • groupHasUsers: Specifies the group attribute holding the name of group. For example, the value is defined as cn for OpenLDAP server, sAMAccountName for ActiveDirectory LDAP server. • userHasGroups: Specifies the name of the group. If this is not specified, the whole DN of the group is used. For example, the value is defined as cn for OpenLDAP server.
<code>com.tibco.trinity.runtime.core.provider.authn.ldap.groupAttributeSubgroupsName</code>	
	<p>Specifies the name of the attribute in each group object denoting subgroups.</p> <p>For example, the value is defined as uniqueMember for OpenLDAP server, member for ActiveDirectory LDAP server.</p> <p>Optional</p>
<code>com.tibco.trinity.runtime.core.provider.authn.ldap.groupAttributeUsersName</code>	

Property	Description
	<p>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.</p> <p>For example, the value is uniqueMember for OpenLDAP, member for ActiveDirectory Server.</p> <p>Required if the groupIndication property has groupHasUsers value.</p>
<code>followReferrals</code>	
	<p>Determines if the client follow referrals are returned by the LDAP server.</p> <p>The default value is false.</p> <p>Optional.</p>
LDAP SSL	
<code>com.tibco.trinity.runtime.core.provider.identity.trust.trustStoreServiceProvider</code>	
	Specifies the Identity Trust Provider configuration to provide SSL support for LDAP
<code>com.tibco.trinity.runtime.core.provider.credential.keystore.truststore.keyStoreLocation</code>	
	Specifies the location of the keystore for the credentials.
<code>com.tibco.trinity.runtime.core.provider.credential.keystore.truststore.keyStorePassword</code>	
	Specifies the location of the keystore for the credentials.
<code>com.tibco.trinity.runtime.core.provider.credential.keystore.truststore.keyStoreRefreshInterval</code>	
	Specifies the refresh interval (milliseconds).
<code>com.tibco.trinity.runtime.core.provider.credential.keystore.truststore.keyStoreType</code>	
	Specifies the keystore type. Supported formats are JKS,PKCS12.

Sample File

The properties and example configuration for LDAP authentication service providers are found in the following sample files:

- `ASG_CONFIG_HOME/default/wss/req_usernetoken_ldapbind.properties`
This file lists the properties with the example configuration for the LDAP server in bind mode.
- `ASG_CONFIG_HOME/default/wss/req_usernetoken_ldapsearch.properties`
This file lists the properties with the example configuration for the LDAP server in search mode.
- `ASG_CONFIG_HOME/default/wss/req_usernetoken_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 Core Engine to verify the signatures when the payload in the incoming request is signed. The 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 the following 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.enableTrustStoreAccess=true

#KCP for TIP Namespace. Used for verifying the signature.
com.tibco.trinity.runtime.core.provider.credential.keystore.truststore.keyStoreType=JKS
com.tibco.trinity.runtime.core.provider.credential.keystore.truststore.keyStoreRefreshInterval=60000
com.tibco.trinity.runtime.core.provider.credential.keystore.truststore.keyStoreLocation=keystore/john_keystore.jks
com.tibco.trinity.runtime.core.provider.credential.keystore.truststore.keyStorePassword=password
```

Properties

Table [Properties for Trust Identify Provider \(TIP\)](#) describes the properties for Trust Identify Provider.

Properties for Trust Identify Provider (TIP)

Property	Description
<code>com.tibco.asg.intent.signature</code>	
	Boolean intent property which 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.
<code>com.tibco.trinity.runtime.core.provider.identity.trust.trustStoreServiceProvider</code>	
	Specifies the name of the credential service provider containing the credentials for establishing trust.
<code>com.tibco.trinity.runtime.core.provider.credential.keystore.truststore.keyStoreType</code>	
	Specifies the keystore type. Supported formats are JKS,PKCS12.
<code>com.tibco.trinity.runtime.core.provider.credential.keystore.truststore.keyStoreLocation</code>	

Property	Description
	Specifies the location of the keystore.
<code>com.tibco.trinity.runtime.core.provider.credential.keystore.truststore.keyStorePassword</code>	
	Specifies the password to unlock the keystore.
<code>com.tibco.trinity.runtime.core.provider.credential.keystore.truststore.keyStoreRefreshInterval</code>	
	Specifies the refresh interval (milliseconds).

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 Core Engine to decrypt the message when the payload in the incoming request is encrypted. The 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 Properties

See the following 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
```

Properties

Table [Properties for Subject Identify Provider \(SIP\)](#) describes the properties for Subject Identify Provider.

Properties for Subject Identify Provider (SIP)

Property	Description
<code>com.tibco.asg.intent.decrypt</code>	
	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.
<code>com.tibco.trinity.runtime.core.provider.identity.subject.identityStoreServiceProvider</code>	
	Specifies the name of the credential service provider containing the private credentials for establishing the subject's identity.
<code>com.tibco.trinity.runtime.core.provider.identity.subject.keyAlias</code>	
	Specifies an alias name for the key corresponding to the private credentials in the credential store for establishing the subject's identity.
<code>com.tibco.trinity.runtime.core.provider.identity.subject.keyPassword</code>	
	Specifies the protection parameter of the private credentials in the credential store for establishing the subject's identity.
<code>com.tibco.trinity.runtime.core.provider.credential.keystore.keyStoreType</code>	
	Specifies the keystore type of the private credentials.
<code>com.tibco.trinity.runtime.core.provider.credential.keystore.keyStoreLocation</code>	
	Specifies the location(s) of the keystore of the private credentials.
<code>com.tibco.trinity.runtime.core.provider.credential.keystore.keyStorePassword</code>	
	Specifies the password to unlock the keystore.
<code>com.tibco.trinity.runtime.core.provider.credential.keystore.keyStoreRefreshInterval</code>	
	Specifies the refresh interval in milliseconds.

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` 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 the following properties:

```
#Example configuration where incoming message is encrypted and signed
com.tibco.asg.intent.signature=true
com.tibco.asg.intent.decrypt=true

#WSS Authn Namespace
#com.tibco.trinity.runtime.core.provider.authn.wss.enableSAML11Assertion=false
com.tibco.trinity.runtime.core.provider.authn.wss.signatureValidationService=class:com.tibco.trinity.runtime.core.provider.identity.subject

#SIP Namespace. Used for token signing service or decrypting the message before authentication.
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=self-signed-Server-key-cert-trusted-by-client
com.tibco.trinity.runtime.core.provider.identity.subject.keyAlias=self-signed-Server-key-cert-noski-trusted-by-client
com.tibco.trinity.runtime.core.provider.identity.subject.keyPassword=#!fgGMyESTOe58y1QEt7sykDYhfWg9mJKMVsJwsShnAC4=
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=#!fgGMyESTOe58y1QEt7sykDYhfWg9mJKMVsJwsShnAC4=
com.tibco.trinity.runtime.core.provider.credential.keystore.keyStoreType=JKS
com.tibco.trinity.runtime.core.provider.credential.keystore.keyStoreRefreshInterval=60000

#TIP Namespace. Used for verifying the signature.
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.truststore.keyStoreType=JKS
com.tibco.trinity.runtime.core.provider.credential.keystore.truststore.keyStoreRefreshInterval=60000
com.tibco.trinity.runtime.core.provider.credential.keystore.truststore.keyStoreLocation=keystores/server-CAs.jks
com.tibco.trinity.runtime.core.provider.credential.keystore.truststore.keyStorePassword=#!fgGMyESTOe58y1QEt7sykDYhfWg9mJKMVsJwsShnAC4=
com.tibco.trinity.runtime.core.provider.credential.keystore.truststore.keyStoreProvider=
```

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:

- [LDAP Authentication Service Provider \(LDAP ASP\) Configuration](#)
- [Trust Identity Provider](#)
- [Configuring Subject Identity Provider](#)

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_and_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.

Web Services Security Authentication

This section explains the procedure to configure the web services security for the Core 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 good practice 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. The 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_usernetoken_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_usernetoken_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 an LDAP system for search mode.

The sample file **req_usernetoken_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. The 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](#)

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](#)

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.

Registering 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.

Procedure

1. Start the GUI, if it has not already been started.
2. Click **WSS** tab.
3. Enter the details for WSS defined as follows:

WSS Configuration

Parameter	Description
WSS Name	Unique name which identifies a WSS configuration.

Parameter	Description
Type	Type of the WSS configuration. Select the type from the drop-down list. Possible values are: <ul style="list-style-type: none"> • WSS • Subject Identity • Trust Identity
New Property File	A new property file which defines the WSS resources configuration. See Define the WSS Configuration Properties file .
Existing Property Files	Select a configuration file of WSS resources configuration.

Defining 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 Config UI.

Procedure

1. On the configuration GUI, click **ROUTING** tab.
2. Click the **Facade Operations** tab.
3. Add a new operation. Enter the details of the Operation. See [Facade Operations](#).
4. Check the **Enable WSS** check box.
5. Enter the details for WSS enabled operation defined as follows:

WSS Enabled Operation Configuration

Parameter	Description
WSS Request	This is the name of the WSS configuration from WSS tab. The property file from this configuration is used for northbound request processing.
WSS Response	This is the name of the WSS configuration from WSS tab. The property file from this configuration is used for northbound response processing.
Encrypt Response	This check box flag indicates whether to encrypt the response message.
Sign Response	This check box flag indicates whether to sign the response message.
Encryption Algorithm	Using this list box, select the algorithm to use for data encryption. Supported values are: TRIPLE_DES, AES_128, AES_256, AES_192

Parameter	Description
Key Algorithm	Using this list box, select the algorithm to use for key encryption. Supported values are: RSA15, RSAOEP, AES128, AES192, AES256, TRIPLEDES
Signing Algorithm	Using this list box, select the algorithm to use for signing. Supported values are: 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
Key Type	Using this list box, choose a key reference method. Supported values are: BST_DIRECT_REFERENCE, ISSUER_SERIAL, X509_KEY_IDENTIFIER, SKI_KEY_IDENTIFIER, EMBEDDED_KEYNAME, EMBED_SECURITY_TOKEN_REF, UT_SIGNING, THUMBPRINT_IDENTIFIER
Keystore Alias	Specifies an alias of the public certificate from the truststore to be used for encryption.

Configure Secure Services with TIBCO API Exchange Gateway

Access secure web services using TIBCO API Exchange Gateway.

TIBCO API Exchange Gateway provides the HTTPs transport for the secure communication with the web services at the back end. You can access the target services with or without client authentication.

The backend services may or may not require X.509 client authentication. The Is Anonymous flag for a target 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.



When the authentication policies are enforced on a SSL enabled target service, make sure to set the class path in the *ASG_HOME/bin/asg-engine.tra* file. The class path must include the *TIBCO_HOME/tools/lib* directory, and can be set using the following variable: `tibco.env.CUSTOM_EXT_PREPEND_CP`.

Altering List of Algorithms (Optional)

Java 7 bundled with TIBCO API Exchange Gateway defines a constraint in the *TIB_JAVA_HOME/lib/java.security* file. This constraint limits the algorithms used by the accepted certificates from the counterpart. To resolve this, follow one of these workarounds:



The default values for the security algorithms are usually excluded by vendors to make the system more secure. By implementing any of this workaround, you are reducing the security level of the system.

Procedure

1. Download Java Cryptography Extension (JCE) Policy Files

To download the Java Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy Files 7, follow these steps:

- Navigate to the following URL: <http://www.oracle.com/technetwork/java/javase/downloads/jce-7-download-432124.html>
- Accept the license agreement.

- c) Right click the `UnlimitedJCEPolicyJDK7.zip` link. Click **Save link as....**
 - d) Save the zip file in the `TIB_JAVA_HOME/jre/lib/security` directory.
2. Disable Algorithms Constraint

To disable the algorithm constraint, follow these steps:

- a) Navigate to the `TIB_JAVA_HOME/jre/lib/security` directory.
- b) Open the `java.security` file in a text editor.
- c) Comment the following property:


```
jdk.certpath.disabledAlgorithms=MD2, RSA keySize < 1024
```
- d) Save changes to the file.

Define DSS Properties for Services

To use the services, define the DSS properties in a file. The DSS properties file is used during the configuration of the service using the Config UI. See [Configuring Services](#).

This section explains the properties required to use the back-end services using the HTTPs transport.

Properties For SSL Authentication (isAnonymous = true)

One way SSL authentication properties for a target operation.

Trust Identity Provider (TIP) properties are used if the `Is Anonymous` flag is set to `true` for any target service. TIBCO API Exchange Gateway supports the one way SSL authentication, that is, when the service is accessed by the 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

See the following properties:

```
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.truststore.keyStoreLocation=E:/keystores/trust_bw.jks
com.tibco.trinity.runtime.core.provider.credential.keystore.truststore.keyStorePassword=password
com.tibco.trinity.runtime.core.provider.credential.keystore.truststore.keyStoreType=JKS
com.tibco.trinity.runtime.core.provider.credential.keystore.truststore.keyStoreRefreshInterval=60000
```

Properties

Table [SSL Authentication Properties for Service](#) explains the properties for SSL authentication (one way SSL authentication) for the service.

SSL Authentication Properties for Service

Property	Description
<code>com.tibco.trinity.runtime.core.provider.identity.trust.trustStoreServiceProvider</code>	
	Specifies that trust store service provider uses keystores for credentials. By default, this is configured to use internal implementation and should not be changed. It is configured as follows: <code>class:com.tibco.trinity.runtime.core.provider.credential.keystore</code>
<code>com.tibco.trinity.runtime.core.provider.credential.keystore.truststore.keyStoreLocation</code>	

Property	Description
	Specifies the location of the keystore.
<code>com.tibco.trinity.runtime.core.provider.credential.keystore.truststore.keyStorePassword</code>	
	Specifies the password to unlock the keystore.
<code>com.tibco.trinity.runtime.core.provider.credential.keystore.truststore.keyStoreRefreshInterval</code>	
	Specifies the refresh interval (milliseconds).
<code>com.tibco.trinity.runtime.core.provider.credential.keystore.truststore.keyStoreType</code>	
	Specifies the keystore type. Supported formats are JKS,PKCS12.

Properties For Mutual SSL Authentication (isAnonymous = false)

Mutual (two way) SSL authentication properties for a target operation.

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.



These properties can be found in the `SslMutual.properties` file of the `ASG_CONFIG_HOME\default\security\` resource directory.

Use Case

Using service when client authentication (mutual SSL authentication) required.

Example Properties

See the following properties:

```
#SIP for mutual SSL
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=server-service-ca
com.tibco.trinity.runtime.core.provider.identity.subject.keyPassword=#IfGqMyESToe58y1QEt7sykDYhfWq9mJkMVsJwsShNAC4=
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/Server-Private-CAs.jks
com.tibco.trinity.runtime.core.provider.credential.keystore.keyStorePassword=#IfGqMyESToe58y1QEt7sykDYhfWq9mJkMVsJwsShNAC4=
com.tibco.trinity.runtime.core.provider.credential.keystore.keyStoreType=JKS
com.tibco.trinity.runtime.core.provider.credential.keystore.keyStoreRefreshInterval=60000

#TIP for one way SSL
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
com.tibco.trinity.runtime.core.provider.credential.keystore.truststore.keyStoreType=JKS
com.tibco.trinity.runtime.core.provider.credential.keystore.truststore.keyStoreRefreshInterval=60000
com.tibco.trinity.runtime.core.provider.credential.keystore.truststore.keyStoreLocation=keystore/Clients-Public-CAs.jks
com.tibco.trinity.runtime.core.provider.credential.keystore.truststore.keyStorePassword=password
```

Properties

Table [Mutual Authentication SSL Properties For service](#) explains the properties for mutual SSL authentication (client authentication) for a service.

Mutual Authentication SSL Properties for service

Property	Description
<code>com.tibco.trinity.runtime.core.provider.identity.subject.identityStoreServiceProvider</code>	
	Specifies that subject service provider uses keystores for credentials. By default, this is configured to use internal implementation and should not be changed. It is configured as follows: <code>class:com.tibco.trinity.runtime.core.provider.credential.keystore</code>
<code>com.tibco.trinity.runtime.core.provider.identity.subject.trustStoreServiceProvider</code>	
	Specifies that identity store service provider uses keystores for credentials. By default, this is configured to use internal implementation and should not be changed. It is configured as follows: <code>class:com.tibco.trinity.runtime.core.provider.credential.keystore</code>
<code>com.tibco.trinity.runtime.core.provider.identity.subject.keyAlias</code>	
	Specifies an alias name for the key corresponding to the private credentials in the credential store for establishing the subject's identity.
<code>com.tibco.trinity.runtime.core.provider.identity.subject.keyPassword</code>	
	Specifies the protection parameter of the private credentials in the credential store for establishing the subject's identity.
<code>com.tibco.trinity.runtime.core.provider.identity.subject.enableCredentialStoreAccess</code>	
	By default, this is configured to use internal implementation and should not be changed.
<code>com.tibco.trinity.runtime.core.provider.identity.subject.enableTrustStoreAccess</code>	
	By default, this is configured to use internal implementation and should not be changed.
<code>com.tibco.trinity.runtime.core.provider.credential.keystore.keyStoreLocation</code>	
	Specifies the location of the keystore of the private credentials.
<code>com.tibco.trinity.runtime.core.provider.credential.keystore.keyStorePassword</code>	
	Specifies the password to unlock the keystore.
<code>com.tibco.trinity.runtime.core.provider.credential.keystore.keyStoreType</code>	
	Specifies the keystore type of the private credentials.

Property	Description
<code>com.tibco.trinity.runtime.core.provider.credential.keystore.keyStoreRefreshInterval</code>	
	Specifies the refresh interval in milliseconds.
<code>com.tibco.trinity.runtime.core.provider.identity.trust.trustStoreServiceProvider</code>	
	By default, this is configured to use internal implementation and should not be changed.
<code>com.tibco.trinity.runtime.core.provider.identity.trust.enableTrustStoreAccess</code>	
	By default, this is configured to use internal implementation and should not be changed.
<code>com.tibco.trinity.runtime.core.provider.credential.keystore.truststore.keyStoreType</code>	
	Specifies the keystore type. Supported formats are JKS,PKCS12.
<code>com.tibco.trinity.runtime.core.provider.credential.keystore.truststore.keyStoreRefreshInterval</code>	
	Specifies the refresh interval (milliseconds).
<code>com.tibco.trinity.runtime.core.provider.credential.keystore.truststore.keyStoreLocation</code>	
	Specifies the location of the keystore.
<code>com.tibco.trinity.runtime.core.provider.credential.keystore.truststore.keyStorePassword</code>	
	Specifies the password to unlock the keystore.

Configuring Services

You can create and configure a service (target operation) using the Config UI.

To configure a target operation, provide a DSS properties file. See [Define DSS Properties for Services](#).

To configure a service, follow these steps:

Procedure

1. Start the Config UI.
2. Log in to the Config UI 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.
6. Select the **Target Operations** tab.

7. Create a new target operation using the "+" icon at the top menu bar.
8. Select from the drop-down list for the Type field.
9. Set the following fields:
 - 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](#).
 - c) **Existing Property Files:** select an existing DSS property file from the drop-down list if the file exists in the wss directory of the configuration.
10. Save the changes to the target services configuration.

Partner Authorization Overview

This section describes the partner authorization.

TIBCO API Exchange Gateway supports the authorization based on following actions:

- [Operation Identification](#)
- [Partner Identification](#)
- [Partner Authorization](#)

Operation Identification

When a client sends the request to TIBCO API Exchange Gateway , the gateway identifies the operation as follows:

- For SOAP requests, the operation is identified from either the SOAP Action header or URI, or both, as defined in the HTTP header.
- For HTTP/XML requests, the operation is identified from the combination of method and URI.
- For HTTP/HTTPs REST requests, the operation is identified from the combination of method, URI, and the value of some named HTTP header.
- For SOAPJMS requests, the operation is identified from the JMS message SoapAction header . The SoapAction of facade Operation must be configured with *SoapAction* where *SoapAction* matches the value of the JMS SoapAction header.
- For ESB requests, the operation is identified from the JMS message Operation header. The SoapAction of facade Operation must be configured with */ESB/[Operation]* where *[Operation]* matches the value of the JMS Operation header.

The operation details are configured in the **Facade Operations** tab of the Config UI.

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 Config UI for a project configuration.

For example, for HTTP or HTTPS transport, the partner is identified as follows:

- Anonymous user

If no user is specified in the incoming request, the Core Engine considers this request as a request from anonymous user which is not authenticated. The Core Engine looks for the partner name

defined by the `tibco.clientVar.ASG/anonymous/PartnerName/Authenticated` property in `ASG_CONFIG_HOME/asg.properties` file. The Core Engine matches the value of this property with the value defined by `Partner Name` field under **Partners** tab on the Config UI. If both the values match, the Core Engine further processes the request.

For example, the property is defined in `ASG_CONFIG_HOME/asg.properties` file as follows:

```
tibco.clientVar.ASG/anonymous/PartnerName/Authenticated=anon_partner
```

To process any unauthenticated requests where no user is specified in the request, configure a partner as `anon_partner` under **PARTNER > Partners** tab on the Config UI.

If there is a mismatch, then the Core 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 Core 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 Core Engine retrieves the user name and issuer CA from the request headers. The Core 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 Config UI.

If there is a mismatch, the Core Engine rejects the partner with Authorization error.

The following table explains the values of partner identification fields for various authentication mechanisms:

Partner Identification Fields

Parameter	Description
SSL Mutual authentication (Apache HTTP Server)	
Partner Serial Number	Subject DN from the X.509 certificate of the client.
Partner Issuer CA	Issuer DN from the X.509 certificate of the client.
Basic Authentication (Apache HTTP Server)	
Partner Serial Number	<i>username</i>
Partner Issuer CA	O=TIBCO;CN=ASG;CN=HTTP;CN=Basic Domain
Basic/UsernameToken Authentication (LDAP)	
Partner Serial Number	LDAP DN of the authenticated user.
Partner Issuer CA	urn:www.tibco.com

Parameter	Description
UsernameToken authentication (File based)	
Partner Serial Number	<i>username</i>
Partner Issuer CA	<i>urn:www.tibco.com</i>

Partner API Key

A partner can be identified by an API key from an incoming request.

TIBCO API Exchange Gateway enables a partner to be identified by an API key from an incoming request. The API key can be sent via HTTP header or the URL. The reason for recommending to have the API key in the header is that it is more secure than having the API in the URL. The API in the URL may be exposed as it may be saved in the browser history or server log.

The API key can be passed in the header using the following format:

ASGAccessKey: nnnnnnnnnnnnnnnnnnn

or in the URL as:

<https://host:port/resource/api& ASGAccessKey = nnnnnnnnnnnnnnnnnnn &...>

If the API key is available in both URL and header, a warning is logged and the API key from the header is used.

If the API key is enabled to use for identifying a partner, the partner identified by the API key then will be used as the sender of the request, instead of the actual authenticated principal. This is useful in the case where the API key owner (or the partner) wants to be responsible for authorization of the invocation and assumes the throttling rate, mapping, and other actions associated with the request.

In a secure environment, using the API key alone to identify the partner is not enough. The request must be authenticated by some other means as the API key may be a stolen key. To protect from stolen key scenario, the API can be further protected by a policy where the sender of must be authenticated. When a request is authenticated, a principal is generated from the authentication. The authenticated principal may or may not be the owner of the API key.

If the authenticated principal is not the owner of the API key, the API key owner should have a choice to deny the use the API key in order to make sure that no else but the owner can use the API key.

In order to allow for the API key to identify the owner (partner) and be able to restrict the API key to be used by the owner, each API key will have the following configuration options:

- API key
- Partner name
- Flag to use API key to identify partner
- Flag to restrict the API key for the partner principal only



This flag will make sure that if a request has and API key as when as a principal, the partner that has the API key must also be the same partner that can be identified by the principal.

The configuration file, `PartnerApiKey.cfg`, is used to configure the partner's API keys.

When a partner request passes through a security processing stage, the authenticated principle will be used to identify a partner. If the API key is in the request, the API key will be used as one of the identification method. Issuer and Serial number from the SSL will also be used to identify the partner, if available. For a request to map to a partner, the API Key or principal and serial number/Issuer

combination are used to identify the partner. If a partner is authorized to invoke the operation, then the rest of the operation processing will be based on the authorized partner. If a partner cannot be authenticated, it will be treated as an unidentified partner. If a partner cannot be identified, it is classified as an anonymous partner and the request is rejected. A request without any partner identifier is treated as an anonymous partner.

The following global variable in `asg.properties` is used as the partner name for an anonymous partner:

```
# Default PartnerName for unauthenticated requests tibco.clientVar.ASG/anonymous/  
PartnerName/Authenticated=anonABC
```

In the above example, the anonymous partner is set to `anonABC`. The default value is `anon` and can be changed.



No certificate issuer and serial number is assigned to the anonymous partner. The issuer and serial number in partner data configuration should be left as empty fields, otherwise they will be ignored.

Partner Authorization

After the operation and partner is identified, TIBCO API Exchange Gateway validates that the identified partner is authorized to invoke the operation. Setup the configuration details under **PARTNER > Facade Access** tab of the Config UI where you specify the operation which the identified partner is allowed to access.

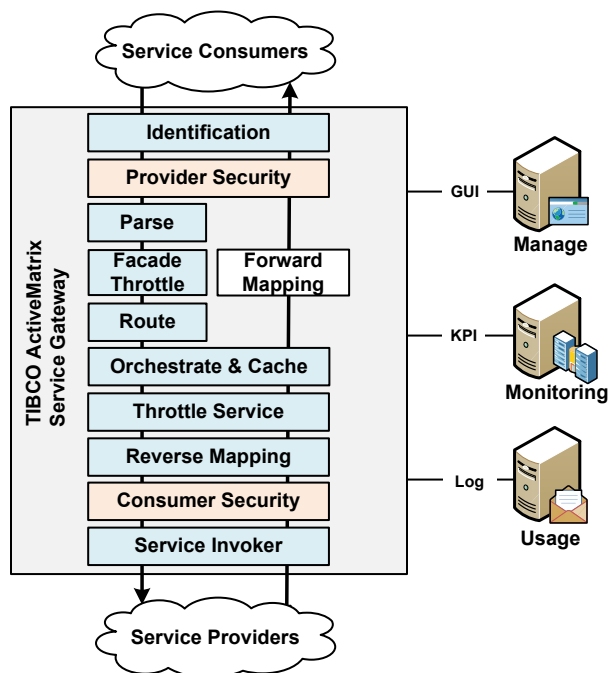
Overview of Security Policies

This section explains the security policies supported by TIBCO API Exchange Gateway software.

TIBCO 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](#) for the details on the supported policies.

The figure below illustrates the security policy enforcement points in the standard request processing pipeline.

Security Enforcement EndPoints



Security Concepts

This section explains the 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. TIBCO 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.

- SiteMinder

In SiteMinder authentication, the credential used for authentication is the SiteMinder session cookie or the username/password from the HTTP headers.

- If no credential is found, a password challenge is returned to request for username/password.
- If the HTTP headers has both SiteMinder session cookie and username/password, the SiteMinder session is used for authentication using the SiteMinder Service provider as specified in the policy.
- If SiteMinder session cookie is not available but username/password is provided, username/password is used to authenticate with the SiteMinder server and the SM session cookie is added to the response after a successful authentication.



SiteMinder authentication policy can be applied to the requests received directly by the HTTP channel of Core Engine, not to the requests which are received through the Apache HTTP Server.

- Kerberos SPNEGO

In SPNEGO authentication, the credential used for authentication is the SPNEGO token from the HTTP headers.

- If no credential is found, a NEGOTIATE challenge is returned to request for the SPNEGO token.
- If the HTTP headers has the SPNEGO token, the SPNEGO token is used for authentication using the Kerberos service provider as specified in the policy.



SPNEGO authentication policy can be applied to the requests received directly by the HTTP channel of Core Engine, not to the requests which are received through the Apache HTTP Server.

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. TIBCO 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](#) 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](#) 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 TIBCO API Exchange Gateway .

Policy Types and SubTypes

Policy Type	Policy Subtype	Endpoints
Authentication	<ul style="list-style-type: none"> • Basic • UsernameToken • SAML • SiteMinder • OAuth • Kerberos SPNEGO 	<ul style="list-style-type: none"> • Facade Operation
Authorization	<ul style="list-style-type: none"> • Role 	<ul style="list-style-type: none"> • Facade Operation
Integrity	<ul style="list-style-type: none"> • Sign • Verify Signature 	<ul style="list-style-type: none"> • Facade Operation • Target Operation
Confidentiality	<ul style="list-style-type: none"> • Encrypt • Decrypt 	<ul style="list-style-type: none"> • Facade Operation • Target Operation
Credential Mapping	<ul style="list-style-type: none"> • Basic • UsernameToken • SAML • OAuth 	<ul style="list-style-type: none"> • Target Operation

Types of Security Policies

The following types of security policies are supported by TIBCO API Exchange Gateway.

- [Authentication](#)
- [Authorization](#)
- [Confidentiality](#)
- [Integrity](#)
- [CredentialMapping](#)

Authentication

TIBCO API Exchange Gateway supports following types of authentication policies:

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.

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 TIBCO 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.

TIBCO API Exchange Gateway supports UsernameToken authentication policy with the password digest using WSS processor for LDAP server. Use the password digest for UsernameToken authentication policy as follows:

- The LDAP server must save plain text passwords which are available to the administrative user.
- Use the OpenLDAP LDAP server.

SAML

TIBCO API Exchange Gateway provides SAML authentication policy, where you can 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

TIBCO 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](#)
- [AuthenticationByUsernameToken.policy](#)
- [AuthenticationBySaml.policy](#)

SiteMinder

TIBCO API Exchange Gateway provides the SiteMinder security policy so that the target operations with HTTP bindings can authenticate the consumer's identity using the SiteMinder session cookie. The consumer's identity is authenticated using a SiteMinder service provider shared resource.

See [AuthenticationBySiteMinder.policy](#).

OAuth

TIBCO API Exchange Gateway provides the authentication by OAuth policy. The authentication by the 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.

To support PingIdentity authorization server for OAuth policies, make sure that you set the `Provider` field correctly in the policy file, as follows:

```
<ns:Provider>PingIdentity</ns:Provider>
```

See [AuthenticationbyOAuth Policy](#).

Kerberos SPNEGO

TIBCO API Exchange Gateway provides the SPNEGO security policy so that the facade operations with HTTP bindings can authenticate the consumer's identity using the SPNEGO token. The SPNEGO token is authenticated using a Kerberos service provider shared resource.

See [AuthenticationbySPNEGO.Policy](#)

To configure the Kerberos SPNEGO policy, refer to [Configuration Setup for Kerberos SPNEGO Authentication Policy](#).

Custom Authentication

TIBCO API Exchange Gateway provides an extensible authentication framework to support non-standard authentication. For example, when you want to verify the user credentials from a request that are stored in a proprietary way, or verify the credentials against a custom identity store, you can extend the base login module as per your requirements.

TIBCO API Exchange Gateway enables the development of an authentication framework using the custom shared resource. See [Authentication Using Custom Shared Resource](#).

Authorization

TIBCO API Exchange Gateway supports following authorization policies:

Role

Authorization by role policy of TIBCO API Exchange Gateway provides a way to authorize the user based on the role.

See [Authorization By Role Policy](#).

Confidentiality

TIBCO 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 the following policies:

- [Encryption.policy](#)
- [Decryption.policy](#)

Integrity

TIBCO 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 the following policies:

- [Sign](#)
- [Verify Signature](#)

CredentialMapping

This section explains the types of Credential Mapping policies.

TIBCO API Exchange Gateway can map the credentials of the subject from the authenticated principal in the form of SAML assertion, or can map the user name and password in the security header or the HTTP Authorization header by virtue of Credential mapping policies.

TIBCO API Exchange Gateway supports following policies for credential mapping:

- Basic
- usernameToken
- SAML
- OAuth

See the following policies:

- [CredentialMappingByUsernameToken Policy](#)
- [CredentialMappingBySAML Policy](#)
- [Credential Mapping by OAuth Policy](#)

Credential Mapping By OAuth

TIBCO 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 TIBCO API Exchange Gateway . Using gateway you can configure various types of policies to support authentication, authorization, integrity, confidentiality and credential mapping.



TIBCO 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](#) lists the sample template policy files for each supported policy.

To manage policies in TIBCO API Exchange Gateway product, you must do the following configuration setup:

- Define the shared resources. See [Configure Shared Resource](#).
- Create policies for the intended usage. See [Create Policy](#).
- Register the policies to the system. See [Registering Policy](#).
- Apply the policies to target operation. See [Applying Policies](#).

Configure Shared Resource

A policy file requires a configured shared resource.

You may configure an appropriate shared resource before you can create a policy. The table [Types of Security Shared Resources](#) explains the types of shared resources supported by TIBCO 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 good practice to use the sample files as templates and edit the properties as per your requirement. See [Shared Resources Properties](#) for details of properties for each supported shared resource.

Sample Files



- The property files for various supported shared resources are located under this directory:
ASG_CONFIG_HOME/default/security/resource
- See [Shared Resources Properties Sample Files](#) for sample file for each shared resource.

Registering Shared Resource with TIBCO API Exchange Gateway

Using the **Shared Resources** tab on the configuration, you can register the shared resources with TIBCO API Exchange Gateway.

To configure a shared resource, follow these steps:

Procedure

1. Start the Config UI server, if not already started. See [Starting GUI](#).
2. Create a new project or select an existing project under **Projects**.
3. Click the **SECURITY** tab on the right hand side.
4. Click **Shared Resources** tab on top menu.
5. Enter the details for shared resource defined as follows:

Shared Resource Name Configuration

Parameter	Description
New Resources File	A new property file which defines the Shared Resources configuration. See Shared Resources Properties for details.
Existing Resources Files	Select a configuration file of Shared Resource configuration. See Shared Resources Properties for details.

6. Save changes to the project.



- The shared resource name is defined in the properties file by the following property:
`com.tibco.governance.sharedresource.name`
For example, the shared resource name is defined in the properties file as follows:
`com.tibco.governance.sharedresource.name=LdapAsp`
- The shared resource name from the properties file must match the shared resource name in the policy file defined by ResourceInstance property.
`<tpa:SharedResourceLoginModule ResourceInstance="LdapAsp"/>`

Create Policy

Use shared resource to create a policy file.

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](#) for details.

You must define the correct shared resource for a specific policy. For example, you must define LDAP shared resource for a username token authentication policy.

The following table lists the shared resource required for a specific policy.

Policy And Shared Resource Property File

Policy	Shared Resource	Shared Resource Property File (Resource Files)
<ul style="list-style-type: none"> UsernameToken authentication Basic authentication 	LDAP Shared Resource	Properties for LDAP Authentication Shared Resource
SAML Authentication	WSS Shared Resource	WssAsp.properties
SiteMinder Authentication	SiteMinder Shared Resource	Properties for SiteMinder Service Provider
SPNEGO Authentication	Kerberos SPNEGO Shared Resource	
Sign	Subject Shared Resource	Properties for Subject Identify Provider (SIP)
Decryption	Subject Shared Resource	Properties for Subject Identify Provider (SIP)
Verify Signature	Trust shared resource	Properties for Trust Identify Provider (TIP)
Encryption	Trust shared resource	Properties for Trust Identify Provider (TIP)
Credential Mapping <ul style="list-style-type: none"> UsernameToken SAML 	<ul style="list-style-type: none"> UsernameToken - Password identity provider Keystore - Password identity provider SAML - Subject Identity Provider 	

Creating Policy File

You can create any supported policy file in the following ways:

Procedure

1. Copy a sample template file from the following location:
`ASG_CONFIG_HOME/default/policy`
2. Edit the parameters in the file as required. For example, you must change the `ResourceInstance` parameter to match the shared resource name defined in the properties file.

```
ResourceInstance="LdapAsp" .
```



- The policy must be a well-formed WS policy.
- All the resource instances in the policy must be defined in the shared resource properties file.

Sample Template Policy Files

Following table lists the policy template (sample) file for each of the supported policy:

Sample Template Policies

Policy Type	Template File
Authentication	Basic Authentication Policy BasicAuthentication.policy
	Username token Authentication Policy AuthenticationByUsernameToken.policy
	SAML Authentication Policy AuthenticationBySaml.policy
	SiteMinder Authentication Policy AuthenticationBySiteMinder.policy
	OAuth Authentication Policy AuthenticationbyOAuth Policy
	SPNEGO Authentication Policy AuthenticationBySPNEGO
Authorization	Authorization By Role Policy Authorization By Role Policy
Confidentiality	Decryption policy Decryption.policy

Policy Type	Template File
	Encryption policy Encryption.policy
Integrity	Sign policy Sign.policy
	VerifySignature policy VerifySignature.policy
CredentialMapping	UsernameToken Credential Mapping UsernameToken Credential Mapping
	SAML Credential Mapping SAML Credential Mapping
	OAuth Credential Mapping CredentialMappingByOAuth.policy

Registering Policy

You can register a policy on the Config UI by uploading a policy file and set the name for a policy.

To register a policy, follow these steps:

Procedure

1. Start the Config UI, if it is not running.
2. Create a new project or select an existing project under Projects.
3. Click the **SECURITY** tab on the right hand side.
4. Click the **Policy Mapping** tab on the top menu.
5. Click the **Add Property** icon to add a new policy mapping.
6. Enter the following parameters for the policy:

Policy Mapping Parameters

Type	Description
Policy Name	<ul style="list-style-type: none"> • Specifies the name for the policy. • Required.
Intent	Set the type of the policy. For example, Authentication. See Policy Types and SubTypes . See Types of Security Policies for details.

Type	Description
Qualifier	Set the policy sub type. For example, UsernameToken See Policy Types and SubTypes . See Types of Security Policies for details.
New Policy File	Specifies the policy definition file. Browse to choose a new policy file. See Create Policy to create the policy definition file.
Existing Policy File	Specifies an existing policy definition file. The policy file must exist in the gateway <code>ASG_CONFIG_HOME/configuration/policy</code> folder. For example, for the default configuration, the policy file must exist in the <code>ASG_CONFIG_HOME/default/policy</code> folder.

7. Save changes to the project or configuration.

Applying Policies

You can apply a policy on the Config UI by associating an existing policy to a target operation or reference endpoint.


Use Policy Binding to associate the policy with one or more target operation endpoints.


To apply any registered policy to a target operation or reference endpoints, follow these steps:

Procedure

1. Start the Config UI, if not running.
2. Create a new configuration or select an existing configuration.
3. Click the **SECURITY** tab on the right hand side.
4. Click the **Policy Binding** tab on the top menu.
5. Click the **Add Property** icon to add a new policy binding.
6. Enter the following parameters for the policy:

Policy Binding Parameters

Parameter	Description
Policy	Specifies a name for the policy. The policy name must be configured under the Policy Mapping tab.
URI	Specifies the URI of the operation to which the policy is applied.  If the URI in facade operation is left blank, then URI field for security binding should use SOAP action instead of URI.
Facade Operation	Specifies the operation to which the policy is applied. The facade operation must be configured in the Facade Operation tab.
Target Operation	Specifies the target operation. The target operation must be configured in the Target Operation tab.

Parameter	Description
Binding	Specifies the binding component that the policy is applied to. This could be either a facade operation (service) or a target operation (reference).
Flow	Specifies the flow of the request or response. The possible values are: <ul style="list-style-type: none"> • in • out
Partner	Specifies the partner to which the policy is applied. This field can be left blank in case no partner needs to be applied.
Type	Specifies the type of the request. <div>  <div>Set this to SOAP for any SOAP request, or to http for any non-SOAP http request.</div> </div>

7. Save changes to the project or configuration.

Policy Use Cases

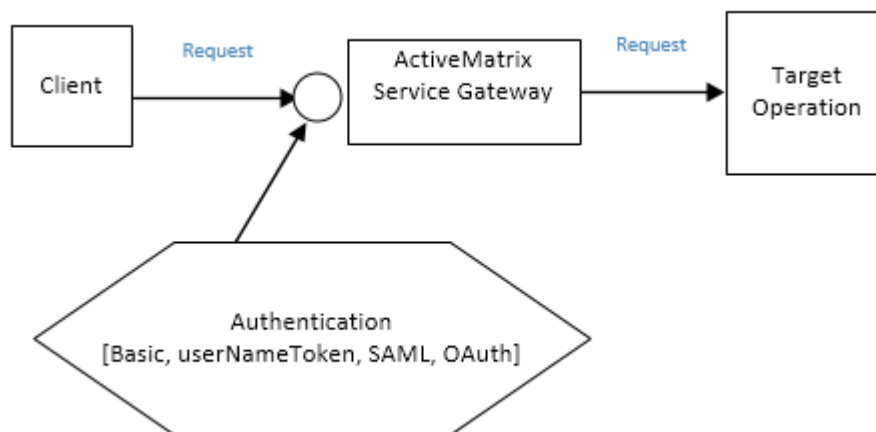
This section describes the use cases for supported policies.

Authentication Policies

This section describes the configuration to apply authentication policies.

Figure [Authentication Policies](#) illustrates how to apply the authentication policies for an incoming request.

Authentication Policies



Configuring Authentication Policies

To configure the authentication policies, follow these steps:

Procedure

1. Configure Shared Resource.

See [Configure Shared Resource](#) for details.

2. Create Policy.

See [Create Policy](#) for details.

3. Register Policy.

See [Registering Policy](#) for detailed steps. You must choose the correct type and subtype to register the policy as shown in the following table:

Policy	Intent	Qualifier
Basic Authentication	Authentication	Basic
UsernameToken Authentication	Authentication	UsernameToken
SAML Authentication	Authentication	SAML
Authentication by OAuth	Authentication	OAuth
SPNEGO Authentication	Authentication	SPNEGO



Use the custom shared resource in an authentication policy to implement a custom login module, which can be used for the authentication in the following ways:

- Extract the user credentials from an incoming request available in a non-standard format.
- Query the user credentials stored in the database to authenticate the user.

See [Authentication Using Custom Shared Resource](#) for details.

4. Apply Policy.

See [Applying Policies](#) for details.

Authentication Policies Types

TIBCO API Exchange Gateway supports the following authentication policies:

Basic

- The client sends the request with user name/password in HTTP basic authentication header.
- The Basic Authentication policy authenticates the request against LDAP.

Example Policy

BasicAuthentication.policy

```
<wsp:Policy
  xmlns:tpa="http://xsd.tns.tibco.com/governance/policy/action/2009"
  xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
  xmlns:wssp="http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702">
  <wsp:All>
    <tpa:AuthenticationByJaas>
      <wssp:SupportingTokens>
        <tpa:Any>

                                <wssp:HttpBasicAuthentication/>

                                </tpa:Any>
      </wssp:SupportingTokens>
    </tpa:AuthenticationByJaas>
  </wsp:All>
</wsp:Policy>
```

```

    <tpa:SharedResourceLoginModule ResourceInstance="LDAPSource">
  </tpa:SharedResourceLoginModule>
</tpa:AuthenticationByJaas>
</wsp:All>
</wsp:Policy>

```

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

```

<wsp:Policy
xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
  xmlns:wssp="http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702"
xmlns:tpa="http://xsd.tns.tibco.com/governance/policy/action/2009" >
  <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

```

<wsp:Policy
xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
xmlns:wssp="http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702"
xmlns:tpa="http://xsd.tns.tibco.com/governance/policy/action/2009">
  <wsp:All>
    <wsp:Policy >
      <tpa:WssProcessor ResourceInstance="WssAsp"/>
    </wsp:Policy>
    <wsp:Policy      xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy">
      <tpa:VerifyAuthentication>
        <wsp:ExactlyOne>
          <wssp:SignedSupportingTokens>
            <wsp:ExactlyOne>
              <wssp:SamlToken>
                <!-- The following will enable verify IssuerName
                  <wssp:IssuerName>urn:test.tibco.com</wssp:IssuerName>
                -->
                <!-- The following will enable verify SAML version
                  <wssp:WssSamlV20Token11/>
                -->
              </wssp:SamlToken>
            </wsp:ExactlyOne>
          </wssp:SignedSupportingTokens>
        </wsp:ExactlyOne>
      </tpa:VerifyAuthentication>
    </wsp:Policy>
  </wsp:All>
</wsp:Policy>

```

```

    </tpa:VerifyAuthentication>
  </wsp:Policy>
</wsp:All>
</wsp:Policy>

```

SiteMinder

- The client sends the request containing the SiteMinder session cookie in the HTTP header.
- SiteMinder authentication policy authenticates the request against SiteMinder service provider.

Example Policy

AuthenticationBySiteMinder.policy

```

<wsp:Policy
<wsp:Policy
  xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
  xmlns:wssp="http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702"
  xmlns:tpa="http://xsd.tns.tibco.com/governance/policy/action/2009" >
  <wsp:All>
    <wsp:ExactlyOne>
      <tpa:AuthenticationByJaas>
        <wssp:SupportingTokens>
          <tpa:ExactlyOne>
            <tpa:SiteminderToken />
          </tpa:ExactlyOne>
        </wssp:SupportingTokens>
        <tpa:SharedResourceLoginModule ResourceInstance="SiteminderAsp" />
      </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. Because the protected facade operation does not access user's resources, the access token is only used for authentication purposes.

Example Policy

AuthenticationbyOAuth Policy

```

<?xml version="1.0" encoding="UTF-8"?>
<wsp:Policy xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy">
  <ns:AuthenticationByJaas xmlns:ns="http://xsd.tns.tibco.com/governance/policy/
action/2009">
    <ns:OAuthToken>
      <ns:Provider>TIBCO</ns:Provider>
      <ns:ClientID>security</ns:ClientID>
      <ns:ClientSecret>ef6e7dca3d52973f73ec3dd0da7087d400f5a05a</ns:ClientSecret>
      <ns:CallbackURI>http://localhost:9322/asg/oauth2/client/callback</
ns:CallbackURI>
    </ns:OAuthToken>
  </ns:AuthenticationByJaas>
</wsp:Policy>

```

Schema for OAuth Policies

The supported OAuth policies use the following OAuth assertion schema:

OAuth Policy Schema

```
<xsd:complexType name="OAuthAssertion">
  <xsd:sequence>
    <xsd:element name="Provider" type="xsd:string" minOccurs="1"
maxOccurs="1" />
    <xsd:element name="ClientID" type="xsd:string" minOccurs="1"
maxOccurs="1" />
    <xsd:element name="ClientSecret" type="xsd:string" minOccurs="1"
maxOccurs="1" />
    <xsd:element name="CallbackURI" type="xsd:string" minOccurs="0"
maxOccurs="1" />
    <xsd:element name="Scopes" type="xsd:string" minOccurs="0"
maxOccurs="1" />
    <xsd:choice>
      <xsd:element name="AuthorizationEndpoint" type="xsd:string"
minOccurs="1" maxOccurs="1" />
      <xsd:element name="TokenEndpoint" type="xsd:string" minOccurs="1"
maxOccurs="1" />
    </xsd:choice>
    <xsd:element name="Username" type="xsd:string" minOccurs="0"
maxOccurs="1" />
    <xsd:element name="Password" type="xsd:string" minOccurs="0"
maxOccurs="1" />
    <xsd:element name="AccessToken" type="xsd:string" minOccurs="0"
maxOccurs="1" />
    <xsd:element name="RefreshToken" type="xsd:string" minOccurs="0"
maxOccurs="1" />
    <xsd:element name="TokenValidator" type="xsd:string" minOccurs="0"
maxOccurs="1" />
    <xsd:element name="TokenType" type="tpa:EnumTokenType" minOccurs="0"
maxOccurs="1" default="Query"/>
  </xsd:sequence>
</xsd:complexType>
```

OAuth Policy File Fields

The OAuth 2.0 policy file contains the following fields:

OAuth Policy File Fields

Field	Description	Example Value
Provider	<p>Specifies the provider for authorization server used for authentication. The supported values are as follows:</p> <ul style="list-style-type: none"> Tibco Use this value for TIBCO authorization server. PingIdentify Use this value for PingIdentity authorization server 	Tibco
ClientID	Specifies the client identifier issued to the client by the authorization server during the registration process.	gateway

Field	Description	Example Value
ClientSecret	Specifies the client secret of the registered application.	UbrJ0YxG
CallbackURI	Optional. Specifies the callback URL.	https://localhost:9333/asg/oauth2/client/callback
Scopes	Specifies a list of comma separated scopes.	public
AuthorizationEndpoint	Specifies an endpoint on the authorization server where the client requests for authorization.	https://localhost:9333/asg/oauth2/authorization
TokenEndpoint	Specifies an endpoint on the authorization server where the client requests for access token.	https://localhost:9333/asg/oauth2/access_token
Username	Optional. Specifies the user name of the resource owner who requests the token.	asgtibcouser
Password	Optional. Specifies the password of the resource owner who requests the token.	asgtibcopass

SPNEGO

- The client sends the request containing the SPNEGO token in the HTTP header.
- SPNEGO authentication policy authenticates the request against the specified Kerberos service provider.

Example Policy

AuthenticationBySPNEGO.policy

```
<wsp:Policy xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
  xmlns:wssp="http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702"
  xmlns:tpa="http://xsd.tns.tibco.com/governance/policy/action/2009">
  <wsp:All>
    <wsp:ExactlyOne>
      <tpa:AuthenticationByJaas>
        <wssp:SupportingTokens>
          <tpa:ExactlyOne>
            <wssp:SpnegoContextToken />
          </tpa:ExactlyOne>
        </wssp:SupportingTokens>
        <tpa:SharedResourceLoginModule
          ResourceInstance="KerberosAsp">
          <tpa:Properties>
            <tpa:Property Name="ServiceName" Value="HTTP/vm-w2k8-
spm13.support.ch.com@SUPPORT.CH.COM" />
          </tpa:Properties>
        </tpa:SharedResourceLoginModule>
      </tpa:AuthenticationByJaas>
    </wsp:ExactlyOne>
  </wsp:All>
</wsp:Policy>
```

Configuration Setup for Kerberos SPNEGO Authentication Policy

This section explains the configuration setup required to process SPNEGO Kerberos authentication requests for TIBCO API Exchange Gateway.

Creating a User Account in the Microsoft Active Directory for TIBCO API Exchange Gateway

You must create a user account in the Microsoft Active Directory for the machine where TIBCO API Exchange Gateway is installed.

Prerequisites

Microsoft Windows Active Directory is installed.

Procedure

1. Click **Start->All Programs-> Control Panel > All Control Panel Items > Administrative Tools->Active Directory Users and Computers**.

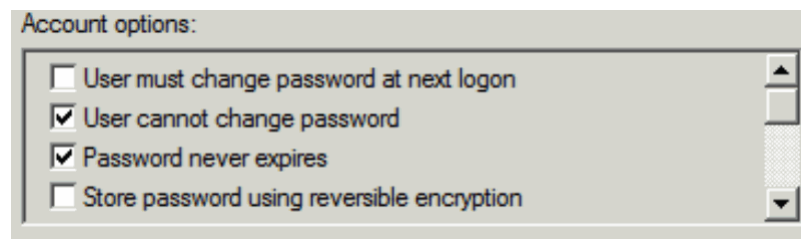
2. Enter a value for **User logon name**.

Use the machine name of the TIBCO API Exchange Gateway installation as the user.

For example, if your machine name is `testtibcoapix.pa.tibco.com`, create a new user in an Active Directory as `testtibcoapix`.

3. Make sure that the account options are chosen as follows:

- Do not select **User must change password at next logon**.
- Select **User can not change password**.
- Select **Password never expires**.



Mapping the Service Principal Name (SPN) to a Microsoft User Account

The Kerberos service principal name is defined as *service name/Full_Qualified_Host_Name*

Prerequisites

A user account is created in the Active Directory.

Procedure

1. Open a command prompt window.
2. Enter the following command to map the Kerberos service principal name (SPN) to a user account:

```
setspn -A service_name/fully_qualified_host_name user account
```

For example, for the `HTTP/testtibcoapix.pa.tibco.com` service principal name, use the following command:

```
setspn -A HTTP/testtibcoapix.pa.tibco.com testtibcoapix
```

- setspn tool is found on the Windows 2000 Resource Kit.
- Refer to the following url for details:

<http://social.technet.microsoft.com/wiki/contents/articles/717.service-principal-names-spn-setspn-syntax-setspn-exe.aspx>

Generating a Keytab File for an SPN

You can generate the Kerberos keytab file (krb5.keytab) for an SPN using the ktpass tool from the Windows Server toolkit.

Prerequisites

- Make sure you have created a user account in the Microsoft Active Directory.
- Ensure that you have mapped the service principal name to the user account.



The instructions in this topic are for Windows platform only.

Procedure

1. Open a command prompt.
2. To generate the keytab file, type the ktpass command:

```
ktpass -out Path_To_Keytab_file
-princ service_name/fully_qualified_host_name -passPasswordValue
-mauser user_logon_name -mapOp set -cryptoEncryption_Key_Type
-pType KRB5_NT_PRINCIPAL

ktpass -out c:\temp\apixg.keytab
-princ HTTP/testtibcoapix.pa.tibco.com -pass testtibcopass -mapUser test
\testtibcoapix -mapOp set -crypto all
-pType KRB5_NT_PRINCIPAL
```

Refer to the following table for command line options:

Command Line Options for ktpass Command

Parameter	Description	Example Value
-out	Specifies the name of the Kerberos keytab output file.	c:\temp\apixg.keytab
-princ	Specifies the principal user name. The value for this field is defined in the form user@REALM . The concatenation of the user logon name, and the realm must be uppercase.	HTTP/ testtibcoapix.pa.tibco.com
-pass	Specifies a password for the principal user name that is specified by the princ option.	testtibcopass
-mapUser	Maps the name of the Kerberos principal, which is specified by the princ option to the specified domain account.	test\testtibcoapix

Parameter	Description	Example Value
-mapOp	Specifies how the mapping attribute is set.	set sets the value for Data Encryption Standard (DES)-only encryption for the specified local user name.
-crypto	Specifies the encryption key type that are generated in the keytab file. The possible values are: <ul style="list-style-type: none"> DES-CBC-CRC AES256-SHA1 all 	all, which indicates that all supported cryptographic types can be used.

Refer to <https://technet.microsoft.com/en-us/library/cc753771.aspx> for ktpass command syntax details.

Authentication Using Custom Shared Resource

You may use a custom shared resource in an authentication policy to authenticate a request.

The custom shared resource contains a custom login module implemented by the user. The custom shared resource is specified using the `ResourceInstance` attribute in a policy.

For example, if the custom shared resource `CelmAsp` is configured in the `CelmAsp.properties` file, use it as follows in an authentication policy:

```
<tpa:WssProcessor ResourceInstance="CelmAsp"/>
```

TIBCO API Exchange Gateway calls the custom login module defined in the `CelmAsp.properties` file.

Refer to [Create Shared Resource Properties File for Custom Authentication](#) for custom shared resource properties.

To use custom shared resource for an authentication, you must complete the following tasks:

- Implement the custom login module. See [Implement Custom Login module](#).
- Package the custom login module. See [Packaging the Custom Login Module Jars](#).

Example Authentication Policy Using Custom Shared Resource

AuthenticationByCelm.policy

```
<wsp:Policy
  xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
  xmlns:wssp="http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702"
  xmlns:tpa="http://xsd.tns.tibco.com/governance/policy/action/2009">
  <wsp:All>
    <wsp:Policy >
      <tpa:WssProcessor ResourceInstance="CelmAsp"/>
    </wsp:Policy>
    <wsp:Policy xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy">
      <tpa:VerifyAuthentication>
        <tpa:ExactlyOne>
          <wssp:UsernameToken />
        </tpa:ExactlyOne>
      </tpa:VerifyAuthentication>
    </wsp:Policy>
  </wsp:All>
</wsp:Policy>
```

Implementing Custom Login Module

The custom login module uses the LoginModule of Java Authentication and Authorization Service (JAAS).

TIBCO API Exchange Gateway provides an abstract LoginModule class that implements the JAAS LoginModule. You must extend the abstract LoginModule, as required.

For example, you can extend the abstract LoginModule for the following functionality:

- Extract security information or credentials such as user name and password from the request.
- Authenticate the extracted credentials as required.



To create your custom login module, refer to [Sample Custom LoginModule](#) class.

Abstract LoginModule

The CelmAbstractLoginModule is an abstract login module class.

You must extend the CelmAbstractLoginModule base class to implement the custom authentication module. This abstract login module class implements the methods of the LoginModule of Java JAAS and populates some fields before passing the control to the custom login module class implemented by the user. When the control is passed to the user specific implementation module, the user specific implementation module may retrieve object such as HTTP request and response or SOAP document depending on the type of authentication is configured.

Custom LoginModule

The custom login module class must be extended from CelmAbstractLoginModule abstract base class.

The CelmAbstractLoginModule class wraps LoginModule methods such as login() and delegates the method to doLogin(). The custom login module class should override the methods of the abstract class such as doLogin() or doCommit() to provide customized authentication processing. Any resource allocation should be cleaned up in the doLogout() or doAbort() methods.

Example doLogin() Method of a Custom LoginModule

The following is an example of doLogin() of a custom LoginModule:

```
/**
 * Perform login.
 * This method is called when login() in LoginModule is called.
 */
@Override
public boolean doLogin() throws LoginException
{
    if (getMessageElement() != null)
        extractUsernamePassword(getMessageElement().getOwnerDocument());
    return authenticateUsernamePassword(getUsername(), getPassword());
}
```

Refer to [Sample Custom LoginModule](#) for a complete sample custom login module class.

Sample Custom LoginModule

The following code defines a sample custom login module
`CelmExampleUsernamePasswordLoginModule` class:

CelmExampleUsernamePasswordLoginModule Sample Class

```

package com.example.security.authentication.provider.celm;
import java.util.HashSet;
import java.util.Map;
import java.util.Set;
import javax.security.auth.Subject;
import javax.security.auth.callback.CallbackHandler;
import javax.security.auth.login.LoginException;
import org.opensaml.xml.util.XMLHelper;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.w3c.dom.Document;
import org.w3c.dom.NodeList;
import com.tibco.asg.security.provider.celm.CelmAbstractLoginModule;
/**
 * CelmExampleUsernamePasswordLoginModule is an example LoginModule
 * to demonstrate how to extend
 * CelmExampleUsernamePasswordLoginModule.
 */
public class CelmExampleUsernamePasswordLoginModule extends
CelmAbstractLoginModule
{
    private static Logger logger =
    LoggerFactory.getLogger(CelmExampleUsernamePasswordLoginModule.class)
    ;
    private String authorizedUsername = null;
    private String authorizedPassword = null;
    private String authorizedRole = null;
    /**
     * Perform initialize.
     * <p>
     * This method is called when login() in initialize is called.
     *
     * @param subject the Subject for authentication.
     * @param callbackHandler the callback handler to provide
     authentication
     * variables like username/password, etc.
     * @param sharedState
     * @param options a map contains the options from properties file.
     */
    @Override
    public void initialize(Subject subject, CallbackHandler
callbackHandler,
final Map<String, ?> sharedState, final Map<String, ?> options)
    {
        super.initialize(subject, callbackHandler, sharedState, options);
        authorizedUsername =
        (String)options.get("com.tibco.asg.security.provider.celm.authorizedU
sername");
        authorizedPassword =
        (String)options.get("com.tibco.asg.security.provider.celm.authorizedP
assword");
        authorizedRole =
        (String)options.get("com.tibco.asg.security.provider.celm.authorizedR
ole");
    }
    /**
     * Perform login.
     * <p>
     * This method is called when login() in LoginModule is called.
     */
    @Override
    public boolean doLogin() throws LoginException
    {
        if (getMessageElement() != null)
        extractUsernamePassword(getMessageElement().getOwnerDocument());
        return authenticateUsernamePassword(getUsername(), getPassword());
    }
}

```

```

/**
 * Perform commit.
 * <p>
 * This method is called when commit() in LoginModule is called.
 *
 */
@Override
public boolean doCommit() throws LoginException
{
    return true;
}
/**
 * Perform cleanup.
 * <p>
 * This method is called when cleanup() in LoginModule is called.
 *
 */
@Override
public void doCleanUp() throws LoginException
{
    super.doCleanUp();
}
//
// The methods below are example to show:
// 1. How to username/password can be extracted from SOAP header.
// 2. AUthenticat the user.
//
private static final String WSSE_NS = "http://schemas.xmlsoap.org/ws/
2002/04/secext";
private static final String USERNAME = "Username";
private static final String PASSWORD = "Password";
/**
 * Extract username and password from the request.
 *
 * This example extract the username and password from the document.
 * It is expecting the username and password is from a element with
 * a specific namespace.
 *
 * This example does not handle and valid replay attack.
 *
 * @param document
 * @return
 * @throws LoginException
 */
private void extractUsernamePassword(Document document) throws
LoginException
{
    try
    {
        NodeList nl = null;
        nl = document.getElementsByTagNameNS(WSSE_NS, USERNAME);
        if (nl.getLength() != 0)
        {
            // save the username
            setUsername(nl.item(0).getFirstChild().getNodeValue());
        }
        nl = document.getElementsByTagNameNS(WSSE_NS, PASSWORD);
        if (nl.getLength() != 0)
        {
            // save the password
            setPassword(nl.item(0).getFirstChild().getNodeValue());
        }
    }
    catch (Exception e)
    {
        logger.error("Unable to extract username/password from request", e);
    }
}
/**
 * Authenticate the given username and password.
 *
 * This example validate the username/password against the username/

```

```

password
* extracted from the properties file in the
* {@link #initialize(Subject, CallbackHandler, Map, Map)} method
above.
*
* Note that name identifier should be set if username is
authenticated.
* The name identifier is used for the Subject's NameIdentifier when
* building the SAML assertion.
*
* There may be other way to validate the username/password, for
example,
* look up password for the user in database.
*
* @param username username to verify.
* @param password password to verify.
* @throws LoginException throw if authentication failed.
*/
private boolean authenticateUsernamePassword(String username, String
password)
throws LoginException
{
if (authorizedUsername == null && authorizedPassword == null)
throw new LoginException("Invalid credentials");
if (username.equals(authorizedUsername) &&
password.equals(authorizedPassword))
{
this.setNameIdentifier(username);
this.setRoles(extractRoles(username));
return true;
}
throw new LoginException("Invalid credentials");
}
/**
* Extract roles for the given user.
*
* This example adds the role extracted from the properties file.
*
* There may be other way to get the roles for the authenticate user,
* for example, look up the roles for the user in database.
*
* @param username user whos roles to be extracted
* @return set of roles for the specified user.
*/
private Set<String> extractRoles(String username)
{
HashSet<String> roleSet = new HashSet<String>();
if (authorizedRole != null)
roleSet.add(authorizedRole);
return roleSet;
}
}

```



To compile the `CelmExampleUsernamePasswordLoginModule` class, you must include `asg-function.jar` in the class path.

Packaging and Deploying the Custom Login Module

You must package the custom login module in a jar file for deployment with TIBCO API Exchange Gateway.

Procedure

1. Create a JAR file containing the implementation class of custom login module.
2. Copy the JAR file and the dependencies to the `ASG_HOME/lib/ext/tpcl` directory.



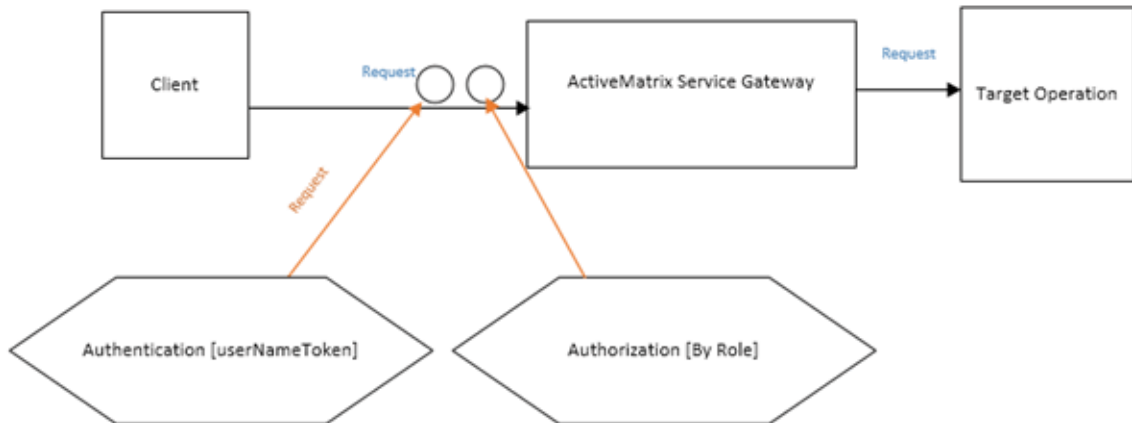
TIBCO API Exchange Gateway loads the custom login module at run time.

Applying Authorization Policies

This section explains the configuration to apply authorization policies.

Figure [Authorization Policies](#) illustrates how to apply an authorization policy based on the role or how you can apply an OAuth policy.

Authorization Policies



Configuring Authorization Policies

To configure a authorization policies, complete the following tasks:

Procedure

1. Configure Shared Resource. See [Configure Shared Resource](#) for details.
2. Create Policy. See [Create Policy](#) for details.
3. Register Policy.
See [Registering Policy](#) for detailed steps. You must choose the correct type and subtype to register the policy as shown in the following table:

Policy	Type	SubType
Authorization By Role	Authorization	Role

4. Apply Policy.
See [Applying Policies](#) for details.

Authorization Policies Types

TIBCO 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 the user name 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 or 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

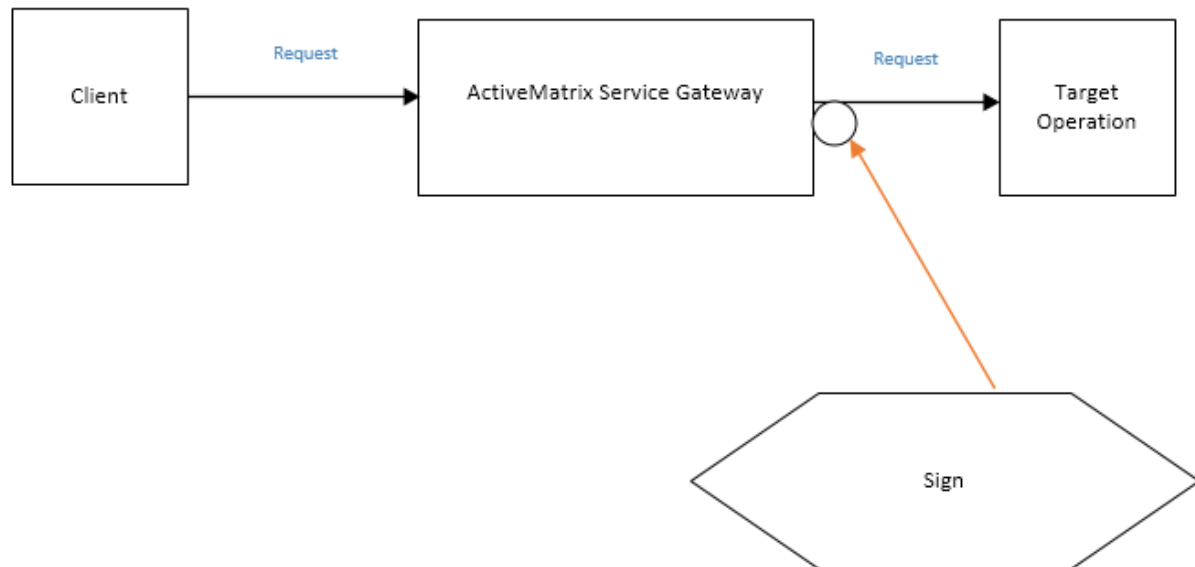
```
<?xml version="1.0" encoding="UTF-8"?>
<wsp:Policy
  xmlns:tpa="http://xsd.tns.tibco.com/governance/policy/action/2009"
  xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
  xmlns:wssp="http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702"
  xmlns:xacml="urn:oasis:names:tc:xacml:2.0:policy:schema:os" 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>
                </xacml:Rule>
              </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:AttributeValue
                                DataType="xsd:string">queryBookByAuthorBW</
                                xacml:Attribute
                                Value>
                                <xacml:AttributeSelector
                                DataType="xsd:string"
                                RequestContextPath="" />
                          </xacml:ActionMatch>
                        </xacml:Action>
                      </xacml:Actions>
                    </xacml:Target>
                  <xacml:Condition>
                    <xacml:Apply FunctionId="is-in">
                      <xacml:AttributeValue
                        DataType="xsd:string">Accounting
                        Managers</
                        xacml:AttributeValue
                      </xacml:Apply>
                    </xacml:Condition>
                  </xacml:Rule>
                </tpa:Operation>
              </tpa:ByRole>
            </tpa:Authorization>
          </wsp:All>
        </wsp:Policy>
      </wsp:All>
    </wsp:Policy>
```


Integrity Policies

This section describes the configuration to apply integrity policies.

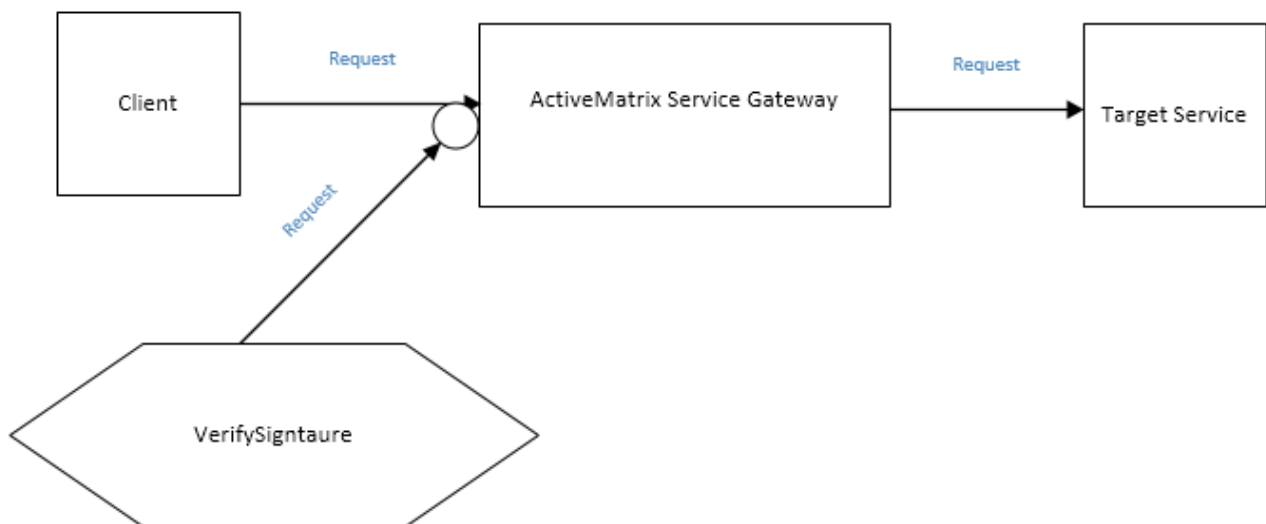
The following figure shows how to apply a sign policy:

Sign Policy



The following figure shows how to apply a verify signature policy:

VerifySignature Policy



Configuring Integrity Policies

To configure the integrity policies, complete the following tasks:

Procedure

1. Configure Shared Resource. See [Configure Shared Resource](#) for details.
2. Create Policy. See [Create Policy](#) for details.
3. Register Policy.
See [Registering Policy](#) for detailed steps. You must choose the correct type and subtype to register the policy as shown in the following table:

Policy	Type	SubType
Sign	Integrity	Sign
Verify Signature	Integrity	Verify

4. Apply Policy.
See [Applying Policies](#) for details.

Integrity Policies Types

TIBCO 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

```
<wsp:Policy xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy">
  <wsp:All>
    <wsp:Policy xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
xmlns:ns1="http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702">
      <ns:Signature xmlns:ns="http://xsd.tns.tibco.com/governance/policy/
action/2009" ResourceInstance="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

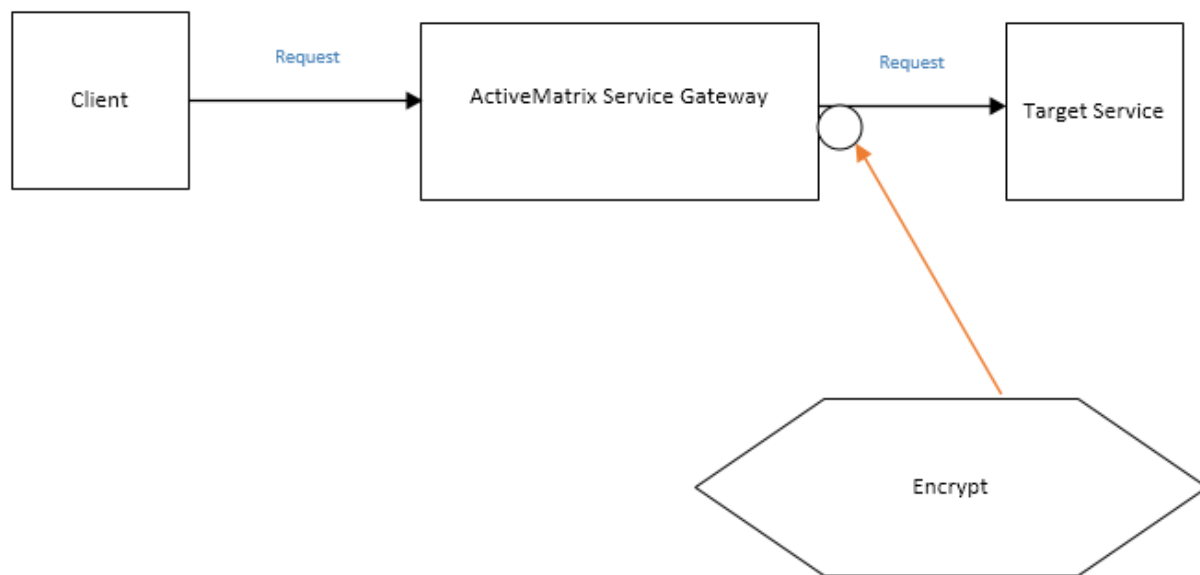
```
<wsp:Policy xmlns:tpa="http://xsd.tns.tibco.com/governance/policy/action/2009"
xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy" xmlns:wssp="http://
docs.oasis-open.org/ws-sx/ws-securitypolicy/200702">
  <wsp:All>
    <wsp:Policy>
      <tpa:WssProcessor ResourceInstance="WssAsp" />
    </wsp:Policy>
    <wsp:Policy>
      <tpa:VerifyAuthentication>
        <wssp:SignedSupportingTokens>
          <wssp:SamlToken />
        </wssp:SignedSupportingTokens>
      </tpa:VerifyAuthentication>
    </wsp:Policy>
    <wsp:Policy>
      <tpa:VerifySignature>
        <wssp:SignedParts>
          <wssp:Header Namespace="http://docs.oasis-open.org/wss/2004/01/
oasis-200401-wsswssecurity-
secext-1.0.xsd" />
          <wssp:Body />
        </wssp:SignedParts>
      </tpa:VerifySignature>
    </wsp:Policy>
  </wsp:All>
</wsp:Policy>
```

Confidentiality Policies

Configuration to apply confidentiality policies.

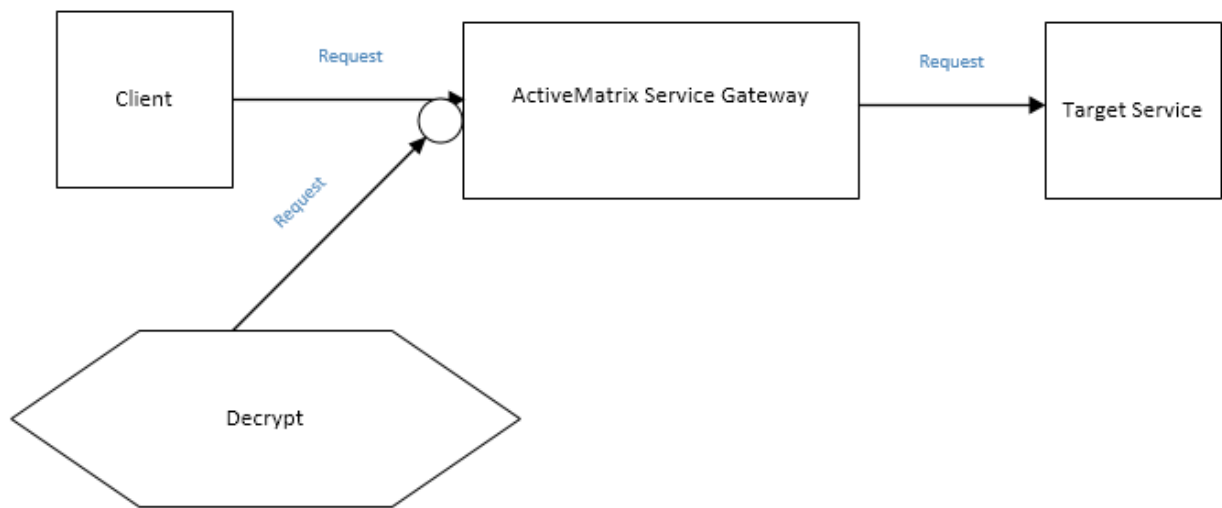
The following figure shows how to apply an encrypt policy.

Encryption Policy



The following figure shows how to apply the decrypt policy.

Decrypt Policy



Configuring Confidential Policies

To configure the confidentiality policies, do the following tasks:

Procedure

1. Configure Shared Resource
See [Configure Shared Resource](#) for details.
2. Create Policy
See [Create Policy](#) for details.
3. Register Policy
See [Registering Policy](#) for detailed steps. You must choose the correct type and subtype to register the policy as shown in the following table:

Policy	Type	SubType
Encrypt	Confidentiality	Encrypt
Decrypt	Confidentiality	Decrypt

4. Apply Policy
See [Applying Policies](#) for details.

Confidentiality Policies Types

TIBCO 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

```
<wsp:Policy xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy">
  <wsp:All>
    <wsp:Policy xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
xmlns:ns1="http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702">
      <ns:Encryption xmlns:ns="http://xsd.tns.tibco.com/governance/policy/
action/2009" 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

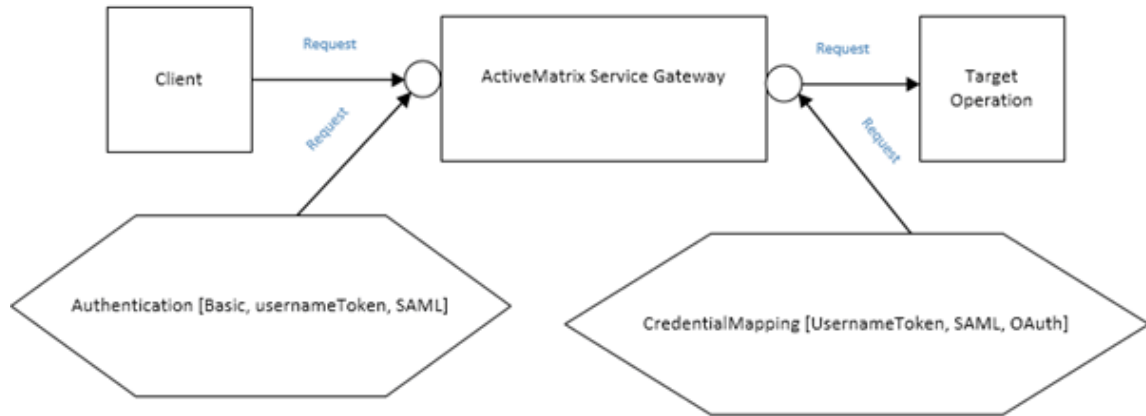
```
<wsp:Policy xmlns:tpa="http://xsd.tns.tibco.com/governance/policy/action/2009"
xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy" xmlns:wssp="http://
docs.oasis-open.org/ws-sx/ws-securitypolicy/200702">
  <wsp:All>
    <wsp:Policy>
      <tpa:WssProcessor ResourceInstance="WssAsp" />
    </wsp:Policy>
    <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

Configuration to apply credential mapping policies.

Figure [CredentialMapping Policies](#) illustrates how you can apply a credential mapping policy.

CredentialMapping Policies



Credential Mapping Policies Types

TIBCO API Exchange Gateway supports following types of credential mapping policies:

UsernameToken Credential Mapping

When applying UsernameToken Credential Mapping, remember the following points:

- 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

CredentialMappingByUsernameToken Policy

```

<wsp:Policy xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
xmlns:wssp="http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702"
xmlns:tpa="http://xsd.tns.tibco.com/governance/policy/action/2009">
  <wsp:All>
    <wsp:Policy xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy">
      <ns:CredentialMapping xmlns:ns="http://xsd.tns.tibco.com/governance/
policy/action/2009">
        <tpa:Fixed>
          <!-- Replace the username/password in the following
UsernameToken -->
          <wssp:UsernameToken xmlns:wsse="http://docs.oasis-open.org/wss/
2004/01/oasis-200401-
wss-wssecurity-secext-1.0.xsd">
            <wsse:Username>schalla</wsse:Username>
            <wsse:Password Type="http://docs.oasis-open.org/wss/2004/01/
oasis-200401-wssusername-
token-profile-1.0#PasswordText">password</wsse:Password>
          </wssp:UsernameToken>
        </tpa:Fixed>
        <wssp:SupportingTokens xmlns:wssp="http://docs.oasis-open.org/ws-
sx/ws-securitypolicy/
200702">
          <wssp:UsernameToken>
            <!-- Uncomment to generate digested password
<wssp:HashPassword />
-->
          </wssp:UsernameToken>
        </wssp:SupportingTokens>
      </ns:CredentialMapping>
    </wsp:Policy>
  </wsp:All>
</wsp:Policy>

```

```

        </wssp:SupportingTokens>
      </ns:CredentialMapping>
    </wsp:Policy>
  </wsp:All>
</wsp:Policy>

```

SAML Credential Mapping

When applying SAML Credential Mapping, remember the following points:

- 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 or 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.



To use credential mapping by SAML policy on a target operation, make sure that `SingleSignonSAMLSigner.properties` file is present in the resource directory of the gateway configuration project. The `SingleSignonSAMLSigner.properties` file is found in the `ASG_CONFIG_HOME\default\security\resource` directory.

Example Policy

CredentialMappingBySAML Policy

```

<wsp:Policy xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy">
  <wsp:All>
    <wsp:Policy xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy">
      <ns:CredentialMapping ResourceInstance="SubjectIsp" xmlns:ns="http://
xsd.tns.tibco.com/governance/policy/action/2009">
        <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

When applying Credential Mapping by OAuth, remember the following points:

- 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 propagates 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 xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy">
  <ns:CredentialMapping xmlns:ns="http://xsd.tibco.com/governance/policy/
action/2009">
    <ns:OAuth>
      <ns:Provider>TIBCO</ns:Provider>
      <ns:ClientID>security</ns:ClientID>
      <ns:ClientSecret>ef6e7dca3d52973f73ec3dd0da7087d400f5a05a</
ns:ClientSecret>
      <ns:CallbackURI>http://localhost:9322/asg/oauth2/client/callback</
ns:CallbackURI>
      <ns:Scope>public</ns:Scope>
      <ns:GrantType>OWNER_CREDENTIAL</ns:GrantType>
      <ns:Username>eric</ns:Username>
      <ns:Password>#!OG7dYlXHx1RknIJxgIx4TE08IXNX6+MhSiSAXov3K34=</ns:Password>
    </ns:OAuth>
  </ns:CredentialMapping>
</wsp:Policy>
```

Types of Security Shared Resources

Table [Types of Security Shared Resources](#) 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](#).

For example, to create authentication policy to authenticate an username against LDAP server, you must register the LDAP shared resource property file.

Types of Security Shared Resources

Type	Description
LDAP	LDAP authentication shared resource provides the ability to authenticate a username and password against an LDAP server.
Trust Identity	<p>The Trust Identity Provider is used for retrieving certificates required for performing trust operations from a credential store.</p> <p>For example, use Trust identity provider (TIP) for verifying signature or encryption and SSL client authentication.</p>
Subject Identity	<p>The Subject Identity Provider is used for retrieving and using private credentials obtained from a credential store.</p> <p>For example, use Subject identity provider (SIP) for signing or decryption.</p>
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](#) describes the properties for LDAP Authentication Shared Resource.

Properties for LDAP Authentication Shared Resource

Property	Description
<code>com.tibco.asg.intent.usernameToken</code>	
	<p>Boolean intent property indicates if the LDAP authentication method can be enforced on the request message or not. Possible values are true or false.</p> <p>If the value of this property set to true, the request message must contain a valid username token.</p>
<code>com.tibco.trinity.runtime.core.provider.authn.ldap.initialCtxFactory</code>	
	<p>Specifies the name of the JNDI Factory to use.</p> <p>The default value is <code>com.sun.jndi.ldap.LdapCtxFactory</code> (Sun's <code>LdapCtxFactory</code>).</p> <p>Optional.</p>
<code>com.tibco.trinity.runtime.core.provider.authn.ldap.serverURL</code>	
	<ul style="list-style-type: none"> Specifies the URL to connect to the LDAP directory server. TIBCO API Exchange Gateway supports list of multiple values separated by comma to configure LDAP server in a high availability and fault tolerant setup. The LDAP URL is defined as: <code>ldap://hostname1:port, ldap://hostname2:port</code> The LDAP SSL URL is defined as: <code>ldaps://hostname1:port, ldaps://hostname2:port,</code> Required.
<code>com.tibco.trinity.runtime.core.provider.authn.ldap.searchTimeout</code>	

Property	Description
	<p>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.</p> <p>Optional.</p>
<code>com.tibco.trinity.runtime.core.provider.authn.ldap.userAttributeUserName</code>	
	<p>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.</p> <p>Optional.</p>
<code>com.tibco.trinity.runtime.core.provider.authn.ldap.userAttributesExtra</code>	
	<p>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.</p> <p>For example, mail givenname</p> <p>Optional.</p>
<code>com.tibco.trinity.runtime.core.provider.authn.ldap.userSearchBaseDN</code>	
	<p>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.</p> <p>For example, ou=people,ou=na,dc=example,dc=org</p> <p>Required in admin (search) mode.</p>
<code>com.tibco.trinity.runtime.core.provider.authn.ldap.userSearchExpression</code>	
	<p>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)).</p> <p>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.</p> <p>This property is relevant only when credentialProvider property is set and the binding is done as administrator; otherwise userDNTemplate is used.</p> <p>Required in admin (search) mode.</p>
<code>com.tibco.trinity.runtime.core.provider.authn.ldap.userDNTemplate</code>	

Property	Description
	<p>Specifies a template to be used when formatting user's DN before binding. It is used as an alternative to admin (search) mode.</p> <p>For example, uid={0},ou=employee,ou=tsi,o=tibco</p> <p>Required for bind mode (not in admin search mode).</p>
<code>com.tibco.trinity.runtime.core.provider.authn.ldap.userAttributeGroupsName</code>	
	<p>If you specified "LDAP user indicates groups" (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.</p>
<code>com.tibco.trinity.runtime.core.provider.authn.ldap.userAttributesExtraList</code>	
	<p>Same as userAttributesExtra property but this is specified in list form.</p> <p>Optional.</p>
<code>com.tibco.trinity.runtime.core.provider.authn.ldap.userSearchScopeSubtree</code>	
	<p>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.</p> <p>Optional.</p>
<code>com.tibco.trinity.runtime.core.provider.authn.ldap.groupSearchBaseDN</code>	
	<p>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.</p> <p>For example, ou=groups,ou=na,dc=example,dc=org</p> <p>The default value is empty.</p> <p>Optional.</p>
<code>com.tibco.trinity.runtime.core.provider.authn.ldap.enableNestedGroupSearch</code>	
	<p>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.</p> <p>The default value is false.</p> <p>Optional.</p>
<code>com.tibco.trinity.runtime.core.provider.authn.ldap.groupSearchExpression</code>	

Property	Description
	<p>Specifies the expression to be used for searching against potential groups. For example, search expression is specified as: (&(uid={0})(objectClass=person)).</p> <p>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.</p> <p>The values might be different for different LDAP server.</p> <p>For example, its defined as uniquemember={0} for SunOne, cn={0} for OpenLDAP, member={0} for Active Directory.</p> <p>Required.</p>
<code>com.tibco.trinity.runtime.core.provider.authn.ldap.groupSearchScopeSubtree</code>	
	<p>A Boolean property which determines if the entire sub-tree is searched or not. If the value true 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.</p> <p>Optional.</p>
<code>com.tibco.trinity.runtime.core.provider.authn.ldap.groupIndication</code>	
	<p>Specifies how the group memberships for users are found.</p> <p>The default value is noGroupInfo.</p> <p>Optional.</p> <p>Possible values are as follows:</p> <ul style="list-style-type: none"> <code>userHasGroups</code> <code>userDNHasGroups</code> <code>groupHasUsers</code> <code>noGroupInfo</code> 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 DN's 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 are not handled.
<code>com.tibco.trinity.runtime.core.provider.authn.ldap.groupAttributeGroupsName</code>	

Property	Description
	<p>Depending on groupIndication's value:</p> <p>groupHasUsers: group attribute holding the group's name. Example value for OpenLDAP: cn, for Active Directory: sAMAccountName. Mandatory.</p> <p>userHasGroups: group's name part holding group's name. If not specified the group's whole DN will be used. Example cn</p> <p>otherwise ignored</p>
<code>com.tibco.trinity.runtime.core.provider.authn.ldap.groupAttributeSubgroupsName</code>	
	<p>Specifies the name of the attribute in each group object denoting subgroups.</p> <p>For example, the value is defined as uniqueMember for OpenLDAP server, member for ActiveDirectory LDAP server.</p> <p>Optional</p>
<code>com.tibco.trinity.runtime.core.provider.authn.ldap.groupAttributeUsersName</code>	
	<p>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.</p> <p>For example, the value is uniqueMember for OpenLDAP, member for ActiveDirectory Server.</p> <p>Required if the groupIndication property has groupHasUsers value.</p>
<code>followReferrals</code>	
	<p>Determines if the client follow referrals returned by the LDAP server.</p> <p>The default value is false.</p> <p>Optional.</p>
LDAP SSL	
<code>com.tibco.trinity.runtime.core.provider.identity.trust.trustStoreServiceProvider</code>	
	Specifies the Identity trust provider configuration to provide SSL support for LDAP.
<code>com.tibco.trinity.runtime.core.provider.credential.keystore.truststore.keyStoreLocation</code>	
	Specifies the location of the keystore for the credentials.
<code>com.tibco.trinity.runtime.core.provider.credential.keystore.truststore.keyStorePassword</code>	
	Specifies the location of the keystore for the credentials.
<code>com.tibco.trinity.runtime.core.provider.credential.keystore.truststore.keyStoreRefreshInterval</code>	
	Specifies the refresh interval (milliseconds).

Property	Description
<code>com.tibco.trinity.runtime.core.provider.credential.keystore.truststore.keyStoreType</code>	
	Specifies the keystore type. Supported formats are JKS,PKCS12.

Sample File

The properties and example configuration for LDAP authentication shared resource is provided in the following sample file:

- See `ASG_CONFIG_HOME/default/security/resource/LdapAsp.properties`, as follows:

[LdapAsp.properties](#)

Configuring SiteMinder Service Provider

Description

The Siteminder Service Provider is used to authenticate SiteMinder session cookie or username/password retrieved from the HTTP header.

Use Case

- Authenticate username and password from the incoming request.
- Authenticate SiteMinder session cookie from the incoming request.

Properties

Table [Properties for SiteMinder Service Provider](#) describes the properties for SiteMinder Service Provider.

Properties for SiteMinder Service Provider

Property	Description
<code>com.tibco.trinity.runtime.core.provider.authn.siteminder.agentName</code>	
	Specifies the name of SiteMinder agent. For example, sm-agent
<code>com.tibco.trinity.runtime.core.provider.authn.siteminder.clientIPAddress</code>	
	Specifies the IP address of the machine where agent is installed. For example, 10.97.107.22
<code>com.tibco.trinity.runtime.core.provider.authn.siteminder.resource</code>	
	Specifies the resource to protect.
<code>com.tibco.trinity.runtime.core.provider.authn.siteminder.smHostConfFileLocationOption</code>	

Property	Description
	Specifies how to identify the SmHost.conf file location. For example, this value can be specified as specifyCustomLocation.
<code>com.tibco.trinity.runtime.core.provider.authn.siteminder.smHostConfFileLocation</code>	
	Specifies the path to the agent configuration file. For example, this value can be specified as /security/resource/SmHost.conf.

Sample File

- See `ASG_CONFIG_HOME/default/security/resource/SiteMinderAsp.properties`, as follows:
[SiteMinderAsp.properties](#)

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 Core Engine to verify the signatures when the payload in the incoming request is signed. The 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 Identify Provider \(TIP\)](#) describes the properties for Trust Identify Provider.

Properties for Trust Identify Provider (TIP)

Property	Description
<code>com.tibco.asg.intent.signature</code>	
	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.
<code>com.tibco.trinity.runtime.core.provider.identity.trust.trustStoreServiceProvider</code>	
	Specifies the name of the credential service provider containing the credentials for establishing trust.

Property	Description
<code>com.tibco.trinity.runtime.core.provider.credential.keystore.truststore.keyStoreType</code>	
	Specifies the keystore type. Supported formats are JKS,PKCS12.
<code>com.tibco.trinity.runtime.core.provider.credential.keystore.truststore.keyStoreLocation</code>	
	Specifies the location of the keystore.
<code>com.tibco.trinity.runtime.core.provider.credential.keystore.truststore.keyStorePassword</code>	
	Specifies the password to unlock the keystore.
<code>com.tibco.trinity.runtime.core.provider.credential.keystore.truststore.keyStoreRefreshInterval</code>	
	Specifies the refresh interval (milliseconds).

Sample File

- See `ASG_CONFIG_HOME/default/security/resource/TrustIsp.properties`, as follows:
[TrustIsp.properties](#)

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 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 \(SIP\)](#) describes the properties for Subject Identify Provider.

Properties for Subject Identify Provider (SIP)

Property	Description
<code>com.tibco.asg.intent.decrypt</code>	

Property	Description
	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.
<code>com.tibco.trinity.runtime.core.provider.identity.subject.identityStoreServiceProvider</code>	
	Specifies the name of the credential service provider containing the private credentials for establishing the subject's identity.
<code>com.tibco.trinity.runtime.core.provider.identity.subject.keyAlias</code>	
	Specifies an alias name for the key corresponding to the private credentials in the credential store for establishing the subject's identity.
<code>com.tibco.trinity.runtime.core.provider.identity.subject.keyPassword</code>	
	Specifies the protection parameter of the private credentials in the credential store for establishing the subject's identity.
<code>com.tibco.trinity.runtime.core.provider.credential.keystore.keyStoreType</code>	
	Specifies the keystore type of the private credentials.
<code>com.tibco.trinity.runtime.core.provider.credential.keystore.keyStoreLocation</code>	
	Specifies the location of the keystore of the private credentials.
<code>com.tibco.trinity.runtime.core.provider.credential.keystore.keyStorePassword</code>	
	Specifies the password to unlock the keystore.
<code>com.tibco.trinity.runtime.core.provider.credential.keystore.keyStoreRefreshInterval</code>	
	Specifies the refresh interval in milliseconds.

Sample File

- See `ASG_CONFIG_HOME/default/security/resource/SubjectIsp.properties`, as follows:
[SubjectIsp.properties](#)

Configuring the Kerberos Service Provider

Description

The Kerberos service provider is used to authenticate the SPNEGO token retrieved from the HTTP header.

Use Case

- Authenticate SPNEGO token from the incoming request.

Properties

The following table describes the properties for Kerberos Service Provider:

Properties for Kerberos Service Provider

Property	Description
<code>com.tibco.trinity.runtime.core.provider.lookup</code>	
	The property value must be <code>com.tibco.trinity.runtime.core.provider.authn.kerberos</code> and should not be changed.
<code>com.tibco.trinity.runtime.core.provider.authn.kerberos.enableSecurityTokenAttribute</code>	
	A boolean property which controls the embedding of original security token in the SAML assertion as an attribute.
<code>com.tibco.trinity.runtime.core.provider.authn.kerberos.realm</code>	
	Specifies the Kerberos realm.
<code>com.tibco.trinity.runtime.core.provider.authn.kerberos.kdc</code>	
	Specifies the KDC hostname. For example,
<code>com.tibco.trinity.runtime.core.provider.authn.kerberos.useTicketCache</code>	
	Set this to <code>true</code> to obtain the TGT from the ticket cache.
<code>com.tibco.trinity.runtime.core.provider.authn.kerberos.storeKey</code>	
	A boolean property used to indicate if the key of principal is stored in the private credentials of subject. Set this property value to <code>true</code> to store the principal's key in the private credentials of subject. The default value is <code>true</code> .
<code>com.tibco.trinity.runtime.core.provider.authn.kerberos.useKeyTab</code>	
	Set this to <code>true</code> if you want the module to get the principal's key from the the keytab. (default value is <code>False</code>) If <code>keyatb</code> is not set then the module will locate the keytab from the Kerberos configuration file. Default is <code>TRUE</code>
<code>com.tibco.trinity.runtime.core.provider.authn.kerberos.keyTab</code>	
	Specifies the path to keytab file.

Property	Description
<code>com.tibco.trinity.runtime.core.provider.authn.kerberos.defaultDomain</code>	
	Specifies the Kerberos domain.
<code>com.tibco.trinity.runtime.core.provider.authn.kerberos.autoGeneratedKrb5ConfFileLocation</code>	
	Specifies the relative file name to use for auto generated kerberos configuration file The auto generated file will be saved in the shared area with this name.
<code>com.tibco.trinity.runtime.core.provider.authn.kerberos.krb5ConfFileLocationOption</code>	
	<p>Specifies the option to identify the <code>krb5.conf/krb5.ini</code> file location. The possible values are as follows:</p> <ul style="list-style-type: none"> • <code>useDefault</code>: Use the system specific default <code>krb5.conf/krb5.ini</code> location. This is the default value. • <code>specifyCustomLocation</code>: Specify the custom file location. • <code>autoGenerate</code>: Auto generate the <code>krb5.conf/krb5.ini</code> file dynamically during initialization using configuration options.

Sample File

- See `ASG_CONFIG_HOME/default/security/resource/KerberosAsp.properties`, as follows:
[SPNEGOAsp.properties](#)

Configuring Custom Shared Resource

The custom shared resource is used to define a login module class which implements the authentication mechanism.

The shared resource file consists of the following properties:

- Property to define the custom login module.
- User-defined list of properties to pass and use by the custom login module to initialize or configure the custom login module.

TIBCO API Exchange Gateway provides a `CelmaSp.properties` sample properties file for the shared resource. Use this file as a template and edit the properties, as required.

Use Case

The custom shared resource is useful for the following scenarios:

- The incoming request contains the user credentials in a non-standard format. The username and password may be embedded within the XML of the message body so the user must extract the credentials specific to application before passing it to TIBCO API Exchange Gateway for authentication.
- Verification of credentials or information is done through a custom database query.
- The authentication mechanism requires third-party code.

Properties

The following table describes the properties for the custom shared resource:

Custom Shared Resource Properties

Property	Description	Example
<code>com.tibco.asg.security.provider.celm.loginModule</code>		
	Name of the login module class that implements the custom authentication.	<code>com.example.security.authentication.provider.celm.CelmExampleUsernamePasswordLoginModule</code>
<code>com.tibco.asg.security.provider.celm.authorizedUsername</code>		
	An example property defined by the user to pass as an argument to the <code>initialize()</code> method of custom login module. This property can be used to set the value of the username.	john
<code>com.tibco.asg.security.provider.celm.authorizedPassword</code>		
	An example property defined by the user to pass as an argument to the <code>initialize()</code> method of custom login module. This property can be used to set the value of the password.	password
<code>com.tibco.asg.security.provider.celm.authorizedRole</code>		
	An example property defined by the user to pass as an argument to the <code>initialize()</code> method of custom login module. This property can be used to set the value of the developer.	developer

Sample File

See `ASG_CONFIG_HOME/BookQuerySecurity/security/resource/CelmAsp.properties`, as follows:

[CelmAsp.properties](#).

Shared Resources Properties Sample Files

This section lists the sample files for different types of security shared resources.

LdapAsp.properties

Sample file for LDAP shared resource.

```
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.serverURL=ldap://10.97.107.23:389,ldap://10.97.108.26:389
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=#!fGqMyEST0e58y1QEt7sykDYhfWq9mjKMsJwsSHnAC4=
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=uniquememberof={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,#!fGqMyEST0e58y1QEt7sykDYhfWq9mjKMsJwsSHnAC4=
com.tibco.trinity.runtime.core.provider.credential.password.protectionParameter=password
```

SiteMinderAsp.properties

Sample file for SiteMinder shared resource.

```
com.tibco.trinity.runtime.core.provider.lookup=com.tibco.trinity.runtime.core.provider.authn.siteminder
com.tibco.trinity.runtime.core.provider.authn.siteminder.agentName=sm-agent
com.tibco.trinity.runtime.core.provider.authn.siteminder.resource=/
com.tibco.trinity.runtime.core.provider.authn.siteminder.smHostConfFileLocationOption=specifyCustomLocation
com.tibco.trinity.runtime.core.provider.authn.siteminder.smHostConfFileLocation=/security/resource/SmHost.conf
com.tibco.trinity.runtime.core.provider.authn.siteminder.clientIPAddress=10.97.107.2
```

```

2
com.tibco.trinity.runtime.core.provider.authn.siteminder.enableSAML11Assertion=false
com.tibco.trinity.runtime.core.provider.authn.siteminder.enableSecurityTokenAttribute=true

```

Kerberos SPNEGOAsp.properties

Sample properties file for Kerberos SPNEGO authentication shared resource.

```

com.tibco.trinity.runtime.core.provider.lookup=com.tibco.trinity.runtime.core.provider.authn.kerberos
com.tibco.trinity.runtime.core.provider.authn.kerberos.enableSecurityTokenAttribute=true
com.tibco.trinity.runtime.core.provider.authn.kerberos.enableSAML11Assertion=true
com.tibco.trinity.runtime.core.provider.authn.kerberos.realm=SUPPORT.CH.COM
com.tibco.trinity.runtime.core.provider.authn.kerberos.kdc=10.97.49.107
com.tibco.trinity.runtime.core.provider.authn.kerberos.useTicketCache=false
com.tibco.trinity.runtime.core.provider.authn.kerberos.storeKey=true
com.tibco.trinity.runtime.core.provider.authn.kerberos.useKeyTab=true
com.tibco.trinity.runtime.core.provider.authn.kerberos.keyTab=apixg.keytab

com.tibco.trinity.runtime.core.provider.authn.kerberos.defaultDomain=support.ch.com
com.tibco.trinity.runtime.core.provider.authn.kerberos.autoGeneratedKrb5ConfFileLocation=apixg-krb5.conf
com.tibco.trinity.runtime.core.provider.authn.kerberos.krb5ConfFileLocationOption=autoGenerate
com.tibco.trinity.runtime.core.provider.authn.kerberos.permittedEncryptionTypes=aes128-cts des3-cbc-sha1 rc4-hmac des-cbc-md5 des-cbc-crc
com.tibco.trinity.runtime.core.provider.authn.kerberos.tktEncryptionTypes=aes128-cts des3-cbc-sha1 rc4-hmac des-cbc-md5 des-cbc-crc
com.tibco.trinity.runtime.core.provider.authn.kerberos.tgsEncryptionTypes=aes128-cts des3-cbc-sha1 rc4-hmac des-cbc-md5 des-cbc-crc

```

SubjectIsp.properties

Sample properties for Subject identity provider shared resource.

```

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

Sample file for Trust identity provider shared resource.

```

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.enableTrustStoreAccess=true
#KCP for TIP Namespace. Used for verifying the signature.
com.tibco.trinity.runtime.core.provider.credential.keystore.truststore.keyStoreType=
JKS
com.tibco.trinity.runtime.core.provider.credential.keystore.truststore.keyStoreRefre
shInterval=60000
com.tibco.trinity.runtime.core.provider.credential.keystore.truststore.keyStoreLocat
ion=keystore/john_keystore.jks
com.tibco.trinity.runtime.core.provider.credential.keystore.truststore.keyStorePassw
ord=password

```

IdentityIsp.properties

```

com.tibco.trinity.runtime.core.provider.lookup=com.tibco.trinity.runtime.core.provid
er.identity.subject
com.tibco.governance.sharedresource.name=IdentityIsp
com.tibco.governance.sharedresource.type=SubjectConfiguration
com.tibco.trinity.runtime.core.provider.identity.subject.identityStoreServiceProvide
r=class:com.tibco.trinity.runtime.core.provider.credential.keystore
com.tibco.trinity.runtime.core.provider.identity.subject.trustStoreServiceProvider=c
lass: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=keystor
e/PAJohnIdentity.jcks
com.tibco.trinity.runtime.core.provider.credential.keystore.keyStorePassword=passwor
d
com.tibco.trinity.runtime.core.provider.credential.keystore.keyStoreType=jcks
com.tibco.trinity.runtime.core.provider.credential.keystore.keyStoreRefreshInterval=
60000

```

WssAsp.properties

Sample file for WS security (WSS) authentication provider shared resource.

```

com.tibco.trinity.runtime.core.provider.lookup=com.tibco.trinity.runtime.core.provid
er.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:c
om.tibco.trinity.runtime.core.provider.identity.subject
com.tibco.trinity.runtime.core.provider.authn.wss.usernameTokenValidationService=cla
ss: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.identityStoreServiceProvide
r=class:com.tibco.trinity.runtime.core.provider.credential.keystore
com.tibco.trinity.runtime.core.provider.identity.subject.trustStoreServiceProvider=c
lass: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=keystor
e/default_keystore.jks
com.tibco.trinity.runtime.core.provider.credential.keystore.keyStorePassword=passwor
d
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=cla

```

```

ss=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.truststore.keyStoreType=
JKS
com.tibco.trinity.runtime.core.provider.credential.keystore.truststore.keyStoreRefre
shInterval=60000
com.tibco.trinity.runtime.core.provider.credential.keystore.truststore.keyStoreLocat
ion=keystore/john_keystore.jks
com.tibco.trinity.runtime.core.provider.credential.keystore.truststore.keyStorePassw
ord=password
#com.tibco.trinity.runtime.core.provider.credential.keystore.truststore.keyStoreProv
ider=
#LDAP namespace. Used to verify user name token.
com.tibco.trinity.runtime.core.provider.authn.ldap.serverURL=ldap://
10.97.107.23:389,ldap://10.97.108.26:389
com.tibco.trinity.runtime.core.provider.authn.ldap.securityAuthentication=simple
com.tibco.trinity.runtime.core.provider.authn.ldap.initialCtxFactory=com.sun.jndi.ld
ap.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,givennam
e
com.tibco.trinity.runtime.core.provider.authn.ldap.userSearchScopeSubtree=true
com.tibco.trinity.runtime.core.provider.authn.ldap.groupSearchExpression=uniquemembe
r={0}
com.tibco.trinity.runtime.core.provider.authn.ldap.groupSearchBaseDN=ou=groups,dc=po
lICY,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.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

```

CelmAsp.properties

Sample properties for custom shared resource.

```

# Custom LoginModule provider. DO NOT MODIFY.
com.tibco.trinity.runtime.core.provider.lookup=com.tibco.asg.security.provider.celm

# LoginModule class which implement the custom authentication
com.tibco.asg.security.provider.celm.loginModule=com.example.security.authentication
.provider.celm.CelmExampleUsernamePasswordLoginModule

# Some properties used by the custom LoginModule
com.tibco.asg.security.provider.celm.authorizedUsername=john
com.tibco.asg.security.provider.celm.authorizedPassword=password
com.tibco.asg.security.provider.celm.authorizedRole=developer

```

Authentication using File-Based Identity Store

Overview of file-based authentication.

TIBCO API Exchange Gateway supports user authentication using the file identity store which can be used for both Basic and UsernameToken authentication. 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.

For user authentication using the file resource, the user credentials are stored in an XML file accessible by the Core Engine. The XML file contains one-way hashes of salted passwords. The Core Engine uses the credentials stored in an XML file to authenticate the user. To protect the credentials of the users, the file should be access-protected.

If the roles information are provided using the <group-name> tag in the XML file, the SAML assertion generated from a successful authentication contains the roles information. The roles can be used in the authorization policy.

Configuring User Authentication Policy using File

To configure the user authentication policy using a file resource, follow these steps:

Procedure

1. Create a shared resource See [Create a Shared Resource Properties File](#).
2. Create an XML file for user credentials. See [Create XML File for Credentials](#).
3. Create policy. See [Creating Policy File](#).
4. Register the policies in the system. See [Registering Policy](#).
5. Apply the policies to the target operation. See [Applying Policy](#).

Create a Shared Resource Properties File

TIBCO API Exchange Gateway uses a shared resource properties file to support the user authentication using a file resource. For example, the shared resource properties file defines a property which refers to an XML file containing the user credentials.

TIBCO API Exchange Gateway provides a sample properties file `XmlAsp.properties` for the shared resource. Use this file as a template and edit the properties, as required.

Sample Property File

The location of the shared resource property file for user authentication using the file resource is as follows:

`ASG_CONFIG_HOME/BookQuerySecurity/security/resource/XmlAsp.properties`

XmlAsp.properties

```
com.tibco.trinity.runtime.core.provider.lookup=com.tibco.trinity.runtime.core.provider.authn.xml
# Location of the users, password, and group file.
com.tibco.trinity.runtime.core.provider.authn.xml.fileLocation=XmlAspUsers.xml
```

Properties

The following table describes the properties for the user authentication using file resource.

Shared Resource Property for XML File Based Authentication

Property	Description
<code>com.tibco.trinity.runtime.core.provider.authn.xml.fileLocation</code>	

Property	Description
	<p>Specifies the name of the XML file containing the credentials of the users required for authentication. The location of the XML file can be either a relative path or an absolute file path. The relative file path is relative to the <code>ASG_CONFIG_HOME/ASGProjectConfiguration/security/resource</code> directory.</p> <p>For example, <code>XmlAspUsers.xml</code> is found in the following directory: <code>ASG_CONFIG_HOME/BookQuerySecurity/security/resource</code>. See Create XML File for Credentials.</p>

Create XML File for Credentials

The username and password used during authentication are set in an XML file. Refer to the `ASG_CONFIG_HOME/BookQuerySecurity/security/resource/XmlAspUsers.xml` file as a template.

The XML file contains the users credentials, which are defined as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<realm xmlns="http://xsd.tns.tibco.com/trinity/realm/2013"
hashAlgorithm="PBKDF2WithHmacSHA256" repetitionCount="128">
  <users>
    <user>
      <name>john</name>
      <!-- specify the password as follows using the <plaintext> element.
      The code will replace <plaintext/> with <password salt=...>hash</password>
      in the file on the first authentication attempt.
      Alternatively use the command-line asg-password-hasher utility.
      Make sure that the hashAlgorithm and repetitionCount attributes at
      line 2 match the input to the command-line tool.

      <plaintext>password</plaintext>

      -->
      <password salt="tHpKLGzd92xa2A4Skkdv/oxxeq0=
      ">ES7VlmB26+h4wXaRfhj6PEze8rwYjUiijzj2/5L3Cd2A=</password>
    </user>
    <user>
      <name>alex</name>
      <!-- the password is secret -->
      <password salt="qEd8Hq70biuzvJUrq6lv1eIRUKYxBAEQc+x6byFFOXg=
      ">s340kB7FjnhZcNm0z3XSvdWKQUKqdsckkjPjXj30+cE8=</password>
    </user>
    <user>
      <name>username1</name>
      <password salt="rBPcqwHagVwVSri3ndbjcHCkEk2TV0zmEnHBnkfbf+U=
      ">836tSikrJDFgKfVDQn332khxjQt/xzeLym3i5dAzqFg=</password>
    </user>
    <user>
      <name>username2</name>
      <password salt="k8YuHe/QxhimlmfFLFMMSCLGL6rx9Kuzb94VXDlx3gg=
      ">xgCmSLnmjoylCCnvXK+D+kiuAaGBPcRSyKkk6Qh1H24=</password>
    </user>
    <user>
      <name>username3</name>
      <password salt="PuPZQnsuVnCOQTTgMA1LWvL7T38yVhKslcQviZfoD1U=
      ">lJPyYXbAmQVdzV13Hrk7UbRVl8WD8DSLD4mKxCedpFQ=</password>
    </user>
    <user>
      <name>username4</name>
      <password salt="kBV40kbnFJcD94kyHl1DJ4ATjStQ/Z8rEGxFJo0Hx1s=
      ">RO3VI95GR/VlM+d8pPpSw/sxPwUN4cj49oG9KzUvcly=</password>
    </user>
    <user>
      <name>username5</name>
      <password salt="tta7NUkzBypyqS7EXnl+gR2MSZ/bT6kV6DVoR4pVmJA=
```

```

        ">q4g4rnJkUfIGS0jkuLlgoN5xgfHgLxATuNXp9MhfVhU=</password>
</user>
<user>
  <name>username6</name>
  <password salt="gTYrCR11ZfTDlp4pZ9hlga50UcpseqiasS0cT98KBto=
">Jic75Qs0U7yktbZyLDkvWXHiYKx8aloI1KSQXSwuI30=</password>
</user>
<user>
  <name>username7</name>
  <password salt="TgB9quAYdUY9St4zvMK8Uqq921Hcb7sUb8jMj5+V1Ks=
">Sf4CwL19/ON9Jmypo12yM9PuQpQW3nqYtHzhLCjOB42U=</password>
</user>
</users>
<group-mapping>
  <group-name>Administrator</group-name>
  <user-name>john</user-name>
</group-mapping>
<group-mapping>
  <group-name>child1subgroup</group-name>
  <user-name>username7</user-name>
</group-mapping>
<group-mapping>
  <group-name>childgroup1</group-name>
  <child-group>child1subgroup</child-group>
  <user-name>username4</user-name>
  <user-name>username5</user-name>
</group-mapping>
<group-mapping>
  <group-name>childgroup2</group-name>
  <user-name>username6</user-name>
</group-mapping>
<group-mapping>
  <group-name>parentgroup1</group-name>
  <child-group>childgroup1</child-group>
  <child-group>childgroup2</child-group>
  <user-name>username1</user-name>
  <user-name>username2</user-name>
  <user-name>username3</user-name>
</group-mapping>
</realm>

```

- The password specified in the XML file can be plain text or hashed.
- Plain text passwords can be specified using the <plaintext> element in the XML file. Any plain text passwords inside <plaintext> elements are converted automatically. The file is rewritten and all plain text entries are replaced with hashed passwords according to the hashAlgorithm and repetitionCount attributes defined in the same XML file.
- To generate the password hashes, you can use the `asg-password-hasher.exe` utility provided by TIBCO API Exchange Gateway. See [asg-password-hasher Tool](#).

For the hashed password, the following attributes must be defined in the XML file:

- hashAlgorithm

The hashAlgorithm attribute can have one of the following values:

- SHA1
- SHA-256
- SHA-384
- SHA-512
- PBKDF2WithHmacSHA1
- PBKDF2WithHmacSHA256
- PBKDF2WithHmacSHA384
- PBKDF2WithHmacSHA512

The default value is PBKDF2WithHmacSHA256

- repetitionCount

repetitionCount is the number of iterations used to compute the hash for the password. The higher the repetitionCount, the harder it becomes for an attacker to crack the password. However, using a higher repetition consumes more CPU time during the password verification.

The default value is 1000.



The values of hashAlgorithm and repetitionCount apply to all hashed passwords in the XML file.

Schema for XML File

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!-- (C) Copyright 2009-2014, TIBCO Software Inc. All rights reserved.
-->
<!-- *****
this is schema is for illustration. The real version is shipped as part of the
code.
*****
-->
<schema targetNamespace="http://xsd.tns.tibco.com/trinity/realm/2013"
xmlns:tns="http://xsd.tns.tibco.com/trinity/realm/2013" xmlns="http://www.w3.org/
2001/XMLSchema"
version="2.0" elementFormDefault="qualified">
  <element name="realm">
    <complexType>
      <sequence>
        <element name="users">
          <complexType>
            <sequence>
              <element name="user" minOccurs="0" maxOccurs="unbounded">
                <complexType>
                  <sequence>
                    <element name="name" type="tns:non-empty-string" />
                    <choice>
                      <element name="plaintext" type="string" />
                      <element name="password">
                        <complexType>
                          <simpleContent>
                            <extension base="base64Binary">
                              <attribute name="salt" type="base64Binary" use="required" />
                            </extension>
                          </simpleContent>
                        </complexType>
                      </element>
                    </choice>
                  </sequence>
                </complexType>
              </element>
            </sequence>
          </complexType>
        </element>
      </sequence>
    </complexType>
  </element>
  <element name="group-mapping" minOccurs="0" maxOccurs="unbounded">
    <complexType>
      <sequence>
        <element name="group-name" type="tns:non-empty-string" />
        <element name="child-group" type="tns:non-empty-string"
minOccurs="0" maxOccurs="unbounded" />
        <element name="user-name" type="tns:non-empty-string"
minOccurs="0" maxOccurs="unbounded" />
      </sequence>
    </complexType>
    <unique name="non-redundant-users">
      <selector xpath="tns:user-name" />
      <field xpath="." />
    </unique>
    <unique name="non-redundant-child-groups">
      <selector xpath="tns:child-group" />
      <field xpath="." />
    </unique>
  </element>
</schema>
```

```

</sequence>
<attribute name="hashAlgorithm" type="tns:non-empty-string"
  default="PBKDF2WithHmacSHA256" />
<attribute name="repetitionCount" type="int" default="1000" />
</complexType>
<key name="uniqueUser">
  <selector xpath="./tns:users/tns:user/tns:name" />
  <field xpath="." />
</key>
<keyref name="group-refers-to-existing-user" refer="tns:uniqueUser">
  <selector xpath="tns:group-mapping/tns:user-name" />
  <field xpath="." />
</keyref>
<unique name="unique-top-level-groups">
  <selector xpath="./tns:group-mapping/tns:group-name" />
  <field xpath="." />
</unique>
</element>
<simpleType name="non-empty-string">
  <restriction base="string">
    <minLength value="1" />
    <maxLength value="255" />
  </restriction>
</simpleType>
</schema>

```

asg-password-hasher Tool

Generates hash passwords.

TIBCO API Exchange Gateway provides a command-line `asg-password-hasher.exe` utility to generate hash passwords, which is located in the `ASG_HOME/bin` directory.

Input Parameters

The `asg-password-hasher.exe` utility prompts you to specify the following parameters:

- HashAlgorithm

The `HashAlgorithm` parameter is configured as `hashAlgorithm` attribute in the XML file. The possible values are as follows:

```

SHA1
SHA-256
SHA-384
SHA-512
PBKDF2WithHmacSHA1
PBKDF2WithHmacSHA384
PBKDF2WithHmacSHA256
PBKDF2WithHmacSHA512

```

If you do not specify any value, the default value `PBKDF2WithHmacSHA256` is accepted.

- Iteration count

The `Iteration count` parameter specifies the `repetitionCount` attribute in the XML file. If you do not specify any value, the default value 1000 is accepted.

- password

Specifies the password to hash.

Output

The `asg-password-hasher.exe` utility generates the password with the salt and hash. The generated hashed password is configured in the XML file.

Sample Output

```
C:\tibco\asg\2.3\bin>asg-password-hasher.exe
***** Calculate the hash for a
given password and random salt.
***** HashAlgorithm
(PBKDF2WithHmacSHA256): [hashAlgorithm=PBKDF2WithHmacSHA256] Iteration count (1000):
[repetitionCount=1000] Type in password to hash (<Enter> or Ctrl-C to stop): <Type
your password at the prompt> Type in same password again: <Re-type the same
password> log4j:WARN No appenders could be found for logger
(com.tibco.security.TIBCOSecurity). log4j:WARN Please initialize the log4j system
properly. <password salt="XLowtTUDQj6ocTCwpWlPkMwv2wbh/ZBCzBVKBgUglOs=">zfDDla/
mcaVUaQj3Vq3kkwnWrA47YPG7kNBfz+8u91g=</password>
```



Ensure that the values of HashAlgorithm and Iteration count parameters for the asg-password-hasher utility match the hashAlgorithm and repetitionCount attributes specified in the XML file.

Creating Policy File

Configuration steps to create a policy file.

Before you create a policy, ensure that you have created the shared resource properties file. See [Create a Shared Resource Properties File](#) for details.

Create the policy file as follows:

Procedure

1. Copy the sample template file for your configuration project from the following location:
ASG_CONFIG_HOME/BookQuerySecurity/security/policy/AuthenticationByXml.policy
2. Edit the parameters in the file, as required. For example, change the ResourceInstance parameter to match the shared resource name as follows:

```
ResourceInstance="XmlAsp"
```

The properties for the XmlAsp shared resource are defined in the XmlAsp.properties file which is found in the *ASG_CONFIG_HOME/BookQuerySecurity/security/resource* directory.



- The policy must be a well formed WS policy.
- All the resource instances in the policy must have a shared resource defined.

Sample Policy

The sample file for user authentication using the XML file is located as follows:

ASG_CONFIG_HOME/BookQuerySecurity/security/policy/AuthenticationByXml.policy

AuthenticationByXml.policy

```
<wsp:Policy
xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
  xmlns:wssp="http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702"
  xmlns:tpa="http://xsd.tns.tibco.com/governance/policy/action/2009" >
  <wsp:All>
    <wsp:ExactlyOne>
      <tpa:AuthenticationByJaas>
        <wssp:SupportingTokens>
          <tpa:ExactlyOne>
            <wssp:HttpBasicAuthentication />
          </tpa:ExactlyOne>
        </wssp:SupportingTokens>
      </tpa:AuthenticationByJaas>
    </wsp:ExactlyOne>
  </wsp:All>
</wsp:Policy>
```

```

        </wssp:SupportingTokens>
        <tpa:SharedResourceLoginModule ResourceInstance="XmlAsp" />
    </tpa:AuthenticationByJaas>
</wsp:ExactlyOne>
</wsp:All>
</wsp:Policy>

```

Registering Policy

Register the authentication policy on the Config UI as follows:

- Upload the policy file.
- Set the name for a policy.

To register a policy, perform the following steps:

Procedure

1. Start the Config UI, if it is not running.
2. Log in to the Config UI using your credentials.
3. Create a new project or select an existing project under **Projects**.
4. Click the **SECURITY** tab.
5. Click the **Policy Mapping** tab on the top menu.
6. Click the **Add Property** icon to add a new policy mapping.
7. Enter the following parameters for the policy:

Policy Mapping Parameters

Type	Description
Policy Name	<ul style="list-style-type: none"> • Specifies the name for the policy. • Required.
Intent	Set the intent for the policy. Select Authentication from the drop-down list.
Qualifier	Set the qualifier for policy intent. Select UsernameToken or Basic from the drop-down list, as per your requirement.
New Policy File	Specifies the policy definition file. Browse to choose a new policy file. See Creating Policy File to create the policy definition file.
Existing Policy File	Specifies an existing policy definition file. The policy file must exist in the gateway <code>ASG_CONFIG_HOME/ASGProjectConfiguration/policy</code> folder. For example, for the <code>BookQuerySecurity</code> project configuration, the policy file must exist in the <code>ASG_CONFIG_HOME/BookQuerySecurity/policy</code> folder. Select AuthenticationByXml.policy from the drop-down list to use the example policy file.

8. Save the changes to the project or configuration.



Applying Policy

To apply the registered policy to a target operation, perform the following steps:

Procedure

1. Start the Config UI, if not running.
2. Log in to the Config UI using your credentials.
3. Create a new configuration or select an existing configuration.
4. Click the **SECURITY** tab.
5. Click the **Policy Binding** tab on the top menu.
6. Click the **Add Property** icon to add a new policy binding.
7. Specify the following parameters for the policy:

Policy Binding Parameters

Parameter	Description
Policy	Specifies a name for the policy. The policy name must be configured under the Policy Mapping tab.
URI	Specifies the URI of the operation to which the policy is applied.  If the URI in facade operation is left blank, then URI field for security binding should use SOAP action instead of URI.
Facade Operation	Specifies the operation to which the policy is applied. The facade operation must be configured in the Facade Operation tab.
Target Operation	Specifies the target operation. The target operation must be configured in the Target Operation tab.
Binding	Specifies the binding component that the policy is applied to. This could be either a facade operation (service) or a target operation (reference).
Flow	Specifies the flow of the request or response. The possible values are: <ul style="list-style-type: none"> • in • out
Partner	Specifies the partner to which the policy is applied. This field can be left blank in case no partner needs to be applied.
Type	Specifies the type of the request.  Set this to SOAP for any SOAP request, or to http for any non-SOAP http request.

8. Save the changes to the project or configuration.

Data Masking and Selective Log Policy

TIBCO API Exchange Gateway masks the sensitive data in a payload and selectively logs only the relevant data.

Configuration Setup For Log Policy

This sections explains how to configure a policy for data masking and selective logging of payload data.

Setting up Properties

Prerequisites

To enable the data masking and selective logging, make sure that you set the log level to 0 (DEBUG) using the `tibco.clientVar.ASG/Logging/MinLogLevel` property in the `ASG_CONFIG_HOME/asg.properties` file.

To configure the payload data masking and selective log policy, you must set the following properties in the `ASG_CONFIG_HOME/asg.properties` file:

Properties for Data Masking

Property	Description
<code>tibco.clientVar.ASG/Logging/enableLogMasking</code>	Enables or disables the log policy. The possible values are <code>true</code> and <code>false</code> . The default value is <code>false</code> .
<code>tibco.clientVar.ASG/Files/PayloadMaskConfigFile</code>	Specifies the path of log policy XML file. The log policy file must be copied to the <code>ASG_CONFIG_HOME</code> directory.

To enable the log policy for a payload, follow these steps:

Procedure

1. Navigate to the `ASG_CONFIG_HOME` directory.
2. Set the following property to `true`, as follows:

```
tibco.clientVar.ASG/Logging/enableLogMasking=true
```

3. Set the following property to set the path to the log policy XML file:

```
tibco.clientVar.ASG/Files/PayloadMaskConfigFile
```

For example, the path can be set as follows:

```
tibco.clientVar.ASG/Files/PayloadMaskConfigFile=ASG_CONFIG_HOME/APIExchange/
security/policy/LogPolicy.xml
```



See [Log Policy Configuration File](#) to create a log policy XML file. The log policy XML filename cannot be changed. Otherwise, the policy will not appear in the drop-down list on the **Log Policy** tab of the Config UI.

4. Save changes to the `asg.properties` file.

Create Log Policy Configuration File

Log policy file is created as an XML file. You can use any XML editor to create the log policy file.

The log policy XML file contains the following type of configurations:

- Data Masking Configuration Parameters. See the following data masking configurations:
 - [Masking the Headers Data](#)
 - [Masking Query Parameters](#)
 - [Masking Payloads](#)
- Selective Logging Configuration Parameters. See [Create Selective Logging Configuration](#)

Creating Data Masking Configuration

The data masking configuration parameters are defined in a XML file. For any request, you can mask the following data:

- Headers Data
- Query String Data
- Payloads Data

To create the data masking configuration in an XML file, refer to the [Sample Log Policy XML File](#).

Masking Headers Data

You can define the headers for masking using the `<MaskHeader>` tag in the log policy XML file. This table explains the field names for masking the headers data in a request.

Parameters For Masking Headers

Parameter	Description
HeaderName	Specifies the name of the header to be masked.

Example of Headers Data for Masking

```
<MaskHeader>
  <HeaderName>client-ip</HeaderName>
  <HeaderName>content-type</HeaderName>
</MaskHeader>
```

Example Header Policy

You can define a **P1** policy with **x** as masking character. When this policy is applied for a facade operation, TIBCO API Exchange Gateway masks the value of `content-type` header in the request depending on the stage of the transaction pipeline processing.

```
<Name>P1</Name>
  <MaskPolicy>
    <MaskChar>x</MaskChar>
    <MaskHeader>
      <HeaderName>content-type</HeaderName>
    </MaskHeader>
```

Masking Query String Parameters Data

You can define the parameters in a query string for masking using the `<MaskQueryString>` tag. This table explains the field names for masking the query string parameters in a request.

Parameters For Masking Headers

Parameter	Description
ParamName	Specifies the parameter of query string to be masked.

Example of Query String for Masking

```
<MaskQueryString>
  <Parameters>
    <ParamName>storenumber1</ParamName>
    <ParamName>rmsskuid</ParamName>
  </Parameters>
</MaskQueryString>
```

Example of Query String Policy

You can define the following policy to mask the value of a author field in the query string of the request. When this policy is applied for a facade operation, TIBCO API Exchange Gateway masks the value of author parameter specified in the query string in the request.

```
<MaskQueryString>
  <Parameters>
    <ParamName>author</ParamName>
  </Parameters>
</MaskQueryString>
```

Masking Payload Data

Using data masking policy, you can mask the data fields of a payload.

TIBCO API Exchange Gateway provides the masking of data for the following formats of payload:

- Text payloads. See [Masking Data in Text Payloads](#).
- XML payloads. See [Masking Data in XML Payloads](#).
- JSON payloads. See [Masking Data in JSON Payloads](#).
- Property based payloads. See [Masking Data in Property Format Payloads](#).

Masking Data in Text Payloads

You can mask the data in the text format payloads using the <TextPayloadMask> tag in the log policy XML file. This table explains the field names for masking the data in a text payload.

Parameters For Masking Data in Text Payload

Parameter	Description
LineNumber	<p>Specifies the line number which contains the string to be masked.</p> <ul style="list-style-type: none"> • If a line number is specified using <LineNumber></LineNumber> tag in the policy file, the Core Engine searches the data string to be masked at that line number only. • If no line number is specified, the Core Engine searches the entire text payload for the data string to be masked.
Starts with	Specifies a prefix for the data string to be masked.
Regex	Specifies a regular expression pattern which matches the data string to be masked.
StartIndex	Specifies an Index of the data string at which masking starts.
EndIndex	Specifies an Index of the data string at which masking ends.
LengthOfClearAtStart	Specifies the number of characters to be left clear at the beginning.
LengthOfClearAtEnd	Specifies the number of characters to be left clear at the end.

Example of Data Masking in Text Payload

```
<TextPayloadMask>
  <MaskText>
    <LineNumber></LineNumber>
    <StartsWith>enerat</StartsWith>
    <Regex></Regex>
    <StartIndex>1</StartIndex>
    <EndIndex>11</EndIndex>
    <LengthOfClearAtStart>1</LengthOfClearAtStart>
    <LengthOfClearAtEnd>1</LengthOfClearAtEnd>
  </MaskText>
  <MaskText>
    <LineNumber>4</LineNumber>
    <StartsWith></StartsWith>
    <Regex>SSN</Regex>
    <StartIndex>0</StartIndex>
    <EndIndex>1</EndIndex>
    <LengthOfClearAtStart>2</LengthOfClearAtStart>
    <LengthOfClearAtEnd>2</LengthOfClearAtEnd>
  </MaskText>
  <MaskText>
    <LineNumber>1</LineNumber>
    <StartsWith></StartsWith>
    <Regex></Regex>
    <StartIndex>4</StartIndex>
    <EndIndex>12</EndIndex>
    <LengthOfClearAtStart>2</LengthOfClearAtStart>
    <LengthOfClearAtEnd>2</LengthOfClearAtEnd>
  </MaskText>
</TextPayloadMask>
```

Masking Data in XML Payloads

You can mask the data in the XML format payloads using the <XMLMaskFields> tag in the log policy XML file. This table explains the field names for masking the data in an XML payload.

Parameters For Masking Data in XML Payload

Parameter	Description
MaskFieldPath	Specifies the XPath for the field. The value of this field is masked.
RemoveField	<p>This is a Boolean field that indicates if the field specified by MaskFieldPath tag is logged or not. The possible values are Y and N.</p> <ul style="list-style-type: none"> If the value of this field is Y, the field specified by MaskFieldPath tag is not logged. If the value of this field is N, the field specified by MaskFieldPath tag is masked and logged.
StartIndex	Specifies an Index of the field value at which masking starts. If the RemoveField field is set to Y, this is ignored.
EndIndex	Specifies an index of the field value at which masking ends. Specifies an Index of the field value at which masking starts. If the RemoveField field is set to Y, this is ignored.

Parameter	Description
LengthOfClearAtStart	Specifies the number of characters to be left clear at the beginning. Specifies an index of the field value at which masking starts. If the RemoveField field is set to Y, this is ignored.
LengthOfClearAtEnd	Specifies the number of characters to be left clear at the end. Specifies an index of the field value at which masking starts. If the RemoveField field is set to Y, this is ignored.

Example of Data Masking in XML Payload

```


<XMLMaskFields>
  <XMLMaskField>
    <MaskFieldPath>/soapenv:Envelope/soapenv:Body/
book:Author</MaskFieldPath>
    <RemoveField>Y</RemoveField>
    <StartIndex>6</StartIndex>
    <EndIndex>12</EndIndex>
    <LengthOfClearAtStart>4</LengthOfClearAtStart>
    <LengthOfClearAtEnd>6</LengthOfClearAtEnd>
  </XMLMaskField>
  <XMLMaskField>
    <MaskFieldPath>BookStore/Book/ISBN</
MaskFieldPath>
    <RemoveField>Y</RemoveField>
    <StartIndex>3</StartIndex>
    <EndIndex>9</EndIndex>
    <LengthOfClearAtStart>4</LengthOfClearAtStart>
    <LengthOfClearAtEnd>6</LengthOfClearAtEnd>
  </XMLMaskField>
</XMLMaskFields>

```

Masking Data in JSON Payloads

You can mask the data in the JSON format payloads using the <JSONMaskFields> tag in the log policy XML file. This table explains the field names for masking the data in an JSON payload.

Parameters For Masking Data in JSON Payload

Parameter	Description
MaskFieldPath	Specifies the JSON path for the field. The value of this field is masked. For example, BookStore.Book.[0].Author
RemoveField	<p>This is a Boolean field that indicates if the field specified by MaskFieldPath tag is logged or not. The possible values are Y and N.</p> <ul style="list-style-type: none"> If the value of this field is Y, the field specified by MaskFieldPath tag is not logged. If the value of this field is N, the field specified by MaskFieldPath tag is masked and logged. <p> The default value is N.</p>

Parameter	Description
StartIndex	Specifies an index of the field value at which masking starts. If the RemoveField field is set to Y, this is ignored.
EndIndex	Specifies an Index of the field value at which masking ends. Specifies an index of the field value at which masking starts. If the RemoveField field is set to Y, this is ignored.
LengthOfClearAtStart	Specifies the number of characters to be left clear at the beginning. Specifies an index of the field value at which masking starts. If the RemoveField field is set to Y, this is ignored.
LengthOfClearAtEnd	Specifies the number of characters to be left clear at the end. Specifies an index of the field value at which masking starts. If the RemoveField field is set to Y, this is ignored.

Example of Data Masking in JSON Payload

```

<JSONMaskFields>
  <JSONMaskField>
    <MaskFieldPath>BookStore.Book.[0].Author</MaskFieldPath>
    <RemoveField>Y</RemoveField>
    <StartIndex>0</StartIndex>
    <EndIndex>10</EndIndex>
    <LengthOfClearAtStart>4</LengthOfClearAtStart>
    <LengthOfClearAtEnd>6</LengthOfClearAtEnd>
  </JSONMaskField>
  <JSONMaskField>
    <MaskFieldPath>BookStore.Book.[0].Title</MaskFieldPath>
    <RemoveField>N</RemoveField>
    <StartIndex>6</StartIndex>
    <EndIndex>12</EndIndex>
    <LengthOfClearAtStart>4</LengthOfClearAtStart>
    <LengthOfClearAtEnd>6</LengthOfClearAtEnd>
  </JSONMaskField>
  <JSONMaskField>
    <MaskFieldPath>BookStore.Book.[0].ISBN</MaskFieldPath>
    <RemoveField>N</RemoveField>
    <StartIndex>3</StartIndex>
    <EndIndex>9</EndIndex>
    <LengthOfClearAtStart>4</LengthOfClearAtStart>
    <LengthOfClearAtEnd>6</LengthOfClearAtEnd>
  </JSONMaskField>
</JSONMaskFields>

```

Masking Data in Property Format Payloads

You can mask data in the property format payloads using the `<PropertiesMaskFields>` tag in the log policy XML file. Using the `<PropertyToMask>`, define the fields for the data masking. This table explains the field names for masking the data in a property format payload.

Parameters For Masking Data in Property Format Payload

Parameter	Description
PropertyKey	Defines the key of the property to be masked. The Core Engine masks the value of key after the key is found.
RemoveField	<p>This is a Boolean field that indicates if the field specified by MaskFieldPath tag is logged or not. The possible values are Y and N.</p> <ul style="list-style-type: none"> If the value of this field is Y, the field specified by MaskFieldPath tag is not logged. If the value of this field is N, the field specified by MaskFieldPath tag is masked and logged.
StartIndex	Specifies an index of the field value at which masking starts. If the RemoveField field is set to Y, this is ignored.
EndIndex	Specifies an index of the field value at which masking ends. Specifies an Index of the field value at which masking starts. If the RemoveField field is set to Y, this is ignored.
LengthOfClearAtStart	Specifies the number of characters to be left clear at the beginning. Specifies an index of the field value at which masking starts. If the RemoveField field is set to Y, this is ignored.
LengthOfClearAtEnd	Specifies the number of characters to be left clear at the end. Specifies an index of the field value at which masking starts. If the RemoveField field is set to Y, this is ignored.

Example of Data Masking in Property Format Payload

```
<PropertiesMaskFields>
  <PropertyToMask>
    <PropertyKey>Tibco</PropertyKey>
    <RemoveField>N</RemoveField>
    <StartIndex>3</StartIndex>
    <EndIndex>9</EndIndex>
    <LengthOfClearAtStart>2</LengthOfClearAtStart>
    <LengthOfClearAtEnd>2</LengthOfClearAtEnd>
  </PropertyToMask>
  <PropertyToMask>
    <PropertyKey>CC</PropertyKey>
    <RemoveField>N</RemoveField>
    <StartIndex>3</StartIndex>
    <EndIndex>9</EndIndex>
    <LengthOfClearAtStart>4</LengthOfClearAtStart>
    <LengthOfClearAtEnd>6</LengthOfClearAtEnd>
  </PropertyToMask>
</PropertiesMaskFields>
```


Sample Log Policy XML File

See the following example log policy file:

```
<?xml version="1.0" encoding="UTF-8"?>
<LogPolicies>
  <LogPolicy>
    <Name>P1</Name>
    <MaskPolicy>
      <MaskChar>X</MaskChar>
      <MaskHeader>
        <HeaderName>client-ip</HeaderName>
        <HeaderName>content-type</HeaderName>
      </MaskHeader>
      <MaskQueryString>
        <Parameters>
          <ParamName>storenumber1</ParamName>
          <ParamName>rmsskuid</ParamName>
        </Parameters>
      </MaskQueryString>
      <PropertiesMaskFields>
        <PropertyToMask>
          <PropertyKey>Tibco</PropertyKey>
          <RemoveField>N</RemoveField>
          <StartIndex>3</StartIndex>
          <EndIndex>9</EndIndex>
          <LengthOfClearAtStart>2</LengthOfClearAtStart>
          <LengthOfClearAtEnd>2</LengthOfClearAtEnd>
        </PropertyToMask>
        <PropertyToMask>
          <PropertyKey>CC</PropertyKey>
          <RemoveField>N</RemoveField>
          <StartIndex>3</StartIndex>
          <EndIndex>9</EndIndex>
          <LengthOfClearAtStart>4</LengthOfClearAtStart>
          <LengthOfClearAtEnd>6</LengthOfClearAtEnd>
        </PropertyToMask>
      </PropertiesMaskFields>
      <XMLMaskFields>
        <XMLMaskField>
          <MaskFieldPath>/Envelope/Body/Title</
MaskFieldPath>
          <RemoveField>Y</RemoveField>
          <StartIndex>0</StartIndex>
          <EndIndex>10</EndIndex>
          <LengthOfClearAtStart>6</LengthOfClearAtStart>
          <LengthOfClearAtEnd>2</LengthOfClearAtEnd>
        </XMLMaskField>
        <XMLMaskField>
          <MaskFieldPath>BookStore/Book/ISBN</
MaskFieldPath>
          <RemoveField>Y</RemoveField>
          <StartIndex>3</StartIndex>
          <EndIndex>9</EndIndex>
          <LengthOfClearAtStart>4</LengthOfClearAtStart>
          <LengthOfClearAtEnd>6</LengthOfClearAtEnd>
        </XMLMaskField>
      </XMLMaskFields>
      <JSONMaskFields>
        <JSONMaskField>
          <MaskFieldPath>BookStore.Book.[0].Author</
MaskFieldPath>
          <RemoveField>Y</RemoveField>
          <StartIndex>0</StartIndex>
          <EndIndex>10</EndIndex>
          <LengthOfClearAtStart>4</LengthOfClearAtStart>
          <LengthOfClearAtEnd>6</LengthOfClearAtEnd>
        </JSONMaskField>
        <JSONMaskField>
          <MaskFieldPath>BookStore.Book.[0].Title</
```

```

MaskFieldPath>
    <RemoveField>N</RemoveField>
    <StartIndex>6</StartIndex>
    <EndIndex>12</EndIndex>
    <LengthOfClearAtStart>4</LengthOfClearAtStart>
    <LengthOfClearAtEnd>6</LengthOfClearAtEnd>
</JSONMaskField>
<JSONMaskField>
    <MaskFieldPath>BookStore.Book.[0].ISBN</
MaskFieldPath>
    <RemoveField>N</RemoveField>
    <StartIndex>3</StartIndex>
    <EndIndex>9</EndIndex>
    <LengthOfClearAtStart>4</LengthOfClearAtStart>
    <LengthOfClearAtEnd>6</LengthOfClearAtEnd>
</JSONMaskField>
</JSONMaskFields>
<TextPayloadMask>
    <MaskText>
        <LineNumber></LineNumber>
        <StartsWith>enerat</StartsWith>
        <Regex></Regex>
        <StartIndex>1</StartIndex>
        <EndIndex>11</EndIndex>
        <LengthOfClearAtStart>1</LengthOfClearAtStart>
        <LengthOfClearAtEnd>1</LengthOfClearAtEnd>
    </MaskText>
    <MaskText>
        <LineNumber>4</LineNumber>
        <StartsWith></StartsWith>
        <Regex>SSN</Regex>
        <StartIndex>0</StartIndex>
        <EndIndex>1</EndIndex>
        <LengthOfClearAtStart>2</LengthOfClearAtStart>
        <LengthOfClearAtEnd>2</LengthOfClearAtEnd>
    </MaskText>
    <MaskText>
        <LineNumber>1</LineNumber>
        <StartsWith></StartsWith>
        <Regex></Regex>
        <StartIndex>4</StartIndex>
        <EndIndex>12</EndIndex>
        <LengthOfClearAtStart>2</LengthOfClearAtStart>
        <LengthOfClearAtEnd>2</LengthOfClearAtEnd>
    </MaskText>
</TextPayloadMask>
</MaskPolicy>
<SelectiveLogPolicy>
    <PropertySelectiveLogFields>
        <PropertyKey>CreditCard</PropertyKey>
        <PropertyKey>SSN</PropertyKey>
    </PropertySelectiveLogFields>
    <XMLSelectiveLogFields>
        <FieldPath>/Envelope/Body/Title</FieldPath>
        <FieldPath>/Envelope/Body/Author</FieldPath>
    </XMLSelectiveLogFields>
    <JSONSelectiveLogFields>
        <FieldPath>BookStore.Book.[0].Author</FieldPath>
        <FieldPath>BookStore.Book.[0].ISBN</FieldPath>
    </JSONSelectiveLogFields>
    <TextSelectiveLog>
        <TextToLog>
            <Prefix>SSN</Prefix>
            <LineNumber>1</LineNumber>
            <StartsWith>XSD </StartsWith>
            <Regex></Regex>
            <StartIndex>1</StartIndex>
            <EndIndex>11</EndIndex>
        </TextToLog>
        <TextToLog>
            <Prefix></Prefix>
            <LineNumber>2</LineNumber>

```

```

        <StartsWith></StartsWith>
        <Regex>XSD</Regex>
        <StartIndex>1</StartIndex>
        <EndIndex>11</EndIndex>
    </TextToLog>
</TextSelectiveLog>
</SelectiveLogPolicy>
</LogPolicy>
<LogPolicy>
    <Name>P2</Name>
    <MaskPolicy>
        <MaskChar>X</MaskChar>
        <MaskHeader>
            <HeaderName>client-ip</HeaderName>
            <HeaderName>content-type</HeaderName>
        </MaskHeader>
        <MaskQueryString>
            <Parameters>
                <ParamName>storenumber1</ParamName>
                <ParamName>rmsskuid</ParamName>
            </Parameters>
        </MaskQueryString>
        <PropertiesMaskFields>
            <PropertyToMask>
                <PropertyKey>SSN</PropertyKey>
                <RemoveField>N</RemoveField>
                <StartIndex>3</StartIndex>
                <EndIndex>9</EndIndex>
                <LengthOfClearAtStart>4</LengthOfClearAtStart>
                <LengthOfClearAtEnd>6</LengthOfClearAtEnd>
            </PropertyToMask>
            <PropertyToMask>
                <PropertyKey>CreditCard</PropertyKey>
                <RemoveField>Y</RemoveField>
                <StartIndex>3</StartIndex>
                <EndIndex>9</EndIndex>
                <LengthOfClearAtStart>1</LengthOfClearAtStart>
                <LengthOfClearAtEnd>1</LengthOfClearAtEnd>
            </PropertyToMask>
        </PropertiesMaskFields>
        <XMLMaskFields>
            <XMLMaskField>
                <MaskFieldPath>/Envelope/Body/Author</
MaskFieldPath>
                <RemoveField>Y</RemoveField>
                <StartIndex>6</StartIndex>
                <EndIndex>12</EndIndex>
                <LengthOfClearAtStart>4</LengthOfClearAtStart>
                <LengthOfClearAtEnd>6</LengthOfClearAtEnd>
            </XMLMaskField>
            <XMLMaskField>
                <MaskFieldPath>BookStore/Book/ISBN</
MaskFieldPath>
                <RemoveField>Y</RemoveField>
                <StartIndex>3</StartIndex>
                <EndIndex>9</EndIndex>
                <LengthOfClearAtStart>4</LengthOfClearAtStart>
                <LengthOfClearAtEnd>6</LengthOfClearAtEnd>
            </XMLMaskField>
        </XMLMaskFields>
        <JSONMaskFields>
            <JSONMaskField>
                <MaskFieldPath>BookStore.Book.[0].Author</
MaskFieldPath>
                <RemoveField>Y</RemoveField>
                <StartIndex>0</StartIndex>
                <EndIndex>10</EndIndex>
                <LengthOfClearAtStart>4</LengthOfClearAtStart>
                <LengthOfClearAtEnd>6</LengthOfClearAtEnd>
            </JSONMaskField>
            <JSONMaskField>
                <MaskFieldPath>BookStore.Book.[0].Title</

```

```

MaskFieldPath>
    <RemoveField>N</RemoveField>
    <StartIndex>6</StartIndex>
    <EndIndex>12</EndIndex>
    <LengthOfClearAtStart>4</LengthOfClearAtStart>
    <LengthOfClearAtEnd>6</LengthOfClearAtEnd>
</JSONMaskField>
<JSONMaskField>
    <MaskFieldPath>BookStore.Book.[0].ISBN</
MaskFieldPath>
    <RemoveField>N</RemoveField>
    <StartIndex>3</StartIndex>
    <EndIndex>9</EndIndex>
    <LengthOfClearAtStart>4</LengthOfClearAtStart>
    <LengthOfClearAtEnd>6</LengthOfClearAtEnd>
</JSONMaskField>
</JSONMaskFields>
<TextPayloadMask>
    <MaskText>
        <LineNumber></LineNumber>
        <StartsWith>enerat</StartsWith>
        <Regex></Regex>
        <StartIndex>1</StartIndex>
        <EndIndex>11</EndIndex>
        <LengthOfClearAtStart>1</LengthOfClearAtStart>
        <LengthOfClearAtEnd>1</LengthOfClearAtEnd>
    </MaskText>
    <MaskText>
        <LineNumber>4</LineNumber>
        <StartsWith></StartsWith>
        <Regex>SSN</Regex>
        <StartIndex>0</StartIndex>
        <EndIndex>1</EndIndex>
        <LengthOfClearAtStart>2</LengthOfClearAtStart>
        <LengthOfClearAtEnd>2</LengthOfClearAtEnd>
    </MaskText>
    <MaskText>
        <LineNumber>1</LineNumber>
        <StartsWith></StartsWith>
        <Regex></Regex>
        <StartIndex>4</StartIndex>
        <EndIndex>12</EndIndex>
        <LengthOfClearAtStart>2</LengthOfClearAtStart>
        <LengthOfClearAtEnd>2</LengthOfClearAtEnd>
    </MaskText>
</TextPayloadMask>
</MaskPolicy>
<SelectiveLogPolicy>
    <PropertySelectiveLogFields>
        <PropertyKey>SSN</PropertyKey>
        <PropertyKey>CreditCard</PropertyKey>
    </PropertySelectiveLogFields>
    <XMLSelectiveLogFields>
        <FieldPath>/SOAP-ENV:Envelope/SOAP-ENV:Body/
ns0:BookStore/ns0:Book/ns0:Author</FieldPath>
        <FieldPath>/SOAP-ENV:Envelope/SOAP-ENV:Body/
ns0:BookStore/ns0:Book/ns0:Publisher</FieldPath>
    </XMLSelectiveLogFields>
    <JSONSelectiveLogFields>
        <FieldPath>BookStore.Book.[0].Author</FieldPath>
        <FieldPath>BookStore.Book.[0].ISBN</FieldPath>
    </JSONSelectiveLogFields>
    <TextSelectiveLog>
        <TextToLog>
            <Prefix>SSN</Prefix>
            <LineNumber>1</LineNumber>
            <StartsWith>XSD </StartsWith>
            <Regex></Regex>
            <StartIndex>1</StartIndex>
            <EndIndex>11</EndIndex>
        </TextToLog>
    </TextSelectiveLog>

```

```
        <Prefix></Prefix>
        <LineNumber>2</LineNumber>
        <StartsWith></StartsWith>
        <Regex>XSD</Regex>
        <StartIndex>1</StartIndex>
        <EndIndex>11</EndIndex>
    </TextToLog>
</TextSelectiveLog>
</SelectiveLogPolicy>
</LogPolicy>
</LogPolicies>
```

Log Policy Schema File

See the following log policy schema (XSD):

```
<xs:schema attributeFormDefault="unqualified"
elementFormDefault="qualified" xmlns:xs="http://www.w3.org/2001/
XMLSchema"> <xs:element name="LogPolicies">
<xs:complexType> <xs:sequence> <xs:element
name="LogPolicy" maxOccurs="unbounded" minOccurs="0">
<xs:complexType> <xs:sequence> <xs:element
type="xs:string" name="Name"/> <xs:element
type="xs:string" name="Format"/> <xs:element
name="MaskPolicy"> <xs:complexType>
<xs:sequence> <xs:element type="xs:string"
name="MaskChar"/> <xs:element
name="MaskHeader">
<xs:complexType>
<xs:sequence> <xs:element type="xs:string"
name="HeaderName" maxOccurs="unbounded" minOccurs="0"/>
> </xs:sequence> </>
xs:complexType> </xs:element>
<xs:element name="MaskQueryString">
<xs:complexType>
<xs:sequence> <xs:element
name="Parameters">
<xs:complexType>
<xs:sequence> <xs:element
type="xs:string" name="ParamName" maxOccurs="unbounded"
minOccurs="0"/> </>
xs:sequence> </>
xs:complexType> </>
xs:element> </>
xs:sequence> </>
xs:complexType> </xs:element>
<xs:element name="XMLMaskFields">
<xs:complexType>
<xs:sequence> <xs:element
name="XMLMaskField" maxOccurs="unbounded"
minOccurs="0">
<xs:complexType>
<xs:sequence> <xs:element
type="xs:string" name="MaskFieldPath"/>
> <xs:element type="xs:string"
name="RemoveField"/> <xs:element
type="xs:string" name="StartIndex"/>
<xs:element type="xs:string" name="EndIndex"/>
> <xs:element type="xs:string"
name="LengthOfClearAtStart"/>
<xs:element type="xs:string" name="LengthOfClearAtEnd"/>
> </>
xs:sequence> </>
xs:complexType> </>
xs:element> </>
xs:sequence> </>
xs:complexType> </xs:element>
<xs:element name="JSONMaskFields">
<xs:complexType>
<xs:sequence> <xs:element
name="JSONMaskField" maxOccurs="unbounded"
minOccurs="0">
<xs:complexType>
<xs:sequence> <xs:element
type="xs:string" name="MaskFieldPath"/>
> <xs:element type="xs:string"
name="RemoveField"/> <xs:element
type="xs:string" name="StartIndex"/>
<xs:element type="xs:string" name="EndIndex"/>
> <xs:element type="xs:string"
name="LengthOfClearAtStart"/>
```

[illegible]


Upload Log Policy XML File

You must use the Config UI to upload the log policy XML file.

Prerequisites

- The log policy XML file is created.
- The log policy XML file exists in the location specified by `tibco.clientVar.ASG/Files/PayloadMaskConfigFile` property in the `ASG_CONFIG_HOME/asn.properties` file.

Procedure

1. Start the Config UI. See [Starting GUI](#).
2. Log in to the Config UI using your credentials.
3. Click your gateway project under **Projects**.
4. Click the **SECURITY** tab.
5. Click **Log Policy**.
6. Click the **Add property**  icon to create a new policy.
7. Enter the following log policy configuration parameters:

New Log Policy Configuration Parameters

Parameter	Description
Operation Name *	<ul style="list-style-type: none"> • Specifies the name of the facade operation. The operation must be configured in the Facade Operations tab of the Config UI. • Select a predefined facade operation from this drop-down list. • This is a required field.
Stage Name *	<ul style="list-style-type: none"> • Specifies the name of the stage in the transaction pipeline processing where the policy has to be applied. Refer to Supported Stages for Log Policy for the stage names where the policy can be applied. • Select a stage name from this drop-down list. • This is a required field.
Policy Type	Select Log from the drop-down list.

Parameter	Description
Policy Name(s)	<ul style="list-style-type: none"> Select the name of policy which has to be applied on the facade operation from this drop-down list. You can configure multiple policies for a facade operation. <p>This policy must be defined using the <Name> tag in the log policy XML file to appear in this drop-down list. See Sample Log Policy XML File.</p>

8. Save changes to the configuration.

Supported Stages for Log Policy

The following table lists the stage names in the transaction pipeline processing where a log policy can be applied.

Stage Names for Log Policy

Stage Names	Description
RequestEntry	Stage when the request from the client enters in the gateway for processing.
ResponseEntry	Stage when the response from target service enters in the gateway.
Service	Specifies the last stage before the request is made and first stage when the response comes in.
ForwardSouthboundMapping	Specifies the stage when the forward southbound mapper is called.
ReverseNorthboundMapping	Specifies the stage when the northbound reverse mapper is called.
Response	Specifies the stage when the final response exits from the gateway.

Create Selective Logging Configuration

You can define the selective logging configuration parameters in a XML file.

For any request payload, you can define the fields in the policy which are logged. The following formats of payloads are supported:

- Text payloads. See [Selective Logging for Text Payloads](#).
- XML payloads. See [Selective Logging for XML Payloads](#).
- JSON payloads. See [Selective Logging for JSON Payloads](#).
- Property based payloads. See [Selective Logging for Property Format Payloads](#).

Refer to the [Sample Log Policy XML File](#) to create the selective logging in an XML file.

Selective Logging for Text Payloads

To select the fields in the text format payloads, use the `<TextSelectiveLog>` tag in the log policy XML file. This table explains the field names for logging in a text payload.

Parameters for Logging in Text Payload

Parameter	Description
Prefix	Specifies a prefix string to be printed before the value of string in the payload to be selectively logged. For example, for <code><Prefix>SSN</Prefix></code> , the selectively log string is SSN : 123 45 6789
LineNumber	Specifies a line number which contains the string to be selectively logged.
StartsWith	Specifies the prefix for the string to be selectively logged.
Regex	Specifies a regular expression pattern which matches the data string to be selectively logged.
StartIndex	Specifies an index of the data string at which masking starts.
EndIndex	Specifies an index of the data string at which masking ends.

Example of Selective Logging Policy for Text Payloads

```
<TextSelectiveLog>
  <TextToLog>
    <Prefix>SSN</Prefix>
    <LineNumber>1</LineNumber>
    <StartsWith>XSD </StartsWith>
    <Regex></Regex>
    <StartIndex>1</StartIndex>
    <EndIndex>11</EndIndex>
  </TextToLog>
  <TextToLog>
    <Prefix></Prefix>
    <LineNumber>2</LineNumber>
    <StartsWith></StartsWith>
    <Regex>XSD</Regex>
    <StartIndex>1</StartIndex>
    <EndIndex>11</EndIndex>
  </TextToLog>
</TextSelectiveLog>
```

Selective Logging for XML Payloads

To select the fields in the XML payloads, use the `<XMLSelectiveLogFields>` tag in the log policy XML file. This table explains the field names for logging in a XML payload:

Parameters for Logging in XML Payload

Parameter	Description
FieldPath	Specifies the XPath for the field to be selectively logged.

Example of Selective Logging Policy for XML Payloads

```
<XMLSelectiveLogFields>
  <FieldPath>/soapenv:Envelope/soapenv:Body/
book:Title</FieldPath>
  <FieldPath>/soapenv:Envelope/soapenv:Body/
book:Author</FieldPath>
</XMLSelectiveLogFields>
```

Selective Logging for JSON Payloads

To select the fields in the JSON payloads, use the `<JSONSelectiveLogFields>` tag in the log policy XML file. This table explains the field names for logging in a JSON payload:

Parameters for Logging in JSON Payload

Parameter	Description
FieldPath	Specifies the JSON path for the field to be selectively logged.

Example of Selective Logging Policy for JSON Payloads

```
<JSONSelectiveLogFields>
  <FieldPath>BookStore.Book.[0].Author</FieldPath>
  <FieldPath>BookStore.Book.[0].ISBN</FieldPath>
</JSONSelectiveLogFields>
```

Selective Logging for Property Format Payloads

To select the fields in the property format payloads, use the `<PropertySelectiveLogFields>` tag in the log policy XML file. This table explains the field names for logging in a property format payloads.

Parameters for Logging in Text Payload

Parameter	Description
PropertyKey	Specifies the key of the property to be selectively logged.

Example of Selective Logging Policy for Property Format Payloads

```
<PropertySelectiveLogFields>
  <PropertyKey>CreditCard</PropertyKey>
  <PropertyKey>SSN</PropertyKey>
</PropertySelectiveLogFields>
```

AntiVirus Scan of Request and Response Payloads

TIBCO API Exchange Gateway can scan the request and response payloads of a facade operation.

TIBCO API Exchange Gateway integrates with McAfee Web Gateway through ICAP to scan the payloads for anti-virus and other malwares.

Configuration Setup of McAfee Web Gateway

This section describes the setup of McAfee Web Gateway for ICAP server configuration.

To setup the McAfee Web Gateway, complete these tasks:

- Download McAfee Web Gateway. See [Downloading McAfee Web Gateway](#).
- Configure McAfee Web Gateway for ICAP Server. See [Configuring McAfee Web Gateway for ICAP Server](#).

Downloading McAfee Web Gateway

You must download and configure McAfee Web Gateway ICAP server to scan the request and response payloads.



Refer to the following website for details on downloading the McAfee Web Gateway: <http://www.mcafee.com/us/products/web-gateway.aspx>

To download the trial version of the software, follow these steps:

Procedure

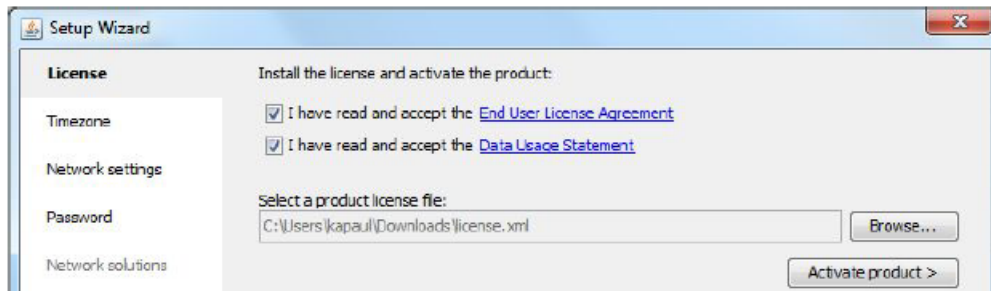
1. Open a web browser and enter the following URL:
<http://www.mcafee.com/apps/downloads/free-evaluations/default.aspx?region=us&pid=aaac14140>
2. Click the McAfee Web Gateway Virtual Appliance Evaluation Request link for trial software registration.
3. Verify that the **Trial Software Registration** page is displayed. Enter the information, as required to obtain a trial registration key.
4. Follow the instructions after you submit the registration form to download and activate the software.

Configuring McAfee Web Gateway for ICAP Server

You must configure the McAfee Web Gateway for ICAP Server.

Procedure

1. Log in to McAfee Web Console.
 - a) Open a web browser window.
 - b) Enter the following URL:
`http://McAfee_Server_Address:Port_Value`
where,
 - *McAfee_Server_Address* is the ip address of McAfee Web Gateway server.
 - *Port_Value* is the port on which McAfee Web Gateway server runs.
 - c) Enter your login credentials.
For example,
 - **username:** admin
 - **password:** webgateway
2. On the **Setup Wizard** dialog, enter the following information:

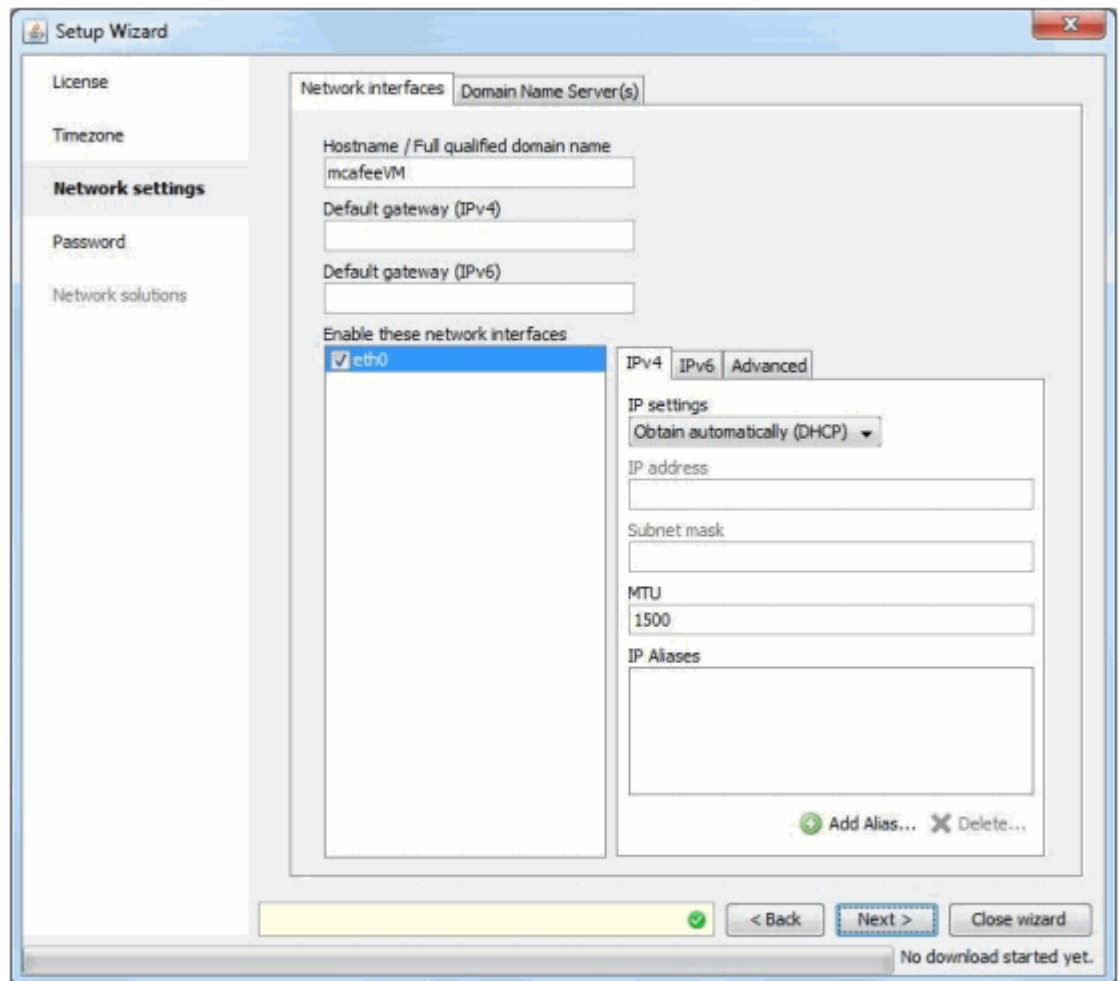


a) Click **License**.

- **Select a product license file:** Click **Browse** to select a valid product license file.
- Click **Activate product**.

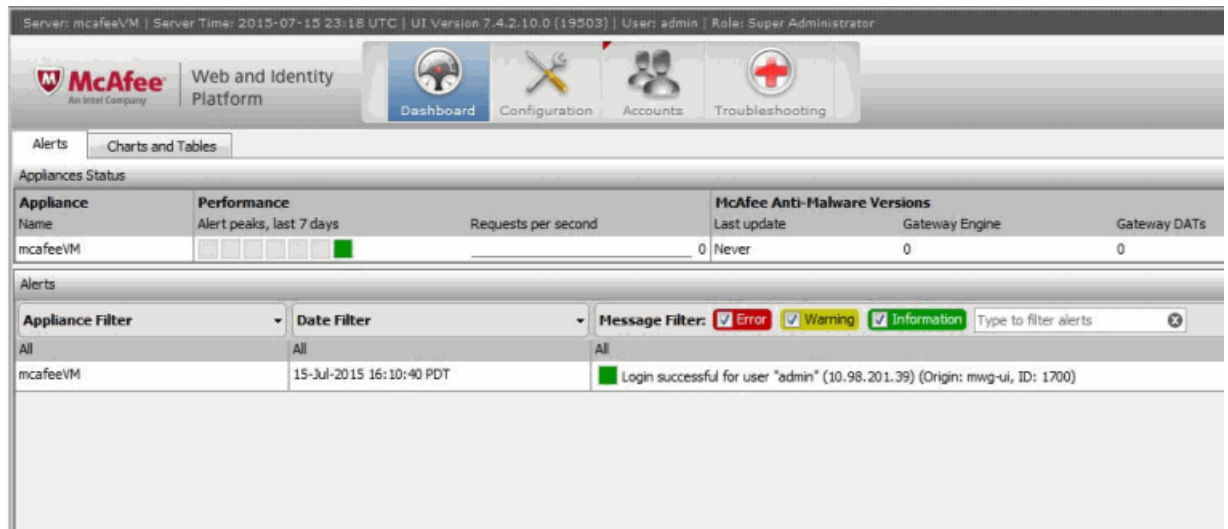
b) Click **Timezone** to set the time zone for system.

c) Click **Network Settings**. Use the default settings to set the network parameters.



d) Click **Password**. Set your account password.

3. Click **Close wizard** to close the **Setup Wizard** dialog.
4. Verify that the dashboard is displayed.



Enabling ICAP Server

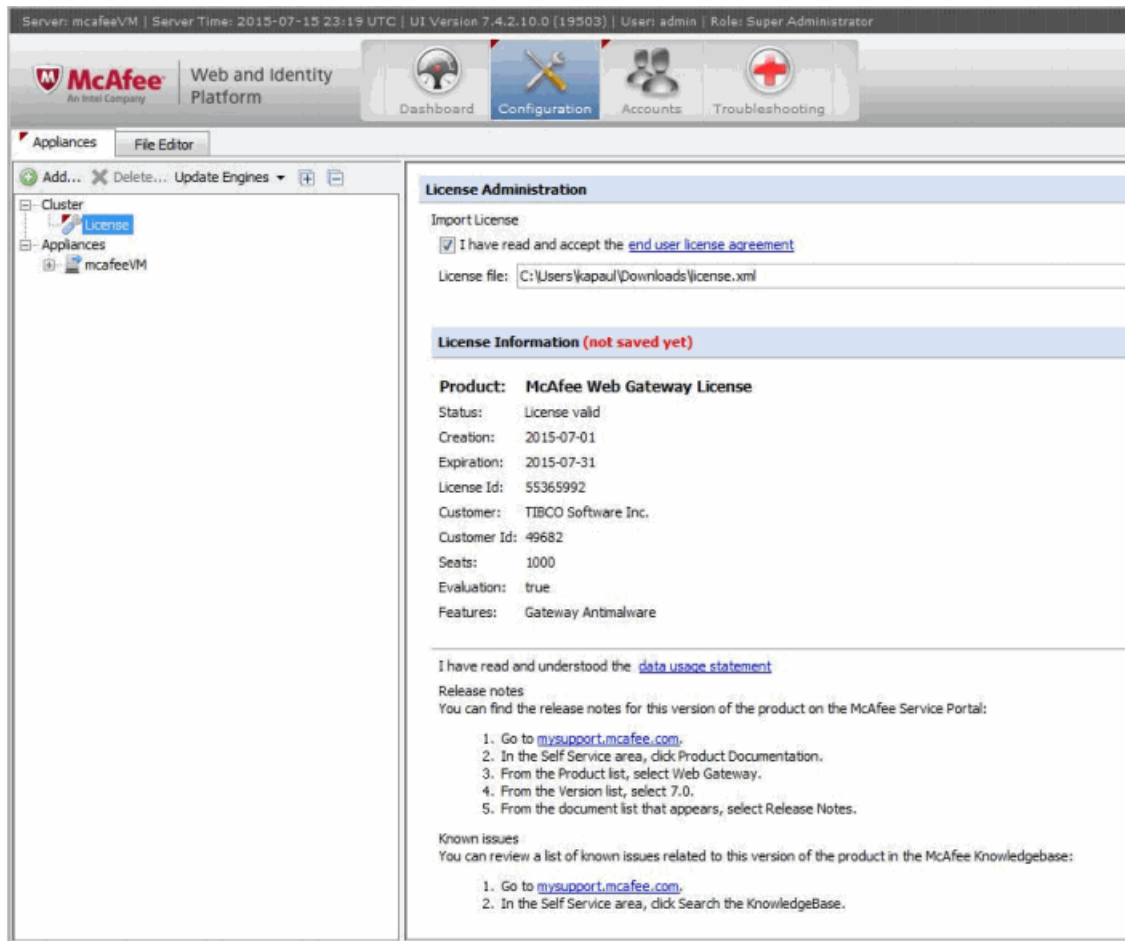
Use McAfee Web Console to enable the ICAP server.

Prerequisites

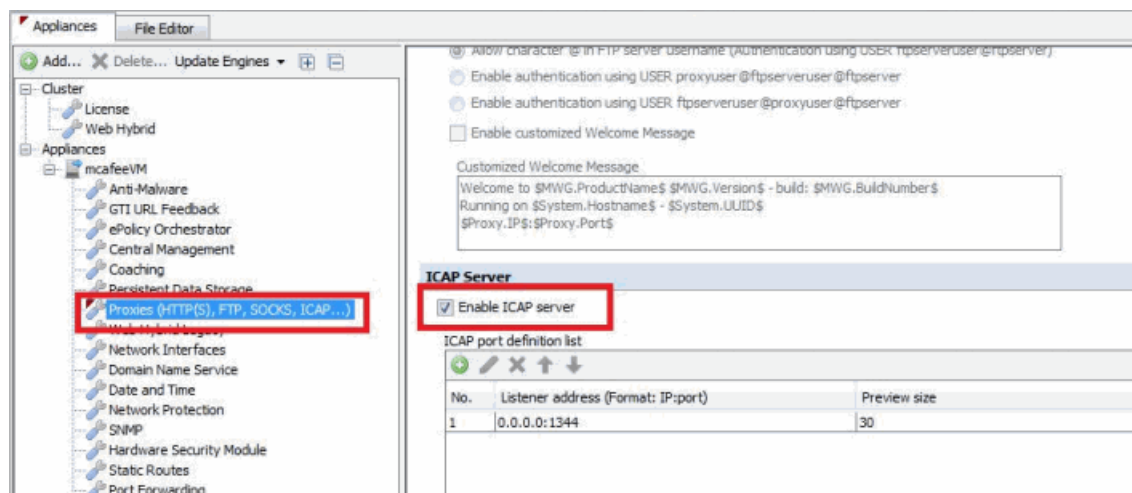
Ensure that you are logged into McAfee Web Console.

Procedure

1. Click **Configuration**.
2. Import the license file, as follows:
 - a) Click the **License** node in the explorer under **Appliances**.
 - b) **License file**: Browse and select a valid license file.



3. Click **Save Changes**.
4. Click the **Proxies** node in the explorer under **Appliances**.



5. Select **Enable ICAP server** check box to enable the ICAP server.
6. Click **Save Changes**.

Configuring ICAP Client for Request and Response

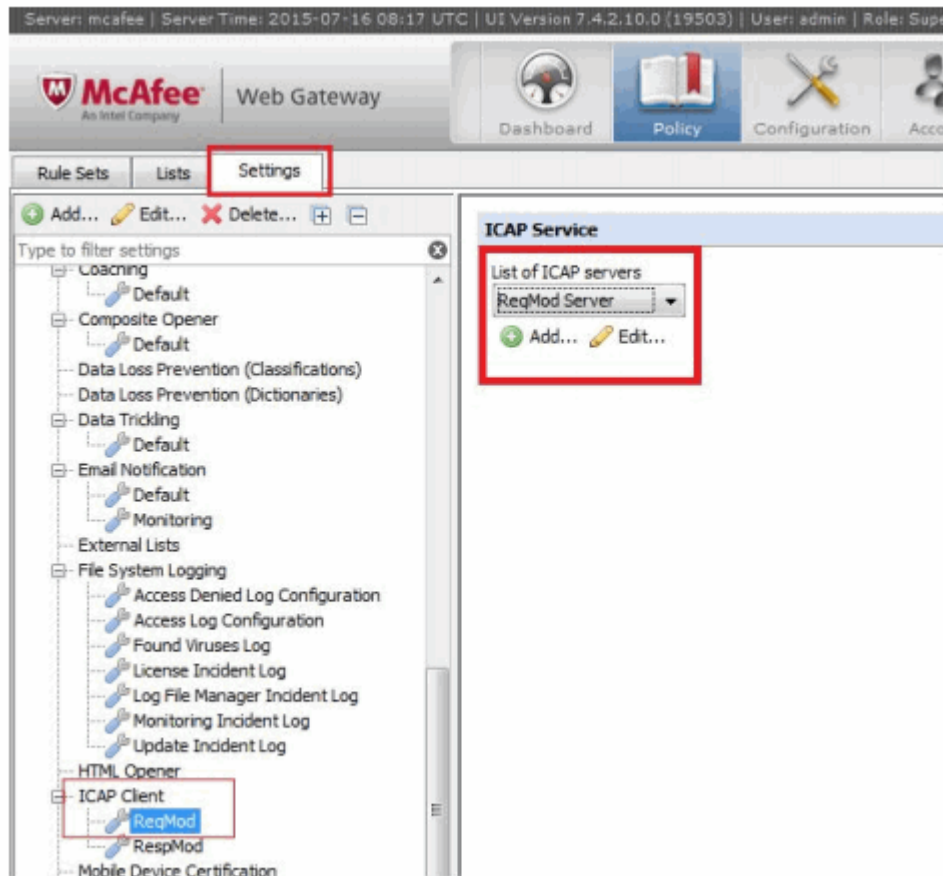
Use McAfee Web Console to configure ICAP client and servers.

Prerequisites

Ensure that you are logged into McAfee Web Console.

Procedure

1. Click **Policy**.
2. Click **Settings**.
3. Expand the **ICAP Client** node.



- a) Select **ReqMod**.
 - b) Under **ICAP Service**, select the ICAP server, as follows:
 - **List of ICAP servers:** Select **ReqMod Server** from the drop-down list.
 - c) Select **RespMod** under **ICAP Client** node .
 - d) Under **ICAP Service**, select the ICAP server, as follows:
 - **List of ICAP servers:** Select **RespMod Server** from the drop-down list.
4. Save your changes.

Configuring McAfee Web Gateway for SSL (Optional)

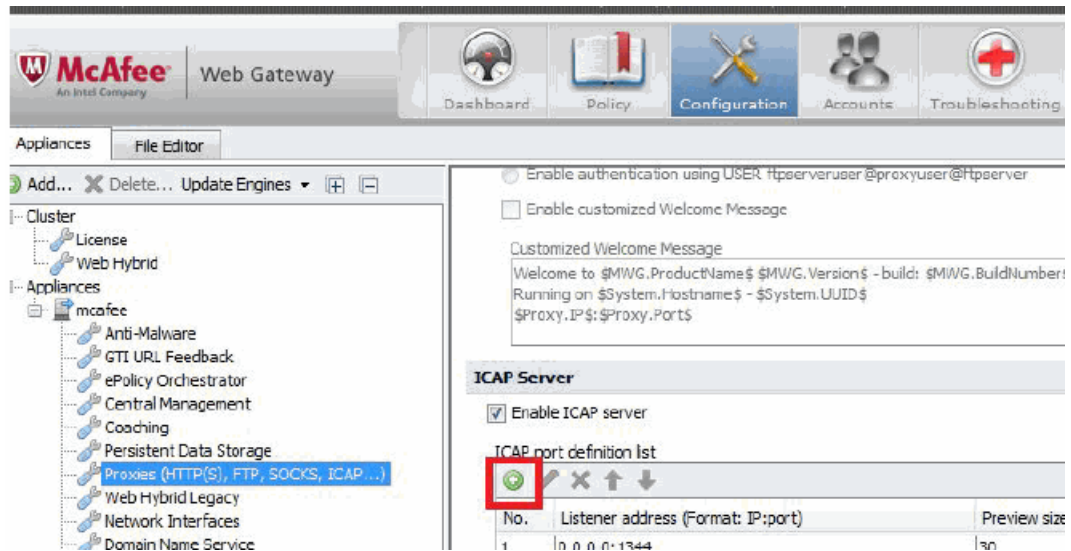
This section explains the configuration setup to use McAfee Web Gateway for SSL communications.

Prerequisites

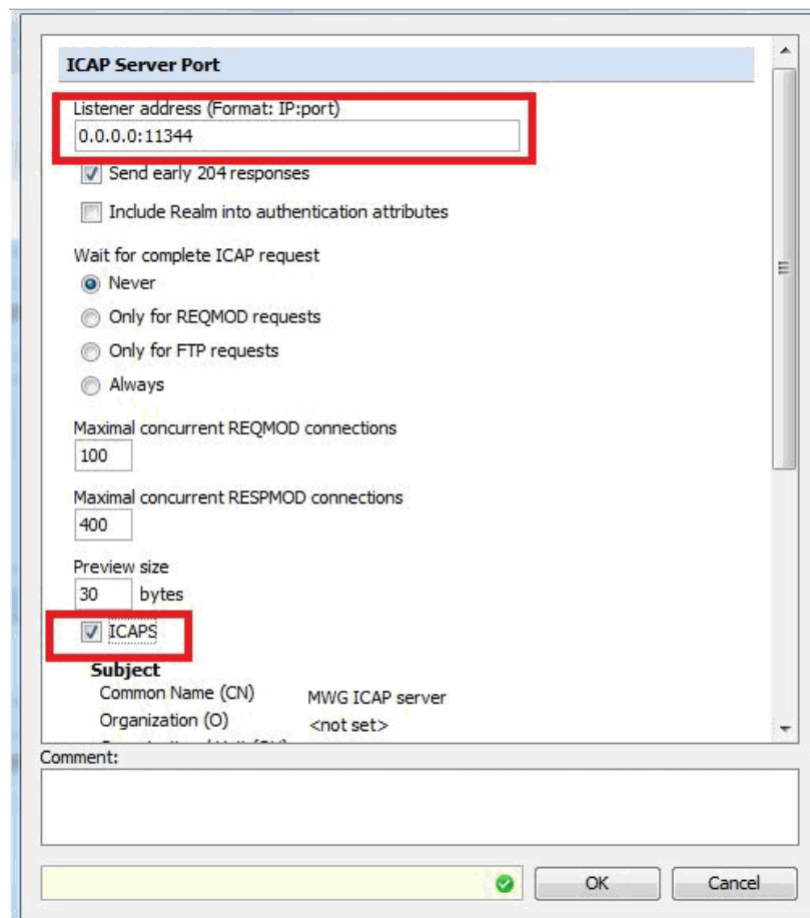
Ensure that you are logged into McAfee Web Console.

Procedure

1. Click **Configuration**.
2. Click the **Proxies** node in the explorer under **Appliances**.



3. Add a new ICAP server port, for example 11344 for SSL.



4. Select the **ICAPS** check box to enable ICAPS.

5. Click **Generate server certificate**.
6. Enter all the required information for server certificate. Click **OK**.
7. Click **Export**.
8. Save the exported certificate in a file with .crt extension. For example, save the exported certificate with the file name as mcafeeserver.crt.
9. Import the public key of the certificate to keystore of TIBCO API Exchange Gateway Server.
 - a) Navigate to *TIBCO_HOME/tibcojre64/1.7.0/bin* directory.
 - b) Enter the following command:

```
keytool -import -trustcacerts -alias rootCA -file
Exported_Certificate_From_ICAPS -keystore KeyStoreFileName
```

For example,

```
keytool -import -trustcacerts -alias rootCA -file mcafeeserver.crt -keystore
mykeystore.jks
```

Configure TIBCO API Exchange Gateway to Enable AntiVirus Scan

This section describes the configuration setup to enable the anti-virus scanning of payloads for TIBCO API Exchange Gateway.

To configure the anti-virus scanning of payloads, complete these tasks:

- Enable anti-virus scan of payloads for a facade operation.
- Set up the runtime properties. See [Setting Runtime Properties of ICAP Server](#).

Setting Runtime Properties of ICAP Server

You must set up the runtime properties of ICAP server to use the anti-virus server.

Procedure

1. Navigate to the *ASG_CONFIG_HOME* directory.
2. Open the *asg.properties* file for editing.
3. Edit the following properties, as required. See [ICAP Server properties](#) in the [Runtime Properties of Core Engine](#) table for the description of the properties. The following table shows the example value of the properties:

Property	Example Value
tibco.clientVar.ASG/ICAP/Host/URL	127.10.180.15
tibco.clientVar.ASG/ICAP/Port	1344
tibco.clientVar.ASG/ICAP/service	mcafee_gateway_service
Properties for SSL Enabled Server	
tibco.clientVar.ASG/ICAP/useSSL	true
tibco.clientVar.ASG/ICAP/keystoreFile	C:/tibco/asg/2.3/bin/ mykeystore.jks
tibco.clientVar.ASG/ICAP/keystoreType	JKS


Property	Example Value
<code>tibco.clientVar.ASG/ICAP/keystorePassphrase</code>	<code>password123</code>
<code>tibco.clientVar.ASG/ICAP/binaryContentTypeList</code>	<code>image/png</code>

4. Save changes to the file.

Enabling AntiVirus Scan for Request Payload of Facade Operation

To enable the scanning of request payload for malwares and viruses, follows these steps:


Procedure

1. Start Config UI. See [Starting GUI](#) to start the Config UI.
2. Create a new configuration or select an existing configuration, as applicable.
3. Click **ROUTING**.
4. Click **Facade Operations**.
5. Create a new operation or select an existing facade operation, as applicable. See [Facade Operations](#).
6. In the **Operation Features** field of the operation, click  (Add Feature) and select `AntiVirusCheckEnabledOnRequest` from the list box.
7. Save changes to the facade operation configuration.

Enabling AntiVirus Scan for Response Payload of Facade Operation

To enable the scanning of response payload for malwares and viruses, follows these steps:

Procedure



1. Start Config UI. See [Starting GUI](#) to start the Config UI.
2. Create a new configuration or select an existing configuration, as applicable.
3. Click **ROUTING**.
4. Click **Facade Operations**.
5. Create a new operation or select an existing facade operation, as applicable. See [Facade Operations](#).
6. In the **Operation Features** field of the operation, click  (Add Feature) and select `AntiVirusCheckEnabledOnResponse` from the list box.
7. Save changes to the facade operation configuration.

Enabling AntiVirus Scan for Request and Response Payloads of Facade Operation

To enable the scanning of request and response payloads for malwares and viruses, follows these steps:

Procedure

1. Start Config UI. See [Starting GUI](#) to start the Config UI.

2. Create a new configuration or select an existing configuration, as applicable.
3. Click **ROUTING**.
4. Click **Facade Operations**.
5. Create a new operation or select an existing facade operation, as applicable. See [Facade Operations](#).
6. In the **Operation Features** field of the operation, add the following features:
 - Click  (**Add Feature**) and select `AntiVirusCheckEnabledOnRequest` from the list box.
 - Click  (**Add Feature**) and select `AntiVirusCheckEnabledOnResponse` from the list box
7. Save changes to the facade operation configuration.

OAuth Server

This section describes how to use the TIBCO API Exchange Gateway OAuth server.

The OAuth 2.0 framework enables a third party application to access private data to which a user has granted permission. OAuth 2.0 is an open standard for authorization that allows a third party application user to share data from a site that owns data, without exposing any credentials to the application that is being accessed. TIBCO API Exchange Gateway supports the OAuth 2.0 framework.

The OAuth 2.0 Authorization Framework specification can be found at the following location:

<https://tools.ietf.org/html/draft-ietf-oauth-v2-31>.

The following topics are explained:

- OAuth server components and interactions
- Configuration setup of OAuth server
- OAuth server endpoints
- APIs supported by OAuth server

Capabilities of the OAuth Server

The OAuth server has the following features:

- Register client application using TIBCO API Exchange Manager.
- Support the following standard OAuth flows. See [OAuth Flows](#).
 - Authorization Code
 - Password Credential
 - Client Credential
- Generate authorization code.
- Generate access tokens from the OAuth flows.
- Provide APIs to manage access tokens. See the following APIs:
 - [Authorization API](#)
 - [Token Request API](#)
 - [Token Validation API](#)
- Provide the interface to build custom (non-default) adapters to plug in your implementations of the owner, client, and scope adapters. See [OAuth Service Provider Interface](#) chapter.

PingIdentity Support

To support PingIdentity authorization server for OAuth policies, make sure that you set the `Provider` field correctly in the policy file, as follows:

```
<ns:Provider>PingIdentity</ns:Provider>
```

See [AuthenticationbyOAuth Policy](#).

OAuth Client Policies

TIBCO API Exchange Gateway supports the OAuth client policies for authentication and credential mapping. See [Security Policies](#) section for details on the OAuth client policies.

OAuth 2.0 Concepts

The OAuth server uses the following concepts from the OAuth 2.0 framework. Refer to the following link for details:

<https://tools.ietf.org/html/draft-ietf-oauth-v2-31>

Resource Owner

A resource owner is an entity capable of granting access to a protected resource. When the resource owner is a person, this is referred to as an end-user.

Client

A client is an application making protected resource requests on behalf of the resource owner, which are authorized by the owner.

Client ID

A client ID is a unique identifier issued to the client by authorization server during the registration process.

Client Secret

A client secret is a password for the client. This should be kept confidential.

Authorization Server

An authorization server issues access tokens to the client after authenticating the resource owner successfully and after obtaining authorization.

Resource Server

A resource server hosts the protected resources and responds to the requests to access the protected resources using access tokens.

Authorization Code

The authorization code is obtained from an authorization server when the resource owner grants the client access to the resource.

Access Token

Access tokens are credentials used to access the protected resources.

Refresh Token

Refresh tokens are credentials used to refresh the access tokens.

Authorization Endpoint

The authorization endpoint is the endpoint on the authorization server where the client requests for authorization. The request is 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 authorization server where the client requests for access token. This includes exchanging an authorization code for an access token or refreshing access token with a refresh token.

Redirect Endpoint

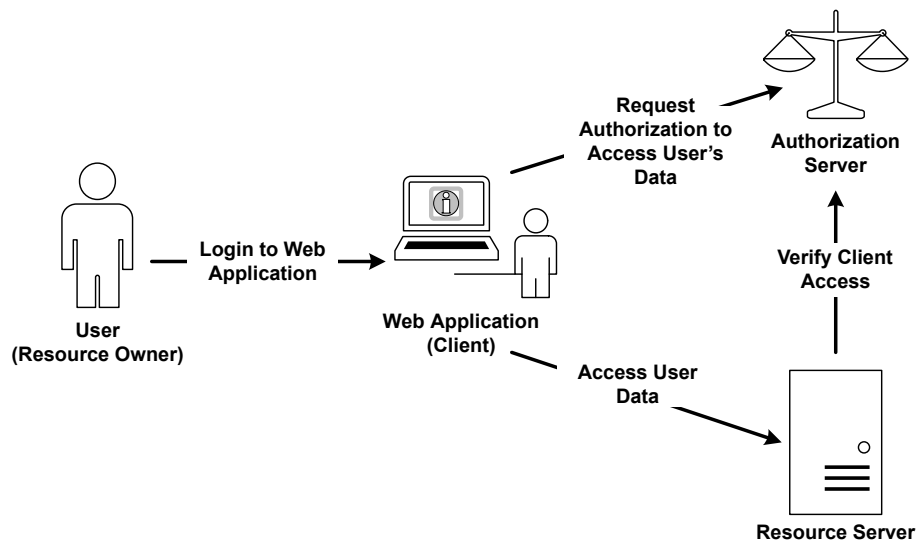
The redirect endpoint is the endpoint in the client application where the authorization server redirects to after the resource owner grants authorization to the client application. The client receives an authorization code which can be used to exchange for an access token.

Example Scenario

For example, a client application can access an API from the TIBCO API Exchange Gateway as long as the owner of the API allows the application to do so.

The following figure illustrates this example scenario.

OAuth Server Overview



The OAuth process flow for the above example is explained as follows:

- A user goes to a website hosting an application that displays some data from APIs on the gateway.
- Before the application accesses 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 asked to login and grant the application to the data that the application wishes to access.
- After the user logs in and grants access, the application receives an authorization code, which is exchanged to obtain an access token.
- The application uses the received access token to access the APIs on the gateway.

Benefits of using the OAuth Server

The OAuth server uses pluggable adapters for the following purposes:

- Authenticate owners
- Authenticate and authorize clients access

- Retrieve scopes for resources

The use of pluggable adapters enables the OAuth server to provide core OAuth2 capability such that the OAuth server can delegate the authentication and authorization for specific domain to pluggable adapters.

For example, the OAuth server can authenticate the resource owners from LDAP, database, or from any third party identity provider using owner adapters. The OAuth server can authorize resources with a scope that manage the resources for a specific domain. The resources are authorized based on the scope of resources for a specific owner.

OAuth Server Components and Interactions

This section describes the main components of OAuth server and the interactions between the components.

Components

The OAuth server has the following main components:

- User Agent
- Relying Party
- Authorization server
- Identity Service Provider
- Resource server
- Owner Adapter

An interface used to manage the owner and authenticate the user credentials. By default, TIBCO API Exchange Gateway provides file-based owner adapter.

- Client Adapter

An interface used to manage the client and validate the client credentials. By default, TIBCO API Exchange Gateway provides file-based client adapter.

- Scope Adapter

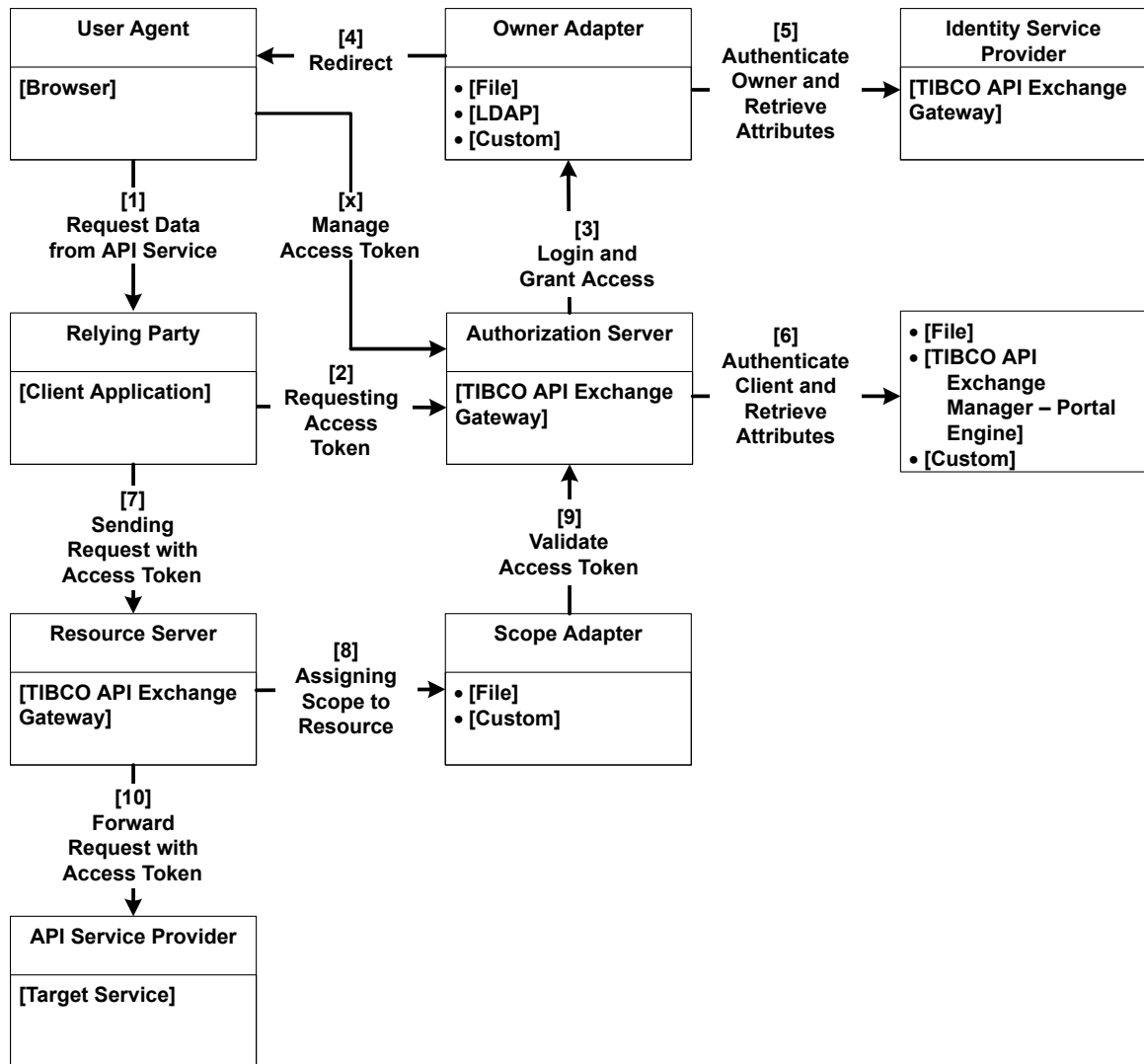
An interface used to manage the scopes for a specific owner. By default, TIBCO API Exchange Gateway provides file-based scope adapter.

- API Provider

Component Interactions

The following diagram illustrates the logical view of interaction between various components that participate in an OAuth protected data request:

Interactions Between Components



The interactions between components of the OAuth server are explained as follows:

1. 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. In a typical OAuth scenario, the resource owner sends the request from a user agent, such as a browser, to the relying party.

2. Requesting Access Token

When the relying party, such as a client application, that is serving the request, receives the request from the resource owner, the application requests an access token from the authorization server. The client application uses one of the following authorization flows to obtain the access token:

- Authorization code
- Password credential
- Client credential

For details of the call for various flows, refer to the RFC 6749, "The OAuth 2.0 Authorization Framework" available at the following location:

<https://tools.ietf.org/html/draft-ietf-oauth-v2-31>

3. Login and Grant Access

In the authorization code flow scenario, to authenticate the resource owner, the authorization server delegates the authentication process 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 makes another request to the authentication server to enable the logged in owner to grant access to the client. When the access is granted to the client, the authorization server generates an authorization code and returns the code to the relying party.



This is only applicable to the authorization code flow.

4. Redirect

When authenticating the user credentials, the owner adapter redirects the login request and grants access request to a browser.

5. Authenticate Owner and Retrieve Attributes

TIBCO API Exchange Gateway uses the owner adapter to perform the user authentication. The adapter may authenticate the owner using LDAP, database, or an SSO provider such as PingFederate™ via custom owner adapter. The successful authentication of owner returns the attributes of the owner such as owner's DN, email, and telephone. Refer to [Owner Service Provider Interface](#) for details.

By default, TIBCO API Exchange Gateway provides the flat file and LDAP based adapter. The OAuth server provides an interface to use the custom owner adapter.

See [OAuth Service Provider Interface](#) for details on how to implement the custom adapters.

6. Authenticate Client and Retrieve Attributes

The authorization server authenticates the client credentials. The successful authentication of the client returns the attributes such as redirect URL, scopes to the authorization server.



You can use the client adapter provided by TIBCO API Exchange Manager. See [Enabling OAuth for Application using TIBCO API Exchange Manager](#).

7. Sending Request with Access Token

When the relying party, such as a client application, has an access token the application forwards the request to get the resource by passing the access token in the request.

8. Assigning Scope to Resource

To perform the resource authorization at runtime, the resource server relies on resource manager to provide information required to perform the authorization.

For example, the resource manager is the TIBCO API Exchange Gateway where the resources such as a facade operation is associated with an API key (client ID).

To authorize a partner for a specific operation, a scope is assigned to a partner operation. When authorizing a request with an access token, the owner of the access token is used to lookup the partner for the request. The client ID of the access token is used as the API key of the request. A partner operation is retrieved using the partner, API key, and the URI of the request. In order to authorize the operation, the operation must have the scope that matches all of the scopes from the access token.

The scope adapter is responsible to return the scopes associated with a facade operation.

9. Validate Access Token

Before the TIBCO API Exchange Gateway resource server forwards the request from the relying party to the API provider, the resource server validates the access token from the request to ensure that the access token is valid. The request is authorized based on the information obtained from the access token.

To validate the access token, the resource server sends the access token to the authorization sever to retrieve the data associated with the access token. The access token is valid when any one of the following scenarios is true:

- It is issued by the authorization server.
- It is not expired.
- It is able to retrieve the scopes for the resource that are associated with the token.
- If the access token is not valid, the request is terminated, and an error is returned to the relying party.
- If the access token is valid, the owner of the access token is used to find the partner of the operation and the request is processed as follows:
 - If the partner is not found using the owner, the request is returned with a partner identification error.
 - The client of the operation is used to find the partner operation. If partner operation is not found, the request is returned with a partner operation not found error.
 - The partner operation must have the scope associated with the access token. If the partner operation does not have a scope that matches the one with the access token, the request is returned with scope mismatch error.

10. Forward Request with Access Token

If the owner, client, and scopes of the access token are valid for the facade operation, the resource server forwards the access token or, alternatively, converts the access token to a SAML assertion and forwards the assertion in the HTTP header.

OAuth Flows

The OAuth server supports the following OAuth flows:

- [Authorization Code](#)
- [Client Credential](#)
- [Password Credential](#)

Authorization Code

In the authorization 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 the 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, refer to "Authorization Code Grant," Section 4.1 of RFC 6749, in the "The OAuth 2.0 Authorization Framework" available at the following location:

<https://tools.ietf.org/html/draft-ietf-oauth-v2-31>

The following is an example scenario of this flow:

- A user uses an application developed by a third party developer that will access photos uploaded to an OAuth server.
- The third party developer registers the application with the OAuth server.
- The user uses the application to view the uploaded photos.
- The application requests an access token using authentication flow before retrieving the photo.

Refer to the following APIs:

- [Authorization API](#)
- [Token Request API](#)
- [Token Validation API](#)

Client Credential

The client does not need the credential of the user who uses the application. The client uses its own credential to get the data from the resource server. The OAuth server authenticates the client using the client ID and client secret of the client. For details of client credential flow, refer to section 4.4 of RFC 6749, "Client Credentials Grant", "*The OAuth 2.0 Authorization Framework*" found at the following location:

<https://tools.ietf.org/html/draft-ietf-oauth-v2-31>

The following is an example scenario of this flow:

- A user uses an application that provides some data that the client has access to.
- The application requests an access token using client credential flow before retrieving the data.

Refer to the following APIs:

- [Token Request API](#)
- [Token Validation API](#)
- [Revoke Token API](#)

Password Credential

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 applies to application design for mobile device where user credential is stored on the mobile device.

For details of password credential flow, refer to "Resource Owner Password Credentials Grant," Section 4.3 of RFC 6749, in *The OAuth 2.0 Authorization Framework* available at the following location:

<https://tools.ietf.org/html/draft-ietf-oauth-v2-31>

The following is an example scenario of this flow:

- A user uses an application on a mobile device that accesses the photos uploaded to an OAuth server.
- The user uses the application to view the uploaded photos.
- The application requests an access token using password flow before retrieving the photo.

Refer to the following APIs:

- [Token Request API](#)
- [Token Validation API](#)
- [Revoke Token API](#)

Configuration Setup of OAuth Server Authorization

This section explains the configuration setup required to use the OAuth server.

Setting OAuth Server Properties

To enable the OAuth server, set the OAuth server properties using the Config UI as follows:

Procedure

1. Navigate to *ASG_HOME/bin* directory.
2. Type `asg-configui` command to start the Config UI.
3. Open a browser window and enter the following URL:
`http://localhost:9200/ConfigUI`
4. Log in to the Config UI using your credentials.
5. On the Home page on the Config UI, select the **Gateway Engine Properties** from the drop-down list.
6. Expand the **Gateway Engine Properties** node.
7. Click the **SECURITY > Security** tab.
8. Expand the **OAuth** node to see the OAuth server properties.
9. Set the OAuth server properties, as required. Refer to the OAuth server properties defined in the **OAuth** section of the [Setting Security Properties](#) table..



- The OAuth server properties described in the [Setting Security Properties](#) can be set in the *ASG_CONFIG_HOME/asg.properties* file. See [Runtime Properties of Core Engine](#) for OAuth server properties names and description.
- It is good practice to use the Config UI to set the properties.
- Refer to the *ASG_CONFIG_HOME/asg.properties* file for the example values of OAuth server related properties.

Enable OAuth Authorization For Gateway (Set Adapter Properties)

The OAuth server uses the following adapters for authenticating owner, client, and to retrieve the scopes for authorizing the client to access resources.

- Owner Adapter - To authenticate the owner. See [Owner Adapter](#) for details.
- Client Adapter - To authenticate the client. See [Client Adapter](#) for details.
- Scope Adapter- To retrieve the scopes to authorize the client. See [Using Scope Adapter](#) for details.

Owner Adapter

The OAuth server provides the following options to authenticate the user credentials.

File-Based Owner Adapter

By default, the OAuth server provides the file- based owner adapter. To use the file-based owner adapter, follow these steps:

Procedure

1. Log in to the Config UI using your credentials.
2. On the home page on the Config UI, select the **Gateway Engine Properties** from the drop-down list.
3. Expand the **Gateway Engine Properties** node.
4. Click the **SECURITY > Security** tab.
5. Expand the **OAuth** node.

6. Set the properties as follows:

Owner Adapter Properties for File

Property	Value
Owner Adapter	com.tibco.asg.oauth.identity.provider.file.OwnerAdapterService
Resource Path Name	/examples/OAuth/resources

7. Click **Save** to save changes.
8. Set the owner credentials in the `ASG_HOME\examples\OAuth\resources\owner.properties` file.



The owner adapter properties can be set in `ASG_CONFIG_HOME\asg.properties` file, as follows:

1. Navigate to the `ASG_CONFIG_HOME` directory.
2. Edit the `asg.properties` file in a text editor.
3. Set the following property:

```
tibco.clientVar.oauth.owner.adapter.class=com.tibco.asg.oauth.identity.provider.file.OwnerAdapterService
```

4. Save changes to the `asg.properties` file.

owners.properties

Parameters for `owners.properties` file.

The following table describes the owner adapter configuration parameters for the `owners.properties` file.

Parameters for owners.properties file

Parameter	Description
name	user ID
password	user password (can be in plain text or obfuscated)
reserve	not in use
attribute	list of key value pair attributes



The following attributes must be present in the attribute list:

- dn
- uid

LDAP

To use the LDAP-based owner adapter, follow these steps:

Procedure

1. Log in to the Config UI using your credentials.
2. On the home page on the Config UI, select the **Gateway Engine Properties** from the drop-down list.
3. Expand the **Gateway Engine Properties** node.
4. Click the **SECURITY > Security** tab.
5. Expand the **OAuth** node.
6. Set the adapter properties as follows:

Owner Adapter Properties for File

Property	Value
Owner Adapter	com.tibco.asg.oauth.identity.provider.file.OwnerAdapterService
Resource Path Name	/examples/OAuth/resources

7. Set the LDAP properties as follows:

LDAP Server Connection Parameters

Property	Description
<code>tibco.clientVar.oauth.identity.provider.ldap.host</code>	
	Specifies the hostname or IP address where LDAP directory server runs. This is required. For example, ldapserver.api.tibco.com
<code>tibco.clientVar.oauth.identity.provider.ldap.port</code>	
	Specifies the port where LDAP directory server runs. This is required. For example, 10389
<code>tibco.clientVar.oauth.identity.provider.ldap.loginDN</code>	
	Specifies the base distinguished name (DN) for the login user. For example, uid=admin,ou=system
<code>tibco.clientVar.oauth.identity.provider.ldap.loginPassword</code>	
	Specifies the password for the login user. For example, root@123
<code>tibco.clientVar.oauth.identity.provider.ldap.searchFilter</code>	

Property	Description
	Specifies the filter to be used for searching in admin mode against potential user objects. For example, search filter is specified as: Objectclass=*
tibco.clientVar.oauth.identity.provider.ldap.ownerSearchTreeDn	
	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. For example, ou=people,ou=na,dc=example,dc=org
tibco.clientVar.oauth.identity.provider.ldap.ownerDnTemplate	
	Specifies a template to be used when formatting user's DN before binding. For example, uid={0},ou=employee,ou=tsi,o=tibco 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 is required for bind mode (not in admin search mode).

8. Click **Save** to save changes.
9. Set the owner credentials in the *ASG_HOME\examples\OAuth\resources\owner.properties* file.



The owner adapter properties can be set in *ASG_CONFIG_HOME\asg.properties* file, as follows:

1. Navigate to the *ASG_CONFIG_HOME* directory.
2. Edit the *asg.properties* file in a text editor.
3. Set the following property:

```
tibco.clientVar.oauth.owner.adapter.class=com.tibco.asg.oauth.identity.provider
```

4. Save changes to the *asg.properties* file.

Client Adapter

The OAuth server provides the following options to validate the client ID and client secret of an application:

File-Based Client Adapter

By default, the OAuth server provides the file-based owner adapter. To use the file-based client adapter, follow these steps:

Procedure

1. Log in to the Config UI using your credentials.
2. On the home page on the Config UI, select the **Gateway Engine Properties** from the drop-down list.
3. Expand the **Gateway Engine Properties** node.
4. Click the **SECURITY > Security** tab.

5. Expand the **OAuth** node.
6. Set the properties as follows:

Client Adapter Properties for File

Property	Value
Client Adapter	com.tibco.asg.oauth.identity.provider.file.ClientAdapterService
Resource Path Name	/examples/OAuth/resources

7. Click **Save** to save changes.
8. Set the properties for the client adapter in the `ASG_HOME\examples\OAuth\resources\client.properties` file.



The client adapter properties can also be set in the `ASG_CONFIG_HOME/asg.properties` file, as follows:

1. Navigate to the `ASG_CONFIG_HOME` directory.
2. Edit the `asg.properties` file in a text editor.
3. Set the following property:

```
tibco.clientVar.oauth.client.adapter.class=com.
tibco.asg.oauth.identity.provider.file.ClientAdapterService
```

4. Save changes to the `asg.properties` file.

clients.properties

Parameters for `clients.properties` file.

The following table describes the client adapter configuration parameters for the `client.properties` file.

Parameters for clients.properties file

Parameter	Description
name	client ID
password	client secret (can be in plain text or obfuscated)
type	client type; if set to <code>public</code> , client authentication does not require client secret.
attribute	list of key value pair attributes



The following attributes must be present in the attribute list:

- dn
- oidClientID
- oidClientRedirectURI
- scopes

TIBCO API Exchange Manager

To use the portal engine of TIBCO API Exchange Manager as the client adapter for the OAuth server, follow these steps:

Procedure

1. Log in to the Config UI using your credentials.
2. On the home page on the Config UI, select the **Gateway Engine Properties** from the drop-down list.
3. Expand the **Gateway Engine Properties** node.
4. Click the **SECURITY > Security** tab.
5. Expand the **OAuth** node.
6. Set the properties as follows:

Client Adapter Properties for TIBCO API Exchange Manager

Property	Value
Client Adapter	com.tibco.asg.portal.engine.oauth.identity.provider.ClientAdapterImpl
Engine URL	http://portal_host_name:9122 where, <i>portal_host_name</i> is the name of machine running the portal engine of TIBCO API Exchange Manager
Resource Path Name	/examples/OAuth/resources—

Note: The client adapter properties are defined in the *ASG_CONFIG_HOME/asg.properties* file, as follows:

- `oauth.client.adapter.class=com.tibco.asg.portal.engine.oauth.identity.provider.ClientAdapterImpl`
- `tibco.clientVar.portal.engine.urlhttp://portal_host_name:9122`

Scope Adapter

TIBCO API Exchange Gateway uses the scope adapter to manage the scope for a specific owner. By default, the OAuth server provides the following options for scope adapter:

File-Based Scope Adapter

To use the file-based scope adapter, follow these steps:

Procedure

1. Log in to the Config UI using your credentials.
2. On the home page on the Config UI, select the **Gateway Engine Properties** from the drop-down list.
3. Expand the **Gateway Engine Properties** node.
4. Click the **SECURITY > Security** tab.
5. Expand the **OAuth** node.
6. Set the properties as follows:

Scope Adapter Properties for File

Property	Value
Scope Adapter	com.tibco.asg.oauth.identity.provider.file.ScopeAdapterService
Resource Path Name	/examples/OAuth/resources

7. Click **Save** to save changes.
8. Set the properties for the scope adapter in the `ASG_HOME\examples\OAuth\resources\scope.properties` file.

Note: The scope adapter properties can also be set in the `ASG_CONFIG_HOME/asg.properties` file, as follows:

1. Navigate to the `ASG_CONFIG_HOME` directory.
2. Edit the `asg.properties` file in a text editor.
3. Set the following property:

```
tibco.clientVar.oauth.scope.adapter.class=com.tibco.asg.oauth.identity.provider.file.ScopeAdapterService
```

4. Save changes to the `asg.properties` file.

scopes.properties

Parameters for `scopes.properties` file

The following table describes the scope adapter configuration parameters for the `scope.properties` file.

Parameters for scopes.properties file

Parameter	Description
scope	the name to identify the scope
description	the description of the scope

Non-Default (Custom) Adapter For Owner Client and Scopes

To use the other resources such as database to authenticate the client, owner and manage the scopes, implement the custom adapters, which can be integrated to the interface provided by TIBCO API Exchange Gateway.

See [OAuth Service Provider Interface](#) section for details.

Starting OAuth Server

The OAuth server is integrated within TIBCO API Exchange Gateway. The OAuth server is run as a Core Engine instance.

To start an instance of OAuth server, follow these steps:

Procedure

1. Set the OAuth server properties. See [Setting OAuth Server Properties](#).
2. To authenticate owner, client and manage the scopes, configure an adapter and set the properties. See [Enable OAuth Authorization For Gateway \(Set Adapter Properties\)](#) section.
3. Start a Core Engine instance using the following command:

On the Windows platform, type the following command:

```
asg-engine -u asg-caching-core -a ASG_Configuration
```

On the UNIX platform, type the following command:

```
./asg-engine -u asg-caching-core -a ASG_Configuration
```

Manage Access Token

To access the target services using access tokens, the OAuth server can perform the following tasks:

- Authorize a request to obtain an access token
- Generate an access token
- Refresh an access token
- Validate an access token
- Retrieve information for a valid access token
- Retrieve list of issued access tokens for a owner
- Revoke an access token

OAuth Server Endpoint

You can secure the applications such as mobile applications, web applications or any type of applications using the OAuth flows supported by the OAuth server.

To access the endpoints of OAuth server, the following information is required:

- Client ID
- Client Secret
- Authorization grant type defined as an OAuth flow
- Additional Information such as user, password, redirect URL, authorization code, and so on required for an OAuth flow

See [OAuth Server Endpoints](#).

Token Management APIs

The OAuth server provides the APIs to manage the access tokens. Refer to the following APIs section for details:

- [Authorization API](#)
- [Token Request API](#)
- [Token Validation API](#)

OAuth Server Endpoints

This section explains various endpoints to access the OAuth server for access token management.

Transport and Port

The OAuth server provides both HTTP and HTTPS transport for the endpoints. In production environments, it is good practice to use HTTPS which provides the transport level security.



By default, the ports for the HTTP and HTTPS transports are enabled with default values as 9322 and 9333 respectively.

If required, change the default value of the ports as follows:

OAuth Server Transport and Port

Parameter	Config UI Field	Runtime Property
HTTP Port	<ul style="list-style-type: none"> Set the Transport scheme to HTTP as follows: Gateway Engine Properties > Security > OAuth > Transport Scheme Set the Port as follows: Gateway Engine Properties > Security > OAuth > Port 	Set in the <code>ASG_CONFIG_HOME/asg.properties</code> file. The default value is 9322. <code>tibco.clientVar.DefaultImplementation/Connections/HTTP/OAuthWebappsConnection/Port</code>
HTTPS Port	<ul style="list-style-type: none"> Set the Transport scheme to HTTPS as follows: Gateway Engine Properties > Security > OAuth > Transport Scheme Set the Port as follows: Gateway Engine Properties > Security > OAuth WebApps SSL > Port 	Set in the <code>ASG_CONFIG_HOME/asg.properties</code> file. The default value is 9333. <code>tibco.clientVar.DefaultImplementation/Connections/HTTPS/OAuthWebappsSSLConnection/Port</code>



If you use HTTPS transport, set the SSL properties listed using the Config UI as follows:

- On the home page of Config UI, go to **Gateway Engine Properties > Security > OAuth WebApps SSL**.
- Set the OAuth SSL properties. See [OAuth WebApps SSL](#) for details

Request Access Token

You must request an access token to access the OAuth enabled target services.

Based on the type of OAuth flow, the endpoint to request an access token requires different parameters. See [Token Request API](#) for details.

The following are the endpoints for each supported flow:

Client Credential Flow

Endpoint for client credential flow.

The client credential flow is used when you want to access the target services using any trusted application. See [Client Credential](#) for details.

The client credential flow requires the following information to send an access token request:

- Client ID
- Client Secret
- grant_type

Use the following endpoint to request an access token:

Token Request Endpoint for Client Credential Flow

Method	URL
POST	<p><code>http://ASGServerHost:httpPort/asg/oauth2/access_token?grant_type=client&client_id=CLIENT_ID_VALUE&client_secret=CLIENT_SECRET_VALUE</code></p> <p>For example,</p> <p><code>http://demoserverapi.tibco.com:9322/asg/oauth2/access_token?grant_type=client&client_id=237-924f4a26-f1a5-4934-a17a-69c22bd52dbe&client_secret=809950e0-c21c-4f84-8dab-239dba1c3187</code></p>

where,

- *ASGServerHost* is the machine running the Core Engine.
- *httpPort* is the port value for HTTP transport.
- *CLIENT_ID_VALUE* is the client ID of the registered application.
- *CLIENT_SECRET_VALUE* is the client secret of the registered application.

Password Credential Flow

Endpoint for password credential flow.

The password flow is used when you want to access the target services from an application that requires a username and password. When any application such as a web or mobile application is launched, and prompts for a username and password, the user credentials are verified by an identity service provider and exchanged for an access token.

The OAuth server validates the credentials before issuing the access token. After the access token is issued, the access token is the key to access the target services.

See [Password Credential](#) for details.

The password credential flow requires the following information to send an access token request:

- Client ID
- Client Secret
- Username
- Password
- grant_type

Use the following endpoint to request an access token:

Token Request Endpoint for Password Flow

Method	URL
POST	<p><code>http://ASGServerHost:httpPort/asg/oauth2/access_token?grant_type=password&client_id=CLIENT_ID_VALUE&client_secret=CLIENT_SECRET_VALUE&username=USERNAME&password=PASSWORD</code></p> <p>For example,</p> <p><code>http://demoserverapi.tibco.com:9322/asg/oauth2/access_token?grant_type=client&client_id=237-924f4a26-f1a5-4934-a17a-69c22bd52dbe&client_secret=809950e0-c21c-4f84-8dab-239dba1c3187&username=john123&password=asgoauth2014</code></p>

where,

- *ASGServerHost* is the machine running the Core Engine.
- *httpPort* is the port value for HTTP transport.
- *CLIENT_ID_VALUE* is the client ID of the registered application.
- *CLIENT_SECRET_VALUE* is the client secret of the registered application.
- *USERNAME* is the username required for application.
- *PASSWORD* is the password required for application.

Authorization Code Flow

Endpoint for authorization code flow.

To request an access token using the authorization code flow, an authorization code is required.

See [Authorization Code](#) for details.

The following steps are required to process the access token request for authorization code flow:

Procedure

1. Request an authorization code

The access token request requires a authorization code. Use the following endpoint to request an authorization code:

Authorization Code Request Endpoint for Authorization Code Flow

Method	URL
POST	<p><code>http://ASGServerHost:httpPort/asg/oauth2/authorize?response_type=code&client_id=CLIENT_ID_VALUE&client_secret=CLIENT_SECRET_VALUE&state=STATE_VALUE&redirect_uri=REDIRECT_URL_VALUE</code></p> <p>For example,</p> <p><code>http://ASGServerHost:httpPort/asg/oauth2/authorize?response_type=code&client_id=237-924f4a26-f1a5-4934-a17a-69c22bd52dbe&client_secret=809950e0-c21c-4f84-8dab-239dba1c3187&state=xyz&redirect_uri=https%3A%2F%2Fclient%2Eexample%2Ecom%2Fcb</code></p>

2. Request an access token

The authorization code flow requires the following information to send an access token request:

- Client ID
- Client Secret
- Authorization code
- Redirect URL
- grant_type

After the authorization code is returned, use the following endpoint to request an access token:

Token Request Endpoint for Authorization Code Flow

Method	URL
POST	<p><code>http://ASGServerHost:httpPort/asg/oauth2/access_token?grant_type=authorization_code&client_id=CLIENT_ID_VALUE&client_secret=CLIENT_SECRET_VALUE&code=AUTH_CODE_VALUE&redirect_uri=REDIRECT_URL_VALUE</code></p> <p>For example,</p> <p><code>http://ASGServerHost:httpPort/asg/oauth2/access_token?grant_type=authorization_code&client_id=237-924f4a26-f1a5-4934-a17a-69c22bd52dbe&client_secret=809950e0-c21c-4f84-8dab-239dba1c3187&code=SpLxIOBeZQQYbYS6WxSbIA&redirect_uri=https%3A%2F%2Fclient%2Eexample%2Ecom%2Fcb</code></p>

where,

- *ASGServerHost* is the machine running the Core Engine..
- *httpPort* is the port value for HTTP transport.
- *CLIENT_ID_VALUE* is the client ID of the registered application.
- *CLIENT_SECRET_VALUE* is the client secret of the registered application.
- *STATE_VALUE* is an arbitrary string that is returned in the callback.
- *AUTH_CODE_VALUE* is the authorization code.
- *REDIRECT_URL_VALUE* is the URL of the third party application which performs the authentication process.

Access Token Response Example

The response for the access token request contains the following information:

- Access Token
- Refresh token (Optional)
- Expiry information for access token
- Authorization grant type
- Additional information depending on the used OAuth flow.

Sample Response

For example, the successful response for password credential flow is returned as follows:

Response:

```
{
  "expires_in": 3600,
  "token_type": "Bearer",
  "refresh_token": "T1amGR21.IdKM.5ecbf91162691e15913582bf2662e0",
  "access_token": "T1amGT21.Idup.298885bf38e99053dca3434eb59c6aa"
}
```

Access Token Error Example

For example, the following is an error response returned for an access token request which contains incorrect grant type:

Sample Response

For example, the error response is returned as follows:

Invalid Response: 400 Bad Request

```
{
  "error": "invalid_grant",
  "error_description": "Invalid username or password"
}
```

Retrieve Access Token Details

You can retrieve the information of an access token as follows:

- Using an access token
- List of access tokens issued by authorization server
- For a specific owner

Using Access Token

Validating Access Token Request

Procedure

- To validate an access token and retrieve the token details, use the following endpoint:

Retrieve Access Token using Token

Method	URL
GET	<p><code>http://ASGServerHost:httpPort/asg/oauth2/access_token/Token_ID</code></p> <p>For example,</p> <p><code>http://ASGServerHost:httpPort/asg/oauth2/access_token/T1amGT21.Idup.3446d8984b48a7c0c531267317334ea7</code></p>

where:

- Token_ID* refers to a valid access token.

Sample Response (Retrieve Access Token)

The following is an example response of retrieving an access token details using an access token for client credential flow:

```
{
  "id": "5d34e914ad73b262dbf88a963dbccb41",
  "accessToken": "T1amGT21.Idup.3446d8984b48a7c0c531267317334ea7",
  "refreshToken": "T1amGR21.IdKM.8a7420fe5079ab1b3bd24b4de1d5fe2",
  "secret": "619307e5-8e20-4997-9045-c083c123e04f",
  "owner": "251-20c8104e-e91a-4384-980c-84545aced892",
  "client": "251-20c8104e-e91a-4384-980c-84545aced892",
  "classification": "reviews",
  "scopes": [
    "reviews"
  ],
  "status": "Active",
  "key": 0,
  "createdOn": 1409942780365,
  "expiresOn": 1409946380427,
  "grantType": "client_credentials",
  "samlToken": null,
  "stringId": "5d34e914ad73b262dbf88a963dbccb41"
}
```

Retrieve List of Tokens

To retrieve the list of tokens maintained by the authorization server, use the following endpoint:

Retrieve List of Tokens

Method	URL
GET	<p><code>http://ASGServerHost:httpPort/asg/oauth2/access_token/owners</code></p> <p>For example,</p> <p><code>http://ASGServerHost:httpPort/asg/oauth2/access_token/owners</code></p>

where:

- *ASGServerHost* is the host name running TIBCO API Exchange Gateway.
- *httpPort* is the port value for HTTP transport.

Retrieve Token for Specific Owner

To retrieve the information for a token issued to a specific owner, use the following endpoint:

Retrieve Access Token for an Owner

Method	URL
GET	<code>http://ASGServerHost:httpPort/asg/oauth2/access_token/owners/owner_ID</code> For example, <code>https://120.107.172.36:9333/asg/oauth2/access_token/owners/ CN=Administrator,CN=Users,DC=config2913vm0,DC=ad</code>

where,

- *ASGServerHost* is the machine running the Core Engine.
- *httpPort* is the port value for HTTP transport.
- *owner_ID* is the ID of owner.

Refresh Token

A refresh token is issued by the authorization server for an authorization code flow. When the current access token expires or is invalid, a refresh token is used. When the authorization server issues the access token, optionally the server can issue the refresh token.

To refresh an access token generated for access token, use the following endpoint:

Refresh Token Endpoint

Method	URL
POST	<code>http://ASGServerHost:httpPort/asg/oauth2/access_token? grant_type=refresh_token&client_id=CLIENT_ID_VALUE&client_secret=CLIENT_SE CRET_VALUE&refresh_token=REFRESH_TOKEN</code> For example, <code>http://demoserverapi.tibco.com:9322/asg/oauth2/access_token? grant_type=refresh_token&client_id=237-924f4a26-f1a5-4934- a17a-69c22bd52dbe&client_secret=809950e0- c21c-4f84-8dab-239dba1c3187&refresh_token=T1amGT21.Idup. 3446d8984b48a7c0c531267317334ea7</code>

where,

- *ASGServerHost* is the machine running the Core Engine..
- *httpPort* is the port value for HTTP transport.
- *CLIENT_ID_VALUE* is the client ID of the registered application.
- *CLIENT_SECRET_VALUE* is the client secret of the registered application.

- *REFRESH_TOKEN* is the refresh token for the access token generated.

Sample Response (Refresh Token)

The following is a sample response for refresh_token request:

```
{
  "expires_in": 3600,
  "token_type": "Bearer",
  "refresh_token": "T1amGR21.IdKM.dee72962103d707169e6e51d7fd5b58",
  "access_token": "T1amGT21.Idup.e684f84d18e4bedec955c75482acef9"
}
```

Revoke Token

To revoke an access token from a specific owner, use the following endpoint:

Revoke Token Endpoint

Method	URL
DELETE	<p><code>http://ASGServerHost:httpPort/asg/oauth2/access_token/Token_ID</code></p> <p>For example,</p> <p><code>http://demoserverapi.tibco.com:9322/asg/oauth2/access_token/T1amGT21.Idup.3446d8984b48a7c0c531267317334ea7</code></p>

where,

- *ASGServerHost* is the host name running TIBCO API Exchange Gateway.
- *httpPort* is the port value for HTTP transport.
- *Token_ID* is the access token to be revoked.

Sample Response (Revoke Token)

The following is a sample response for revoke token request:

```
{
  "id": "429730f4beacf9882992b19f739f4d2e",
  "accessToken": "T1amGT21.Idup.e3a4999fa859deffe8d430e4464382",
  "refreshToken": "T1amGR21.IdKM.a18216e8acbd12fff96dc6f8fe7e8766",
  "secret": "49a8933c-152f-4486-954e-cf85c7eedb16",
  "owner": "CN=Administrator,CN=Users,DC=config2913vm0,DC=ad",
  "client": "379-be9ad58f-ac8a-4c47-bfe8-3ceb9f57ef39",
  "classification": "oauthsoap",
  "scopes": [
    "oauthsoap"
  ],
  "status": "Active",
  "key": 0,
  "createdOn": 1412193378135,
  "expiresOn": 1412196978229,
}
```

```

"grantType": "password",
"samlToken": null,
"stringId": "429730f4beacf9882992b19f739f4d2e"
}

```

Accessing Token Persistence

The OAuth server uses ActiveSpaces as caching and persistence layer. The access tokens are persisted in the database or memory. Use the database in production systems to store the access tokens. If you choose the memory to store the access tokens, they are lost if the OAuth server goes down.

To use ActiveSpaces for access tokens persistence, follow these steps:

Procedure

1. Start the Config UI, if not running.
2. Log in to the Config UI using your credentials.
3. On the home page of the Config UI, select the **Gateway Engine Properties** from the drop-down list.
4. Click the **Security** link.
5. Set the OAuth Metaspace properties as explained in the [OAuth Data Space](#).
6. Save the changes.

You can set the following properties for OAuth Data Space in `ASG_CONFIG_HOME/asg.properties` file:

Properties for Access Token Persistence

<code>tibco.clientVar.oauth.dataspace.metaspace.name</code>	
	<ul style="list-style-type: none"> • Specifies the metaspace name used by the OAuth server. • The default value is ASG-OAuth-Tokens.
<code>tibco.clientVar.oauth.dataspace.local.discovery</code>	
	<p>Specifies the discovery URL for this OAuth instance of the metaspace discovers the current metaspace members.</p> <p>For example, <code>tcp://machine1_IP_Address:6300;machine2_IP_Address:6300</code></p>
<code>tibco.clientVar.oauth.dataspace.local.listen</code>	
	<p>Specifies the listening URL for this OAuth instance of the metaspace.</p> <p>For example, <code>tcp://machine1_IP_Address:6300</code></p>
<code>tibco.clientVar.oauth.dataspace.load.batch.size</code>	

<ul style="list-style-type: none"> Specifies the maximum number of entries to return when querying data such as an access token. The default value is 1024. Newest tokens are loaded first.
<code>tibco.clientVar.oauth.dataspace.persister.store</code>
<ul style="list-style-type: none"> Defines the type of persistence store. The possible values are: <ul style="list-style-type: none"> InMemory Database <p>If the Database is set, define the properties for database server connection.</p>
<code>tibco.clientVar.oauth.dataspace.capacity</code>
<p>Specifies the maximum number of tokens to store in the local cache.</p> <p>The default value is 1024.</p>
<code>tibco.clientVar.oauth.dataspace.replication.count</code>
<p>Specifies the number of seeders that are used to replicate the token. If you have n number of OAuth servers, set this property to n-1 to replicate the token to all servers. The default value is -1.</p> <p>For example, setting this property to 1 means that the token is replicated to one additional seeder.</p>
<code>tibco.clientVar.oauth.dataspace.replication.policy</code>
<p>Specifies the OAuth access token replication policy when more than one OAuth servers are configured in a cluster. The possible policy options are:</p> <ul style="list-style-type: none"> sync: as the tokens are added to the OAuth servers, they are replicated immediately to all seeders in the cluster. async: as the tokens are added to the OAuth servers, it does not guarantee that the tokens are replicated immediately. <p>The default value is async.</p>
<code>tibco.clientVar.oauth.dataspace.eviction.policy</code>
<p>Specifies how a token lookup request is received for a token that exists in the database but not in the cache, and if the cache capacity is reached.</p> <p>The default value is LRU. This means the token is read from the database and one token from the cache is evicted.</p>
Properties For OAuth Server Persister Store of Database Type
<code>tibco.clientVar.oauth.dataspace.persister.jdbc.driver</code>

Specifies the database jdbc driver when the database is used as OAuth persistence store.
<code>tibco.clientVar.oauth.dataspace.persister.jdbc.url</code>
Specifies the jdbc url for the database server when the database is used as OAuth persistence store.
<code>tibco.clientVar.oauth.dataspace.persister.jdbc.username</code>
Specifies the user to connect to the database server when the database is used as OAuth persistence store.
<code>tibco.clientVar.oauth.dataspace.persister.jdbc.password</code>
Specifies the password of the user to connect to the database server when the database is used as OAuth persistence store.
<code>tibco.clientVar.oauth.access.token.retention.period</code>
Specifies the expiration time (in minutes) for an access token. The default value is 60 minutes. When the access token passes expiration time as specified by this property, it is no longer valid but still remains in the database. The access token is removed from the database based on the value specified by the <code>tibco.clientVar.oauth.access.token.retention.time</code> property.
<code>tibco.clientVar.oauth.access.token.retention.time</code>
<p>Specifies the retention period (in minutes) for an access token. The default value is 1440 minutes (1 day) . When the access token passes retention period as specified by this property, the token is removed from the database. By default, the access token is removed from the database after 1 day.</p> <ul style="list-style-type: none"> • The value of <code>tibco.clientVar.oauth.access.token.retention.period</code> property must be less than the value specified by the <code>tibco.clientVar.oauth.access.token.retention.time</code> property. • Note: In a multi-instance TIBCO API Exchange environment, if you change the TTL value on a gateway instance, shut down all instances that connect to that metaspace and then restart the instances.

Enabling OAuth for Application using TIBCO API Exchange Manager

Using the TIBCO API Exchange Manager, you can access the target services deployed on the TIBCO API Exchange Gateway, as follows:

- Using the API key. This is the default behavior. See the following guides on how to use API key to access the target services:
 - TIBCO API Exchange Manager Administration
 - Adapter Code for TIBCO® API Exchange and Joomla! User's Guide
 - Adapter Code for TIBCO® API Exchange and Joomla! Administration
- Using the access token issued by the OAuth server. See [OAuth Server Endpoints](#).

This section explains how you can use TIBCO API Exchange Manager to register an application for OAuth authorization and generate the keys (client ID and client secret). After the client ID and client secret are sent to an application, exchange these keys to obtain an access token.

The following are the high level steps to secure your application for OAuth authorization using TIBCO API Exchange Manager:

Procedure

1. Enable Application for OAuth

To enable any application for OAuth authorization using TIBCO API Exchange Manager, follow these steps:

- a) Log in to the TIBCO API Exchange Manager Joomla Administrator.
- b) Click the **System > Global Configuration > API manager Config and Email template** link.
- c) Set **Enable OAuth** to **Normal**.
- d) Save your settings.

Refer to following guides for details:

- *Adapter Code for TIBCO® API Exchange and Joomla! User's Guide*
- *Adapter Code for TIBCO® API Exchange and Joomla! Administration*



Adapter Code for TIBCO® API Exchange and Joomla! User's Guide and *Adapter Code for TIBCO® API Exchange and Joomla! Administration* manuals are available at the following location: <https://github.com/API-Exchange/JoomlaAdapter/wiki>

2. Register Application for OAuth

When you create an application, set the following parameters of the application for OAuth authorization under **Scopes**:

- a) Set **Enable OAuth** to **Yes**.
- b) Select the **Scope** for an application, such as public.
- c) Enter **Redirect URL** which is required for authorization code flow.
- d) Save changes to the application.

Refer to following guides on how to setup a new application:

- *Adapter Code for TIBCO® API Exchange and Joomla! User's Guide*
- *Adapter Code for TIBCO® API Exchange and Joomla! Administration*

Adapter Code for TIBCO® API Exchange and Joomla! product manuals are available at the <https://github.com/API-Exchange/JoomlaAdapter/wiki> location.

3. Request Key for an Application

To receive the client ID and client secret for an application, follow these steps:

- a) Select the application.
- b) Click **Request Key** tab.
- c) Verify that the client ID and client secret are returned on the screen.

4. Use Client ID and Client Secret to Request Access Token

After the client ID and client secret are generated for the application, use a REST client such as POSTMAN to request an access token. For the endpoints details, see [Authorization API](#).

5. Use Access Token to Access Target Services

After the access token is sent to the application by the OAuth server, use the access token to access the target services hosted by TIBCO API Exchange Gateway.

For example, to query books by author using the access token, use the following URL:

`http://ASGGatewayHost:ASGGatewayPort/Books/BookOperations/Author/Vivek Ranadive?
access_token=T1amGT21.Idup.e684f84d18e4bedec955c75482acef9`

Authorization API

The OAuth server provides the following API to authorize a request:

Name

`/authorize`

Description

Process an authorization request.

Authorization Request

Use the following parameters to send an authorization request. The parameters can be added to the query component of the authorization endpoint URI using the "application/x-www-form-urlencoded" format.

Authorization Request Parameters

Parameter	Description
<code>response_type</code>	The value must be set to code. Required.
<code>client_id</code>	The client identifier issued to the client by the authorization server during the registration process. Required.
<code>redirect_uri</code>	The URL where the authorization server sends the authorization code. Optional. Refer to Section 3.1.2 of RFC 6749 for details.
<code>scope</code>	Specifies the scope of the access request. Refer to Section 3.3 of RFC 6749 for details. Optional.
<code>state</code>	Specifies an opaque value used by the client to maintain the state between the request and callback. The authorization server includes the state value when redirecting the user-agent back to the client. Optional.

Authorization Request Example

The client directs the user-agent to make the following HTTP request using TLS:

```
GET /asg/oauth2/authorize?response_type=code&client_id=s6BhdRkqt3&state=xyz
&redirect_uri=https%3A%2F%2Fclient%2Eexample%2Ecom%2Fcb HTTP/1.1
Host: server.example.com
```

Authorization Response

The authorization server processes the authorize request from client as follows:

- Issues an authorization code
- Adds the parameters as described in the following table to the query component of the redirection URI using "application/x-www-form-urlencoded" format
- Delivers the authorization code to the client

Authorization Response Parameters

Parameter	Description
code	The authorization code generated by the authorization server. Required. Refer to Section 4.1.2 of RFC 6749 for details.
state	Refers to the exact state parameter value as received from the client. This is required if the state parameter was present in the client authorization request.

Authorization Response Example

The authorization server redirects the user-agent by sending the following HTTP response.

```
HTTP/1.1 302 Found
Location: https://client.example.com/cb?code=Sp1x10BeZQYbYS6WxSbIA
```

Authorization Error

The authorization server returns an error response if the request processing fails. The processing of the request fails due to one of the following reasons:

- Missing, invalid, or mismatching redirection URI
For this case, the authorization server informs the resource owner of the error and does not automatically redirect the user-agent to the invalid redirection URI.
- Missing or invalid client identifier
For this case, the authorization server informs the resource owner of the error and does not automatically redirect the user-agent to the invalid redirection URI.

- Resource owner denies the access request

The authorization server informs the client by adding the following parameters to the query component of the redirection URI using the `application/x-www-form-urlencoded` format.

Authorize Request Error Parameters

Parameter	Description
error	Specifies a single error code returned from the authorization server. Required. Refer to Authorize Request Error Codes table for the error codes.
state	Refers to the exact state parameter value as received from the client. This is required if the state parameter was present in the client authorization request.

Authorize Request Error Codes

Error Code	Description
invalid_request	
	The request is missing a required parameter, includes an invalid parameter value, includes parameter more than once, or is otherwise malformed.
unauthorized_client	
	The client is not authorized to request an authorization code using this method.
access_denied	
	The resource owner or authorization server denied the request.
unsupported_response_type	
	The authorization server does not support obtaining an authorization code using this method.
invalid_scope	
	The requested scope is invalid, unknown, or malformed.
server_error	
	The authorization server encountered an unexpected condition that prevented it from fulfilling the request. This error code is needed because a 500 Internal Server Error HTTP status code cannot be returned to the client via an HTTP redirect.

Authorization Error Example

The authorization server redirects the user-agent by sending the following HTTP response:

```
HTTP/1.1 302 Found
```

```
Location: https://client.example.com/cb?error=access_denied
```

Token Request API

To request an access token from the OAuth server, use the following API:

Name

/access_token

Description

Processes an access token request.

Access Token Request

The client makes a request to the token endpoint by sending the following parameters using the application/x-www-form-urlencoded format with a character encoding of UTF-8 in the HTTP POST request:

Access Token Request Parameters

grant_type	Required. The value must be set to authorization_code.
code	Required. The authorization code received from the authorization server.
redirect_uri	Required, if the redirect_uri parameter was included in the authorization request. The value must match the redirect_uri value sent in the authorization request.
client_id	Required, if the client is not authenticating with the authorization server.
scope	Optional. Specifies the scope of the access request.

Access Token Request Example

To request an access token for the authorization code flow, the client makes the following HTTP POST request using TLS:

```
POST /asg/oauth2/access_token HTTP/1.1
```

```
Host: server.example.com
```

```
Authorization: Basic czZCaGRSa3F0MzpnWDFmQmF0M2JW
```

```
Content-Type: application/x-www-form-urlencoded
grant_type=authorization_code&code=Sp1xl0BeZQQYbYS6WxSbIA
&redirect_uri=https%3A%2F%2Fclient%2Eexample%2Ecom%2Fcb
```

Access Token Response

If the access token request is valid and authorized, the authorization server issues an access token and an optional refresh token. The refresh token may be returned only for authorization code flow.

Access Token Response Example

The following is a successful response:

```
HTTP/1.1 200 OK
Content-Type: application/json; charset=UTF-8
Cache-Control: no-store
Pragma: no-cache
{
  "access_token": "2YotnFZFEjr1zCsicMWpAA",
  "token_type": "example",
  "expires_in": 3600,
  "refresh_token": "tGzv3JOkF0XG5Qx2TlKWIA",
  "example_parameter": "example_value"
}
```

Access Token Request Error

If the request client authentication failed or is invalid, the authorization server returns an error response. The authorization server responds with an HTTP 400 status code (unless specified otherwise) and includes the following parameters with the response:

Access Token Error Parameter

Parameter	Description
error	<p>Specifies a single error code returned from the authorization server.</p> <p>Required.</p> <p>Refer to table Access Token Request Error Codes for the error codes.</p>

The following table lists the error codes for the error returned for an invalid token request:

Access Token Request Error Codes

Error Code	Description
invalid_request	
	The request is missing a required parameter, includes an unsupported parameter value (other than grant type), repeats a parameter, includes multiple credentials, utilizes more than one mechanism for authenticating the client, or is otherwise malformed.
invalid_client	
	Client authentication failed (e.g., unknown client, no client authentication included, or unsupported authentication method). The authorization server MAY return an HTTP 401 (Unauthorized) status code to indicate which HTTP authentication schemes are supported. If the client attempted to authenticate via the "Authorization" request header field, the authorization server MUST respond with an HTTP 401 (Unauthorized) status code and include the "WWW-Authenticate" response header field matching the authentication scheme used by the client.
invalid_grant	
	The provided authorization grant (e.g., authorization code, resource owner credentials) or refresh token is invalid, expired, revoked, and does not match the redirection URI used in the authorization request, or was issued to another client.
unauthorized_client	
	The authenticated client is not authorized to use this authorization grant type.
unsupported_grant_type	
	The authorization grant type is not supported by the authorization server.
invalid_scope	
	The requested scope is invalid, unknown, malformed, or exceeds the scope granted by the resource owner.
error_description	
	Optional. Human-readable ASCII [USASCII] text providing additional information, used to assist the client developer in understanding the error that occurred.
error_uri	
	Optional. A URI identifying a human-readable web page with information about the error, used to provide the client developer with additional information about the error.

Access Token Error Example

The following is an example of the error response for an access token request:

```
HTTP/1.1 400 Bad Request
Content-Type: application/json;charset=UTF-8
Cache-Control: no-store
Pragma: no-cache
{
  "error": "invalid_request"
}
```

Token Validation API

To validate an access token issued by the authorization server of TIBCO API Exchange Gateway, use the following API:

Name

/access_token

Description

Validates the access token and returns the token information.

Token Validation Request

To send a validate access token request, use the following parameters to the query component of the authorization endpoint URI using the "application/x-www-form-urlencoded" format in the HTTP GET request:

Token Validation API Parameters

Parameter	Description
access_token	Specifies the access_token issued by the authorization server. Required.

Token Validation Request Example

To validate an access token issued by the authorization server, send an HTTP GET request using TLS with the following parameter:

Send the following HTTP request using TLS:

```
GET /asg/oauth2/access_token/2YotnFZFEjr1zCsicMWpAA HTTP/1.1
Host: server.example.com
```

Token Validation Response

If the access token request is valid and authorized, the resource server returns the data associated with the access token, which can be used to query the resources.

Token Validation Response Parameters

Parameter	Description
access_token	Specifies the access_token issued by the authorization server. Required.
resource_uri	Optional, if identical to the scope requested by the client; otherwise, Required.
access_token	Optional, if identical to the scope requested by the client; otherwise, Required.

Token Validation Response Example

The successful response is as follows:

```
HTTP/1.1 200 OK
Content-Type: application/json; charset=UTF-8
Cache-Control: no-store
Pragma: no-cache
{
  "access_token": "2YotnFZFEjr1zCsicMWpAA",
  "expires_in": 3600,
  "resource_uri": "https://fairlawn.bookclub.org/member/friend",
  "scope": "friends"
}
```

Token Validation Error

The authorization server responds with an HTTP 400 (Bad Request) status code (unless specified otherwise) and includes the following parameters with the response:

Token Validation Error Parameters

Parameter	Description
error	<p>Specifies a single error code returned from the authorization server.</p> <p>Required.</p> <p>Refer to table Token Validation Error Codes for the error codes.</p>

Token Validation Error Codes

Error Code	Description
invalid_request	
	The request is missing a required parameter, includes an unsupported parameter value (other than grant type), repeats a parameter, includes multiple credentials, utilizes more than one mechanism for authenticating the client, or is otherwise malformed.
invalid_grant	
	The provided authorization grant (e.g., authorization code, resource owner credentials) or refresh token is invalid, expired, revoked, does not match the redirection URI used in the authorization request, or was issued to another client.
error_description	
	Optional. Human-readable ASCII [USASCII] text providing additional information, used to assist the client developer in understanding the error that occurred.
error_uri	
	Optional. A URI identifying a human-readable web page with information about the error, used to provide the client developer with additional information about the error.



Values for the `invalid_grant`, `error_description`, and `error_uri` parameters **must not** include characters outside the set `%x20-21 / %x23-5B / %x5D-7E`.

Token Validation Error Example

The error response is a HTTP response using the "application/json" media type as follows:

```
HTTP/1.1 400 Bad Request
```

```

Content-Type: application/json;charset=UTF-8
Cache-Control: no-store
Pragma: no-cache
{
  "error": "invalid_grant"
}
```

```
}
```

Retrieve Access Token

The authorization server maintains the list of tokens that it has issued. The tokens can be retrieved using the REST API.

The following are the REST APIs to retrieve the tokens:

Retrieving all tokens

```
GET /asg/oauth2/access_token/owners HTTP/1.1
Authorization: Basic czZCaGRSa3F0MzpnWDFmQmF0M2JW
Host: server.example.com
```

Retrieving tokens for specific owner

```
GET /asg/oauth2/access_token/{ownerID} HTTP/1.1
Authorization: Basic czZCaGRSa3F0MzpnWDFmQmF0M2JW
Host: server.example.com
```

Retrieve Token Response Example

The response of the API is list of tokens in JSON format. The example of 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"
}
```

```
{
...
}
]
```

Revoke Token API

Send a revoke token request to remove the client permissions associated with a valid token.

Name

/access_token

Description

Revoke the permissions associated with a valid access token.

Revoke Token Request

To send a request to revoke an access, use the following parameters to the query component of the authorization endpoint URI using the "application/x-www-form-urlencoded" format:

Token Validation Request Parameters

Parameter	Description
access_token	Specifies the access_token issued by the authorization server. Required.

Revoke Token Request Example

Send the following HTTP DELETE request to revoke an access token:

```
DELETE /asg/oauth2/access_token/{tokenID} HTTP/1.1
Host: server.example.com
Authorization: Basic czZCaGRSa3F0MzpnWDFmQmF0M2JW
Content-Type: application/x-www-form-urlencoded
```

OAuth Service Provider Interfaces

Overview of OAuth service provider interfaces.

This section describes the service provider interfaces supported by the TIBCO API Exchange Gateway OAuth server.

The OAuth server is capable of extending the OAuth authorization processing via Java-based service provider interfaces. The service provider interface implementations are implemented in plug-ins which are loaded during startup. Using this functionality, you can implement custom adapters.

The following service provider interfaces are supported:

- Owner service provider interface

- Client service provider interface
- Scope service provider interface

Owner Service Provider Interface

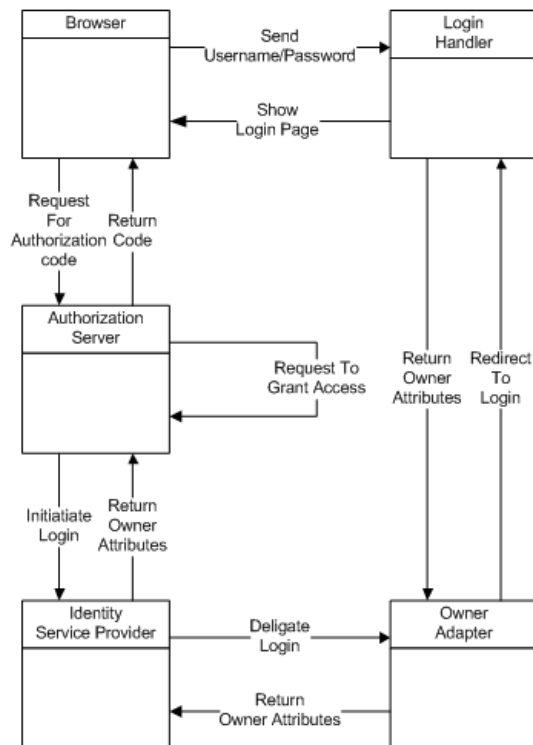
Provides an interface to authenticate the resource owner.

The owner service provider interface is used by the OAuth server to authenticate the resource owner during the authorization code flow, and to obtain attributes of the resource owner. In addition to authenticate the owner, the owner service provider interfaces are responsible for redirecting the client applications to login and access grant page.

Owner Service Provider Interface (SPI) Flow

The following figure illustrates the flow for owner service provider interface.

Owner SPI Flow



Owner Service Provider Interface (SPI) Java API

The following is the Java API of the owner service provider interface:

```

/**
 * 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

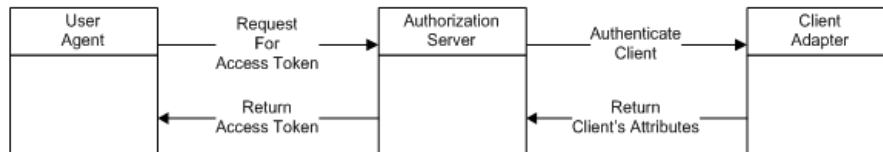
Provides an interface to authenticate a client.

The client service provider interface is used by the OAuth server to authenticate a client when the client is requesting the access token during the 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

The following figure illustrates the flow for client service provider interface.

Client SPI Flow



Client Service Provider Interface (SPI) Java API

The following is the Java API of the client service provider interface:

```

/**
 * 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.
     */
    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 follows:

Required Attributes For getAttributes

Attribute	Description
dn	Specifies the distinguish name for the client
oidcClientID	Specifies the unique ID for the client
oidcClientRedirectURI	Specifies the redirect or callback URL
scopes	Specifies a list of comma separated scopes

Sample attributes:

The following table shows the sample attributes:

Attribute	Sample Value
oidcClientID	playground
dn	oidcClientIDplayground
ou	client
dc	tibco
scopes	public playground
oidcClientRedirectURI	https://redirectHost/site/client/redirectEndpoint
oidcAppName	Playground
oidcAppLogoURL	https://redirectHost/site/client/logo
oidcClientEmail	playground.asg@tibco.com

Scope Service Provider Interface

Provides an interface to retrieve the scope details.

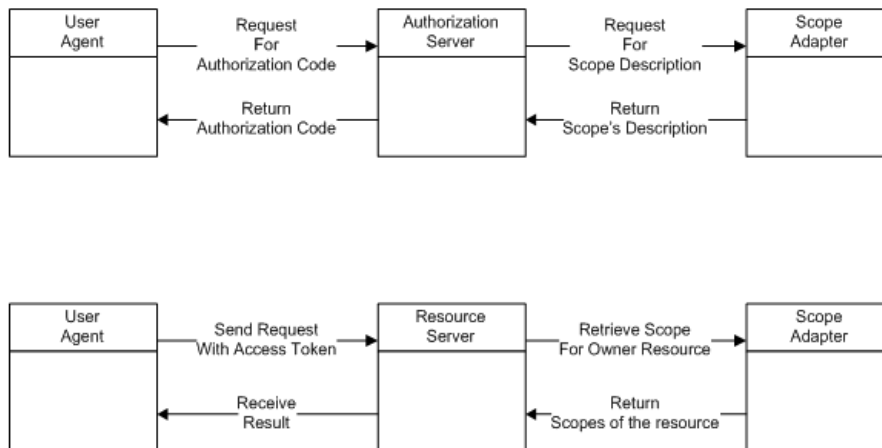
The scope service provider interface is used by the OAuth authorization server for the following actions:

- Retrieve the description of the scopes
- Retrieve the scopes for a specified owner and resource to access

Scope Service Provider Interface (SPI) Flow

The following figure [Scope SPI Flow](#) illustrates the flow for scope service provider interface.

Scope SPI Flow



Scope Service Provider Interface (SPI) Java API

The following is the Java API of the scope (token) service provider interface:

```

/**
 * ScopeAdapter is the interface use 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 descriptons 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;
}

```


Deploying Custom Adapters

Use the custom adapters to implement the service provider interfaces.

The custom adapters implement the service provider interfaces for OAuth server. To enable the OAuth server to use the custom adapters, deploy TIBCO API Exchange Gateway on the same machine where the Identity service provider of the OAuth server runs.

To deploy the custom adapters, follow these steps:

Procedure

1. Copy the Adapters Jar Files.

Copy the jars containing the implementation class of the adapters and their dependencies to `ASG_HOME/lib/ext/tpcl` directory.

2. Set OAuth Adapter Properties

Set the OAuth adapter properties using the Config UI. OAuth adapter properties settings are described in the [Properties For OAuth Adapters](#) section..



- You can set the OAuth server adapter properties in the `ASG_CONFIG_HOME/asg.properties` file. Refer to [Properties For OAuth Adapters](#) for properties details.
- If the custom adapters use external resources and require configuration, the custom adapters will need to obtain the configuration accordingly.

For example, the file adapters for owner, client and scope use the following resource files containing the configuration and are found in `ASG_HOME/examples/OAuth/resources` directory :

- `clients.properties`
- `owners.properties`
- `scopes.properties`

Default Adapters

By default, TIBCO API Exchange Gateway provides the following adapters:

Client Adapter

List of default client adapters.

Out of the box, the following client adapters are supported:

- File
- Portal Engine of TIBCO API Exchange Manager

Owner Adapter

List of default owner adapters.

Out of the box, the following owner adapters are supported:

- File
- LDAP

Scope Adapter

List of default scope adapters.

Out of the box, the following owner adapters are supported:

- File

Gateway Management Features

Overview of Gateway management components.

This section explains the functionality of TIBCO API Exchange Gateway management components. It consists of following topics:

- Central Logger
- Central Logger Database
- Global Throttle Manager
- Cache Cleanup Agent

Central Logger

This section explains the functionality of Central Logger component of TIBCO API Exchange Gateway.

Overview

The Central Logger component provides the centralized logging. Each instance of the Core Engine 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. The Core Engine instance aggregates events in memory and publishes the messages at configured intervals in order to reduce the disk load during high transaction rates.

The Central Logger communicates with the Core Engine using the following transports:

- Rendezvous (RV): This is the default transport.
- Java Message Service (JMS). See [Enabling JMS Channel for Central Logger](#).
- ActiveSpaces (AS). See [Enabling AS Transport](#).

Pre-requisites

The Central Logger component audit logs the transactions in a database. Make sure to configure and setup a database server for the functionality of the Central Logger component. See [Setup Database and Configuring Central Logger](#)

Database Setup and Configuration for Central Logger

This section explains the steps required to configure the database and the database drivers for the Central Logger component of TIBCO API Exchange Gateway.

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 the Central Logger component.

Task A Setup Database Driver

- Copy the appropriate JDBC driver jar file to the *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-5.1.22-bin.jar
```

For example, database jar file for Oracle database is:

```
ojdbc6.jar
```

For example, database jar for **SQL Server** database is:

```
sqljdbc4.jar
```

For example, database jars for **DB2** database is:

```
db2jcc.jar  
db2jcc_license_cu.jar
```

Task B Creating a Database

Depending on 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, DB2 and MS SQL Server database server, ask your database administrator to create a database such as asgstat for the Central Logger component of TIBCO API Exchange Gateway.

This section lists the steps to create a database for testing purposes using MySQL database server. You must contact your database administrator to use appropriate database for production requirements.

Procedure

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.



- Refer to <http://dev.mysql.com/doc/refman/5.5/en/creating-database.html> link for more information.

Task C Creating 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, SQL Server and DB2 database server, work with your database administrator to create a database user in the appropriate database.

Procedure

1. Verify that you are connected to the MySQL database server as a root user. If not, type the command:

```
mysql -u root -p
```
2. Type the following command at the mysql command prompt to create a new user:

```
create user 'asguser' identified by 'asgpass';
```

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.

- **For DB2 Database**

Ask your database administrator to create a database user (for example asguser) and grant ALL, CONTROL, CONNECT, insert, SELECT privileges to this user.

Task D Setting up the Database Schema

This section explains the steps to create the database tables in the database required for the Central Logger component.

For MySql Database

Procedure

1. Navigate to the following directory:

```
ASG_HOME/templates/database/mysql
```

2. Type the following command at the command prompt:

```
mysql -D asgstat -u asguser -pasgpass < createAsgTransactions.sql
mysql -D asgstat -u asguser -pasgpass < createAsgKpis.sql
```

For Oracle Database

Procedure

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

Procedure

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
```

For DB2 Database

Procedure

1. Navigate to the following directory:

```
ASG_HOME/templates/database/db2
```

2. Type the following command on DB2 console:

```
db2 -tvvf createAsgTransactions.sql
db2 -tvvf createAsgKpis.sql
```

Task E Setting up the Database Connection Parameters

This section explains the steps required to setup the parameters to connect to the database.

Procedure

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, the values are shown for MySQL database as follows:

```
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, the values are shown for Oracle database as follows:

```
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, the values are shown for SQL Server database as follows:

```
tibco.clientVar.CL/Database/
Driver=com.microsoft.sqlserver.jdbc .SQLServerDriver
tibco.clientVar.CL/Database/Url=jdbc:sqlserver://localhost:
1433 ;databaseName=asgstat
tibco.clientVar.CL/Database/Username=asguser
tibco.clientVar.CL/Database/Password=asgpass
tibco.clientVar.CL/Database/Schema=dbo
```

For example, the values are shown for DB2 database as follows:

```
tibco.clientVar.CL/Database/Driver=com.ibm.db2.jcc.DB2Driver
tibco.clientVar.CL/Database/Url=jdbc:db2://localhost:50000/asgstat
```

```
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, as required.

Runtime Properties For Central Logger

See [Runtime Properties of Central Logger](#) for details.

Enabling Reporting to the Central Logger

By default, the reporting is not enabled to the Central Logger. Enable the reporting as follows:

Procedure

1. Open the `ASG_CONFIG_HOME/asg.properties` file for editing.
2. Search 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 changes in the file and close the editor.



When the detail level logging is enabled for the Central Logger, the Core Engine logs the complete HTTP headers (such as Client IP, Host), SOAPAction, and Service URI of the request message into the `ASG_TRANSACTIONS_DETAILS` database table. The SOAPAction, Service URI and headers information is stored in the `DET_SOAPACTION`, `DET_SERVICEURI`, `DET_HTTPHEADERS` columns respectively.

Running the Central Logger

This section explains the steps to run the Central Logger.

Procedure

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_config_name
```

For example: `asg-engine -u asg-cl -a APIExchange`



Before you run the Central Logger, confirm that the Core Engine is running in Cache Agent enabled mode. This means that the Core Engine is running with `asg-caching-core` processing unit.

Central Logger Database

The Central Logger stores the transactions and transaction details in the database tables.



When the database is down, there is a failure to store the data in the EMS message queues until the Central Logger establishes communication with the database.

Database Tables

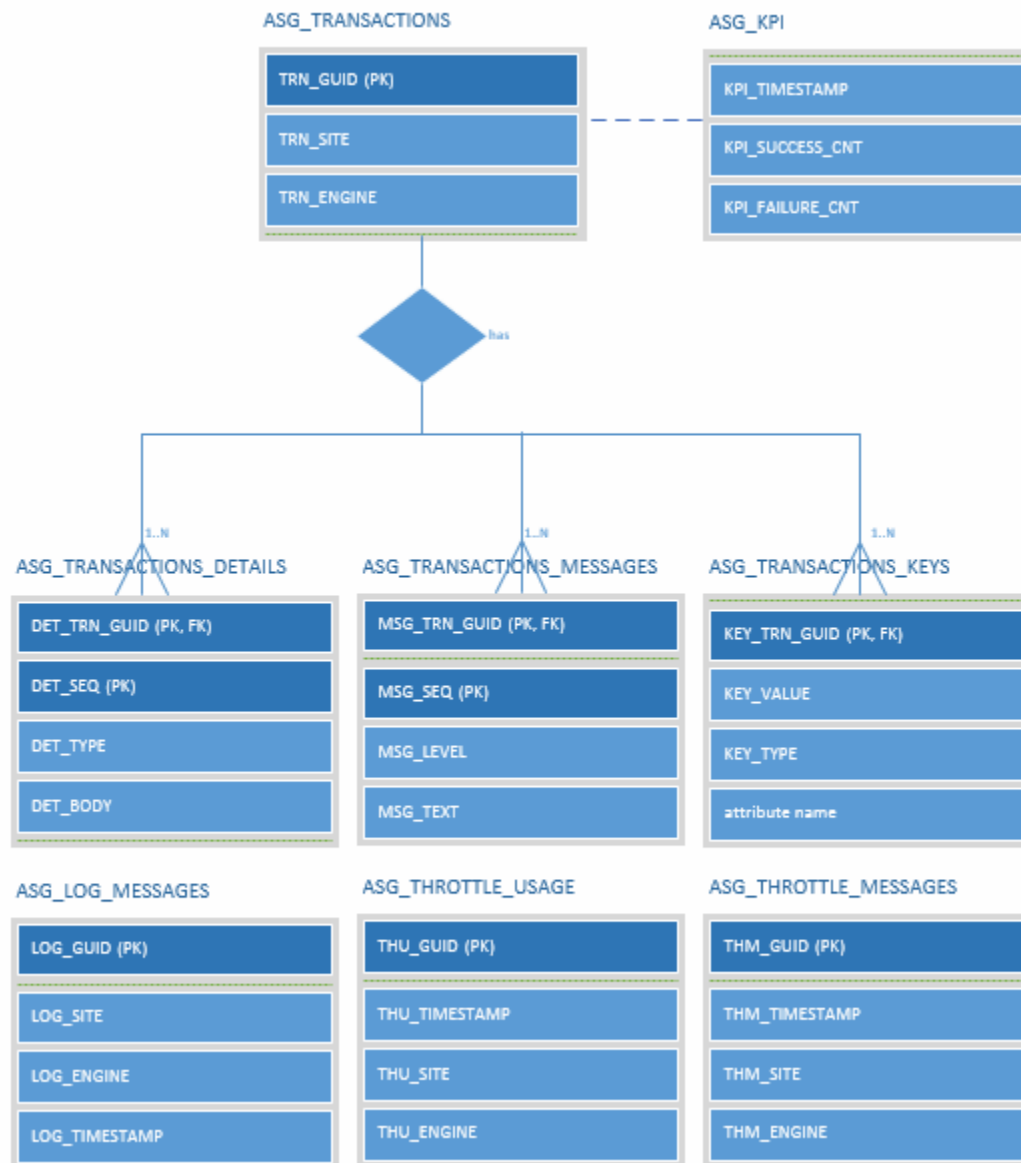
The database schema includes the following tables:

- ASG_TRANSACTIONS
- ASG_TRANSACTION_DETAILS
- ASG_TRANSACTION_MESSAGES
- ASG_TRANSACTION_KEYS
- ASG_THROTTLE_MESSAGES
- ASG_THROTTLE_USAGE
- ASG_KPI
- ASG_LOG_MESSAGES

Schema Details

The following figure shows the schema details for the Central Logger database.

Central Logger Database Schema



ASG_TRANSACTIONS Table

The ASG_TRANSACTIONS table contains the basic statistics such as facade operation name, target operation name, routing key, the name of the Core Engine for each request. The Central Logger inserts one transaction record for each request in this table.

Columns of ASG_TRANSACTION Table

Field Name	Description
TRN_GUID	<p>The primary key as an unique reference for the transaction.</p> <p>GUID is generated automatically by the Central Logger component.</p>
TRN_TIMESTAMP	The time of the request.
TRN_SITE	<p>The site name which accepts the request. You can use different gateway sites or environments for TIBCO API Exchange Gateway product.</p> <p>The value for the TRN_SITE field is populated from the Common/Deployments/SiteNumber global variable defined in the <i>ASG_CONFIG_HOME/asg_cl.properties</i> file. The default value of Common/Deployments/SiteNumber is 1.</p> <p>Note: For 1.2.x release, the value for the TRN_SITE field is populated from the CL/Logging/siteNumber global variable defined in the <i>ASG_CONFIG_HOME/asg_cl.properties</i> file.</p>
TRN_ENGINE	<p>The name of the engine which processes the transaction request.</p> <p>You can assign different engine names when starting the Gateway core engine. The engine name can be specified using -n parameter to asg-engine command on the command line.</p>
TRN_SOURCE	The partner name, such as anon, who is the requestor of the operation.
TRN_TARGET	The value returned from the OpCo field of the parsing step of the request processing.
TRN_FACADE_OPERATION	The facade operation name.
TRN_FACADE_SERVICE	The facade service name. Not currently used, might be used in future.
TRN_ROUTING_KEY	The evaluated routing key used to determine the target operation.
TRN_TARGET_OPERATION	The target operation name or the target operation group.
TRN_TARGET_SERVICE	The target service name. Not currently used, might be used in future.
TRN_TRANSACTION_ID	The transaction id passed with the request.

Field Name	Description
TRN_FACADE_I_TIME	The ingress (reception) time (in milliseconds) for the facade operation message.
TRN_ROUTER_I_TIME	The egress (emission) time (in milliseconds) for the first target operation message.
TRN_FACADE_E_TIME	The egress (emission) time (in milliseconds) for the northbound message.
TRN_ROUTER_E_TIME	The (reception) time (in milliseconds) for the last target operation message.
TRN_FACADE_DURATION	The total time (in milliseconds) spent in the gateway.
TRN_ROUTER_DURATION	The total time (in milliseconds) waiting for the response from the back-end service (target operation).
TRN_STATUS	The completion status of the transaction.
TRN_ERROR_CODE	The error code for the transaction, if the transaction finished with error.
TRN_ERROR_MESSAGE	The error message for the transaction, if the transaction finished with error.
TRN_APP_ID	The application ID.
TRN_APP_NAME	The name of the application.
TRN_PRODUCT_ID	The product ID.
TRN_PRODUCT_NAME	The name of the product.
TRN_PARTNER_ID	The ID of the partner.
TRN_CORRELATION_ID	The correlation id of the request message.
TRN_REQUEST_PAYLOAD_SIZE	
TRN_RESPONSE_PAYLOAD_SIZE	

ASG_TRANSACTION_DETAILS Table

The ASG_TRANSACTION_DETAILS table contains the transaction details of the request and response messages exchanged between the facade operation and target operation of the back-end service. The Central Logger inserts six detail rows for each transaction request.



The ASG_TRANSACTION_DETAILS table is populated only when you enable the detail logging. The detail level logging is set in the `ASG_CONFIG_HOME/asg.properties` file, using the following property:

```
tibco.clientVar.ASG/Logging/clLogLevel
```

Columns of ASG_TRANSACTION_DETAILS Table

Field Name	Description
DET_TRN_GUID	The reference to the transaction in the ASG_TRANSACTIONS table.
DET_SEQ	The sequence order number of the transaction.
DET_TYPE	The type of the message. The possible values are: <ul style="list-style-type: none"> • nb.request • cn.request • sb.request • sb.response • cn.response • nb.response
DET_BODY	The body of the request or response payload.
DET_SERVICEURI	The Service URI from the HTTP headers of request payload.
DET_SOAPACTION	The SOAPAction from the HTTP headers of request payload.
DET_HTTPHEADERS	The complete HTTP headers of request payload.
DET_PAYLOAD_SIZE	

ASG_TRANSACTION_MESSAGES Table

The ASG_TRANSACTION_MESSAGES table contains the record of log messages reported during the message processing.

Columns of ASG_TRANSACTION_MESSAGES Table

Field Name	Description
MSG_TRN_GUID	The reference to the transaction in the ASG_TRANSACTIONS table.
MSG_SEQ	The sequence number of the transaction log message.
MSG_TIMESTAMP	The time of the transaction log message.
MSG_LEVEL	The severity of the transaction log message.
MSG_TEXT	The text of the transaction log messages related to stages in the lifecycle.

ASG_TRANSACTION_KEYS Table

The ASG_TRANSACTION_KEYS table stores the reference data passed in the body of request message. An example is the customer reference data sent in the request message.

Columns of ASG_TRANSACTION_KEYS Table

Field Name	Description
KEY_TRN_GUID	The foreign key reference to the transaction in the ASG_TRANSACTIONS table.
KEY_VALUE	The transaction key name such as tel:4366412345678.
KEY_TYPE	The key type of the transaction message. The type of the transaction message is populated from the address element in the transformation xslt file specified for the parse output document.

ASG_THROTTLE_USAGE Table

The ASG_THROTTLE_USAGE table stores the usage information related to throttles and subscriptions.

Columns of ASG_THROTTLE_USAGE Table

Field Name	Description
THU_GUID	An unique ID of the throttle.
THU_TIMESTAMP	The timestamp of the throttle record entry.
THU_SITE	<p>The site name which accepts the request. You can use different gateway sites or environments for TIBCO API Exchange Gateway product.</p> <p>The value for the THU_SITE field is populated from the Common/Deployments/SiteNumber global variable defined in the ASG_CONFIG_HOME/asg_cl.properties file. The default value of Common/Deployments/SiteNumber is 1.</p>
THU_ENGINE	The name of the Core Engine which sends the throttle information. You can assign different engine names when starting the Core Engine. The engine name can be specified using -n parameter to asg-engine command on the command line.
THU_THROTTLE	The name given to the throttle for identification.
THU_MAX	The maximum threshold count of the throttle.
THU_CUR	The current usage count of the throttle.
THU_PCT	The current percentage usage of the throttle.

ASG_THROTTLE_MESSAGES Table

The ASG_THROTTLE_MESSAGES table stores the messages related to throttles.

Columns of ASG_THROTTLE_MESSAGES Table

Field Name	Description
THM_GUID	A unique IID of the throttle.
THM_TIMESTAMP	The timestamp of the throttle record entry.
THM_SITE	<p>The site name which accepts the request. You can use different gateway sites or environments for TIBCO API Exchange Gateway product.</p> <p>The value for the THM_SITE field is populated from the Common/Deployments/SiteNumber global variable defined in the <i>ASG_CONFIG_HOME/asg_cl.properties</i> file. The default value of Common/Deployments/SiteNumber is 1.</p>
THM_ENGINE	<p>The name of the Core Engine which sends the throttle information. You can assign different engine names when starting the Core Engine in any of the following ways:</p> <ul style="list-style-type: none"> • Use the "-n" parameter to asg-engine command on the command line. • Change the name of the PU in the <i>ASG_HOME/bin/asg_core.cdd</i> file.
THM_MESSAGE	<p>The text of the message related to the throttle. The message can be one of the following types:</p> <ul style="list-style-type: none"> • INFO • ERROR • DEBUG • WARN
THM_LEVEL	<p>The severity of the message related to the throttle. The severity of the message is populated based on the log level set in the <i>ASG_CONFIG_HOME/asg.properties</i> file using the following global variable:</p> <ul style="list-style-type: none"> • tibco.clientVar.ASG/Logging/ThrottleMessageLogLevel <p>The default value is 0.</p>

ASG_KPI Table

The ASG_KPI table stores information regarding the key performance indicators (KPI) of the transactions stored in the ASG_TRANSACTION table to support visualizations. The central logger flushes the records in regular time frames as per the KPI configuration. You should use the ASG_TRANSACTION table for historical data analysis.

Columns of ASG_KPI Table

Field Name	Description
KPI_TIMESTAMP	The timestamp.
KPI_SUCCESS_CNT	The count of successful transactions with status = OK.
KPI_FAILURE_CNT	The count of failed transactions with status = INVALID/FAIL.
KPI_TIMEOUT_CNT	The count of the timed out transactions.
KPI_ERROR_CNT	The count of the transactions with error as response.
KPI_TOTAL_CNT	The total number of transactions in a chosen period.
KPI_AVG_LATENCY	<p>The average time spent in the gateway, which measures the gateway overhead. The average KPI latency time for a transaction is calculated from the ASG_TRANSACTION table as below:</p> <p>(Façade E Time – Façade Ingress Time – Router Egress Time – Router Ingress Time)</p>
KPI_AVG_DURATION	<p>The average time between request and response message.</p> <p>The average KPI duration for a transaction is calculated from the ASG_TRANSACTION table as below:</p> <p>(Façade Egress Time – Façade Ingress Time)</p>
KPI_TYPE	<p>The KPI dimension. The possible values are:</p> <ul style="list-style-type: none"> • Engine • Operation • Partner • Service
KPI_ENTITY	Unique value for the KPI dimension.
KPI_FREQUENCY	The name of aggregation group such as kpi1m , kpi5m, kpi1h.

Field Name	Description
KPI_END_TIMESTAMP	Specifies the end interval for the KPI type.

ASG_LOG_MESSAGES Table

The ASG_LOG_MESSAGES table contains the log messages generated by the gateway core engine. When the detail level logging is enabled for the central logger, the ASG_LOG_MESSAGES table is populated.

Columns of ASG_LOG_MESSAGES Table

Field Name	Description
LOG_GUID	The unique ID of the message.
LOG_TIMESTAMP	The timestamp of the message.
LOG_SITE	<ul style="list-style-type: none"> The site name which processes the request. You can use different gateway sites or environments for TIBCO API Exchange Gateway product. The value for the TRN_SITE field is populated from the Common/Deployments/SiteNumber global variable defined in the <code>ASG_CONFIG_HOME/asg.cl.properties</code> file. The default value of Common/Deployments/SiteNumber is 1.
LOG_ENGINE	The name of the Core Engine which generates the message. You can assign different engine names when starting the Core Engine. The engine name can be specified using -n parameter to asg-engine command on the command line.
LOG_MESSAGE	The text of the transaction log message.
LOG_LEVEL	The severity of the log message. The severity of the message is mentioned by the level of logging set in <code>ASG_CONFIG_HOME/asg.properties</code> file. The possible values for the level of logging are 0,1,2,3,4.

Write Transactions Data to File

TIBCO API Exchange Gateway has the ability to log the transactions data of facade operations to a file.

By default, TIBCO API Exchange Gateway stores the transactions data of the facade operations in the ASG_TRANSACTIONS database table. You can write the transactions data to a file instead of logging to the database table when the detail level logging is enabled.

To write the transactions of facade operation in a file, you must specify the facade operation name in the `tibco.clientVar.CL/Logging/fileFilter` property. See [Enabling Transaction Data to a File](#) to configure logging of transactions data to a file.

By default, TIBCO API Exchange Gateway write the transactions data of the following operations (as separated by | character) to the file. These operations are internal to gateway and do not have to be stored in a database.

```
test|ping|updateConfiguration|addConfiguration|deleteConfiguration|getConfiguration|
getConfigurationStatus|publishConfiguration|unlockConfiguration|
restoreConfiguration|CentralLoggerRuleFunctionOp|portalEventResource|
analyticsResource|apiKeyResource|oauthClientAuthenticateResource|
oauthClientAttributesResource|loadResourceListing|LogLevelRuleFunctionOp
```



The Central Logger uses the regular expression search pattern to match the facade operation name in the `tibco.clientVar.CL/Logging/fileFilter` property. For any facade operation, the operation name must not contain any string which matches the facade operation name in the `tibco.clientVar.CL/Logging/fileFilter` property.

For example, When a facade operation name is configured, this should not contain `test` or `ping` as they are listed in the `fileFilter` property.

Enabling Transaction Data to a File

Configure the properties to write the transaction data of facade operations to a file instead of database.

Prerequisites

The detail level logging must be enabled for the Central Logger.

Procedure

1. Verify that the detail level logging is enabled for the Central Logger, as follows:
 - a) Navigate to `ASG_CONFIG_HOME` directory.
 - b) Open the `asg.properties` file.
 - c) Ensure that the following property is set to `true`. If not, set the value of the property to `true`.

```
tibco.clientVar.ASG/Logging/reportingEnabled=true
```

- d) Save any changes to the file.
2. Navigate to `ASG_CONFIG_HOME` directory.
3. Open the `asg_cl.properties` file for editing.
4. Set the following properties:

```
tibco.clientVar.CL/Logging/fileFilter
tibco.clientVar.CL/Logging/files/directory
tibco.clientVar.CL/Logging/files/transactions
tibco.clientVar.CL/Logging/files/maxcount
tibco.clientVar.CL/Logging/files/maxsize
```



- Refer to [Runtime Properties of Central Logger](#) for the description of the properties.
- To configure the properties using the Config UI, refer to the **Central Logger** section of [Monitoring Properties](#).
- See [Enabling Transaction Data to a File](#) for example values of the properties.

5. Save changes to the file.

Example Values

```
tibco.clientVar.CL/Logging/fileFilter=test|ping|updateConfiguration|
addConfiguration|deleteConfiguration

tibco.clientVar.CL/Logging/files/directory=C:/TIBCO_HOMECONFIGAPIX220/tibco/
cfgmgmt/asg/logs

tibco.clientVar.CL/Logging/files/transactions=trans_log.txt

tibco.clientVar.CL/Logging/files/maxcount=3

tibco.clientVar.CL/Logging/files/maxsize=5000000
```

Format of Transaction Data Log File

This section lists the format of the output log file when the transactions data of facade operations are logged to a file.

The Core Engine writes the transactions data in the following format:

```
TRN_TIMESTAMP,      extID,      TRN_GUID,      TRN_SITE,      TRN_ENGINE,      TRN_SOURCE,
TRN_TARGET,         TRN_FACADE_OPERATION,      TRN_FACADE_SERVICE,      TRN_ROUTING_KEY
TRN_TARGET_OPERATION,      TRN_TARGET_SERVICE,      TRN_TRANSACTION_ID,
TRN_FACADE_I_TIME,      TRN_ROUTER_I_TIME,      TRN_ROUTER_e_TIME,
TRN_FACADE_E_TIME,      TRN_FACADE_DURATION,      TRN_ROUTER_DURATION,
TRN_STATUS,         TRN_ERROR_CODE,      TRN_ERROR_MESSAGE
```

Sample Transaction Data Log File

The following sample output log file contains the transaction data of the ping facade operation:

```
20150730 11:15:41,null,b43711f2-5b0f-4209-895b-a2f18f74ac13,1,asg-
caching-
core,anon,noop.ping,internal_ping,RESERVED,default,service_noop.ping,
RESERVED,null,
1438235141195,1438235141295,1438235141296,1438235141340,145,1,ok,null
,null
```

Recording Error Events to Central Logger

The Central Logger consumes the event messages published by the Core Engine and stores them in the database for reporting purposes.

You can enable the Core Engine to publish all events, including successful and failed transactions, to the Central Logger by setting the `tibco.clientVar.ASG/Logging/errorReportingEnabled` property in the `ASG_CONFIG_HOME/asg.properties` file.

With TIBCO API Exchange Gateway, you can also selectively publish the error events to the Central Logger and store the failed transactions. This helps to reduce the volume of transactions stored in the Central Logger database.

Publishing Error (Failed) Transactions

To selectively publish the error transactions to the Central Logger, follow these steps:

Procedure

1. Open the `ASG_CONFIG_HOME/asg.properties` file for editing.
2. Set the properties as shown below:

```
tibco.clientVar.ASG/Logging/errorReportingEnabled=true
```

```
tibco.clientVar.ASG/Logging/reportingEnabled=false
```

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

3. Save the file and close the editor.

The failed transactions are stored in the following tables:

- ASG_TRANSACTIONS
- ASG_TRANSACTIONS_DETAILS
- ASG_TRANSACTIONS_KEYS
- ASG_TRANSACTIONS_MESSAGES

If the detail level logging is disabled (that is, `tibco.clientVar.ASG/Logging/clLogLevel` property is set as 0 in `ASG_CONFIG_HOME/asg.properties` file), then the failed transactions are stored in the following tables:

- ASG_TRANSACTIONS
- ASG_TRANSACTIONS_KEYS



The `tibco.clientVar.ASG/Logging/reportingEnabled` property takes precedence over `tibco.clientVar.ASG/Logging/errorReportingEnabled` property. Therefore, if you set `tibco.clientVar.ASG/Logging/reportingEnabled` property as true, the Core Engine publishes all types of the messages, and the Central Logger records both success and failure type transactions.

Correlation ID

TIBCO API Exchange Gateway supports correlation ID to help the users in auditing and debugging the message transactions. The correlation ID is used to group the related message requests and responses.

For HTTP, the correlation header can be configured by setting the `tibco.clientVar.ASG/Request/CorrelationHeaderName` property in the `asg.properties` file. By default, it is set to `X-Request-ID`.

For JMS and ESB, the correlation header is the `JMSCorrelationId` header.

When the `NorthBoundRequest` protocol and the `SouthBoundResponse` protocol are different, an XSLT needs to be used to map the incoming Correlation ID header to the outgoing Correlation ID header. For example, When an HTTP request comes into TIBCO API Exchange Gateway with `X-Request-ID` header, the headers value should be mapped onto a `JMSCorrelationId` header when the outgoing request from TIBCO API Exchange Gateway is JMS or ESB. A similar transformation needs to be performed when the `NorthBound` protocol is ESB/JMS and the `SouthBound` protocol is HTTP.

When the detail level logging is enabled for the Central Logger, and once the successful roundtrip of the request is completed, the correlation ID is logged in the Central Logger database (in the `TRN_GUID` column of `ASG_TRANSACTIONS` table) for analysis purposes.



- If the correlation ID is not set in the incoming request, TIBCO API Exchange Gateway generates an ID dynamically and assigns to the request.
- The correlation ID is forwarded as the HTTP header to the target operation.
- The correlation ID is added to the header of response message from the target operation.

Setting Correlation ID for HTTP Header

You can change the default name of the correlation ID for HTTP header.

Procedure

1. Navigate to the `ASG_CONFIG_HOME` directory.

2. Edit the `asg.properties` file.
3. Set the following property:

```
tibco.clientVar.ASG/Request/CorrelationHeaderName
```

For example, if you set the property, as follows:

```
tibco.clientVar.ASG/Request/CorrelationHeaderName=X-Request-Correlation-ID, the
HTTP header name is set as X-Request-Correlation-ID in the incoming request.
```

4. Save changes to the file.



You must restart the Core Engine after saving the changes to the property.

Enable JMS Channel for Central Logger

The Central Logger component of the TIBCO API Exchange Gateway receives the messages from the 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 Core Engine.

By default, the messages from the 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 Core Engine to Central Logger.

When the Rendezvous transport is used by the 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 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 Core Engine and the Central Logger.

Enabling JMS Channel for Central Logger

You must enable EMS channel to use JMS transport for Central Logger in the `asg_coe.cdd` and `asg_cl.cdd` files.

By default, the JMS channel for the Central Logger is disabled using the property defined in the `asg_core.cdd` and `asg_cl.cdd` files.

To enable the JMS channel for the Central Logger, follow these steps:

Procedure

1. Navigate to the `ASG_HOME/bin` directory.
2. Open the `asg_core.cdd` file in a text editor and edit it as follows:
 - a) Locate the **`be.channel.deactivate`** property as follows:

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

<property name="be.channel.deactivate" value="/Common/Channel/AS,/
DefaultImplementation/Channels/OAuthWebappsHTTPSChannel,/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,/Common/Channel/CentralLoggerJMS"/>
```

- b) If you have not taken the backup of the **be.channel.deactivate** property, copy the **be.channel.deactivate** property to **be.channel.deactivate.backup**, as follows:

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

<property name="be.channel.deactivate.backup" value="/Common/Channel/AS,/
DefaultImplementation/Channels/OAuthWebappsHTTPSChannel,/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,/Common/Channel/CentralLoggerJMS"/>
```

- c) Remove the following entry from the **be.channel.deactivate** property value:

```
/Common/Channel/CentralLoggerJMS
```

- d) Ensure that the value of **be.channel.deactivate** property is as follows:

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

<property name="be.channel.deactivate" value="/Common/Channel/AS,/
DefaultImplementation/Channels/OAuthWebappsHTTPSChannel,/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"/>
```

- e) Save the changes to the file.

3. Open the `asg_cl.cdd` file in a text editor and edit it as follows:

- a) Locate the **be.channel.deactivate** property as follows:

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

<property name="be.channel.deactivate" value="/Common/Channel/AS,/ASG/Channels/
RvMappingChannel,/ASG/Channels/RvCacheableChannel,/ASG/Channels/modRV_Channel,/
Common/Channel/Channel,/DefaultImplementation/Channels/SouthboundEsb0Channel,/
DefaultImplementation/Channels/SouthboundEsb1Channel,/DefaultImplementation/
Channels/SouthboundEsb2Channel,/DefaultImplementation/Channels/
North_ESBChannel,/DefaultImplementation/Channels/North_HTTPChannel,/ASG/
Channels/SOAPJMSChannel,/ASG/Channels/SOAPJMSChannel_North,/ASG/Channels/
SOAPJMSChannel_South,/Common/Channel/CentralLoggerJMS,/DefaultImplementation/
Channels/OAuthWebappsChannel,/DefaultImplementation/Channels/
OAuthWebappsHTTPSChannel"/>
```

- b) If you have not taken the backup of the **be.channel.deactivate** property, copy the **be.channel.deactivate** property to **be.channel.deactivate.backup**, as follows:

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

<property name="be.channel.deactivate.backup" value="/Common/Channel/AS,/ASG/
Channels/RvMappingChannel,/ASG/Channels/RvCacheableChannel,/ASG/Channels/
modRV_Channel,/Common/Channel/Channel,/DefaultImplementation/Channels/
SouthboundEsb0Channel,/DefaultImplementation/Channels/SouthboundEsb1Channel,/
DefaultImplementation/Channels/SouthboundEsb2Channel,/DefaultImplementation/
Channels/North_ESBChannel,/DefaultImplementation/Channels/
North_HTTPChannel,/ASG/Channels/SOAPJMSChannel,/ASG/Channels/
SOAPJMSChannel_North,/ASG/Channels/SOAPJMSChannel_South,/Common/Channel/
CentralLoggerJMS,/DefaultImplementation/Channels/OAuthWebappsChannel,/
DefaultImplementation/Channels/OAuthWebappsHTTPSChannel"/>
```

- c) Remove the following entry from the **be.channel.deactivate** property value:

```
/Common/Channel/CentralLoggerJMS
```

- d) Ensure that the **be.channel.deactivate** property value looks as follows:

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

<property name="be.channel.deactivate" value="/Common/Channel/AS,/ASG/Channels/
RvMappingChannel,/ASG/Channels/RvCacheableChannel,/ASG/Channels/modRV_Channel,/
Common/Channel/Channel,/DefaultImplementation/Channels/SouthboundEsb0Channel,/
DefaultImplementation/Channels/SouthboundEsb1Channel,/DefaultImplementation/
Channels/SouthboundEsb2Channel,/DefaultImplementation/Channels/
North_ESBChannel,/DefaultImplementation/Channels/North_HTTPChannel,/ASG/
Channels/SOAPJMSChannel,/ASG/Channels/SOAPJMSChannel_North,/ASG/Channels/
SOAPJMSChannel_South,/DefaultImplementation/Channels/OAuthWebappsChannel,/
DefaultImplementation/Channels/OAuthWebappsHTTPSChannel"/>
```

- e) Save the changes to the file.

Setting JMS Transport for Central Logger

Set the JMS transport for the Central Logger in the asg.properties file as follows:

Procedure

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 Core Engine and Central Logger. Setting the value to JMS specifies that JMS transport is used for communication between the Core Engine and the Central Logger.

Configuring JMS Transport Properties

The [JMS Transport Properties](#) table shows the properties you must define to use the JMS transport for the Central Logger. The JMS transport properties are defined in the Core Engine properties asg.properties file as well as in the Central Logger properties asg_cl.properties file.

To set the JMS transport properties, follow these steps:

Procedure

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](#) 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](#) table.
7. Save the changes to the file.

JMS Transport Properties

Property	Description
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

Property	Description
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 TopicConnectionFactory object stored in JNDI. This object is used to create a topic connection with JMS server for the Central Logger. The default value is TopicConnectionFactory
tibco.clientVar.Common/Connections/JMS/ CL_JMSConnection/QueueConnectionFactoryName	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. The default value is QueueConnectionFactory
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/JMSUsername	Specifies the user name to use to authenticate to the JMS server.
tibco.clientVar.Common/Connections/JMS/ CL_JMSConnection/JMSPassword	Specifies the password to use to authenticate to the JMS server.
tibco.clientVar.Common/Connections/JMS/ CL_JMSConnection/TransactionReportDestinationName	Specifies the name of the JMS destination to which the transaction reports are sent to the Central Logger by the Core Engine. For example, asg.cl.transaction.queue

Property	Description
<code>tibco.clientVar.Common/Connections/JMS/CL_JMSConnection/TransactionReportDestinationType</code>	<p>Specifies the type of the JMS destination to which the transaction reports are sent to the Central Logger by the Core Engine.</p> <p>Possible values are queue or topic.</p> <p>The default value is queue</p> <p>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.</p>



You must restart the Core Engine and Central Logger instance after setting the properties in the `asg.properties` and `asg_cl.properties` file, if they are running.

Global Throttle Manager

Overview of Global Throttle Manager.

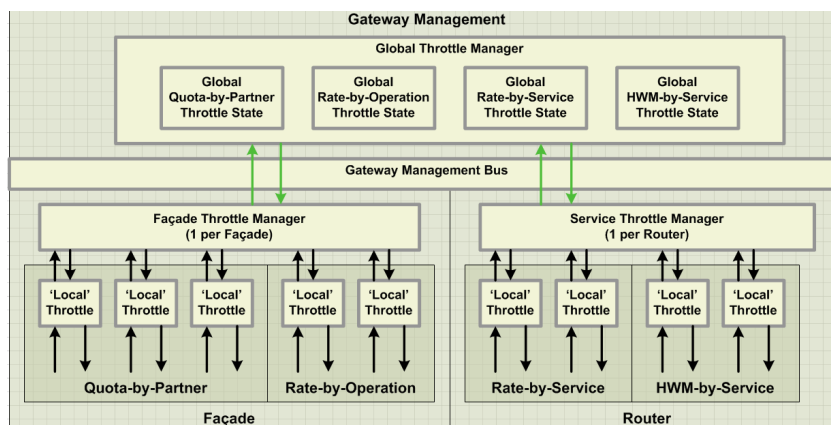
The Global Throttle Manager manages the Façade Throttle Manager and Service Throttle Manager. The Global Throttle Manager reports throttle usage to the Central Logger.

The Global Throttle Manager component maintains the state of all the global throttles in both Facades (Facade Throttles) and Routers (Service Throttles). The Global Throttle Manager exchanges the states of global throttles with active Facade Throttle Managers and Service Throttle Managers.

The Global Throttle Manager component provides the mechanism to evenly distribute the global throttles among TIBCO API Exchange Gatewayserver 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 Facade 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/ThrottleInterval)
```

- 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:

$\text{Ceiling}(0.4) \times \text{active engines} = 1 \times 20 = 20$

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.

Running Global Throttle Manager

This section explains the steps to run the Global Throttle Manager.

Procedure

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 Core Engine are running before you start the Global Throttle Manager.
- 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.

Enabling AS Transport

This section describes the ActiveSpaces transport communication for Central Logger and Global Throttle Manager.

Overview

TIBCO API Exchange Gateway supports an ActiveSpaces channel that enables the Global Throttle Manager and Central Logger to communicate with the Core Engine. If you set the ActiveSpaces transport for Global Throttle Manager, both the Central Logger and Global Throttle Manager communicate with the Core Engine using the ActiveSpaces channel.

When RV transport is configured for Global Throttle Manager, and the transport for the Central Logger is set as JMS, the Central Logger uses the JMS transport to log the transactions in the database.

Configuration

Configuration to enable ActiveSpaces transport.

To set the ActiveSpaces transport for Global Throttle Manager and Central Logger, follow these steps:

Procedure

1. Set Properties

Set the ActiveSpaces transport properties as follows:

- a) Start the Config UI, if not running.
- b) Log in to the Config UI using your credentials.
- c) On the home page of the Config UI, select the **Gateway Engine Properties** in the drop-down list.
- d) Click the **Transport** link.
- e) Set the ActiveSpaces transport properties as explained in the [Transport Properties](#) table.
- f) Save changes to the project configuration.

You can set the following runtime properties to enable ActiveSpaces channel in the *ASG_CONFIG_HOME/asg.properties* file:

```
tibco.clientVar.ASG/transport=AS
#Properties to connect to AS metaspace
#asLogLevel values: 0-7
#0 - INFO, 1 - WARN, 2 - ERROR, 3 - FATAL,
#4 - FINE, 5 - FINER, 6 - FINEST, 7 - NONE
tibco.clientVar.ASG/AS/MetaspaceName=APIXMS
tibco.clientVar.ASG/AS/DiscoveryUrl=tcp://127.0.0.1:13000
tibco.clientVar.ASG/AS/ListenUrl=tcp://127.0.0.1:13000-*/
tibco.clientVar.ASG/AS/asLogLevel=0
```

tibco.clientVar.ASG/AS/asLogDir=c:/tibcocfg/tibco/cfgmgmt/asg/logs

2. Activate AS Channel in asg_core.cdd File

Activate the AS channel, as follows:

- a) Navigate to the *ASG_HOME/bin* directory.
- b) Edit the *asg_core.cdd* file in a text editor.
- c) Search the following property:

```
<property name="be.channel.deactivate" value="/Common/Channel/AS,/
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,/Common/Channel/
CentralLoggerJMS"/>
```

- d) If you have not taken the backup of the **be.channel.deactivate** property, copy the **be.channel.deactivate** property to **be.channel.deactivate.backup**, as follows:

```
<property name="be.channel.deactivate.backup" value="/Common/Channel/AS,/
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,/Common/Channel/
CentralLoggerJMS"/>
```

- e) Remove **/Common/Channel/AS** from the value of the **be.channel.deactivate** property. The modified value of the property is as follows:

```
<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,/Common/Channel/CentralLoggerJMS"/>
```

- f) Save changes to the file.

3. Activate AS Channel in asg_cl.cdd File

Activate the AS channel as follows:

- a) Navigate to the *ASG_HOME/bin* directory.
- b) Edit the *asg_core.cdd* file in a text editor.
- c) Search the following property:

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

<property name="be.channel.deactivate" value="/Common/Channel/AS,/ASG/Channels/
RvMappingChannel,/ASG/Channels/RvCacheableChannel,/ASG/Channels/modRV_Channel,/
Common/Channel/Channel,/DefaultImplementation/Channels/SouthboundEsb0Channel,/
DefaultImplementation/Channels/SouthboundEsb1Channel,/DefaultImplementation/
Channels/SouthboundEsb2Channel,/DefaultImplementation/Channels/
North_ESBChannel,/DefaultImplementation/Channels/North_HTTPChannel,/ASG/
Channels/SOAPJMSChannel,/ASG/Channels/SOAPJMSChannel_North,/ASG/Channels/
SOAPJMSChannel_South,/Common/Channel/CentralLoggerJMS,/DefaultImplementation/
Channels/OAuthWebappsChannel,/DefaultImplementation/Channels/
OAuthWebappsHTTPSChannel"/>
```

- d) If you have not taken the backup of the **be.channel.deactivate** property, copy the **be.channel.deactivate** property to **be.channel.deactivate.backup**, as follows:

```
<property name="be.channel.deactivate.backup" value="/Common/Channel/AS,/ASG/
Channels/RvMappingChannel,/ASG/Channels/RvCacheableChannel,/ASG/Channels/
modRV_Channel,/Common/Channel/Channel,/DefaultImplementation/Channels/
SouthboundEsb0Channel,/DefaultImplementation/Channels/SouthboundEsb1Channel,/
DefaultImplementation/Channels/SouthboundEsb2Channel,/DefaultImplementation/
Channels/North_ESBChannel,/DefaultImplementation/Channels/
North_HTTPChannel,/ASG/Channels/SOAPJMSChannel,/ASG/Channels/
SOAPJMSChannel_North,/ASG/Channels/SOAPJMSChannel_South,/Common/Channel/
CentralLoggerJMS,/DefaultImplementation/Channels/OAuthWebappsChannel,/
DefaultImplementation/Channels/OAuthWebappsHTTPSChannel"/>
```

- e) Remove **/Common/Channel/AS** from the value of the **be.channel.deactivate** property. The modified value of the property is as follows:

```
<property name="be.channel.deactivate" value="/ASG/Channels/
RvMappingChannel,/ASG/Channels/RvCacheableChannel,/ASG/Channels/modRV_Channel,/
Common/Channel/Channel,/DefaultImplementation/Channels/SouthboundEsb0Channel,/
DefaultImplementation/Channels/SouthboundEsb1Channel,/DefaultImplementation/
Channels/SouthboundEsb2Channel,/DefaultImplementation/Channels/
North_ESBChannel,/DefaultImplementation/Channels/North_HTTPChannel,/ASG/
Channels/SOAPJMSChannel,/ASG/Channels/SOAPJMSChannel_North,/ASG/Channels/
SOAPJMSChannel_South,/Common/Channel/CentralLoggerJMS,/DefaultImplementation/
Channels/OAuthWebappsChannel,/DefaultImplementation/Channels/
OAuthWebappsHTTPSChannel"/>
```

- f) Save changes to the file.

4. Deactivate the Central Logger RV and JMS Channel in asg_cl.cdd File

- a) Navigate to the **ASG_HOME/bin** directory.
 b) Edit the **asg_cl.cdd** file in a text editor.
 c) Search the following property:

```
<property name="be.channel.deactivate" value="/Common/Channel/AS,/ASG/Channels/
RvMappingChannel,/ASG/Channels/RvCacheableChannel,/ASG/Channels/modRV_Channel,/
Common/Channel/Channel,/DefaultImplementation/Channels/SouthboundEsb0Channel,/
DefaultImplementation/Channels/SouthboundEsb1Channel,/DefaultImplementation/
Channels/SouthboundEsb2Channel,/DefaultImplementation/Channels/
North_ESBChannel,/DefaultImplementation/Channels/North_HTTPChannel,/ASG/
Channels/SOAPJMSChannel,/ASG/Channels/SOAPJMSChannel_North,/ASG/Channels/
SOAPJMSChannel_South,/Common/Channel/CentralLoggerJMS,/DefaultImplementation/
Channels/OAuthWebappsChannel,/DefaultImplementation/Channels/
OAuthWebappsHTTPSChannel"/>
```

- d) If you have not taken the backup of the **be.channel.deactivate** property, copy the **be.channel.deactivate** property to **be.channel.deactivate.backup**, as follows:

```
<property name="be.channel.deactivate.backup" value="/Common/Channel/AS,/ASG/
Channels/RvMappingChannel,/ASG/Channels/RvCacheableChannel,/ASG/Channels/
modRV_Channel,/Common/Channel/Channel,/DefaultImplementation/Channels/
SouthboundEsb0Channel,/DefaultImplementation/Channels/SouthboundEsb1Channel,/
DefaultImplementation/Channels/SouthboundEsb2Channel,/DefaultImplementation/
Channels/North_ESBChannel,/DefaultImplementation/Channels/
North_HTTPChannel,/ASG/Channels/SOAPJMSChannel,/ASG/Channels/
SOAPJMSChannel_North,/ASG/Channels/SOAPJMSChannel_South,/Common/Channel/
CentralLoggerJMS,/DefaultImplementation/Channels/OAuthWebappsChannel,/
DefaultImplementation/Channels/OAuthWebappsHTTPSChannel"/>
```

- e) Add **/Common/Channel/CentralLoggerRV** and **/Common/Channel/CentralLoggerJMS** to the **be.channel.deactivate** property. The modified value of the property is as follows:

```
<property name="be.channel.deactivate" value="/Common/Channel/CentralLoggerRV,/
Common/Channel/CentralLoggerJMS,/Common/Channel/AS,/ASG/Channels/
RvMappingChannel,/ASG/Channels/RvCacheableChannel,/ASG/Channels/modRV_Channel,/
Common/Channel/Channel,/DefaultImplementation/Channels/SouthboundEsb0Channel,/
DefaultImplementation/Channels/SouthboundEsb1Channel,/DefaultImplementation/
Channels/SouthboundEsb2Channel,/DefaultImplementation/Channels/
North_ESBChannel,/DefaultImplementation/Channels/North_HTTPChannel,/ASG/
Channels/SOAPJMSChannel,/ASG/Channels/SOAPJMSChannel_North,/ASG/Channels/
SOAPJMSChannel_South,/Common/Channel/CentralLoggerJMS,/DefaultImplementation/
Channels/OAuthWebappsChannel,/DefaultImplementation/Channels/
OAuthWebappsHTTPSChannel"/>
```

- f) Save the changes to the CDD file.

5. Deactivate the Central Logger RV and JMS Channel in asg_core.cdd File

- a) Navigate to the **ASG_HOME/bin** directory.
 b) Edit the **asg_core.cdd** file in a text editor.
 c) Search the following property:

```
<property name="be.channel.deactivate" value="/Common/Channel/AS,/ASG/Channels/
RvMappingChannel,/ASG/Channels/RvCacheableChannel,/ASG/Channels/modRV_Channel,/
Common/Channel/Channel,/DefaultImplementation/Channels/SouthboundEsb0Channel,/
DefaultImplementation/Channels/SouthboundEsb1Channel,/DefaultImplementation/
Channels/SouthboundEsb2Channel,/DefaultImplementation/Channels/
North_ESBChannel,/DefaultImplementation/Channels/North_HTTPChannel,/ASG/
Channels/SOAPJMSChannel,/ASG/Channels/SOAPJMSChannel_North,/ASG/Channels/
```

```
SOAPJMSChannel_South,/Common/Channel/CentralLoggerJMS,/DefaultImplementation/
Channels/OAuthWebappsChannel,/DefaultImplementation/Channels/
OAuthWebappsHTTPSChannel"/>
```

- d) If you have not taken the backup of the **be.channel.deactivate** property, copy the **be.channel.deactivate** property to **be.channel.deactivate.backup**, as follows:

```
<property name="be.channel.deactivate.backup" value="/Common/Channel/AS,/ASG/
Channels/RvMappingChannel,/ASG/Channels/RvCacheableChannel,/ASG/Channels/
modRV_Channel,/Common/Channel/Channel,/DefaultImplementation/Channels/
SouthboundEsb0Channel,/DefaultImplementation/Channels/SouthboundEsb1Channel,/
DefaultImplementation/Channels/SouthboundEsb2Channel,/DefaultImplementation/
Channels/North_ESBChannel,/DefaultImplementation/Channels/
North_HTTPChannel,/ASG/Channels/SOAPJMSChannel,/ASG/Channels/
SOAPJMSChannel_North,/ASG/Channels/SOAPJMSChannel_South,/Common/Channel/
CentralLoggerJMS,/DefaultImplementation/Channels/OAuthWebappsChannel,/
DefaultImplementation/Channels/OAuthWebappsHTTPSChannel"/>
```

- e) Add /Common/Channel/CentralLoggerRV and /Common/Channel/CentralLoggerJMS to the **be.channel.deactivate** property. The modified value of the property is as follows:

```
<property name="be.channel.deactivate" value="/Common/Channel/CentralLoggerRV,/
Common/Channel/CentralLoggerJMS,/Common/Channel/AS,/ASG/Channels/
RvMappingChannel,/ASG/Channels/RvCacheableChannel,/ASG/Channels/modRV_Channel,/
Common/Channel/Channel,/DefaultImplementation/Channels/SouthboundEsb0Channel,/
DefaultImplementation/Channels/SouthboundEsb1Channel,/DefaultImplementation/
Channels/SouthboundEsb2Channel,/DefaultImplementation/Channels/
North_ESBChannel,/DefaultImplementation/Channels/North_HTTPChannel,/ASG/
Channels/SOAPJMSChannel,/ASG/Channels/SOAPJMSChannel_North,/ASG/Channels/
SOAPJMSChannel_South,/Common/Channel/CentralLoggerJMS,/DefaultImplementation/
Channels/OAuthWebappsChannel,/DefaultImplementation/Channels/
OAuthWebappsHTTPSChannel"/>
```

- f) Save the changes to the CDD file.

Cache Cleanup Agent

The Cache Cleanup Agent provides mechanism to clear the cache. The Cache Cleanup Agent 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 Cleanup Agent evaluates all cache entries on a scheduled time basis and evicts these entries whose time-to-live has expired.

Running Cache Cleanup Agent

This section explains the steps to run the Cache Cleanup Agent:

Procedure

1. Open a terminal window.
2. Navigate to *ASG_HOME/bin* directory.
3. Type the following command:

```
./asg-engine -u asg-cache-cleanup
```



It is good practice to start the Cache Cleanup Agent 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.

Reporting

Overview of reporting functionality supported by TIBCO API Exchange Gateway using TIBCO Spotfire.

The Central Logger component stores reporting information to the database. 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
- Defect detection
- System Usage

Spotfire Configuration

This section describes the steps for TIBCO Spotfire Professional configuration. The Spotfire client retrieves the data stored in TIBCO API Exchange Gateway database to display the reports.

Configuring 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 perform the following tasks:

- Create library contents (such as data sources, information links).
- Import and Export the library content.
- Store the analysis files.

Procedure

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.

Setting up a Spotfire Data source

Set up a data source for Spotfire.

Procedure

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 follows:

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 follows:

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 follows:

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**.
5. Click **Save** again, if pop window appears.
6. Verify that this data source appears in the left panel of the Information Designer dialog with a plus (+) sign next to it.

Creating a Spotfire Information Model for Central Logger

Procedure

1. Within TIBCO Spotfire Professional, open the Information Designer from the menu bar as: **Tools -> Information Designer**.
2. Select the **Elements** tab.
3. Create a new directory as follows:
 - Click **New -> Folder**
 - Enter the name of the new folder. For example, `ASGTEST`.
4. Select the **Data Sources** tab. In the left panel of the Information Designer dialog, expand the data source name by clicking on the + sign. Expand the `asguser` schema.
5. 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
6. Press Shift and 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** link from the context menu.
 7. On the next window, select **Destination folder** from the Library as ASGTEST.
 8. Click **OK** on the next new dialog window.
 9. 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 **OK**.
 10. 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 window on the **Elements** tab.
 11. In the Information Designer window, follow these steps to create the join between the ASG_TRANSACTIONS and ASG_TRANSACTION_KEYS tables:
 - a) Click **New** and select Join from the drop-down list.
 - b) From the database schema, expand the ASG_TRANSACTIONS table. Select TRN_GUID and click **Add** on the right side of the Information Designer window.
 - c) From the database schema, expand the ASG_TRANSACTION_KEYS table. Select KEY_TRN_GUID and click **Add** on the right side of the Information Designer window.
 - d) Click **Save** to save changes.
 - e) On the next window, select **Destination folder** from the Library as ASGTEST. Verify that this creates ASG_TRANSACTIONS - ASG_TRANSACTION_KEYS Inner Join join under ASGTEST folder.
 12. In the Information Designer window, follow these steps:
 - a) Go to **Elements** tab and select ASGTEST folder.
 - b) Click **New** and select Information Link from the drop-down list.
 - c) Select the ASG_TRANSACTIONS data model folder. Add TRN_GUID and TRN_STATUS columns.
 - d) Select the ASG_TRANSACTION_KEYS data model folder. Add KEY_TRN_GUID, KEY_TYPE, KEY_VALUE columns.
 - e) Expand the **Join Path** tab. Add the ASG_TRANSACTIONS - ASG_TRANSACTION_KEYS Inner Join join.
 - f) Save the Information link as Transaction-Transaction Keys.
 13. Click **Close** to close the Information Designer.

Deploying Default TIBCO API Exchange Gateway Audit Trail DXP File on TIBCO Spotfire Server

Procedure

1. Within TIBCO Spotfire Professional, open the *ASG_HOME/templates/spotfire/ASG_LogData.dxp* Spotfire visualization file by selecting the menu option **File - > Open**.

2. If the analysis file opens with *Missing Information Link* warnings, follow the steps:
 - Select the **Browse for the missing information link** radio button and click **OK** .
 - In the Select Information Link dialog window, select the ASG_TRANSACTION information link (with the paperclip icon next to it) and click **OK** .
 - Repeat these steps for the following missing information links:
 - ASG_KPI
 - ASG_TRANSACTION
 - ASG_TRANSACTION_DETAILS
 - ASG_TRANSACTION_MESSAGES
 - ASG_TRANSACTION_KEYS
 - Transaction-Transaction Keys
3. Note that the analysis file opens without any errors. This file does not show any data as the Central Logger database is empty.
4. Save the 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 **Finish**.
5. Click **Close** 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.

Basic Deployment

Overview of the process for basic deployment of the Core Engine and its components.

After the project configuration is completely defined using the Config UI, test the configuration by running the Core Engine and its components at the command line. After testing the configuration, you can deploy the gateway components on production systems.

This section explains the process and configuration required to deploy the Core Engine and its components in a single server or distributed environment.

Deploying TIBCO API Exchange Gateway Processing Units

Full deployment of TIBCO API Exchange Gateway contains the deployment of following components. The following components can be deployed on a single server or distributed across multiple servers.

- Core Engine
- Central Logger
- Global Throttle Manager
- Cache Manager (Optional)
- Cache Cleanup Agent (Optional)

For deployment details of runtime components, see [High Availability Deployment 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. Ensure that the third party jars are available at run time, if applicable.

In case, if any customizations are made in the ASG_DefaultImplementation project of the gateway, configure the CDD file settings and build the EAR file.

Deployment Options

TIBCO API Exchange Gateway provides the following ways to deploy the Core Engine and other processing units:

- Using TIBCO API Exchange Gateway Monitoring and Management Server. See [Deploy Using Monitoring And Management Server](#).
- Run the Core Engine and run time components at the command line. See [Running Processing Units At Command Line](#).
- Using TIBCO Administrator. See [Deploying Gateway Components Using TIBCO Administrator](#).

Deploy Using Monitoring And Management Server

Using the Monitoring and Management Server of TIBCO API Exchange Gateway, perform the following actions:

- Deploy the cache-based Core Engines.
- Monitor and manage a deployed cluster.

The Monitoring and Management Server works with the following sub-components:

- Site topology editor

The site topology editor is a canvas-based editor, which you can use to set the deployment configuration of the cluster by specifying its hosts and deployment mappings. See [Working With Site Topology Editor](#).

- MM Console

MM Console is a web-based dashboard that you can use 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](#) for details on site topology file.

TIBCO API Exchange Gateway packages a sample site topology file to deploy the caching enabled Core Engine and Global Throttle Manager on a single host.

Sample Site topology file

A sample of the site topology file is shipped with the product installation. This file exists as follows:

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.

Configuring JMX properties in Core Engine TRA File

After the Core Engines are started and join the cluster, they use JMX MBeans to expose monitoring and management information to the Monitoring and Management (MM) Server, and for 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
```

Procedure

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 run time. 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 PU, in the site topology file.

To Enable JMX MBeans Authentication

The following property enables authentication:

```
java.property.be.engine.jmx.connector.authenticate=true
```

Editing MM.cdd File

The CDD file used by the MM server is the MM.cdd. In this file, you must specify the path to the site topology file. The MM.cdd file is an XML file and can be edited using any XML or text editor.



Before you make any changes to the MM.cdd file, back up this file.

Procedure

1. Navigate to the `ASG_HOME/mm/bin` directory.
2. Edit the MM.cdd file and change the value of **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 the MM server to load.

By default, the property is set to the sample of a site topology file shown as follows:

```
<property name="be.mm.topology.file" value="ASG_HOME/mm/bin/asg_default.st"/>
```

The sample site topology file exists as follows:

```
ASG_HOME/mm/bin/asg_default.st
```



Whenever you make changes to the MM.cdd file, restart the MM server so that it uses the updated values.

Editing Site Topology File in a Text Editor

The site topology file contains deploy time information such as what processing units to deploy to specific computers and hosts in your environment. You need to know information about the computers that will host the agents you intend to deploy, such as the information about the machine's operating system and IP address.

An annotated site topology file that can be used as a template exists as follows:

```
ASG_HOME/mm/bin/asg_default.st
```

You only need to edit the topology file if the defaults do not serve your purposes. This happens typically when the Core 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 `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>
```

See [Site Topology Reference](#) for details on the XML elements defined in the topology file.



Backup the site 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.

Software Options for Deployment, Remote Start, and Remote Method Invocation

Software	Deployment	Remote Start, Stop	Remote Method Invocation
SSH	Yes	Yes	
PsTools (Windows)		Yes	
TIBCO Hawk®		Yes	Yes
JMX (required)			Yes

Although it is possible to use more than one utility for the machines in the cluster, it is good practice to 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](#).

SSH

Only SSH software enables MM to deploy Core 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 below.

Installing and Configuring an SSH Server

If you want to use SSH on Windows machines, 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 the 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, 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 user name suggested when configuring the OpenSSH server and provide a password for the user name.
- For deployment and starting PUs, use a user name that is SSH enabled to login to the remote host. For example, `ssh username@hostname`. The credentials of the user name can be specified in the host settings of the site topology file, Host Settings User and Password fields. See [Host Settings](#).

However, if you choose to use a different user name, ensure that the user name 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.

Deploying and Managing Processing Units

Complete these steps to deploy the processing units:

Procedure

1. Starting MM Server

To start the MM server, follow these steps:

- Navigate to the `ASG_HOME/bin` directory.
- Type the following command:

On the Windows platform,

```
asg-mm.exe
```

On the Unix platform,

```
./asg-mm
```

2. Logging On to MM Console

To log on to MM Console, follow these steps:

- a) Refer to the [Configure MM Console Properties](#) to set the MM Console properties, as required.
- b) Open a browser window and enter the following URL:

`http://localhost:9000/index.html`

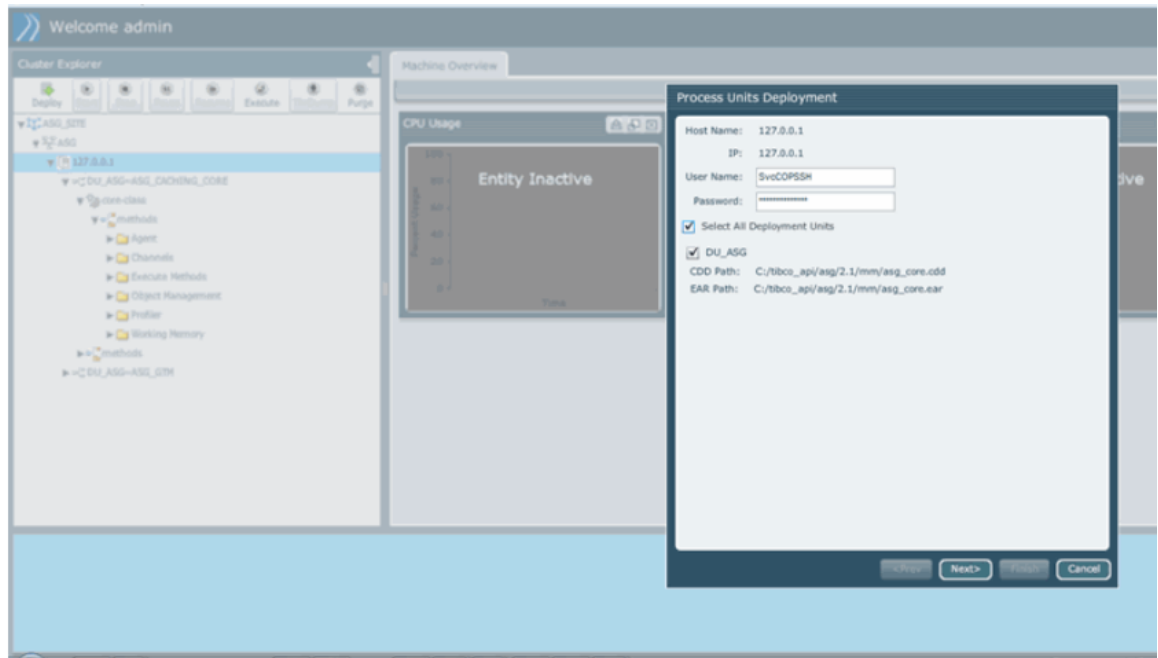
The host name and port are configured in the MM.cdd file.

- c) Log on to MM Console using your login credentials. The default user name and password are admin and admin.
- d) Verify that you are successfully logged into the MM Console.

3. Deploying Processing Unit in MM Console

To deploy the processing units of TIBCO API Exchange Gateway, follow these steps:

- a) Log on to MM Console. See [Logging On to MM Console](#).
- b) From the Cluster Explorer, select the machine node you want to deploy.

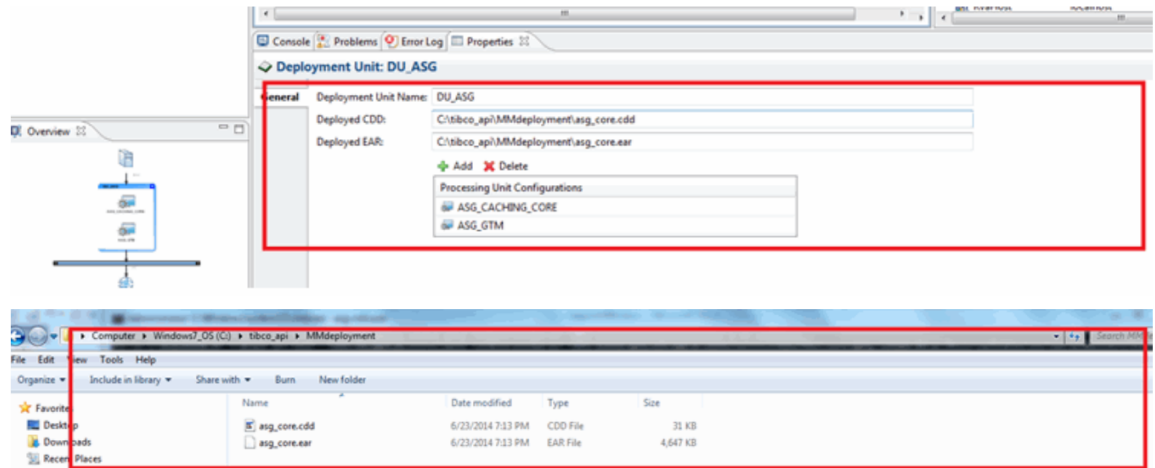


- c) Select the icon of the host machine where you want to deploy and click **Deploy**.
- d) Verify the login details or provide them. Click **Next**.
- e) If you want to override any global variables, select the **Global Variables** tab and do so. See [Overriding Global Variables in MM](#) for details. The processing unit that you configured to deploy on that machine in the site topology file is deployed.
- f) Click **OK**. The engine or engines configured to deploy to that machine deploys.



- To see if an engine or agent is deployed, hover the pointer over its name in the explorer panel. A tooltip shows if it is deployed or undeployed.
- Deployment time information is saved to a file located under `ASG_HOME/mm/` deployed and the last deployment time is displayed in the UI.

- g) Verify that after successful deployment, the EAR and CDD files are generated in the folder as specified in the site topology file for the deployment unit.



4. Starting Processing Unit in MM Console

After the successful deployment of processing unit, follow these steps to start a processing unit:

- Select the processing unit you want to start.
- Click **Start**.
- Verify that you get the success message dialog on the MM console. The log file is created in the `ASG_HOME\asg\2.3\mm\bin\logs` folder.

Overriding Global Variables in MM

You can override the global variable settings of the project in the following ways:

- When command-line deployment is used, set the global variables in the master CDD file.
- Using TIBCO BusinessEvents Monitoring and Management settings. If you plan to deploy using MM, override global variables using MM, instead of in the CDD file.
- You can set values at the machine level (but not at the engine level). The global variable overrides affect all engines deployed to the selected machine.
- The global variable settings are appended to the CDD file that is deployed to a machine.

Procedure

- Log on to MM Console. See [Logging On to MM Console](#).
- In Cluster Explorer, select a machine and select the **Global Variables** tab in the panel on the right. The global variable names and default values are displayed.
- In the **Current Value** column, replace the current value with the desired override value.
- Click **Save**.
- Click **Refresh** to ensure that your value was the last entered.

Configure MM Console Properties

Configure the properties dealing with the MM Console in the following file:

`ASG_HOME/MM/web-root/app_config.xml`

Update the file as needed to set values for the following properties.

MM Console Configuration Properties

Property	Notes
<code>debugMode</code>	Set to <code>true</code> to enable more detailed error messages. The default value is <code>false</code> .
<code>Demo Mode</code>	If demo mode is enabled, chart updates are faked with random values based on the most recent value. The default value is <code>false</code> .
<code>updateInterval</code>	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. The default value is five.
<code>failedPaneThreshold</code>	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, Lost connection to data server. The user clicks OK and is logged out. The default value is 0.2.
<code>logoURL</code>	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.
<code>chartStyles</code>	You can configure preferences such as colors used for various chart elements. Follow the documentation in the file for each element.

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](#) and [Configure Cluster Discovery and Internal Communication](#).

Configuring ASG_DefaultImplementation Project's CDD

Procedure

1. Start the Studio.
2. Open the ASGDefaultImplementaion 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.

Configure 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.

Configuring 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.

Procedure

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](#)
 - [Unicast \(Well-Known Address\) Cluster Member Discovery](#)
 - [Configuring the TIBCO BusinessEvents DataGrid Listen URL](#)
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 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](#)).

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 as follows.



A TIBCO BusinessEvents DataGrid cluster is also known as a *metaspace*.

A Core Engine is a *node* in the metaspace.

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.

Parameter	Notes
<code>destinationPort</code>	<p>Specifies the destination port used by the PGM transport.</p> <p>Must be the same value on all machines in the cluster.</p> <p>The default value is 7888.</p>
<code>network</code>	<p>Specifies the IP address of the interface to be used for sending multicast packets, and the multicast group address to be used.</p> <p>The format is as follows: <i>interface;multicast group address</i></p> <p>The value for <i>interface</i> 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.</p> <p>The value for <i>multicast group address</i> must be the same on all machines in the cluster.</p> <p>The default value for <i>interface</i> is the first available interface provided by the operating system hosts file for the machine.</p> <p>Note</p> <p>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 <code>/etc/hosts</code> 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:</p> <pre>SYS_ERROR (multicast_error - (8) grp_iface not a valid multicast interface)</pre> <p>To resolve this issue, either modify the hosts file, or provide the desired interface explicitly in the <i>network</i> argument.</p> <p>The default value for multicast group address is the multicast group address as 239.8.8.9.</p>

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](#)). 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.

Parameter	Notes
<i>interface</i>	<p>To specify a value, use the desired IP address.</p> <p>The value for <i>interface</i> 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.</p> <p>The default value for <i>interface</i> is the first available interface provided by the operating system for the machine.</p>
<i>port</i>	<p>To specify a single port use the port number in the listen URL, as shown in this example:</p> <pre>tcp://interface:6000/</pre> <p>You can use an auto-incrementing feature, as explained in Auto-incrementing Within a Range of Ports.</p> <p>The default value is the first available port in the 50000+ range.</p>

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, 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, such as 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, recreate the site topology file using the updated CDD.
- If you change the site topology, you must restart the MM server.

Procedure

1. Configure Cluster Properties.
2. Add Deployment Units (DU's).

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 will 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.

Specify the host configuration, including the software used on the remote hosts to start remote Core Engine (processes). Map the DU's to the hosts where you want to deploy them. Multiple hosts can reuse the same deployment unit configuration.

References in Site Topology File

References of CDD and EAR files in the site topology file.

The site topology file refers two locations of the CDD and EAR files. The files in each location must be *exact copies*:

- Master CDD and EAR files: 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, 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 might 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 be 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, link it only to the appropriate host, for deployment.

Site Topology Reference

References to the various settings in the site topology file.

Site Settings

Site Topology — Site Settings

Property	Notes
Site Name	Site name. The default value is the name of the site topology file.
Description	Description of the site, as desired.

Cluster Settings

Site Topology — Cluster Settings

Property	Notes
Cluster Name	Read-only field displaying the cluster name specified in the CDD. This name is set in the Cluster Name field of the CDD editor.

Property	Notes
Project CDD	Location and name of project CDD. This is the location used by the Studio for configuration of the site topology.
Master CDD	Location and name of the master CDD. This is the location used by the MM server.
Master EAR	Location and name of the master EAR. This is the location used by the MM server.

Deployment Unit Settings

Site Topology — Deployment Unit Settings

Property	Notes
Deployment Unit Name	<p>Name of the deployment unit. Name must be unique.</p> <p>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, put the host name in the DU name.</p> <p>Note : 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.</p> <p>The default value is DU_<i>n</i> where <i>n</i> is a number that increments each time you add a DU to the diagram.</p>
Deployed CDD	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.
Deployed EAR	Absolute file path to the location where the MM server will deploy the copy of the master EAR used by this DU.
Processing Unit Configurations	Displays a list of processing unit configurations.

Processing Unit Settings

Site Topology — Processing Unit Settings

Property	Notes
Processing Unit Configuration Name	<p>The name that identifies this configuration of the processing unit, as specified in the Processing Unit property setting. The name must be unique within a DU.</p> <p>The processing units settings are configured in the CDD.</p>
Use As Engine Name	Select this check box to use the value of the Processing Unit Configuration Name field as the engine name. Use the same choice across all processing units in the cluster.

Property	Notes
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), 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:

Site Topology – Host Settings

Property	Notes
General Settings	
host name	<p>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.</p> <p>To validate the host name, ping the host using this name from the MM server machine.</p> <p>Note: Specify the exact name of the host. Errors in the host name result in the host appearing in the MM Console UI as an unpredefined machine. Do not, for example, use localhost.</p>
ip	<p>IP address of the host machine. Used for remote access. Required.</p> <p>Use 127.0.0.1 (localhost loop back IP address) for engines running on the same machine as the MM server.</p>
username	<p>User name to log onto the host machine.</p> <p>The user credentials are used for remote deployment and execution, including starting a process unit.</p> <p>At run time, 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. Enter different values at run time as needed.</p> <p>If you do not provide the credentials here, then provide them at the pop-up dialog.</p> <p>Specify a local user or a domain user.</p> <p>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.</p>

Property	Notes
password	Password of the user referenced in the User Name field. The password is encrypted. See notes in User Name section.
os-type	Operating system of the host machine. See the product readme for a list of supported platforms.
Start PU Method Setting	
Start-PU-Method	<p>Choose the method that MM will use to start this processing unit on remote machines:</p> <p>Use Hawk</p> <p>Use PsTools</p> <p>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 a good practice because it is also the default port used by most Linux SSH servers.</p> <p>Note that a username and password for the remote machines are required for MM to connect (see notes for User Name and Password fields).</p> <p>See Install and Configure Software for Remote Start and Deployment for details on each option.</p> <p>The default value is SSH. Default SSH port number is 22.</p>

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 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 (Core 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, Core Engines, and agents are all *members* of the TIBCO API Exchange Gateway cluster.

Navigating Cluster Explorer

To navigate the cluster explorer, follow any of these steps:

Procedure

1. Expand Cluster Explorer and select the member you want to work with or whose metrics you want to see. Metrics display on the right.
2. Click an inactive cluster member to display the last available health metrics for that member.
3. Click the minimize button in the Cluster Explorer title bar to minimize the explorer panel.

Starting Stopping Pausing and Resuming Core Engines

Procedure

1. From Cluster Explorer, select the engine you want to start, stop, pause, or resume.
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 the Studio. Before you begin ensure that you have a valid CDD file. The processing units that you deploy to the various hosts are defined in the CDD.

See [Site Topology Overview](#) for important information. See [Site Topology Reference](#) for detailed information on the settings.

Adding a Site Topology Diagram

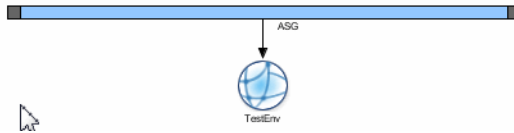


Disable Animation to Avoid Display Issues

Animation can cause display problems. To disable animation, go to **Windows > Preferences > TIBCO BusinessEvents > Diagram**. In the Animation section, clear the **Allow** checkbox.

Procedure

1. Open the project in 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 the 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:



Configuring the Site Topology

Procedure

1. On the canvas, click the site icon (the globe). In the **Site Properties** tab, 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 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 the Studio, you must copy the master files to the specified location so they are available for use by MM.

In the topology file, 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, specify a locally available copy of the project CDD, used only at design time for configuring the topology file in the Studio.
- Master CDD and EAR files: Also in the cluster configuration tab, 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, 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](#) 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. 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.

See Site Topology Reference: [Processing Unit Settings](#) 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](#) 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 Core Engine](#) for the processing units details of TIBCO API Exchange Gateway.

You can run any processing unit as follows:

Procedure

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

```
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 Cleanup Agent.

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 gateway project configuration of TIBCO API Exchange Gateway.

Deploying Gateway Components Using TIBCO Administrator

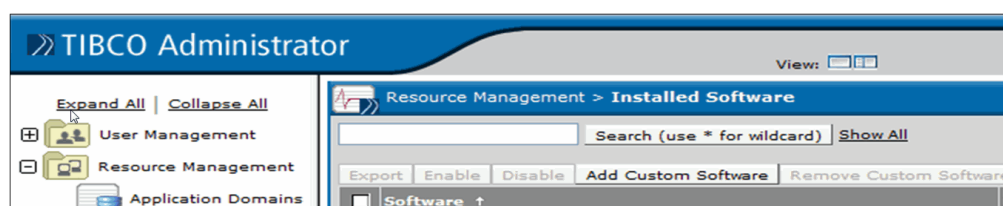
To administer the Core Engine or any component such as Central Logger or Global Throttle Manager of TIBCO API Exchange Gateway within the TIBCO Administrator, follow these steps:

Procedure

1. **Back up Files**
 - a) Navigate to the *ASG_HOME/bin* directory.
 - b) Open the *asg-engine.tra* file in a text editor.
 - c) Ensure that the *tibco.env.HAWK_HOME* property is set correctly to the location of TIBCO Hawk installation home. For example, the property is set to *c:/tibco/hawk/4.9*.
 - d) Ensure that the *tibco.env.RV_HOME* property is set correctly to the location of TIBCO Rendezvous installation home. For example, the property is set to *c:/tibco/tibrv/8.3*.
 - e) Ensure that the *tibco.env.ASG_CONFIG_HOME* property is set correctly to the location of gateway configuration directory. For example, the property is set to *C:/TIBCOAPIX_Exchange211GA_CONFIG/tibco/cfgmgmt/asg*.
 - f) Save changes to the file, if any.
 - g) Back up the *asg-engine.tra* file on all the machines where the components of TIBCO API Exchange Gateway are deployed.
 - h) Back up the *asg-core.ear* file.
2. **Register TIBCO API Exchange Gateway Component**

To register the TIBCO API Exchange Gateway component within TIBCO Administrator, follow these steps:

- a) Start the TIBCO Administrator, if not running.
- b) Log in to the TIBCO Administrator using your credentials.



- c) Expand the **Resource Management** node.
- d) Click **Installed Software**.
- e) Click **Add Custom Software**. Verify that the list of machines associated with the domain are displayed.
- f) Click the radio button to select the machine where TIBCO API Exchange Gateway is installed.
- g) Click **OK**.

The screenshot shows the 'New Custom Software' dialog box with the 'General' tab selected. The 'Machine' field is populated with 'Anikkah2-T420' and has a 'change...' link. Below it are input fields for 'Software Type', 'Software Display Name', 'Version', and 'Executable (Full Path)'. There are two checkboxes: 'Software is an adapter' (unchecked) and 'Java Software' (checked). Below these are input fields for 'Java Start Class', 'Java Start Method', 'Java Stop Method', and 'Java Classpath'.

- h) Enter the values as explained in the following table:

New Custom Software Properties

Parameter	Value	Description
Software Type	be-engine	Specifies the name of custom software. This is a required field and should not be changed.
Software Display Name	<i>User_Specified_Software_Name</i>	Specifies a name of the software. The name of the software displayed under Installed Software in TIBCO Administrator is a concatenation of this display name and the version value as entered in the Version field.
Version	5.1.4	Specifies the version of TIBCO BusinessEvents used by TIBCO API Exchange Gateway. This is a required field. For example, for TIBCO API Exchange Gateway 2.3, enter 5.1.4
Executable (Full Path)	<i>Full_Path of asg-engine.exe</i>	Specifies the full path of the executable file on the machine. For example, c:\tibco\asg\2.3\bin\asg-engine.exe
Software is an adapter	Select the check box.	
Java Software	Select the check box.	

Parameter	Value	Description
Java Start Class	com.tibco.asg.container.standalone.ASGMain	Specifies the JVM to be used to launch this application. This value must match the value entered in the <i>ASG_HOME/bin/asg-engine.tra</i> file.
Java Start Method	main	Specifies the main function in start the application. This method must be main as entered in the <i>ASG_HOME/bin/asg-engine.tra</i> file.

i) Click **Save**.

3. Load New Application

- a) Navigate to **Application Management > All Applications**.
- b) Click **New Application**.
- c) Click **Choose File** and select the *ASG_HOME/bin/asg_core.ear* file.
- d) Click **OK**.
- e) Verify the settings on the **New Application Configuration** screen.
- f) Enter name of application and deployment name under **Application Parameters**, as follows:
 - **Name:** Enter a name for application. For example, BookQuery. The default value is *ASG_Core*.
 - **Deployment Name:** Enter a name for deployment. For example, APIX-BookQuery. The default value is *domain_name-application name*.
- g) Click **Save**.
- h) Expand the *Application_Name* node.
- i) Click **ASG_Core.bar**.
- j) In the **General** section, select the **Enable Service** check box.
- k) Click **Save**.

4. Deploy Application

- a) Expand **Application Management > Application_Name > Configuration**.
- b) Click *Application_Name*.
- c) Click **Advanced**. Edit the default values of parameters, if required.
- d) Click **Save**.
- e) Delete the newly generated *asg-engine.tra* file.
- f) Copy the original *asg-engine.tra* back up file to *asg-engine.tra*.
- g) Click **Deploy**.
- h) Clear the **Start successfully deployed services** check box.
- i) Click **OK**.

When any TIBCO API Exchange Gateway component is deployed for the first time, a new *asg-engine.tra* file is created under *ASG_HOME/bin* directory, which does not contain full information. To correct this, follow these steps:

Using the Administrator, follow these steps to perform the force deployment on all machines:

- Navigate to **Application Management > Application_Name > Configuration**.
- Click **Deploy**.
- Select the **Force redeployment of all services** check box.
- Click **OK**.



5. Modify Generated TRA File

After the TIBCO API Exchange Gateway components are deployed, a corresponding TRA file is created in the *Tibco_Domain_name/application/Application_Name* directory.

For example, for the BookQuery application, the default file name is BookQuery-Process_Archive.tra file.

Follow these steps to edit these files for correct parameter settings:

- a) Open the TRA file in a text editor.
- b) Search for the application.args=%APP_ARGS% string. Ensure that there are no additional parameters after APP_ARGS in this line. For example, if the deployment has generated additional parameters within this line such as -p, remove the text.
- c) Search for the tibco.hawk.microagent.name=COM.TIBCO.ADAPTER.be-engine line. Replace this line with Hawk.AMI.DisplayName=COM.TIBCO.ADAPTER.be-engine.
- d) Search for the tibco.clientVar.DirTrace line. Replace this line with: Engine.Log.Dir.
- e) Save changes to the file.

6. Modify Generated CMD File

After the TIBCO API Exchange Gateway components are deployed, a corresponding CMD file is created in the *Tibco_Domain_name/application/Application_Name* directory.

For example, for the BookQuery application, the default file name is BookQuery-Process_Archive.cmd file.

- a) Open the CMD file in a text editor.
- b) Append the parameters -u, -a, -n, -c and their associated values to the end of the asg-engine.exe command. Refer to *TIBCO API Exchange Gateway User's Guide* for details on these parameters.

For example,

```
C:/tibco/be/be/5.0/bin/asg-engine.exe --run --propFile "C:/tibco/classic/tra/
Tibco_Domain_name/application/BookQuery/BookQuery-ASG_Core.tra" -c "C:\tibco\classic\asg
\2.3\bin\asg_core_deploy.cdd" -u asg-caching-core -a BookQuery -n BookQuery
ASG_HOME/bin/asg_core.ear
```

- c) Save changes to the file.

7. Start Engine Instance

- a) Navigate to **Application Management > ASG_Core > Service Instances**.
- b) Click the *Machine_Name-Application_name* service instance.
- c) Click **Start**.
- d) To ensure that the service instance is started successfully, follow these steps:

- Verify that the state of application is changed to Running.
- Ensure there are no errors in the *Tibco_Domain_name/application/Application_Name/logs* directory.
- Using a browser, enter the following URL to submit a ping operation request:
http://IP_Address_Of_machine_name:listening_port_value/ping

The gateway must return a successful response.

Advanced Features

This section describes some advanced functionality of TIBCO API Exchange Gateway.

Cache Agent

A cache agent manages the cache data.

The Cache Agent stores cache data and is responsible for object management. A processing unit can have one only Cache Agent.

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 Core 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 the cache.



Use an unlimited cache only if you deploy enough Cache Agents to handle the data. Otherwise out of memory errors may occur.

Running Cache Agent

Procedure

1. Open a terminal window.
2. Navigate to *ASG_HOME/bin* directory.
3. Type the following command to start up the stand-alone Cache Agent:

```
./asg-engine -u asg-cache
```

The following command starts up the Core Engine in cache enabled mode:

```
./asg-engine -u asg-caching-core
```

Hot Deployment Overview

Allows the configuration changes at runtime.

You can make certain changes to a set of TIBCO API Exchange Gateway configuration without shutting down the Core Engine. This is known as hot deployment.

Hot deployment process suspends north inbound channels and checks for any pending transactions to complete in memory. If there are any pending transactions, the Core Engine waits for the specified delay time before hot deploying the configuration. If there are no transactions in memory, the configuration is reloaded immediately.

Enabling Hot Deployment

By default, hot deployment is not enabled. To enable the hot deployment feature, follow these steps:

Procedure

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 Core 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 Core Engine.

`refreshConfig` method uses following parameters:

- The first parameter is the location of the configuration directory used by the Core Engine.
- The second parameter is the delay in milliseconds. If the delay value is -1, the Core Engine does not shutdown on any error during hot deployment. Otherwise, the Core 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 Studio.

Extension mechanism implements the following features:

Association Cache

The Core 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 Core 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 Core Engine to process the requests faster and also to 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 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 Core 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 a product ID in the request but the back-end service requires the product name, then the Core Engine calls an east side service to translate the product ID into the 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 faster processing of requests.

Content based Authorization

TIBCO API Exchange Gateway supports 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. TIBCO API Exchange Gateway supports the authorization of each 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. TIBCO API Exchange Gateway provides a set of catalog and custom rule functions to support this functionality.

Response Caching

Overview of Response caching.

TIBCO API Exchange Gateway can cache response messages from the target services. The response caching functionality is supported for HTTPS and REST or HTTPS and SOAP requests. To enable caching of response messages from the target services for any facade operation, see [Enabling Response Caching](#).

Response caching is supported by the CacheEnabled operation feature of TIBCO API Exchange Gateway. If the CacheEnabled keyword is specified in the **Operation Feature** and the caching is not explicitly configured on the Config UI for a facade operation, TIBCO API Exchange Gateway uses the default caching parameters as follows:

- Cache Type: Simple Cache
- Time To Live: 1 day (24 hours)

Types of Response Caching

The following types of response caching are supported:

Facade Response Cache (Simple Cache)

The Facade Response Cache is known as Simple Cache. If the Simple Cache type is enabled as response caching for a facade operation, the Core Engine processes the incoming facade requests as follows:

- The Core Engine retrieves the response message from the cache for the matching cache response key for every facade request from the client.
- If the response message is found in the cache and is not expired, the Core Engine sends the response message from the cache as the facade response message to the client.
- If the response message is not found in the cache, the Core Engine forwards the facade request to the target operation and sends the response message from the target service to the client. The Core Engine stores the response message into the cache just before sending the response message to the client. This ensures that the response message is available in the cache for subsequent requests from the client. The Time to Live parameter specifies the duration of the response message in the cache. See [Response Caching Parameters](#) table.
- If the response message in the cache for the same cache response key is expired, the Core Engine deletes the response message from cache and forwards the facade request to the target operation.

- When the response message is received from the target service by the Core Engine for the first time, the Core Engine stores the response message including the headers in the cache using the cache response key. See [Cache Response Key](#). For every subsequent requests from the client, the response from the cache is returned with all the response headers, similar to the response headers from the target service. The X-Cache: Hit in the response message header indicates that the response message is returned from the cache.
- If the response from the target operation is an error, the Core Engine does not cache the response.



- When a response message is found in the cache for a facade operation request, the Core Engine adds the following header in the response message:

X-Cache: Hit

- If the HTTP header of the response message from the target service has a Cache Control header with either the no-cache or no-store directive, the response is not cached.

Sample Request Header

The following is the sample request header for the facade request with accept header:

```
<h:request xmlns:h="http://www.tibco.com/asg/protocols/http">
<h:server-ip>localhost</h:server-ip>
<h:server-port>9222</h:server-port>
<h:client-ip>0:0:0:0:0:0:0:1</h:client-ip>
<h:client-port>0</h:client-port>
<h:scheme>http</h:scheme>
<h:method>GET</h:method>
<h:request-uri>/Books/BookOperations/Title/The Power of Now</h:request-uri>
<h:protocol-version>HTTP/1.1</h:protocol-version>
<h:query-string/>
<h:header name="host">localhost:9222</h:header>
<h:header name="connection">keep-alive</h:header>
<h:header name="accept">
text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
</h:header>
```

Sample Response Headers

The following are sample response headers for the facade response caching type:

Sample Response Headers from Target Service

```
Access-Control-Allow-Origin: *
Date: Wed, 13 Aug 2014 20:59:06 GMT
Server: Apache-Coyote/1.1
Connection: Keep-Alive
Transfer-Encoding: chunked
Access-Control-Allow-Methods: GET, POST, DELETE, PUT
Content-Type: application/json;charset=ISO-8859-1
```

Sample Response Headers from Cached Response

```
Access-Control-Allow-Origin: *
Date: Wed, 13 Aug 2014 20:59:06 GMT
X-Cache: Hit
Server: Apache-Coyote/1.1
Connection: Keep-Alive
Transfer-Encoding: chunked
Access-Control-Allow-Methods: GET, POST, DELETE, PUT
Content-Type: application/json;charset=ISO-8859-1
```

Target Backup Response Cache

The Target Backup Response Cache is known as Backup Cache. The Target Backup Response Cache ensures that the response message is sent from the cache if existing, to the client in case the response is not received from the target service within a time period.

If the response caching of Improve SLA type is enabled for a facade operation, the Core Engine processes the incoming facade request as follows:

- The Core Engine routes the request to the appropriate target service.
 - The Core Engine sends the response message from the target service to the client if a successful response message is received.
 - The Core Engine stores the response message in the cache.
 - The Core Engine processes the subsequent requests from the client as follows:
 - For any request from the client, if the response from the target service is not received within a configured time period specified by the Short Wait Timer parameter, the Core Engine fetches the response message stored in the cache.
1. If the response is found in the cache and is not expired, the Core Engine sends back the response message to the client.
 2. If the response is found in the cache and is expired, the Core Engine deletes the response from the cache and waits for the response message from the target service.
 3. If the response is not found in the cache, the Core Engine waits for the target service to respond and sends back the response to client when received. If the response message is not received from the target service, the Core Engine sends a timeout error to the client.
- If the response message from the target operation is received after the specified time period, the Core Engine just updates the cached response message with the received response message.

Cache Response Key

TIBCO API Exchange Gateway uses the cache response key to check if a cached response exists in the cache for an incoming request.

By default, the Core Engine constructs the value of the cache response key for REST/HTTP requests using the following parameters:

`routingKey+requestURI(minus the API Key)+Accept-Header+Soap Action`

where,

- **Routing Key:** when the routing key is added to the cache response key, this ensures that for each incoming request, the cached response is only returned for a particular routing key.
- **requestURI:** this is the URI that contains the path and query parameters. The API key is removed from the URI.
- **Accept-Header:** The Accept HTTP request-header may be used to pass a content-type preference to the target service. For example, a service may support an Accept-Header of application/xml or application/json to preferentially return the response message in XML or JSON format. For the purpose of caching, the request with Accept-Header of application/xml or application/json are managed as two separate requests. The Accept-Header is added to the cache response key as an additional discriminator.

For example, if a user requests the JSON data for a URI such as `/Books/BookOperations/Title/Power`, the cached response is only returned if the cache value has a JSON response for that URI in the cache.

- Soap Action: specifies the SOAP Action header for a HTTP SOAP request used to construct the default cache response key.

For HTTP/REST and HTTP/SOAP requests, the Core Engine creates a default caching key using the above parameters..



You can override the default cache response key using the custom XSLT specified in the parsing step of a facade operation request. See [Overriding Cache Response Key and Parameters](#) on how to create a cache response key using XSLT.

Response Caching with Proxy Server

For the Proxy operation feature of TIBCO API Exchange Gateway, the response caching is supported as follows:

- If the caching is explicitly configured for a facade operation on the Config UI, TIBCO API Exchange Gateway uses the Cache Type and other caching parameters such as Time To Live as configured for the facade operation.
- If the caching is not explicitly configured for a facade operation on the Config UI, TIBCO API Exchange Gateway uses the default caching parameters, as follows:
 - Cache Type: Simple Cache
 - Time To Live: 1 day (24 hours)

Enabling Response Caching

To enable the response caching for a facade operation request, follow these steps::

Procedure

1. Start the Config UI, if not running.
2. Log in to the Config UI using your credentials.
3. Add a new project or select an existing project under **Projects**.
4. Click the **ROUTING** tab on the right-hand side.
5. Click the **Facade Operations** tab.
6. Select a facade operation to enable the response caching.
7. Select the **Enable Caching** check box.
8. Enter the parameters as explained in the [Response Caching Parameters](#) table.

Response Caching Parameters

The following table explains the parameters required for response caching:

Response Caching Parameters

Parameter	Description
Cache Type	Specifies the type of the response caching. The possible values are: <ul style="list-style-type: none"> • Simple Cache • Backup Cache

Parameter	Description
Time To Live	Specifies the time (in milliseconds) that a response is retained in the cache. The default value is 86400000 (ms).
Short Wait Timer	<ul style="list-style-type: none"> Specifies the time that the Core Engine waits for a response from the target operation before sending a response back from the cache. This parameter is not applicable for Simple Cache response caching type.

Clearing Cached Items

The cached messages are deleted after the time specified by Time To Live parameter. Start the Cache Cleanup Agent to clear the cached response messages as follows:

Procedure

1. Navigate to the *ASG_HOME/bin* directory.
2. Type the following command:
On Windows platform,

```
./asg-engine.exe -u asg-cache-cleanup
```


On Unix platform,

```
asg-engine -u asg-cache-cleanup
```

Overriding Cache Response Key and Parameters

Using TIBCO API Exchange Gateway, you can override the following parameters for a specific facade operation request:

- Cache Response Key. See [Cache Response Key](#).
- Cache Response Parameters.

The Cache Response Key and Cache Response parameters can be overwritten using a custom XSLT file.

To override the cache response key and cache response parameters, follow these steps:

Procedure

1. Create XSLT File.

Create an XSLT file with the following key types to specify the new values for the cache response. Refer to [Sample XSLT File](#).

Key Tags for XSLT File

KeyType	Description
CacheResponse_Key	Specifies the cache response key for a facade operation request used to check if a cached response exists for this request.

KeyType	Description
CacheResponse_ShortWait	Specifies the time that the Core Engine waits for a response from the target service before sending a response back from the cache.
CacheResponse_TTL	Specifies the time (in seconds) that a response is retained in the cache.

2. Upload XSLT File.

To upload the XSLT file, follow these steps:

- Start the Config UI, if not running.
- Log in to the Config UI using your credentials.
- Add a new project or select an existing project under **Projects**.
- Click the **ROUTING** tab on the right-hand side.
- Click the **Facade Operations** tab.
- Select a facade operation to enable the response caching.
- Upload the XSLT file in the **New ProcessBody Transform** field.
- Save the changes to your configuration.



If you copy the XSLT file to the `ASG_CONFIG_HOME/ASGProjectConfig/xslt` directory, select the XSLT file in the **ProcessBody Transform** field of the **Facade Operations** tab.

Sample XSLT File

The following is an example XSLT file which can be used to override the cache response key and response caching parameters:

```
<!-- The key used to check if a cached response exists for this request -->
<key type="CacheResponse_Key"><xsl:value-of select="$httpRequest/h:request/
h:request-uri"/></key>

<!-- This is how long (in milliseconds) we wait for the response from the service
before sending a cached response instead -->
<key type="CacheResponse_ShortWait">4000</key>

<!-- This is the time (in seconds) that the response will be retained in the cache
-->
<key type="CacheResponse_TTL">50</key>

<!-- What caching semantics are applied?
simpleCache - return a value from cache if present, otherwise invoke service and
put response in cache.
improveSLA - use a cached value if the service takes more than ShortWait ms to
reply. Freshen value in cache with real response.
-->
<key type="xCacheResponse_Type">simpleCache</key>
```

Response Caching Example

Out of the box, TIBCO API Exchange Gateway provides a BookQueryBE example. By default ,the queryBookByTitle facade operation is enabled for simple cache response type.

Performance Tuning Parameters

To improve the performance of TIBCO API Exchange Gateway, tune the following parameters:

Procedure

1. Set the following property in the *ASG_CONFIG_HOME*/asg.properties file:

```
tibco.clientVar.ASG/HttpClient/useSynchHttpClient=true
```

2. Increase the number of threads to handle the load of incoming requests, as follows:
 - a) Navigate to the *ASG_HOME*/bin directory.
 - b) Edit the asg_core.cdd file in a text editor.
 - c) Search the <inference-agent-class id="core-class"> under <agent-classes> element.
 - d) For the <inference-agent-class id="core-class">, go to the <shared-queue> node.
 - e) Expand the <shared-queue> node to edit the <workers> element value, as follows:

```
<shared-queue>
  <size>1024</size>
  <workers>10</workers>
</shared-queue>
```

- f) Edit the the value of <workers> element, as required. The default is 10.
 - g) Save changes to the asg_core.cdd file.
3. If you use native HTTP channel, select the Caller's Thread threading model as follows:
 - a) Navigate to the *ASG_HOME*/bin directory.
 - b) Edit the asg_core.cdd file in a text editor.
 - c) Search the <destinations id="DefaultImplementationDestinations">/<destination id="FacadeRequest_5449883F"> under <destination-groups> element.
 - d) For the <destination id="FacadeRequest_5449883F">, go to the <threading-model> node.
 - e) Edit the <threading-model> element value, as follows:

```
<threading-model>caller</threading-model>
```

Using the Caller's Thread threading model, by default, 200 threads managed by the underlying HTTP server resource are used to handle the incoming HTTP requests. When the Caller's Thread is set as threading model for native HTTP channel, the settings for <workers> under <shared-queue> for <inference-agent-class id="core-class"> under <agent-classes> do not apply for native HTTP channel.

- f) Save changes to the file.

Large Payload Limit Settings

The following global variables limit the size of payload at which logging is truncated.

The following global variables will limit the size of payload at which logging is truncated:

```
tibco.clientVar.ASG/ForwardLargePayloadLimit=10000
```

```
tibco.clientVar.ASG/ReverseLargePayloadLimit=10000
```

The default value is -1, which disables this behavior.

Enable Access Logs in HTTP Channel

Use the following settings to add support for access logs on Native HTTP Channels.

To add support for access logging for the native HTTP channel, change the following settings found in the **Advanced** tab of the HTTP channel resource editor:

1. Select the **Debug Request Info** check box. This enables two additional debug fields - **Debug Log Folder** and **Debug Log Pattern**.

2. In the **Debug Log Folder** field, specify where the location of the log file should be generated. The default location is the currently running folder/logs.
3. In the **Debug Log Pattern** field, the default log pattern is used. To change the log pattern, refer to the Tomcat documentation.



You can use global variables in all fields of the **Advanced** tab.

The following properties need to be modified for the access logs.

Parameters to enable access logging for native HTTP Channel:

- `tibco.clientVar.DefaultImplementation/Connections/HTTP/FacadeHTTPConnection/DebugFolder`
For example, `tibco.clientVar.DefaultImplementation/Connections/HTTP/FacadeHTTPConnection/DebugFolder=C:/tibco/asg20/c/tibco/cfgmgmt/asg/logs`
- `tibco.clientVar.DefaultImplementation/Connections/HTTP/FacadeHTTPConnection/DebugPattern`
For example, `tibco.clientVar.DefaultImplementation/Connections/HTTP/FacadeHTTPConnection/DebugPattern=%{yyyy MMM dd HH:mm:ss.SSS 'GMT'X}t %A %I [%m] '%U' [%s] %bbytes %Dms`

Parameters to enable access logging for native HTTPS Channel:

- `tibco.clientVar.DefaultImplementation/Connections/HTTP/FacadeHTTPSSLConnection/DebugFolder`
For example, `tibco.clientVar.DefaultImplementation/Connections/HTTP/FacadeHTTPSSLConnection/DebugFolder=C:/tibco/asg20/c/tibco/cfgmgmt/asg/logs`
- `tibco.clientVar.DefaultImplementation/Connections/HTTP/FacadeHTTPSSLConnection/DebugPattern`
For example, `tibco.clientVar.DefaultImplementation/Connections/HTTP/FacadeHTTPSSLConnection/DebugPattern=%{yyyy MMM dd HH:mm:ss.SSS 'GMT'X}t %A %I [%m] '%U' [%s] %bbytes %Dms`

High Availability Deployment Of Runtime Components

Fault tolerance and high availability configuration setup of runtime components of API Exchange Gateway software.

This section describes the deployment of runtime components in a high availability setup. The runtime components are 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 are highly available in production systems to achieve maximum functionality of the gateway.

TIBCO API Exchange Gateway 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 must add additional instances of Core Engines. The architecture of Core 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 is a corresponding increase in management activity. To avoid the possible impact of management activity upon the Core Engines of the TIBCO API Exchange Gateway, the management components (Global Throttle Manager, Cache Cleanup Agent, and Central Logger) and the TIBCO Spotfire servers should be moved onto separate servers.

If multiple instances of the Core Engines 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 runtime components of the gateway. The runtime 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.

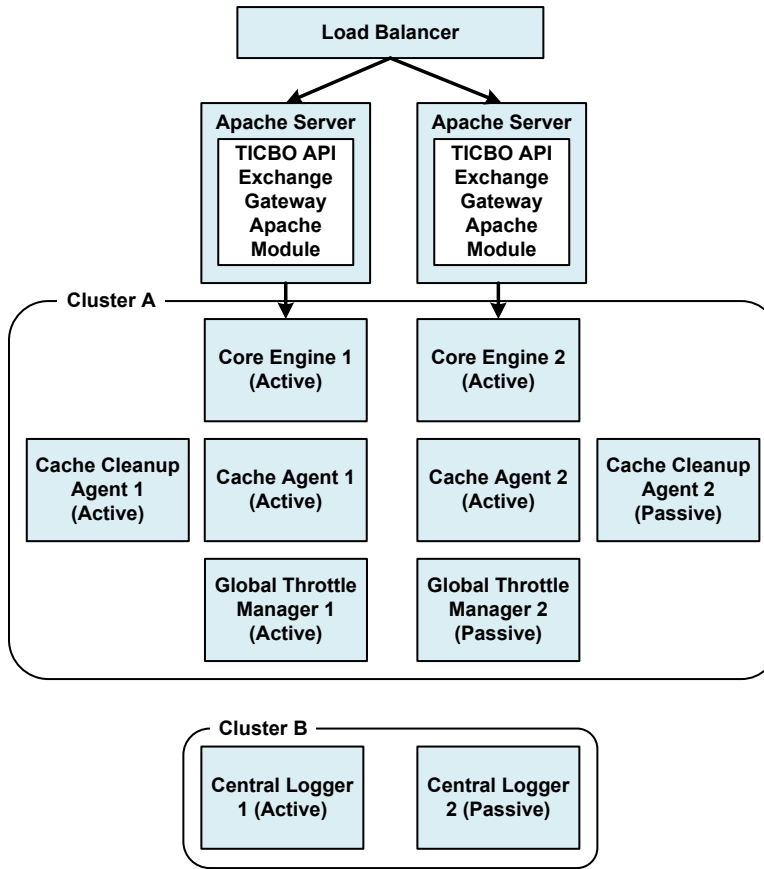
The administrators deploy two or more instances of the Core Engines in production to achieve high availability through load balancing. That is, a load balancer routes request messages to multiple Core Engine instances. The Core Engines together can handle more messages than just one Core Engine instance running. Also, such deployment makes the runtime 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 TIBCO 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](#) shows a high level overview of a deployment model of runtime components in a cluster for high availability setup.

Overview of Runtime components For High Availability



Operational Layer Components

This section briefly describes the information for load balancing setup of the Core Engines and Cache Agents.

Load Balancing 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 [Configure Load Balancer](#) 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 Core Engine. If one of the 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 Core Engine which results in all those requests timing out. For this reason, it is good practice that you do 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:

- There should be a single Apache server per Core Engine instance. For example, if you plan to run two instances of 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 Core Engines are up and running. See [Configure Load Balancer](#) for configuration details. If any of the Apache server or the 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

TIBCO 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.



TIBCO API Exchange Gateway does not support backing store for cache management.

Gateway Management Layer Components

The components of the Gateway Management Layer should be deployed once in a 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 Core Engine operates without loss of functionality.

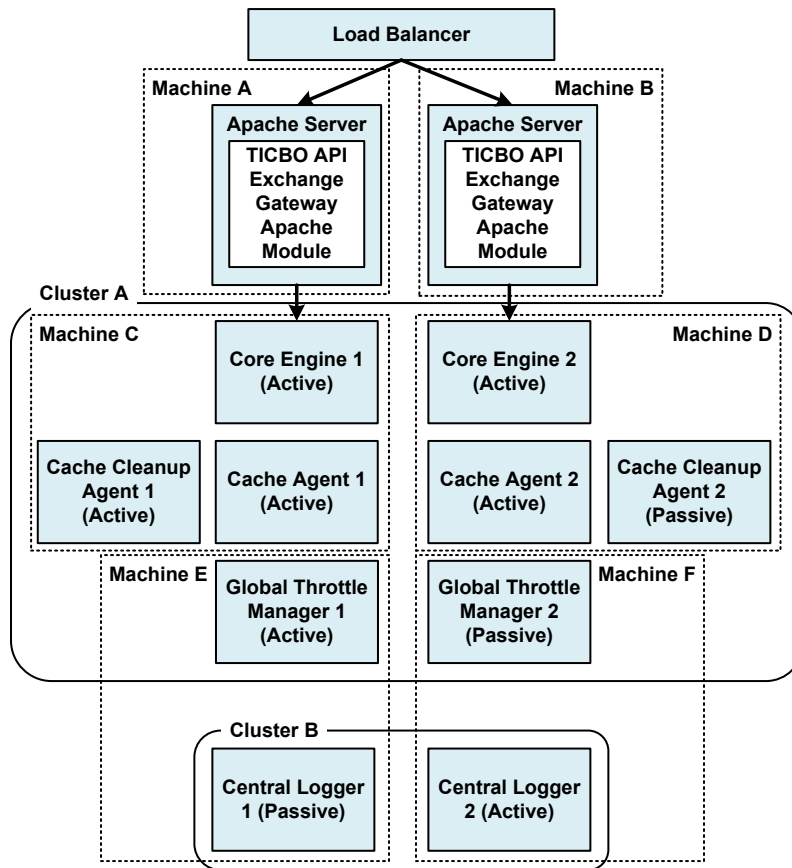
Therefore the Central Logger and Global Throttle Manager are deployed in fault tolerant configuration with one active engine (Central Logger instance or Global Throttle Manager instance) and one or more standby agents on a separate host servers. Such fault-tolerant engine setup can be configured in the 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 good practice 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](#) illustrates the deployment of runtime components on the machines in a cluster environment for high availability.

Deployment of Runtime Components In a Cluster




It is good practice to have the runtime components deployed in a cluster:

- Core Engines - You must deploy all the instances of the 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](#) illustrates an example deployment model in which the components are deployed as follows:

- All the instances of the run time 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 follows:
 - The Core Engine instance 1 runs as an Active instance.

- One instance of the Cache Agent runs.
 - The Cache Cleanup Agent instance 1 runs as an Active instance.
 - On machine D in Cluster A, the components are deployed as follows:
 - The Core Engine instance 2 runs as an Active instance.
 - Second instance of the Cache Agent 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 follows:
 - One instance of the Global Throttle Manager runs as an Active instance.
 - One instance of the Central Logger runs as an Passive instance.
 - On machine F in Cluster A and Cluster B, the components are deployed as follows:
 - Second instance of the Global Throttle Manager runs as an Passive instance.
 - Second instance of the Central Logger runs as an Active instance.
- 
 - All the run time components communicate using the Rendezvous channel. In the example deployment model as shown in [Deployment of Runtime Components In a Cluster](#), it is 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 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.

`asg_core.cdd` file defines the configuration for the following processing units:

- default, asg-core, asg-caching-core - These processing units refer to the Core Engine. The configuration settings of core-class (Inference) agent class defines the runtime behavior 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.

- Configure an IP based load balancer. See [Configure Load Balancer](#).
- Configure Apache module per Core Engine instance. See [Configure Apache Modules for Core Engines](#).
- Configure machines for cluster. See [Cluster Configuration For Runtime Components](#).
- Configure the gateway core instances. See [Configuring Core Engines](#).
- Configure the Cache Agent instances. See [Configuring Cache Agent](#).
- Configure the Cache Cleanup Agent instances. See [Configuring Cache Cleanup Agent](#).
- Configure the Global Throttle Manager instances. See [Configuring Global Throttle Manager](#).
- Configure the Central Logger instances. See [Configuring Central Logger](#).

Configure 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.

Creating A Health Monitor For Core Engines

To create a monitor for the load balancer, follow these steps:

Procedure

1. Go to the Health Monitor tab of the navigation pane.
2. Expand Monitors node under Local Traffic on left. Click the "+" to create a new monitor.
3. Verify that the New Monitor screen appears.
4. 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
5. For Configuration field, select **BASIC** from drop-down list.
6. Input the values for the following fields under Configuration section.
 - 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).
7. Click **Finished**.


Creating 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.

Procedure

1. Go to the **Main** tab of the navigation pane.
2. On the left, under Local TrafficVirtual Servers, select Pools node. Click the "+" 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.

 If the Create button is not available, make sure that you have permission to create a pool. Ask your administrator to grant Create Pool permissions to your user role.
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 [Creating A Health Monitor For Core Engines](#) section.
 - b) Click **Move** (<<) 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 the Resources setting, select an appropriate algorithm from the drop-down list for the **Load Balancing Method** field. For example, you can select Round Robin.
9. Add the pool members as follows:
 - a) Select the New Address option.
 - b) In the Address box, type the IP address of the machine where the 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**.


Creating 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.

Procedure

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 **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 [Creating A Load Balancing Pool](#) section.
11. From the Default Persistence profile setting, select profile_none as the profile.
12. Click **Finished**.

Configure 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 Core Engine runs, you must configure the Rendezvous session connection parameters as described in [Rendezvous Session Connection Parameters for Apache Module](#) and [Rendezvous Session Connection Parameters for Core Engine](#).

Setting the Rendezvous Connection Parameters For Apache Module

For each machine running the Apache server, set the Rendezvous parameters for Apache module.

Procedure

1. Browse to `ASG_HOME/modules/http_server/apache` directory.
2. Edit the `mod_ASG.conf` file.
3. Set the `AsgSubject` parameter, which must be defined uniquely for each machine. For example, this can be set be follows:
`AsgSubject ASG_CoreEngine1_Subject1`
4. You must set the Rendezvous connection parameters as described in the table [Rendezvous Session Connection Parameters for Apache Module](#), if the Rendezvous daemon is running with non-default session parameters.
5. Save the changes in the file.

Setting the Rendezvous Connection Parameters for Core Engine

For each machine running the Core Engine instance, set the properties for Rendezvous connection.

Procedure

1. Navigate to the `ASG_CONFIG_HOME` directory.
2. Edit `asg.properties` file.
3. Set the `tibco.clientVar.ASG/modRV/north_request` property, which must be defined unique for each machine. For example, this can be set as follows:

```
tibco.clientVar.ASG/modRV/north_request=ASG_CoreEngine1_Subject1
```



The value of `tibco.clientVar.ASG/modRV/north_request` property must match the `AsgSubject` defined in the `mod_ASG.conf` file for that machine.

4. You must set the Rendezvous connection parameters as described in the table [Rendezvous Session Connection Parameters for Core Engine](#), if the Rendezvous daemon is running with non-default session parameters.
5. Save the changes in the file.

Cluster Configuration For Runtime Components

This section explains the configuration to setup the cluster properties. A cluster defines the machines running the Core Engines, Global Throttle Manager, Cache Agent and Cache Cleanup Agents. You must define all runtime components in the same cluster except the Central Logger component. The Central Logger instance must be deployed in a separate cluster.

Configuration Files

The cluster properties are defined in the `asg_core.cdd` and `asg_cl.cdd` files.



- All the runtime components (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 Core 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 Core 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 (Core Engine, Cache Agent, Global Throttle Manager, Cache Cleanup Agent).

Configuring 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 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 [Editing asg_core.cdd File To Set Discover URL \(using text editor\)](#).
- 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 [Editing asg_cl.cdd File To Set Discover URL \(using text editor\)](#).

Editing asg_core.cdd File To Set Discover URL (using text editor)

To edit the discover URL and listen URL.

Procedure

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 Core Engine instance 1 is running on Machine C, 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 follows, 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.

Editing 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.

Procedure

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 follows:

```
<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 the Studio to set the discover URL, See [Editing CDD File using Studio](#).

Configuring Fault Tolerance Parameters

For the 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 Core Engines](#).

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](#), [Configuring Global Throttle Manager](#) and [Configuring Central Logger](#).

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.

The following parameters define the fault tolerant configuration of runtime components:

Fault Tolerant Configuration Parameters

Parameter	Description
Max Active	<ul style="list-style-type: none"> 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.
Priority	<ul style="list-style-type: none"> Specifies the priority of the agent for fault tolerant setup of agents. The priority is set at the processing unit level. 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. 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.

This section explains the steps for setting the maximum number of active instances and priority required for the high availability of runtime components.

Configuring Core Engines

You must run the multiple 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 Core Engine Active Active instances, Max Active and Priority parameters are not applicable.

Example Settings for Core Engine Instance

Just as a reference, this section shows the sample settings for the Core Engines in the *ASG_HOME/bin/asg_core.cdd* (XML file). By default, no changes are required to run multiple instances of the Core Engine.

```
<inference-agent-class id="core-class">
  <load>
    <max-active/>
  </load>
```

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 the table below 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.

CDD Cluster Tab Cache OM Settings

Property	Notes
Cache Agent Quorum	<p>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.</p> <p>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).</p> <p>The default value is 1.</p>
Number of Backup Copies	<p>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.</p> <p>Value of 0 means that in the case of abnormal termination, some portion of the data in the cache will be lost.</p> <p>A backup count of 1 means one server plus one backup are needed, that is, two Cache Agents.</p> <p>The default value is 1.</p>

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>
</cache-manager>
</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](#) for description of parameters.

To set the number of the active instances and priority for the Cache Cleanup Agent instances.

Procedure

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 the Studio. See [Edit Cluster Deployment Descriptor \(CDD\) File](#).
3. If you use the Studio, set Max Active property for cache cleanup agents as follows:
 - a) Select the **Agent Classes** tab.
 - b) Select the cache-cleanup-scheduler, Inference agent.
 - c) On the right side, set the following property as follows:

Max Active 1
 - d) Select the cache-cleanup-esp, Query agent.

e) On the right side, set the following property as follows:

Max Active 1

4. Save the changes.
5. Optionally, you can set the Priority for the processing units as follows using the Studio:
 - a) Select the **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:

```
<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](#) for description of parameters.

You can set the number of the active instances and priority for the Global Throttle Manager instances.

Procedure

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 the Studio. See [Edit Cluster Deployment Descriptor \(CDD\) File](#).

3. Set the Max Active property as follows for the Agent Classes > gtm-class (Inference) agent. See [Setting Max Active](#) using the Studio.

Max Active 1
4. Save the changes.
5. Optionally, you can set the priority for the asg-gtm processing unit using the Studio as follows. See [Setting Priority](#).
 - a) Go to **Processing Units** tab.
 - b) Select the asg-gtm node.
 - c) Go to the 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.

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:

```
<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/>
      <priority>2</priority>
    </agent>
  </agents>
```

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](#) for description of parameters.

You can set the number of the active instances and priority for the Central Logger instances:

Procedure

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 the Studio. See [Edit Cluster Deployment Descriptor \(CDD\) File](#). If you use the Studio, follow these steps:
 - a) Open the *ASG_HOME/bin/asg_cl.cdd* file using the steps described in [Editing CDD File using Studio](#).
 - b) Select **Agent Classes** tab.
 - c) Select the BusinessEvents_Archive (Inference) node.
 - d) On the right side, set the following property as follows:

Max Active 1
 - e) Save the changes.
3. Optionally, you can set the priority for the asg-cl processing unit using the Studio as follows. See [Setting Priority](#).

- a) Go to the **Processing Units** tab.
- b) Select the `asg_cl` node.
- c) Go to the 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:

```
<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>
```

Save the changes to the file.

Configure Rendezvous Session Connection Parameters

You must configure the Rendezvous session connection parameters for the 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 Core Engine and Global Throttle Manager Communication](#)
- [Rendezvous Session Connection Parameters For Core Engine and Central Logger Communication](#)

The instances of the runtime components illustrated in the deployment [Deployment of Runtime Components In a Cluster](#) assumes the following points:

- The Core Engine instances and the Global Throttle Manager instances are running on machines which are in the same subnet.
- The Core Engine instances and the Central Logger instances are running on machines that are in the same subnet.

Appendix A

This appendix describes some configuration parameters required for high availability setup configuration.

Edit 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 Studio.

You can edit any CDD file in the following ways:

- Using text editor
- Using 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.

Procedure

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 follows:

```
<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 Studio

Procedure

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 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 select 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 [Setting Discover URL](#), [Setting Max Active](#), [Setting Priority](#).
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.

Setting Discover URL

Procedure

1. 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
```
2. 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/
```
3. Save the changes to the resource.

Setting Max Active

This task explains the steps to setup the Max Active property for an agent.

Procedure

1. Select **Agent Classes** tab.
2. 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.
3. On the right side, click the following property to set a value as follows:

Max Active 1
4. Save the changes to the resource.

Setting Priority

This section explains the steps to setup the Priority property for a processing unit.

Procedure

1. Select **Processing Units** tab.
2. 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.
3. On the right side, in the Agents section, select the row with `gtm-class` agent.
4. Double-click the `Priority` column to set a value. For example, you can set this value to 2.
5. Save the changes to the resource.

Configuration Tasks

This appendix explains a few configuration tasks required for some functionality of the product.

Enabling 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 using:

- Text editor
- Studio

This section explains how to add the CORS filter properties in the `asg_core.cdd` file using a text editor.

Procedure

1. Open the `ASG_HOME/bin/asg_core.cdd` file for editing in a text editor.
2. Set the properties as follows:

```
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 not contain a period (.) or an equal to (=) 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 period (.)
- *filterName* is used both by TIBCO 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 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`

Adding Filter Parameters

To add filter parameters, set the properties by following the structure:

`be.http.filter.cors.param.paramName=paramValue`

where:

- *cors* identifies an instance of the filter.
- *paramName* is the parameter name. This should not contain an 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 for a list of available parameter names and values.

Configuring 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 back-end 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 [Configuring TIBCO Enterprise Message Service](#). You can override the default JMS destinations by configuring the destination and type.

To configure a destination and type for an ESB service type, follow these steps:

Procedure

1. Start the GUI, if it has not already started. See [Starting GUI](#) for details.
2. Add a new service of ESB Type. See [Adding a New Target Operation](#) for details.
3. Set the following parameters to configure a destination for the service:

Parameter	Description
ESB Channel	Define the number of predefined ESB channels.
Destination Name	<p>Define a name of the custom queue or topic destination for JMS channel. This topic or queue should exist on the EMS server.</p> <p>For example, type a queue name as <code>asg.custom.requestQ</code></p> <p>For example, type a topic name as <code>asg.custom.requestT</code></p>
Destination Type	<p>Select the type of the destination for JMS channel from the drop-down list. Valid values are QUEUE or TOPIC. If no value is given, then the QUEUE type is used as default.</p> <p>QUEUE type sets the JMS destination as queue.</p> <p>TOPIC type sets the JMS destination as topic.</p>

4. Save the service configuration.

Configuring 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 back-end service in async mode, no response is expected from the southbound service, and the gateway does not wait for any southbound response from the back-end service

After the async request is sent to the back-end service, a default northbound response payload is created and sent back as the northbound response. The default northbound response is:

```
<?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 :

Procedure

1. Start the GUI, if it has not already started. See [Starting GUI](#) for details.
2. Add a new service of ESB Type. See [Adding a New Target Operation](#) for details.
3. Set the following field for the service as follows:

Mode: ASYNC (select from the drop-down list.)

4. Save the service configuration.

Configuring Retry parameters for HTTP HTTP(s) Transport

TIBCO API Exchange Gateway provides the retry mechanism in case when it fails to send the southbound request message to the back-end 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 and HTTPS service type are available on the services tab of the configuration GUI.

Set the retry parameters for a HTTP and HTTPS service using the configuration GUI:

Procedure

1. Start the GUI, if not already started. See [Starting GUI](#) for details.
2. Add a new service of HTTP and HTTPS type. See [Adding a New Target Operation](#) for details.
3. Set the following fields:

Parameter	Description
Timeout	Set the value in milliseconds which is used as a timeout to use when invoking the back-end service. The indicates the maximum time to wait before the northbound response is returned from a back-end service.
Retry Count	Set the value to use as a number of retries for HTTP connection.
Retry Interval	Set the value to use as an interval between HTTP connection retries. If set as 0, it means no retry is done between HTTP connection retries.
Retry Timeout	Set the value in milliseconds which is used as a timeout value on each attempt of HTTP connection. If set as 0, it means no time to wait for each attempt of HTTP connection.

4. Save the service configuration.

Enabling detail level logging for Gateway

This section explains the steps required to enable the detail level logging for basic authentication.

Procedure

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 follows:

```
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 follows:

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

Configuring TIBCO Enterprise Message Service

This section describes the steps to setup and configure TIBCO Enterprise Messaging services destination required by the Core Engine at run time. This section assumes that TIBCO API Exchange Gateway has access to a running instance of TIBCO Enterprise Messaging Service.

Procedure

1. Make sure the TIBCO EMS server is running.
2. Create the following queues in the TIBCO EMS server using the **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, edit the TIBCO Enterprise Message Service configuration parameters in the `ASG_CONFIG_HOME/asg/asg.properties` file.

For example, use the **tibemsadmin** tool of TIBCO Enterprise Message Service.

3. Navigate to the TIBCO Enterprise Message Service installation directory.
4. Change to the `EMS_HOME/bin` directory.
5. Start the **tibemsadmin** tool, as follows:

```
tibemsadmin server "tcp://host:port" user adminuser
```

6. 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
```

7. 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.

Configuring TIBCO Designer

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.

Procedure

1. Open 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:
 - a) Configure the JMS Connection parameter. Select a pre-configured JMS connection resource.
 - b) Click the **Advanced** tab. For the JMS Application Properties field, select the configured JMS Application Properties resource (for example, JMSPROPERTY1). Click Apply.
 - c) 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 facade **Operations** tab of the gateway configuration UI.
 - d) Click **Apply** and save the project.

Configuring Operations

Procedure

1. Start the Config UI.
2. Click **Facade Operations** tab. Configure a new operation as follows:
 - 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 CDD File

By default, the ESB channels are disabled. To enable the channels, see [Enabling ESB Channels](#).

Updating TIBCO Enterprise Message Service Libraries

To include the TIBCO Enterprise Message Service libraries, perform only one of the following steps:

Procedure

1. 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`
2. Copy the `jms.jar` and `tibjms.jar` files from TIBCO Enterprise Message Service installation (`EMS_HOME/lib`) to `ASG_HOME/lib/ext/tpcl`.

Change the Stack Size

To address the high memory usage requirements of TIBCO API Exchange Gateway server, it is good practice to set a smaller stack size, for example 256 KB.



You must have super user privileges to change the stack size.

Changing the Stack Size Permanently

Procedure

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.

Changing the Stack Size Temporarily

You can change the stack size for the current session at the command line. To do this, 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 Core 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 Core 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.

Parameter	Notes
<i>interface</i>	<p>To specify a value, use the desired IP address.</p> <p>The value for <i>interface</i> 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.</p> <p>The default value for <i>interface</i> is the first available interface provided by the operating system for the machine.</p>
<i>port</i>	<p>To specify a single port use the port number in the listen URL, as shown in this example:</p> <pre>tcp://interface:6000/</pre> <p>The default value is the first available port in the 50000+ range.</p>

Editing the Discover URL and Listen URL (using text editor)

To edit the discover URL and listen URL.

Procedure

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.

Editing the Discover URL and Listen URL (using Studio)

You can edit the discover and listen URL in the Cluster Deployment Descriptor (*asg_core.cdd*).

Procedure

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 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:
`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.
13. Double-click 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.

Generate Private Keys And Public Certificates with OpenSSL

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:

- Generating a self-signed certificate using the OpenSSL toolkit.
- Creating your own certificate authority that you use to sign your own generated request by using the OpenSSL toolkit.

Generating Self-Signed SSL Certificates

Creating 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.

Generating 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, it's 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:

```
$ openssl req -new -key asgserver01.key -out asgserver01.csr
```

This command retrieves the public key from the asgserver01.key key file and prompts the user to gather information to construct a Distinguished Name for your server's identity. Follow the prompts to enter the relevant information which will be incorporated into your certificate request including a Distinguished Name or a DN. Also enter a password that is used to encrypt the CSR.



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:

```
$ 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.

Using the CA.pl (or CA.sh) script you can 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 (perl) or CA.sh (shell) script that is shipped with OpenSSL.

Creating CA Hierarchy

This section explains the steps to create CA hierarchy for your private CA. This is a one time action. After you have created your CA hierarchy, it is used for every key/certificate pair you want to generate and sign with this CA.

Procedure

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.

Creating 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.

Procedure

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`.

Creating 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.

Procedure

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

Procedure

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 follows:

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 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.

Procedure

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 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: `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.

13. Double-click the `asg_core.cdd` file.
14. Edit the properties, as needed. For example, to edit the cluster properties, perform the following tasks:
 - a) Select Cluster tab **General** on the left. On the right, expand the cluster node to edit the following properties:


```
be.engine.cluster.as.discover.url
be.engine.cluster.as.listen.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://127.0.0.1:6000/
be.engine.cluster.as.listen.url=tcp://127.0.0.1:6000-*/
```
15. Save the file.
16. Make a back up copy of the original `asg_core.cdd` file in the `ASG_HOME/bin` directory.
17. 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.

Connection Problem to TIBCO DataGrid

If the IP address of the machine where TIBCO API Exchange Gateway software is installed changes, the Core 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 Core 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] Connecting to
[Metaspace:ASG] on ports [tcp://10.105.178.29:6000-*/] and [tcp://
10.105.178.29:6000/]

asg-caching-core Warning [main] -
[com.tibco.cep.runtime.service.dao.impl.tibas.ASDaoProvider] Failed connecting to
[Metaspace:ASG] with error TIBAS_SYS_ERROR (no_port_available -
ip=10.105.178.29:6000-65535). Retrying in [10] seconds
```

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 Core 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

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.

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.

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*

T

transaction object

An instance that gets created for each request and contains various information related to the processing of that request. The transaction object is deleted once the request processing is complete, for example, when a response is sent back and log reported is completed.