



TIBCO FTL[®] - Enterprise Edition

Administration

Version 7.0.1 | December 2024

Contents

| | |
|--|-----------|
| Contents | 2 |
| About this Product | 14 |
| Product Overview | 15 |
| Package Contents | 17 |
| Directory Structure Reference | 17 |
| Sample Programs Reference | 18 |
| Brief Definitions of Key Concepts | 20 |
| Messaging Concepts | 20 |
| Infrastructure Concepts | 21 |
| Administrative Concepts | 22 |
| Persistence Concepts | 23 |
| FTL Server and Services | 24 |
| FTL Server Executable | 25 |
| FTL Server: Message Broker | 27 |
| FTL Server Configuration | 28 |
| Example Configuration File for FTL Servers | 29 |
| FTL Server Configuration Parameters | 31 |
| Realm Service Configuration Parameters | 61 |
| Persistence Service Configuration Parameters | 66 |
| Bridge Service Configuration Parameters | 72 |
| eFTL Service Configuration Parameters | 74 |
| FTL Administration Utility | 78 |
| Realm Administration Tools | 90 |

| | |
|--|------------|
| Realm Definition Terminology | 91 |
| Application and Endpoint Concepts | 92 |
| Configuration Model | 92 |
| Endpoints | 93 |
| Abilities | 94 |
| Implementation: Endpoints (Micro) | 96 |
| Transports | 96 |
| Transport Connectors | 97 |
| Connectors to Cover Required Abilities | 99 |
| Implementation: Application (Macro) | 100 |
| Application Definitions | 101 |
| Application Instance Definitions | 103 |
| Default Instance and Named Instances | 103 |
| Instance Matching | 103 |
| Instances Determine Subscribers and Durables | 104 |
| Administrative Requirements | 105 |
| Administrative Options | 105 |
| Multiple Transports and Serial Communications | 106 |
| Sample Configuration 1: Default with tibsend and tibrecv | 107 |
| Sample Configuration 2: tibsendex and tibrecvex | 108 |
| Sample Configuration 3: tibrequest and tibreply | 110 |
| Transport Concepts | 112 |
| Transport Definitions Must be Unique | 114 |
| Unitary and Fragmentary Transport Definitions | 114 |
| Related Transports | 115 |
| Bus Topologies | 116 |
| Unitary Mesh | 116 |
| Asymmetric Multicast Topologies | 117 |
| Assembling Larger Topologies from Pair Connections | 118 |
| Pair Connections | 120 |

| | |
|--|------------|
| Listen End and Connect End | 121 |
| Connect Requests are Specific to IP Addresses | 121 |
| Subscriber Interest | 122 |
| Send: Blocking versus Non-Blocking | 123 |
| Shared Memory Transports and One-to-Many Non-Blocking Send | 123 |
| Blocking Send and Inline Mode | 124 |
| Inline Mode for Administrators | 124 |
| Transport Restrictions with Inline Mode | 124 |
| Receive Spin Limit | 129 |
| Polling for Data | 129 |
| Receive Spin Limit Tuning | 130 |
| Inheritance of Receive Spin Limit From Transports to Dispatch | 131 |
| UDP Packet Send Limit | 132 |
| Transport Protocol Types | 134 |
| Dynamic TCP Transport | 134 |
| Dynamic TCP Transport: Parameters Reference | 136 |
| Static TCP Transport | 137 |
| Static TCP Transport: Parameters Reference | 137 |
| Auto Transport | 139 |
| Multicast Transport (mcast) | 139 |
| Multicast Group as Shared Communication Medium | 140 |
| Multicast Transport: Parameters Reference | 140 |
| Multicast Port Ranges | 145 |
| Configuring Multicast Inbox Communication | 147 |
| Multicast Retransmission Suppression | 147 |
| Process Transport (PROC) | 148 |
| Process Transport: Parameters Reference | 148 |
| Shared Memory Transport | 149 |
| Configuring a Shared Memory Transport for Access by Different Users: UNIX | 150 |
| Configuring a Shared Memory Transport for Access by Different Users: Windows | 150 |
| Shared Memory Transport: Parameters Reference | 151 |

| | |
|---|------------|
| Direct Shared Memory Transport | 152 |
| Reliable UDP Transport (RUDP) | 154 |
| RUDP Transport: Parameters Reference | 154 |
| Buffer and Performance Settings for Transports | 157 |
| Formats | 159 |
| Formats: Managed, Built-In, and Dynamic | 159 |
| Defining Formats | 160 |
| Built-In Formats | 162 |
| Affiliated FTL Servers | 163 |
| Server Roles | 163 |
| Modifications to the Realm Definition of Affiliated Servers | 166 |
| Disconnected Satellite | 167 |
| Configuring Satellite FTL Servers | 167 |
| Realm Service | 169 |
| Developing Secure Applications | 170 |
| Ensuring FTL System Security: Tasks for Administrators | 172 |
| Enabling TLS for FTL Server | 173 |
| Enabling TLS with User-defined Certificates | 174 |
| Enabling TLS with FTL-Generated Certificates | 175 |
| Securing Transport Bridges | 176 |
| Securing Persistence Services | 177 |
| Securing eFTL Services | 178 |
| Logging | 179 |
| Authentication and Authorization | 180 |
| Authentication | 181 |
| Using the Built in Flat-File Authentication Service | 182 |
| Using the Built in LDAP Authentication Service | 184 |

| | |
|---|------------|
| Using the HTTP/HTTPS Authentication Service | 187 |
| Using the External JAAS Authentication Service | 187 |
| Using the external custom HTTP / HTTPS based authentication service | 189 |
| Using the Built in mTLS Based Authentication Service | 189 |
| Using the built in OAuth 2.0 based authentication service | 191 |
| Single Sign-On with OAuth 2.0 | 194 |
| Authorization | 195 |
| FTL Server Authorization Groups | 195 |
| Mapping Authorization Groups | 195 |
| Permissions | 197 |
| Configuring Permissions | 199 |
| Required Permissions for API Calls | 201 |
| Migrating to FTL 6.8.0 when Using Permissions | 204 |
| FTL Prometheus Metric Naming | 206 |
| FTL Server and Interfaces | 224 |
| FTL Server GUI Address | 224 |
| Swagger REST API Interface | 224 |
| GUI Overview | 225 |
| GUI Top Bar | 226 |
| GUI Status Tables | 227 |
| GUI Left Column Menu | 228 |
| GUI Grids | 228 |
| GUI Details Panels | 230 |
| Restrictions on Names | 230 |
| Reserved Names | 231 |
| Length Limit | 231 |
| Size Units Reference | 231 |
| Realm Definition Storage | 232 |
| Modification Lock (Edit Mode On) | 232 |
| The Deploy Transaction | 233 |

| | |
|--|------------|
| Successful Test | 236 |
| Realm Modifications Reference | 236 |
| FTL Server GUI: Configuration | 244 |
| Realm Properties Details Panel | 245 |
| Message Formats Administration | 249 |
| Applications Grid | 250 |
| Application Definition Details Panel | 254 |
| Endpoint Details Panel | 255 |
| Transports Grid | 256 |
| Transport Details Panel | 259 |
| Formats Grid | 259 |
| Validation Results | 261 |
| Deployment History | 262 |
| New Clients during a Deployment | 264 |
| FTL Server GUI: Monitoring | 265 |
| Clients Status Table | 265 |
| Client Status | 267 |
| Conditions for Disabling Clients | 268 |
| Client Status Details | 269 |
| FTL Servers Status Page | 271 |
| Realm Services Status Page | 272 |
| FTL Server Web API | 274 |
| HTTP Request Addressing | 274 |
| JSON Attribute Values | 275 |
| HTTP Authentication | 276 |
| HTTP Headers | 276 |
| Response Status Codes | 276 |
| Responses | 277 |
| Pagination | 278 |

| | |
|--|------------|
| Example | 279 |
| Semantics of Web API Objects and Methods | 279 |
| Configuring the Realm Definition Using the Web API | 282 |
| Creating or Modifying a Definition | 283 |
| Workspace | 284 |
| Deployments | 288 |
| Definition Objects | 292 |
| Application Definition Objects | 292 |
| Transport Definition Objects | 302 |
| Persistence Definition Objects | 304 |
| Bridge Definition Objects | 342 |
| Format Definition Objects | 345 |
| Realm Definition and Properties | 347 |
| Status Objects | 355 |
| FTL Server Status Objects | 355 |
| Clients Status Objects | 367 |
| Bridges Status Objects | 374 |
| Group Server Status Object | 376 |
| Persistence Status Objects | 381 |
| Metrics and Monitoring Objects | 407 |
| GET monitoring | 408 |
| Authenticating to FTL Server | 410 |
| Authenticating with Basic Authentication | 410 |
| Authenticating with mTLS | 411 |
| Authenticating with OAuth 2.0 | 411 |
| Catalog of Metrics | 414 |
| Catalog of Application Metrics | 414 |
| Catalog of Persistence Metrics | 418 |
| Catalog of eFTL Metrics | 429 |
| Catalog of FTL Server and Service Metrics | 430 |

| | |
|---|------------|
| Groups of Applications | 432 |
| Introduction to Groups | 432 |
| Group Communication | 432 |
| Group Service | 433 |
| Fault Tolerance of the Group Service | 433 |
| Groups Status Table | 433 |
| Transport Bridge | 436 |
| Transport Bridge Use Cases | 437 |
| Transport Bridge to Shift Fanout for Efficiency | 437 |
| Transport Bridge to Extend beyond a Transport | 438 |
| Transport Bridge: Multiple Transports | 439 |
| Transport Bridge to Cross the Boundary of Shared Memory | 441 |
| Transport Bridge to Isolate Spokes | 442 |
| Transport Bridge Topologies | 442 |
| One Bridge | 442 |
| Two Serial Bridges | 443 |
| Hub-and-Spoke Bridges | 444 |
| Transport Bridge Restrictions | 445 |
| Transport Bridge Configuration | 446 |
| Bridges Grid | 446 |
| Bridge Service | 448 |
| Arranging Fault-Tolerant Bridge Services | 449 |
| Bridges among Dynamic TCP Meshes | 449 |
| Bridges Status Table | 452 |
| Persistence: Stores and Durables | 454 |
| Purposes of Persistence | 454 |
| Persistence Architecture | 455 |
| Persistence Service Disk Capacity | 460 |
| Compact Disk Persistence Files with Persistence Service Online | 462 |
| Compact Disk Persistence Files with Persistence Service Offline | 464 |

| | |
|--|-----|
| Coordination for Persistence | 465 |
| Stores for Delivery Assurance | 465 |
| Delivery Assurance: Topology | 465 |
| Delivery Assurance: Retention and Delivery | 466 |
| Delivery Assurance: Larger Networks of Endpoints | 467 |
| Delivery Assurance: Multiple Stores | 469 |
| Delivery Assurance: Durable Collision | 471 |
| Stores for Apportioning Message Streams | 472 |
| Stores for Last-Value Availability | 473 |
| Stores for Message Broker | 475 |
| Default Durable | 477 |
| Durable Behavior | 477 |
| Types of Durables: Standard, Shared, Last-Value, Map | 478 |
| Standard Durables | 479 |
| Shared Durables | 482 |
| Last-Value Durables | 485 |
| Message Interest | 488 |
| Acknowledgment Mode | 489 |
| Retention Time | 490 |
| Rewinding a Durable | 491 |
| Endpoint Store Inboxes | 491 |
| Inbox Durable Templates | 492 |
| Wide-Area Forwarding | 493 |
| Wide-Area Forwarding Zone Types | 494 |
| Example: Wide-Area Behavior | 497 |
| Arranging a Wide-Area Store | 498 |
| Wide-Area Forwarding: Run Time Conditions | 500 |
| Persistence Services and Clusters | 503 |
| Quorum and Leader | 504 |
| Quorum Conditions: General Rule | 504 |
| Cluster Size | 504 |

| | |
|--|-----|
| Quorum Behaviors | 505 |
| Persistence Service Fault Tolerance | 507 |
| Configuring Persistence | 507 |
| Stores Grid | 509 |
| Store Detail Panel | 514 |
| Durable Details Panel | 515 |
| Clusters Grid | 521 |
| Cluster Details Panel | 525 |
| Persistence Service Details Panel | 531 |
| Zones Grid | 533 |
| Zone Settings Panel | 534 |
| Defining a Static Durable | 535 |
| Enabling Dynamic Durables | 536 |
| Enabling Key/Value Maps | 537 |
| Built-In Dynamic Durable Templates | 537 |
| Configuration of Durable Subscribers in an Application or Instance | 538 |
| Persistence Service Transports | 540 |
| Publisher Mode | 542 |
| Persistence Limits | 545 |
| Memory Reserve for Persistence Services | 546 |
| Starting a Persistence Service | 548 |
| Stopping or Restarting a Persistence Service | 548 |
| Persistence Monitoring and Management | 549 |
| Persistence Clusters Status Table | 549 |
| Persistence Stores Status Table | 554 |
| Before Forcing a Quorum | 560 |
| Disk Persistence Backup and Restore | 561 |
| Saving and Loading Persistence State | 562 |
| Suspending a Persistence Cluster | 563 |
| Saving the State of a Persistence Service | 564 |
| Restarting a Persistence Cluster with Saved State | 565 |

| | |
|--|------------|
| Starting a New Persistence Service with Saved State | 566 |
| Clock Synchronization, Affiliated FTL Servers, and Persistence | 567 |
| Message Tracing | 567 |
| Disaster Recovery | 569 |
| Scope of the Disaster Recovery Feature | 571 |
| Disaster Recovery Prerequisites | 571 |
| DNS Remapping | 572 |
| Persistence Service Sets: Primary and Standby | 572 |
| Data Gaps and WAN Capacity | 574 |
| Preparing FTL Servers for a Disaster Recovery Site | 575 |
| Recovering after Disaster | 577 |
| Fail Back to the Original Site | 578 |
| Disaster Recovery for Routes | 582 |
| Enabling Disaster Recovery for Routed Persistence Clusters | 583 |
| Failover to the Disaster Recovery Site | 584 |
| Planned Failback to the Primary Site | 585 |
| Failover to the Standby Satellite Site | 587 |
| Planned Fail-back to the Active Satellite Site | 587 |
| IPv4 and IPv6 | 590 |
| Multithreading with Direct Publishers and Subscribers | 591 |
| Docker Containerization for FTL | 592 |
| Transports and Docker | 592 |
| Building a Docker Image | 592 |
| Starting a Default FTL Server in a Docker Container | 593 |
| Three Default FTL Servers | 594 |
| Starting a Cluster of FTL Servers in Docker Containers | 594 |
| Coordination Forms | 597 |

| | |
|---|------------|
| Upgrading or Migrating to a New Release | 598 |
| Upgrading from Release 6.x | 598 |
| Upgrading from Release 6.7.1 or Later | 598 |
| Migration With Disk Persistence | 599 |
| Enabling Disk Persistence for the First Time | 601 |
| Migrating to a Different Host | 601 |
| Eliminating the TIBCO FTL Keystore (Authentication Only) | 602 |
| Eliminating FTL-Generated Certificates (Authentication and TLS) | 603 |
| TIBCO FTL Processes as Windows Services | 605 |
| Installing the FTL Server as a Windows Service | 605 |
| Uninstalling a Windows Service | 606 |
| TIBCO Documentation and Support Services | 607 |
| Legal and Third-Party Notices | 609 |

About this Product

TIBCO® is proud to announce the latest release of TIBCO FTL® software.

This release is the latest in a long history of TIBCO products that use the power of Information Bus® technology to enable truly event-driven IT environments. TIBCO FTL software is part of TIBCO Messaging®. To find out more about TIBCO Messaging software and other TIBCO products, please visit us at www.tibco.com.

Product Overview

TIBCO FTL® software is a messaging infrastructure product. It features high speed, structured data messages, and clearly defined roles for application developers and application administrators. TIBCO FTL software can achieve low message latency with consistent performance.

Components

TIBCO FTL software consists of these main components.

API Libraries

Developers use APIs in application programs.

FTL Server

The FTL server is a multi-purpose executable component that can consolidate several services into one server process: realm service, persistence service, transport bridge service, group service, authentication service, and eFTL service.

Combining services can simplify usage, especially in cloud-based installations.

FTL Services

An FTL server can provide one or more of the following services, as specified in its configuration file.

Realm Service

Applications obtain operational metadata from a realm service. The realm service funnels operational metrics from application clients to external monitoring points.

Authentication Service

The realm service can authenticate users against a separate authentication service or an internal authentication service.

Persistence Service

Persistence services store messages for durable subscribers.

Transport Bridge Service

Bridge services forward messages between two endpoints.

eFTL Service

eFTL services forward messages between eFTL clients and FTL clients. With eFTL, the messaging backbone can communicate with a variety of client devices, applications, and other things—smartphones, tablets, browsers, sensors.

Other Messaging Systems

TIBCO FTL can interoperate with the other component systems of TIBCO Messaging software, such as monitoring using [TIBCO® Messaging Monitor for TIBCO FTL®](#).

Package Contents

Directory Structure Reference

The TIBCO FTL installation directory contains the subdirectories in this table.

TIBCO FTL Directories

| Directory | Description |
|-------------|---|
| bin | DLL files (for Windows). FTL server executable. |
| bin/modules | The FTL server manages the executable services of other related products, such as TIBCO eFTL. To enable this functionality, product installers for those products place their service components into the <i>FTL</i> product tree under this directory. |
| include | C API header files (in subdirectories). |
| lib | Library files (for UNIX and macOS). Java .jar files. |
| monitoring | Scripts to start and stop the monitoring gateway and the InfluxDB data base. |
| samples | Sample programs. For more detail, see Sample Programs Reference . |
| scripts | Post-installation scripts for Linux platforms. |

Sample Programs Reference

Sample programs serve as models for your own programs. You can use sample programs to experiment with TIBCO FTL software, test for correct installation, and demonstrate performance in your network environment.

Sample Programs

These tables describe some of the sample programs, however, the list is not exhaustive. For instructions on running the sample programs, and for descriptions of other sample programs, see the readme files in each directory.

Each sample program prints a usage summary in response to the `-h` command line option.

Sample Programs for One-to-Many Communication

| Programs | Description |
|-----------|---|
| tibrecvex | tibrecvex creates a subscriber and outputs the messages it receives. Start this program first. |
| tibsendex | tibsendex publishes messages for tibrecvex. |

Sample Programs for One-to-One Communication

| Programs | Description |
|------------|--|
| tibreply | tibreply creates a subscriber, and replies to request messages it receives. Start this program first. |
| tibrequest | tibrequest creates an inbox subscriber, sends request messages to tibreply, and receives replies at its inbox. |

Files

Files in Samples Directory

| File | Description |
|------------|---|
| readme.txt | Instructions to compile and run sample programs. |
| setup | Script that defines environment variables for the samples, such as PATH, LD_LIBRARY_PATH and CLASSPATH. Start with this script. |

Directories

Samples Directories

| Directory | Description |
|-----------|--|
| bin | Executable sample programs, precompiled from C sources. |
| config | More sample configuration files, for use as instructional models, but not related to the sample programs. |
| docker | Docker resources and information. |
| helm | Sample to deploy FTL on a Kubernetes cluster with the default configuration. |
| jaas | Files that configure JAAS authentication and authorization for the sample scripts. |
| scripts | <p>Sample scripts to start and stop the FTL server. Before running the sample programs you <i>must</i> start the FTL server with this script.</p> <p>Sample configuration file for the FTL server.</p> <p>The readme.txt file contains instructions for using these samples.</p> |
| src | <p>Sample program source code in C, Java, and .NET. You may use these samples as models for your application programs.</p> <p>The readme.txt file contains instructions for compiling.</p> |
| yaml | FTL server samples. The readme.txt file contains instructions. |

Brief Definitions of Key Concepts

These brief definitions are reminders of the key concepts of TIBCO FTL software.

For a more intuitive picture of TIBCO FTL software and its concepts, please read [TIBCO FTL Concepts](#) as your first introduction to this product.

Messaging Concepts

Application programs (clients) use TIBCO FTL software to communicate by sending messages to one another. Programs send messages using *publisher* objects, and receive messages using *subscriber* objects.

FTL typically uses a message broker architecture. For low latency, FTL can be configured such that messages travel directly between peer application programs.

Messages and Fields

A *message* is a structured unit of data. The structure is flexible, and developers tailor the structure to the needs of the application.

Each message consists of a set of named *fields*. Each field can contain a value of a specific data type.

One-to-Many Communication

Publishers can send messages using *one-to-many* communication. These messages can reach potentially many subscribers.

A subscriber can use a *content matcher* to receive only those messages in which a particular set of fields contain specific values.

One-to-One Communication

For efficient *one-to-one* communication, publishers can also send messages that can reach one specific *inbox subscriber*. The *inbox* data structure uniquely identifies an

inbox subscriber.

Inbox subscribers cannot use content matchers. An inbox subscriber receives all messages directed to it.

Infrastructure Concepts

Applications and FTL servers communicate through an infrastructure and FTL topology.

Endpoint

An *endpoint* is the connection point for publishers and subscribers in communicating programs. Application architects and developers determine the set of endpoints that an application requires. Administrators formally define those endpoints. Programs create publisher and subscriber objects using those endpoints.

Transport

A *transport* is the underlying medium that moves message data between endpoints. For example, TCP transports and multicast transports can move messages among programs connected by a network, and a shared memory transport can move messages among separate program processes on a multi-core hardware platform.

Affiliated Servers and Sites

For larger-scope messaging systems, FTL servers at different geographic sites can cooperate to distribute services. These affiliated servers are typically arranged as a *primary* site with one or more *satellite* sites. Each primary or satellite site can consist of multiple cooperating *core* and *auxiliary* servers. A third type of site, Disaster Recovery (*DR*) site, serves as a complete backup/standby site in the system.

Persistence Stores and Durables

You can use persistence stores and durables to strengthen message delivery assurance, to apportion message streams among a set of cooperating subscribers, to provide last-value availability, to provide a key/value map, or to simply serve as a message broker. Stores can be replicated and/or written to disk within a persistence service to provide persisted messaging.

Persistence Clusters

Multiple persistence services can form a persistence cluster, where the persistence services form a quorum with an active persistence service, and backup, synchronized standby persistence services.

Administrative Concepts

A *realm* is a namespace that defines resources and configurations available to application programs. (Those resources include endpoints, transports, and formats.)

Administrators can use the following types of interfaces to define the realm:

- FTL Administrative GUI (Graphical User Interface)
- REST API (either via a command line tool like curl, or the Swagger extension on the GUI: REST API Reference)
- YAML file configuration options, which are read at FTL server startup
- FTL server command-line options (typically within a script)

The TIBCO FTL base library caches the realm definition in a *realm object* and consults the local realm object for information about endpoints, transports, and message formats. Application programs connect to the *realm service* to read the relevant parts of the realm definition. If the realm definition changes, the realm service updates its application clients.

Realm definition parameters fall into the following categories:

- Realm Properties - General and common settings for the entire realm
- Applications - Client programs that utilize publish, subscribe, monitoring, or other FTL client API features.
- Transports - Note that transports are not visible to application developers, rather, they are strictly the responsibility of the administrator. From the developer's perspective, these details are hidden behind endpoints.
- Persistence - Configuration of stores, persistence clusters, and zones.
- Bridges - A service that forwards messages among sets of transport buses.
- Formats - Definitions of message formats

Architects, administrators, and developers coordinate the detailed requirements of applications using *coordination forms* (see [Coordination Forms](#)).

When configuration changes are edited and stored, they then can be deployed, per a controlled deployment procedure.

Persistence Concepts

The realm definition can also include persistence-service definitions to manage stores and durables and their organization into a hierarchy of clusters and zones. Persisted messaging provides for reliable delivery of messages in the event subscribers become temporarily unavailable. You can tune the depth and scope of persistence via configuration options and administrative settings.

For detailed information, see [Persistence: Stores and Durables](#).

FTL Server and Services

Release 6.0 introduced a new FTL server component, which consolidates all of the separate executable components from earlier releases (such as the realm server, persistence server, and transport bridge). This topic defines the new terminology.

Terminology

FTL Server

An FTL server is the consolidated server executable that provides and manages one or more services. Management includes starting a service, and automatically restarting it when it is not running.

Service

Each service is a module that runs within an FTL server. Each service implements part of the functionality available in TIBCO FTL.

Examples of FTL services include the realm service, persistence service, transport bridge service, eFTL service, authentication service, and group service.

Message Broker

A message broker is an FTL server that provides server-based messaging using a message broker pattern.

Core Server

A *core server* is an FTL server that you designate in the list of `core.servers`.

A set of core servers forms a quorum.

A set of core servers can cooperate for fault tolerance, or a single core server can operate solo. The number of core servers must be odd.

A core server implicitly provides a group service.

Auxiliary Server

An *auxiliary server* is an FTL server that you designate and configure for a specialized purpose, and do *not* specify in the list of `core.servers`.

For example, an auxiliary server might provide an eFTL service to expand capacity, or a backup bridge service for fault tolerance. You can use auxiliary servers during migration procedures.

FTL Server Executable

Each FTL server requires a minimum of 2GB of free disk space. In addition, each persistence service using disk persistence or swap requires an additional minimum of 1GB of free disk space. For details on these features, see [Persistence Architecture](#).

Administrators use `tibftlserver`, the FTL server command line executable, to start an FTL server process. Each time an FTL server is started, the FTL server process must be executed by the same user. Each time an FTL server is started, the FTL server process must be executed by the same user.

| Parameter | Arguments | Description |
|-----------------|----------------------|--|
| -h --help | | The FTL server prints a usage message summarizing its command line parameters. |
| -c --config | <path> | The FTL server reads the configuration file at <path>. |
| -n --name | <name> | <p>This parameter designates the name of the FTL server.</p> <p>This name selects one of the core servers.</p> <p>The FTL server monitoring GUI displays the name you supply so you can identify individual FTL server processes.</p> <p>To run an auxiliary server in a Docker container with non-default host and port values, specify the name argument with this form:</p> <pre><name>@<host>:<port></pre> |
| --init-security | <password_specifier> | When present, the FTL server generates new TLS data, and then immediately exits. |

| Parameter | Arguments | Description |
|-------------------------------|-------------------------------|---|
| | | <p>The TLS data files include <code>ftl-trust.pem</code> (trust) and <code>ftl-tport.p12</code> (keystore).</p> <p>The FTL server writes the TLS data files to its data directory. If it cannot write to the data directory, the server writes these files to the current directory.</p> <p>When this flag is present, the FTL server ignores its <code>--name</code> argument.</p> <p>The FTL server uses the <code><password_specifier></code> argument to secure the keystore file. For details, see "Password Security."</p> <p>If the FTL server detects existing TLS files, it does not generate them anew. However, the FTL server does not decrypt or inspect existing files.</p> |
| <code>--init-auth-only</code> | | <p>When present, the FTL server generates new authentication data, and then immediately exits.</p> <p>The security data files include <code>ftl-trust.pem</code> (trust) and <code>ftl-tport.p12</code> (keystore).</p> <p>The FTL server writes the data files to its data directory. If it cannot write to the data directory, the server writes these files to the current directory.</p> <p>For details, see Authentication and Authorization .</p> <p>If the FTL server detects existing authentication files, it does not generate them anew. However, the FTL server does not decrypt or inspect existing files.</p> |
| <code>--core.servers</code> | <code><URL_list></code> | <p>Optional.</p> <p>This parameter is a convenience for use with Docker.</p> <p>When starting an FTL server in a Docker image using a default empty configuration file, specify the cluster of core servers in this <code><URL_list></code>.</p> |

| Parameter | Arguments | Description |
|--------------------------|-----------|--|
| | | <p><URL_list> is a pipe-separated list of core servers. Each server specification has this form:</p> <pre><name>@<host>:<port></pre> |
| -l --loglevel <level> | | <p>When present, the FTL Server logs at this level of detail. Supply one of these strings:</p> <ul style="list-style-type: none"> • off • severe • warn • info • verbose • debug <p>Note that the output from verbose and debug can result in a very large output, and this level of detail is generally not useful unless TIBCO staff specifically requests it.</p> <p>When this option is absent, the default value is info.</p> |

FTL Server: Message Broker

Default values configure an FTL server to operate as a message broker.

That is, when you run one FTL server without supplying any configuration file, it automatically operates as a message broker. Moreover, when you run three FTL servers without supplying any configuration file, they automatically join together to operate as a message broker.

- Each FTL server provides a realm service and a persistence service.
- The persistence services arrange themselves in a default cluster.
- The default realm definition includes a set of default persistence stores.
- The realm defines a default application which defines several endpoints, offering a

mix of store and durable types.

- Any application definition can define an endpoint that uses one of the default persistence stores. Additionally, for message broker operation, the endpoint should have no direct path transport (in the administrative GUI the absence of a direct-path transport is displayed as a "Server" transport). An application that uses an endpoint like this will automatically connect with the persistence services of the default message broker. For more information, see [Configuring Persistence](#).
- Clients connecting to any of the FTL servers can send and receive messages through the message broker.

Non-Default Message Broker

To use a non-default message broker, see:

- [Persistence Services and Clusters](#)
- [Configuring Persistence](#)
- [Stores for Message Broker](#)

FTL Server Configuration

Specify all the values for a localized cluster of cooperating FTL servers, along with the services they provide in a single YAML formatted configuration file. A cluster of three or five servers in a quorum are recommended. A cluster of seven servers is not supported.

One FTL server command line parameter specifies the file name of the configuration file. A second command line parameter selects one named configuration from within that file.

For clarity, specify sections in the order listed in the following table.

i Note: Configuration file examples in the documentation are for explanatory purposes. If you copy and paste excerpts from these examples, ensure that the resulting configuration file obeys correct YAML syntax and specifies a valid configuration.

i Note: Data directories and files used by FTL server should not be read/write accessible by non-privileged users.

Example Configuration File for FTL Servers

```
# The "globals" section defines parameters that apply to all FTL servers
# in the cluster.
globals:
  # The "core.servers" map defines which FTL servers participate in a
  # quorum
  # for managing realm configuration.
core.servers:
  ftlserver1: localhost:8585
  ftlserver2: localhost:8685
  ftlserver3: localhost:8785
# The "servers" section defines parameters that apply to a specific
# FTL server and its services. Include an entry for each core server
# Auxiliary servers are optional.
servers
  # Configuration for ftlserver1.
  ftlserver1:
    # The "ftlserver.properties" element defines parameters that apply
    # to the entire FTL server.
    - ftlserver.properties:
      # Location of the log file.
      logfile: TIBCO_HOME/ftlserver1/logs/ftlserver1.log
# The "realm" element defines parameters that apply to the realm
service.
- realm:
  # Location for storing realm configuration data.
  data: TIBCO_HOME/ftlserver1/realm/data
# Define optional services by adding "bridge", "persistence", or "eftl"
# elements.
```

```
# Configuration for ftlserver2.
ftlserver2:
  # The "ftlserver.properties" element defines parameters that apply
  # to the entire FTL server.
  - ftlserver.properties:
    # Location of the log file.
    logfile: TIBCO_HOME/ftlserver1/logs/ftlserver2.log
# The "realm" element defines parameters that apply to the realm
service.
- realm:
  # Location for storing realm configuration data.
  data: TIBCO_HOME/ftlserver1/realm/data
```

```
# Define optional services by adding "bridge", "persistence", or "eftl"
# elements.
```

```
# Configuration for ftlserver3
ftlserver3:
  # The "ftlserver.properties" element defines parameters that apply
  # to the entire FTL server.
  - ftlserver.properties:
    # Location of the log file
    logfile: TIBCO_HOME/ftlserver1/logs/ftlserver3.log
  # The "realm" element defines parameters that apply to the realm
  service.
  - realm:
    # Location for storing realm configuration data.
    data: TIBCO_HOME/ftlserver1/realm/data
  # Define optional services by adding "bridge", "persistence", or "eftl"
  # elements.
```

```
# Auxiliary FTL servers (additional servers to expand capacity) may be
# optionally defined
```

Sections in the FTL Server Configuration File

| Section | Description |
|----------|--|
| globals: | <p>Required.</p> <p>The globals section can contain key/value pairs that directly affect the operation of FTL servers.</p> |
| servers: | <p>Optional.</p> <p>The servers section contains a map of the FTL servers you are configuring. Each item in the map pairs a server name with a list of services that the server provides. The map <i>must</i> contain an item configuring each core server, and may also include optional items configuring auxiliary servers.</p> <p>Each FTL server item in this map begins with a unique <i>name</i>. The FTL server command line parameter <code>--name</code> selects one of the server names from this map. The resulting process instantiates an FTL server with that name.</p> |

| Section | Description |
|---------|---|
| | For each server, configure a list of services that that server provides. An FTL server provides a service if and only if the service is configured at this level. |
| | You may omit the servers section if the file specifies only one core server, and specifies its extended server name with host and port. |


FTL Server Configuration Parameters

Parameters in the “globals” section of the configuration file apply to all FTL servers in the cluster, including all core and auxiliary servers.

Parameters in the “ftlserver.properties” item apply to the FTL server as a whole. The “ftlserver.properties” item is part of the definition of one FTL server in the “servers” section of the configuration file.

Certain parameters must be defined in the “ftl” item instead. These also apply to the FTL server as a whole. The “ftl” item is part of the definition of one FTL server in the “servers” section of the configuration file. In general, the “ftl” item is used only to define auxiliary servers.

Refer to the tables below to determine where each parameter belongs.

 **Important:** Do not use relative paths.

Servers and Locations

| Parameter | Location | Arguments | Description |
|--------------|----------|-----------|--|
| core.servers | globals | servers | Required. Supply a map from server names to their locations. Use the following form for each server: |

| Parameter | Location | Arguments | Description |
|------------|----------|----------------|--|
| | | | <pre><server_name>: <host>:<port></pre> <p>Each location is a connect address, used by other FTL servers in the cluster when connecting to the given FTL server.</p> <p>By default, FTL server binds all interfaces on the specified port. However, if <host> is “localhost”, FTL server binds only “localhost”.</p> <p>Include in this map only core servers. Omit any auxiliary servers.</p> <p>For more information, see “Core Servers”</p> |
| server | ftl | <host>:<port> | <p>Specify the location of an auxiliary server. The location is a connect address, used by other FTL servers in the cluster when connecting to the given FTL server.</p> <p>By default, FTL server binds all interfaces on the specified port. However, if <host> is “localhost”, FTL server binds only “localhost”.</p> <p>For more information, see “Auxiliary Servers”</p> |
| spin.limit | ftl | <microseconds> | <p>Optional.</p> <p>Set the limit for the amount of spin time , in microseconds, for an</p> |

| Parameter | Location | Arguments | Description |
|-----------|----------|-----------|--|
| | | | FTL server receiving data from a connection. The default is 0. |
| | | | Use caution with this parameter and adjust it only when necessary and with full understanding of its effect on CPU and thread performance. The default value should provide optimal performance in most cases. |

Authentication Providers

When one or more authentication providers are configured, authentication is enabled for the REST API, UI, realm connections from FTL clients, and connections between affiliated FTL servers. Authentication for other services (for example, persistence services or ectl services) may be optionally enabled through the realm configuration. For more information, see [Authentication and Authorization](#)



Important: Files and passwords should not be read/write accessible to non-privileged use

| Parameter | Location | Arguments | Description |
|----------------|----------|-----------------|---|
| auth.providers | globals | <provider_list> | <p>Optional. Set this parameter to enable authentication.</p> <p>The <provider_list> is a comma-separated list of one or more auth providers.</p> <p>The following are valid providers. Specify at most one of each type.</p> <ul style="list-style-type: none"> • “file:<path>” |

| Parameter | Location | Arguments | Description |
|---------------|----------|-------------|---|
| | | | <ul style="list-style-type: none"> • “http://<host>:<port>/<path>” or “https://<host>:<port>/<path>” • “ldap://<host>:<port>” or “ldaps://<host>:<port>” • “mtls” • “oauth2” <p>Example: “file:/path/users.txt,oauth2”</p> <p>For information about how to use each provider type, see Authentication</p> |
| auth.user | globals | <user_name> | <p>Optional.</p> <p>The FTL server identifies itself to an external authentication service using this username credential.</p> <p>This parameter is used only for the http(s) and ldap(s) auth providers.</p> |
| auth.password | globals | <password> | <p>Optional.</p> <p>The FTL server identifies itself to an external authentication service using this password credential.</p> <p>This parameter is used only for the http(s) and ldap(s) auth providers.</p> <p>To hide the password from</p> |

| Parameter | Location | Arguments | Description |
|--------------|----------------------|-----------|--|
| | | | casual observers, see Password Security |
| auth.trust | globals | <path> | <p>Optional.</p> <p>When using an https authentication provider, use this parameter to specify the location of the authentication service's public certificate file (in PEM format). The FTL server uses the certificate to verify the identify of the external authentication service.</p> <p>If not specified, FTL server uses the system trust store to verify the identity of the authentication service.</p> <p>This parameter is only used for https auth providers.</p> |
| auth.timeout | globals | <seconds> | <p>Optional.</p> <p>Timeout for requests to a remote authentication service, in seconds. If the remote authentication service does not respond after this period, the FTL server returns an error. The default value is 15 seconds.</p> |
| auth.rolemap | ftlserver.properties | <path> | <p>Optional.</p> <p>This file can be used to map an authorization group returned by an auth provider</p> |

| Parameter | Location | Arguments | Description |
|-----------------------|----------------------|-----------|--|
| | | | <p>to different authorization groups. If mapped, FTL server will consider the user to be a member of both the original and the mapped groups.</p> <p>For example, a group from LDAP might be mapped to the “ftl-admin” group in a particular FTL environment.</p> <p>You may specify different mappings for different auth providers. The format of this file is described in “Mapping Authorization Groups”. Also see FTL Server Authorization Groups</p> |
| ldap.config | globals | <path> | <p>Required when the “ldap(s)” authentication provider is configured. The file specifies configuration for connecting to the LDAP server and authenticating user. The contents of this file are described in Using the Built in LDAP Authentication Service</p> |
| tls.server.trust.file | ftlserver.properties | <path> | <p>Required when the “mtls” auth provider is configured.</p> <p>When clients or other FTL servers connect to this FTL server using mTLS authentication, the incoming connection will include a client certificate.</p> |

| Parameter | Location | Arguments | Description |
|-----------------------|----------|-----------|---|
| | | | <p>This file contains the trust certificate(s) used by FTL server to verify the incoming client certificate.</p> <p>When the “mtls” provider is configured, you must also enable TLS user-defined certificates by specifying “tls.server.cert”.</p> <p>For more information, see Using the Built in mTLS Based Authentication Service</p> |
| oauth2.claim.roles | globals | <string> | <p>Required when the “oauth2” authentication provider is configured.</p> <p>When validating an oauth2 access token, FTL server will interpret this claim as an array of authorization groups.</p> <p>Default: group</p> |
| oauth2.claim.username | globals | <string> | <p>Required when the “oauth2” authentication provider is configured.</p> <p>When validating an oauth2 access token, FTL server will interpret this claim as a username.</p> <p>Default: preferred_username</p> |
| oauth2.audience | globals | <string> | <p>Required when the “oauth2” authentication provider is configured.</p> |

| Parameter | Location | Arguments | Description |
|-----------------------|----------------------|-----------|---|
| | | | When validating an access token, FTL server will ensure that the audience (“aud”) claim matches this value. If it does not match, the token is rejected. |
| oauth2.validation.key | ftlserver.properties | <url> | <p>Required when the “oauth2” authentication provider is configured. When a client or another FTL server authenticates to this FTL server using an oauth2 access token</p> <p>FTL server will validate the signature of the access token using one of the given validation keys.</p> <p>The <url> specifies the location of the validation key(s). There are two valid forms:</p> <ul style="list-style-type: none"> • “file:<path>” • “http://<host>:<port>/<path>” or “https://<host>:<port>/<path>” <p>If a “file” url is specified, the file may be a PEM file or a JWKS file.</p> <p>If an “http(s)” url is specified, FTL server will fetch the JWKS file by making an http(s) request to the specified</p> |

| Parameter | Location | Arguments | Description |
|----------------------------|----------------------|-----------|--|
| | | | <p>url.</p> <p>For “https” urls, “oauth2.provider.trust.file” may be specified as well.</p> |
| oauth2.provider.trust.file | ftlserver.properties | <path> | <p>Optional.</p> <p>If “oauth2.validation.key”, “oauth2.ui.endpoint.auth”, “oauth2.ui.endpoint.token”, or “oauth2.srv.endpoint.token” is an https url, this parameter specifies the location of the oauth2 provider’s public certificate file (in PEM format). The FTL server uses the certificate to verify the identity of the oauth2 provider.</p> <p>If not specified, FTL server uses the system trust store to verify the identity of the oauth2 provider.</p> |
| disable.default.security | globals | | <p>Optional.</p> <p>This parameter is intended for backward compatibility with FTL 6.x in certain unusual situations. If you are upgrading normally from FTL 6.x, there is no need to use this parameter.</p> <p>This parameter only takes effect when there is no pre-existing realm database, authentication is enabled, and</p> |

| Parameter | Location | Arguments | Description |
|-----------|----------|-----------|--|
| | | | <p>TLS is not enabled.</p> <p>When set, the default persistence cluster and the group service do not use secure transports by default. This preserves compatibility with 6.x clients.</p> <p>Set this parameter only if you are starting FTL servers with no pre-existing realm configuration, where authentication is enabled and TLS is not enabled, and where compatibility with 6.x clients is required.</p> |

Single Sign-On with OAuth2

By default, users of the UI must authenticate directly to FTL server using basic authentication (username/password). To enable single sign-on via an oauth2 provider, use the following parameters. The “oauth2” auth provider must also be configured.

| Parameter | Location | Arguments | Description |
|--------------------------|----------|-----------|--|
| oauth2.ui.endpoint.auth | globals | <url> | <p>Required for SSO with oauth2.</p> <p>Specifies the URL of the oauth2 auth endpoint.</p> <p>For “https” urls, you may need to specify “oauth2.provider.trust.file”</p> |
| oauth2.ui.endpoint.token | globals | <url> | Required for SSO with oauth2. |

| Parameter | Location | Arguments | Description |
|---------------------------|----------------------|------------|--|
| | | | <p>Specifies the URL of the oauth2 token endpoint.</p> <p>For “https” urls, you may need to specify “oauth2.provider.trust.file”.</p> <p>.</p> |
| oauth2.ui.client.id | ftlserver.properties | <string> | <p>Required for SSO with oauth2.</p> <p>Specifies the oauth2 client id to use for authorization code</p> <p>Specifies the oauth2 client id to use for authorization code flow with the UI.</p> |
| oauth2.ui.client.secret | ftlserver.properties | <password> | <p>Required for SSO with oauth2.</p> <p>Specifies the oauth2 client secret to use for authorization code flow with the UI.</p> <p>To hide the password from casual observers, see Password Security.</p> |
| oauth2.ui.endpoint.logout | string | <password> | url of the oauth2 logout endpoint; if not specified, FTL UI with oauth2 may not logout properly |
| oauth2.validation.key | ftlserver.properties | <url> | Required when the “oauth2” authentication provider is configured. When a client or another FTL server authenticates to this FTL server using an oauth2 access token |

| Parameter | Location | Arguments | Description |
|-----------|----------|-----------|---|
| | | | <p>FTL server will validate the signature of the access token using one of the given validation keys.</p> <p>The <url> specifies the location of the validation key(s). There are two valid forms:</p> <ul style="list-style-type: none"> • “file:<path>” • “http://<host>:<port>/<path>” or “https://<host>:<port>/<path>” <p>If a “file” url is specified, the file may be a PEM file or a JWKS file.</p> <p>If an “http(s)” url is specified, FTL server will fetch the JWKS file by making an http(s) request to the specified url.</p> <p>For “https” urls, “oauth2.provider.trust.file” may be specified as well.</p> |

Authenticating to other FTL Servers

When authentication is enabled, FTL server must authenticate itself to other FTL servers. FTL server must be configured to do basic authentication, mTLS authentication, or oauth2 authentication. It is an error to specify multiple authentication modes for outgoing connections. (It is not an error to specify multiple authentication providers for incoming connections.)

For more information, see [Authenticating to FTL Server](#)

| Parameter | Location | Arguments | Description |
|-----------|----------------------|------------|---|
| user | ftlserver.properties | <username> | <p>Used for basic authentication to other FTL servers.</p> <p>If specified alongside “oauth2.svr.endpoint.token”, this parameter is instead used for the oauth2 password credentials grant.</p> <p>It is an error to specify this parameter alongside “tls.client.cert” or “oauth2.access.token”.</p> |
| password | ftlserver.properties | <password> | <p>Used for basic authentication to other FTL servers.</p> <p>Used for basic authentication to other FTL servers. Required when “user” is present.</p> <p>If specified alongside “oauth2.svr.endpoint.token”, this parameter is instead used for the oauth2 password credentials grant.</p> <p>It is an error to specify this parameter alongside “tls.client.cert” or “oauth2.access.token”.</p> <p>To hide the password</p> |

| Parameter | Location | Arguments | Description |
|------------------------|----------------------|-----------|--|
| | | | <p>from casual observers, see “Password Security”.</p> |
| tls.client.cert | ftlserver.properties | <path> | <p>Used for mTLS authentication to other FTL servers.</p> <p>Specifies the location of the identity certificate used when FTL server connects as a TLS client to another FTL server. The file must be in PEM format. Intermediate certificates may be appended to the file.</p> <p>It is an error to specify this parameter alongside “user”, “oauth2.svr.endpoint.token”, or “oauth2.access.token”.</p> |
| tls.client.private.key | ftlserver.properties | <path> | <p>Used for mTLS authentication to other FTL servers.</p> <p>The file must contain the private key corresponding to the identity certificate in “tls.client.cert”. The file must be in PEM format.</p> <p>The private key may be</p> |

| Parameter | Location | Arguments | Description |
|---------------------------------|----------------------|------------|--|
| | | | <p>encrypted or unencrypted.</p> <p>It is an error to specify this parameter alongside “user”, “oauth2.svr.endpoint.token”, or “oauth2.access.token”</p> |
| tls.client.private.key.password | ftlserver.properties | <password> | <p>Used for mTLS authentication to other FTL servers. Required when “tls.client.private.key” contains an encrypted private key.</p> <p>The password is used to decrypt the private key.</p> <p>It is an error to specify this parameter alongside “user”, “oauth2.svr.endpoint.token”, or “oauth2.access.token”.</p> <p>To hide the password from observers, see Password Security</p> |
| oauth2.svr.endpoint.token | globals | <url> | <p>Used for oauth2 authentication to other FTL servers.</p> <p>FTL server will fetch an access token from this endpoint</p> |

| Parameter | Location | Arguments | Description |
|--------------------------|----------------------|------------|---|
| | | | <p>For “https” urls, you may need to specify “oauth2.provider.trust.file”.</p> <p>It is an error to specify this parameter alongside “tls.client.cert” or “oauth2.access.token”.</p> |
| oauth2.svr.client.secret | ftlserver.properties | <password> | <p>Used for oauth2 authentication to other FTL servers. Required when “oauth2.svr.endpoint.token” is specified.</p> <p>Specifies the oauth2 client secret for the oauth2 client credentials grant. If “user” and “password” are specified, the oauth2 password credentials grant is used instead.</p> <p>It is an error to specify this parameter alongside “tls.client.cert” or “oauth2.access.token”.</p> <p>To hide the password from observers, see Password Security</p> |
| oauth2.access.token | ftlserver.properties | <password> | Used for oauth2 authentication to other FTL servers. |

| Parameter | Location | Arguments | Description |
|-----------|----------|-----------|--|
| | | | <p>Instead of fetching an access token from an oauth2 endpoint, FTL server will use this access token as long as it is running. The access token must be long-lived.</p> <p>It is an error to specify this parameter alongside “user”, “tls.client.cert” or “oauth2.svr.endpoint.token”.</p> <p>To hide the password from observers, see Password Security</p> |

TLS Security

TLS can be enabled for FTL server using FTL-generated certificates or user-defined certificates. It is an error to configure both. When enabling TLS, authentication must also be enabled.

When set, by default TLS is enabled for the REST API, UI, realm connections from FTL clients, connections from eFTL clients, and connections between affiliated FTL servers. TLS for other services (for example, persistence services) may be optionally enabled through the realm configuration. For more details, see [Enabling TLS for FTL Server](#).

| Parameter | Location | Arguments | Description |
|------------|----------|------------|---|
| tls.secure | globals | <password> | When present, TLS is enabled for FTL server using FTL-generated |

| Parameter | Location | Arguments | Description |
|-----------------|----------------------|-----------|--|
| | | | certificates. The password argument decrypts the TLS key file, which was generated before starting the FTL server. It is an error to combine this with “tls.server.cert”. For more information, see Enabling TLS for FTL Server . To hide the password from observers, see Password Security |
| tls.server.cert | ftlserver.properties | <path> | <p>When present, TLS is enabled for FTL server using user-defined certificates. The file must contain the FTL server's identity certificate in PEM format. Intermediate certificates, also in PEM format, may be appended to the file.</p> <p>The certificate must have one or more subject alternative names. When clients or other FTL servers connect to this FTL server, they will compare the host in</p> |

| Parameter | Location | Arguments | Description |
|---------------------------------|----------------------|------------|---|
| | | | <p>their connect address to the set of subject alternative names. One of the names must match. One level of wildcarding is permitted in DNS names.</p> <p>It is an error to combine this with "tls.secure".</p> |
| tls.server.private.key | ftlserver.properties | <path> | <p>Required when "tls.server.cert" is specified.</p> <p>The file must contain the private key corresponding to the identity certificate in "tls.server.cert". The file must be in PEM format.</p> <p>The private key may be encrypted or unencrypted.</p> |
| tls.server.private.key.password | ftlserver.properties | <password> | <p>Required when "tls.server.private.key" contains an encrypted private key.</p> <p>The password is used to decrypt the private key. To hide the password from</p> |

| Parameter | Location | Arguments | Description |
|--------------------|----------|-----------|--|
| | | | observers, see Password Security |
| tls.security.level | globals | <level> | <p>Optional.</p> <p>When specified, set the openssl security level for TLS connections. The <level> must be an integer from 1 to 5.</p> <p>When absent, the default value is 2.</p> <p>In general, a higher security level will cause FTL to enforce stricter requirements for key sizes and encryption. Refer to the openssl documentation for full details.</p> <p>For example, if an RSA key is too small for the given security level, FTL will reject the TLS connection.</p> <p>The openssl security level applies to:</p> <ul style="list-style-type: none"> • TLS connections among FTL clients and FTL servers |

| Parameter | Location | Arguments | Description |
|------------------------------------|-----------------------------------|---------------------------|--|
| | | | <ul style="list-style-type: none"> • TLS connections accepted by FTL server for the REST API, UI, or eFTL clients |
| <code>tls.client.trust.file</code> | <code>ftlserver.properties</code> | <code><path></code> | <p>Optional.</p> <p>It is an error to specify this parameter when using FTL-generated certificates (i.e., when “tls.secure” is set). When using FTL-generated certificates, administrators should place the “ftl-trust.pem” file in the realm data directory of the FTL server. See Enabling TLS for FTL Server</p> <p>Otherwise, when using user-defined certificates (“tls.server.cert”), this file identifies the trust certificate(s) when FTL server connects as a TLS client to other FTL servers. The file must be in PEM format.</p> |

| Parameter | Location | Arguments | Description |
|--------------------|----------|-----------|--|
| | | | When absent, FTL server will load the system trust store when connecting to other FTL servers. |
| tls.trust.everyone | globals | | <p>Optional. The FTL server trusts any other FTL server without verifying trust in the other service's certificate.</p> <p>Warning: Do not use this parameter except for convenience in development and testing. It is not secure.</p> |

REST Requests

| Parameter | Location | Arguments | Description |
|-----------------|----------|-----------|---|
| origins.allowed | globals | <urls> | <p>When absent, the default behavior rejects all cross-origin requests.</p> <p>When present, allow REST requests that originate from URLs in an array of strings. The realm service rejects cross-origin requests from any other origin.</p> <p>The special value * allows all origins.</p> |

| Parameter | Location | Arguments | Description |
|-----------|----------|-----------|---|
| | | | Warning: Do not supply this parameter except when necessary. Supplying a null string in the argument list of this parameter is not a recommended security practice. |

Affiliated FTL Servers

For information about disaster recovery servers, see [Disaster Recovery Prerequisites](#)

| Parameter | Location | Arguments | Description |
|--------------|----------|------------|---|
| satellite of | globals | <urls> | <p>When present, designate this FTL server as a satellite of a set of primary FTL servers.</p> <p>Supply a pipe-separated URL list of primary FTL servers. For example:</p> <p>https://<host1>:<port1> https://<host2>:<port2> https://<host3>:<port3></p> <p>A satellite receives its realm definition and accepts realm updates from any primary realm service.</p> <p>The realm service in a satellite does not accept client connection requests until it first receives a realm definition from a primary.</p> |
| drfor | globals | <URL_list> | <p>When present, this FTL server starts as a disaster recovery server for a set of primary FTL servers.</p> <p>Supply a pipe-separated URL list of primary FTL servers.</p> <p>The primaries initiate the connection to disaster recovery servers.</p> |

| Parameter | Location | Arguments | Description |
|-----------|----------|---------------|---|
| drto | globals | <URL_list> | <p>When present, the primary FTL server attempts to connect to a set of disaster recovery servers.</p> <p>Supply a pipe-separated URL list of disaster recovery FTL servers.</p> <p>(You must separately specify the “drfor” parameter in the configuration file of the disaster recovery servers.)</p> |
| timeout | globals | <timeout> | <p>Optional.</p> <p>FTL servers use this timeout for communication between other FTL servers in the same cluster. For example, this timeout applies to communication among core and auxiliary FTL servers at a given primary, satellite, or DR site</p> |
| heartbeat | globals | <hb_interval> | <p>Optional.</p> <p>An FTL server sends its heartbeat signal at <hb_interval>, in seconds.</p> <p>Supply a positive number.</p> <p>When absent, the default value is 1 second. This applies to communication among core and auxiliary FTL servers at a given primary, satellite, or DR site.</p> |

Logging

| Parameter | Location | Arguments | Description |
|-----------|----------------------|-----------|---|
| loglevel | ftlserver.properties | <level> | When present, the FTL server logs events at this level of |

| Parameter | Location | Arguments | Description |
|-----------|----------------------|------------------|--|
| | | | <p>detail.</p> <p>You may specify any of the standard log level strings . See “Tuning the Log Level” and its sub-topics in TIBCO FTL Development. You can set a general log level, or custom log levels for different elements.</p> <p>When this parameter is absent, the default level is info.</p> <p>See also Logging in the context of security.</p> |
| logfile | ftlserver.properties | <logfile_prefix> | <p>When present, the FTL server logs to a rolling set of log files instead of the console. The <logfile_prefix> argument may denote a path.</p> |

| Parameter | Location | Arguments | Description |
|--------------|----------------------|-----------|---|
| | | | <p>All of the directories in the path must already exist.</p> <p>For more information about rotating log files, see Log Output Targets in TIBCO FTL Development.</p> <p>When absent, the FTL server sends log output to the console, ignoring the parameters max.log.size and max.logs.</p> |
| max.log.size | ftlserver.properties | <size> | <p>Optional</p> <p>Available if logfile is set.</p> <p>Limits the maximum size, in bytes, of log files. The value must be greater than 100 kilobytes (102400 bytes).</p> <p>If max.log.size is not specified</p> |

| Parameter | Location | Arguments | Description |
|-----------|----------------------|-----------|---|
| | | | than the default of 20 MB is used. |
| max.logs | ftlserver.properties | <logs> | Optional. Available if logfile is set. Limits the maximum number of rolling log files. max.logs can be: Not specified: The FTL server uses the default value of 20. Equal to 1 : The max.log.size is ignored. A number greater than 1 and less than 1000. |

Core Servers

A *core server* is an FTL server to which FTL clients connect.

A core server provides an explicitly configured realm service, which is essential for client operations.

A cluster of core servers can cooperate for fault tolerance, or a single core server can operate solo. It is best practice to specify an odd number of core servers. You must specify at least one core server.

Specify the set of core servers in the `core.servers` map within the `globals` section of the FTL server configuration file.

```
globals:
  core.servers:
    ftl1: host1:8585
    ftl2: host2:8585
    ftl3: host3:8585
```

Specify the host and port location of each core server in this map.

The *host* can specify a host computer or an interface of a host computer.

The *port* specifies the port for all communications with the server. Even if an FTL server provides several services, network administrators need open only one port. The server automatically multiplexes the request stream to its services. The default port number is 8585.

Auxiliary Servers

An *auxiliary server* is an FTL server that you designate and configure for a specialized purpose, and do *not* designate in the list of `core.servers`. For example, an auxiliary server might provide an eFTL service to expand capacity, or a backup bridge service for fault tolerance, or provide a persistence service.

Auxiliary servers are optional. In many ordinary situations you do not need any auxiliary servers.

You can add auxiliary servers without restarting core servers, so auxiliary servers are ideal for expanding the client capacity of eFTL services.

An auxiliary server does not participate in a quorum.

An auxiliary server does not provide a group service.

An auxiliary server does not provide a default message broker.

Specify auxiliary servers in the `servers` section of the FTL server configuration file. For each auxiliary server, include an `ftl` configuration map, in which you specify its host and port location.

Do *not* include auxiliary servers in the core servers map. In the following example, notice that the auxiliary server `ftl4` is not in the core servers map.

```
globals:  # These values apply to all servers.
```



```

    core.servers: # The set of core servers and their locations.
        ftl1: host1:8585
        ftl2: host2:8585
        ftl3: host3:8585

servers:

    ftl1:
    # ...

    ftl4: # Aux server to expand eFTL capacity and backup bridge
service.
    - ftl:
        server: host4:7890
    - ectl: {}
    - bridge: {}

```

URL List

This topic describes URL list arguments in greater depth.

```
https://<host1>:<port1>|https://<host2>:<port2>|https://<host3>:<port3>
```

A URL list is an argument that specifies a set of interchangeable cooperating FTL servers. URL lists are important in the following contexts:

- FTL server configuration file parameters such as `satelliteof`, `drfor`, and `drto` accept arguments in this form.
- Command line parameters that specify servers can accept a URL list.
- API call parameters that specify FTL servers can accept a URL list.

The *host* can specify a host computer or an interface of a host computer.

The *port* specifies the FTL server port for all communications with the server. Even if an FTL server provides several services, network administrators need open only one port. The server automatically multiplexes the request stream to its services. The default port number is 8585.

Password Security

Passwords are sensitive information, and keeping them secure is critical to the security of your FTL processes. You can supply a password as a command line argument in any of several ways, which vary in the level of protection they provide for the password.

Password Scope

Server and service passwords authenticate the identity of a server or service process to other processes, for example, an FTL server authenticating itself to an affiliated FTL server or to an authentication service. For transparent fault tolerance, you can use the same password for equivalent servers or services in a fault-tolerant arrangement.

Keystore passwords encrypt key files, such as the private key file that FTL servers use to identify themselves to clients and to other servers.

Passwords can be masked. You can mask passwords using `tibftladmin`. Masked passwords have `$mask$` at the beginning of the string. Masked passwords are unmasked before being sent to the realm service.

Password Argument

When you supply a password, that argument is visible to casual observers. For example, command line arguments appear in the output of the UNIX `ps` command, even after you have cleared the shell's command history. Passwords in a file are visible to anyone who can access or intercept that file.

You can supply the password in any of the following forms. Each form results in a different level of security for the password, along with associated precautions you must take. Choose exactly one form.

| Form | Description |
|-----------------------------|---|
| <code>file:file_path</code> | <p>This form can provide excellent security: only the file path is visible to observers.</p> <p>You must create a text file that contains only the password itself, store that file on the file system of the FTL server's host computer, and ensure the security of that file.</p> |

| Form | Description |
|-----------------------------|--|
| env: environment_ var | <p>This form can provide excellent security.</p> <p>You must set an environment variable in the shell where you run the FTL server. The value of that variable is the password string. You must ensure that only authorized personnel have access to that shell.</p> |
| pass:password | <p>This form is <i>not</i> secure: the password is in the configuration file itself.</p> <p>Do <i>not</i> use these forms except during development and early testing phases.</p> |

Realm Service Configuration Parameters

This topic presents the FTL server configuration parameters that apply to the realm service. Also see: [Realm Service](#).

Example

This example configuration file illustrates the use of realm services configuration.

```
globals:  # These values apply to all servers.

core.servers: # The set of core servers and their locations.
  ftl1: host1:8585
  ftl2: host2:8585
  ftl3: host3:8585

# All servers inherit this value.
auth.url: file:/myhome/ftlserver/secure/myauthfile.txt

servers:
  ftl1:
    - ftlserver.properties:
        logfile: TIBCO_HOME/logging/ftl2/logs/ftl1.log
    - realm:
        loglevel: debug
    - persistence:
        name: p1
    - bridge:
        names:
```

```

    - bridgeA
    - bridgeB

ftl2:
  - ftlserver.properties:
      logfile: TIBCO_HOME/logging/ftl2/logs/ftl2.log
  - persistence:
      name: p2
  - bridge:
      names:
        - bridgeA
        - bridgeB
  - ectl:
      loglevel: debug

ftl3:
  - ftlserver.properties:
      logfile: TIBCO_HOME/logging/ftl3/logs/ftl3.log
  - persistence:
      name: p3
  - ectl:
      loglevel: debug

```

Logging

| Parameter | Arguments | Description |
|-----------|-----------|---|
| loglevel | <level> | <p>When present, the realm service logs protocol communication at this level of detail.</p> <p>You may specify any of the standard log level strings. See “Tuning the Log Level” and its sub-topics in TIBCO FTL Development. You can set a general log level, or custom log levels for different elements.</p> <p>When this parameter is absent, the default level is <code>info</code>.</p> |

Initial Configuration

| Parameter | Arguments | Description |
|--|--|--|
| <code>initial.realm.config</code> | <code><filepath/filename></code> | <p>During the initial deployment, if this parameter is set, the contents of this file are deployed. Otherwise, the default realm configuration asset is deployed. This parameter is not used on subsequent runs of the FTL server.</p> <p>Note that clients are unable to connect to the realm server until after the initial deployment has successfully completed.</p> <p>The FTL server logs any validation errors regarding the contents of the initial deployment file, and deployment will fail.</p> |
| <code>default.cluster.disk.swap</code> | <i>boolean</i> | <p>Enable (true) or disable (false) message swapping for the default cluster.</p> <p>When message swapping is enabled, the server can swap messages from process memory to disk. Swapping allows the server to free process memory for more incoming messages, and to process message volume in excess of memory limit. When the server swaps a message to disk, a small record of the swapped message remains in memory.</p> <p>This can be enabled with or without disk persistence. If enabled without disk persistence, data is written to a temporary swap file. This option can apply to replicated or non-replicated stores.</p> <p>Note: When using message swapping, if all FTL servers in the cluster are restarted simultaneously, messages in the store prior</p> |

| Parameter | Arguments | Description |
|--------------------------------------|------------------|--|
| | | to the restart are not retrievable after the restart unless disk persistence is also enabled. |
| default.cluster. disk.persistence | <i>disk mode</i> | <p>Select the disk persistence mode.</p> <ul style="list-style-type: none"> • none: Disable disk persistence. • sync: The client returns from a send-message call after the message has been written to a majority of disks. This mode generally provides consistent data and robustness, but at the cost of increased latency and lower throughput. If the cluster restarts, no data is lost; performance is subject to disk performance. • async: The client may return from a send-message call before the message has been written to disk by majority of the FTL servers. This mode generally provides less latency and more throughput, but messages could be lost if a majority of servers restart shortly after the API call. |
| default.cluster. disk.nocompact | disk mode | Auto compaction is disabled by default in the initial realm configuration for the default cluster. |

TLS Security

| Parameter | Arguments | Description |
|-----------|-----------------|-------------|
| tls.san | <i>SAN spec</i> | Optional. |

| Parameter | Arguments | Description |
|-----------|-----------|---|
| | | <p>Add a SAN (Subject Alternative Name) to the certificate generated by the FTL server.</p> <p>When no custom certificate is configured, this certificate is presented to:</p> <ul style="list-style-type: none"> • Secure eFTL clients (WSS) • HTTPS clients of the eFTL REST API • HTTPS clients of the FTL realm UI or realm REST API <p>This parameter can be used to customize the certificate in cases where the clients above connect to a hostname unknown to FTL server (for example, the hostname of a load balancer).</p> <p>Example: IP:1.2.3.4,DNS:myhost.com</p> |

Affiliated FTL Servers

| Parameter | Arguments | Description |
|---------------------|----------------|---|
| satellite.heartbeat | <i>seconds</i> | Communication with satellite realm services. The default is 1 second. |
| satellite.timeout | <i>seconds</i> | <p>Communication with satellite realm services. The default is 3 seconds.</p> <p>An FTL server waits for this timeout interval before repeating its connection request to an affiliated FTL server.</p> |
| backup.heartbeat | <i>seconds</i> | Communication with disaster recovery realm services. The default is 1 second. |
| backup.timeout | <i>seconds</i> | Communication with disaster recovery realm |

services. The default is 3 second.

An FTL server determines that an affiliate is unavailable when the affiliate's heartbeat signal is silent for this timeout interval.

Persistence Service Configuration Parameters

This topic presents the FTL server configuration parameters that apply to the persistence service.

Example

The example configuration file has two sections: globals and servers.

```
#Sample yaml file that demonstrates how to start three FTLServers with
non-default persistence cluster.
globals:
  core.servers:
    ftlserver1: localhost:8585
    ftlserver2: localhost:8685
    ftlserver3: localhost:8785

servers:
  ftlserver1:
    - realm:
        # Update this field accordingly (for windows use backward
slashes)
        data: TIBCO_HOME/persistence/ftlserver1/realm/data
        # Starts a non-default persistence server name pserver1
        # that belongs to a non-default persistence cluster
        # a persistence cluster with this server name needs to be configured
    - persistence:
        name: pserver1
        # Update this field accordingly (for windows use backward
slashes)
        data: TIBCO_HOME/persistence/ftlserver1/pserver1/data
        savedir: TIBCO_HOME/persistence/ftlserver1/pserver1/dumpdata
  ftlserver2:
    - realm:
        # Update this field accordingly (for windows use backward
slashes)
```



```

    data: TIBCO_HOME/persistence/ftlserver2/realm/data
  - persistence:
    name: pserver2
    # Update this field accordingly (for windows use backward
slashes)
    data: TIBCO_HOME/persistence/ftlserver2/pserver2/data
    savedir: TIBCO_HOME/persistence/ftlserver2/pserver2/dumpdata
  ftlserver3:
  - realm:
    # Update this field accordingly (for windows use backward
slashes)
    data: TIBCO_HOME/persistence/ftlserver3/realm/data
  - persistence:
    name: pserver3
    # Update this field accordingly (for windows use backward
slashes)
    data: TIBCO_HOME/persistence/ftlserver3/pserver3/data
    savedir: TIBCO_HOME/persistence/ftlserver3/pserver3/dumpdata

```

Name

| Parameter | Arguments | Description |
|-----------|--------------------|---|
| name | <service_ name> | <p>Required.</p> <p>Set the name of this persistence service. The <service_ name> must match one of the service names defined in the persistence cluster definition.</p> <p>To specify parameters for the default persistence service, set this to <code>_default</code>.</p> <div> <p>Note: The service name must be distinct for each FTL server.</p> </div> |

File I/O

| Parameter | Arguments | Description |
|-----------|-----------|---|
| data | <path> | <p>Optional.</p> <p>When present, the persistence service stores its working data files in this <path> location. (The directory at <path> must already exist, as the persistence service does not create it automatically.)</p> <p>When absent, the default <path> is the current directory.</p> <p>Ephemeral metadata is stored in this directory. If disk persistence is enabled, persistence data, including messages and acknowledgments, are written to this directory. If message swapping is enabled, data messages are swapped out to this directory by default. See swapdir later in this table.</p> |
| savendir | <path> | <p>Optional.</p> <p>When present, the persistence service can write its state file in this <path> location. (The directory at <path> must exist, as the persistence service does not create it automatically.)</p> <p>When absent, the default <path> is the current directory.</p> <p>The state file name is always <service_name>.state.</p> <p>A persistence service writes its state file <i>only</i> in response to an explicit command. This parameter specifies the location, but does</p> |

| Parameter | Arguments | Description |
|-----------|-----------|--|
| | | <p>not trigger a save operation. See Saving the State of a Persistence Service and POST persistence/clusters/<clus_name>/servers/<svc_name>.</p> <p>This parameter does <i>not</i> affect loading a state file.</p> |
| load | <path> | <p>Optional.</p> <p>Use this parameter when moving operations to a new physical location. After saving the state of the persistence service, copy the state file to <path> at the new location. Then use the load parameter to load that state when the persistence service starts at the new location.</p> <p>When present, the persistence service loads its state from this state file. (Specify the complete file pathname, not only the directory path.)</p> <p>When absent, the service loads its state from the default state file, <service_name>.state.</p> <p>For more information, see Restarting a Persistence Cluster with Saved State.</p> |
| swapdir | <path> | <p>Optional.</p> <p>When present and disk_swap is set to true, persistence service can write its message swap file in this <path> location. (The directory at <path> must exist, as the persistence service does not create it automatically.)</p> <p>When absent, the default <path> is the designated data directory.</p> |

| Parameter | Arguments | Description |
|---------------------------------|---------------------------|--|
| <code>max.disk.fraction</code> | <code><path></code> | <p>Optional.</p> <p>The <code>max.disk.fraction</code> parameter monitors disk capacity and prevents a disk full state. You must also enable <code>disk_persistence</code> in the realm configuration.</p> <p>The persistence service measures disk usage across the whole volume. Client publish calls fail once total disk usage approaches the <code>max.disk.fraction</code> setting multiplied by the capacity of the disk containing the persistence data directory.</p> <p>The default value is 0.95. Values less than 0 and more than 1 are not allowed. A value of 0 disables the feature.</p> <p>For more information, see Persistence Architecture, Setting Automatic Disk Persistence File Compaction.</p> |
| <code>disk.prealloc.size</code> | <code><size></code> | <p>Optional.</p> <p>This parameter only takes effect if disk persistence is enabled for the persistence service.</p> <p>Specify an integer representing the number of bytes to pre-allocate on disk. When specified, the persistence service will extend an existing database to this size, or create a new database of this size</p> <p>Enabling pre-allocation can avoid the overhead of extending the database file in situations where the message backlog is growing.</p> |
| <code>force.disk.load</code> | | By default, if the persistence service |

| Parameter | Arguments | Description |
|-----------|-----------|---|
| | | <p>encounters a corrupt record in its database, the persistence service stops loading the database and exit. If other quorum members are running and up to date, the recommended course of action is to move or delete the corrupted database. On restart the persistence service syncs from the other quorum members.</p> <p>If this is not possible because no other quorum members are running, you can use this parameter to force the persistence service to load the database. The corrupted record is discarded.</p> |

Also see: [Saving and Loading Persistence State](#)

Memory Reserve

| Parameter | Arguments | Description |
|--------------------------|---------------------------|--|
| <code>mem.reserve</code> | <code><size></code> | <p>Optional.</p> <p>When present, the persistence service reserves memory so it can continue limited operations after exhausting available memory.</p> <p>The <code><size></code> argument specifies the amount of memory in bytes. The minimum size is 100 megabytes.</p> <p>When absent, the persistence service allocates the minimum size reserve.</p> <p>For more information, see Memory Reserve for Persistence Services.</p> |

Logging

| Parameter | Arguments | Description |
|-----------|-----------|--|
| loglevel | <level> | <p>When present, the persistence service logs protocol communication at this level of detail.</p> <p>You may specify any of the standard log level strings. See “Tuning the Log Level” and its sub-topics in TIBCO FTL Development. You can set a general log level, or custom log levels for different elements.</p> <p>When this parameter is absent, the default level is info.</p> |

Bridge Service Configuration Parameters

This topic presents the FTL server configuration parameters that apply to the bridge service.

Example

The example configuration file has three sections: globals, services, and servers. Set bridge services configurations in the server section.

```
# Sample FTLServer yaml configuration file that demonstrates starting a
# primary and backup bridge service
# Since we are starting two bridge servers, one of them will end up
# being the backup.
#
globals:
  core.servers:
    ftlserver1: localhost:8585
    ftlserver2: localhost:8685
    ftlserver3: localhost:8785
servers:
  ftlserver1:
    - ftlserver.properties:
        logfile: bridge1.log
    - realm:
        # Update this field accordingly (for windows use backward
slashes)
        data: TIBCO_HOME/samples/yaml/ftlserver1/bridge
    - bridge:
        names:
```

```

      - brdg
      ft.weight: 100

ftlserver2:
  - ftlserver.properties:
      logfile: bridge2.log
  - realm:
      # Update this field accordingly (for windows use backward
slashes)
      data: TIBCO_HOME/samples/yaml/ftlserver2/bridge
  - bridge:
      names:
        - brdg
      ft.weight: 200
ftlserver3:
  - ftlserver.properties:
      logfile: bridge3.log
  - realm:
      # Update this field accordingly (for windows use backward
slashes)
      data: TIBCO_HOME/samples/yaml/ftlserver3/bridge
  - bridge:
      names:
        - brdg
      ft.weight: 300

```

Bridge Service Configuration Parameters

| Parameter | Arguments | Description |
|-----------|-------------------|---|
| names | <bridge_ name> | Optional. A list of bridge names. A bridge service implements one or more bridge objects. When absent, the bridge service implements only the bridge object named default. |
| label | <label> | Optional. The bridge status page of the FTL server GUI presents client information about bridge objects. This parameter can help you can distinguish among bridge objects in fault-tolerant bridge services. |

| Parameter | Arguments | Description |
|------------------------|-----------------------------|--|
| | | <p>Specify the <code>label</code> parameter at the same level as the <code>names</code> parameter.</p> <p>When present, the client label of a bridge object is a string that concatenates this <code><label></code> argument with the bridge name.</p> <p>When absent, a bridge object uses only the bridge name argument as its client label.</p> |
| <code>ft.weight</code> | <code><weight></code> | <p>Optional.</p> <p>If a group of fault-tolerant bridges is configured, the bridge service with the highest weight becomes active. This applies to all logical bridges managed by the bridge service.</p> <p>The value must be a positive integer greater than 0.</p> <p>See Arranging Fault-Tolerant Bridge Services.</p> |

eFTL Service Configuration Parameters

This topic presents the FTL server configuration parameters that apply to the eFTL service.

Example

The configuration file sets a parameter for the eFTL services in all servers. Next, it specifies a unique client name for the eFTL services on each of the three servers.

```
globals:

  core.servers: # The set of core servers and their locations.
    ftl1: host1:8585
    ftl2: host2:8585
    ftl3: host3:8585

servers:
  ftl1:
    - ectl:
        name: my_ectl_cluster
```



```

    client.label: ectl-svc-1

ftl2:
- ectl:
    name: my_ectl_cluster
    client.label: ectl-svc-2

ftl3:
- ectl:
    name: my_ectl_cluster
    client.label: ectl-svc-3

```

| Parameter | Arguments | Description |
|--------------|--------------------|---|
| name | <cluster_ name> | Optional. Cluster name. An eFTL service belongs to exactly one eFTL cluster. You may supply the name of the cluster to which this service belongs. When absent, the default value is Cluster. |
| client.label | <label> | Optional. When present, the eFTL service uses this string as its client label. You can use this string to distinguish among fault-tolerant eFTL services in the GUI. |

Security: EMS Server

If you are using eFTL with an EMS channel, set the following to the location of the EMS client library:

- Linux: LD_LIBRARY_PATH
- MacOS: DYLD_LIBRARY_PATH
- Window: PATH

| Parameter | Arguments | Description |
|------------|-----------|--|
| ssl.params | <path> | <p>Required for secure connections to an EMS server.</p> <p>The eFTL service reads parameters from this file, and uses them when connecting to a secure EMS server.</p> <p>For details, see "SSL Parameters for EMS Connections" in <i>TIBCO eFTL™ Administration</i>.</p> |

Messages

An eFTL service can automatically append user and client identification information to the messages that clients publish. See "Client Information Fields" in *TIBCO eFTL Development*.

| Parameter | Arguments | Description |
|-------------------|-----------|--|
| publish.client.id | <boolean> | <p>Optional.</p> <p>When present and true, the eFTL service appends the <code>_client_id</code> field to every message that any eFTL client publishes.</p> <p>See "Client ID Field" in <i>TIBCO eFTL Concepts</i>.</p> |
| publish.user | <boolean> | <p>Optional.</p> <p>When present and true, the eFTL service appends the <code>_user</code> field to every message that any eFTL client publishes.</p> <p>See "User Field" in <i>TIBCO eFTL Concepts</i>.</p> |

Logging

| Parameter | Arguments | Description |
|-----------|-----------|---|
| loglevel | <level> | When present, the eftl service logs protocol communication at this level of detail. |

| Parameter | Arguments | Description |
|-----------|-----------|--|
| | | <p>You may specify any of the standard log level strings. See “Tuning the Log Level” and its sub-topics in TIBCO FTL Development. You can set a general log level, or custom log levels for different elements.</p> <p>When this parameter is absent, the default level is info.</p> |

OAuth 2.0 Parameters for EMS connections

If you configure any EMS channels in an eFTL cluster, you must also arrange a configuration file with parameters from the following table. The eFTL service uses these parameter values when connecting to an EMS server and EMS server is set up for OAuth 2.0.

For information about these parameters, see “tibemsOAuth2Params and Environment Variables” in TIBCO Enterprise Message Service™ User Guide.

For syntax, see the example configuration file in the samples directory.

Supply OAuth 2.0 parameters in a file, and specify the file name as the value of the FTL server configuration file parameter `oauth2.params`. `oauth2.params` (similar to `ssl.params`) that we use for eFTL.

Note: All the EMS channels in a cluster must use the same set of OAuth 2.0 parameter values for connecting to their respective EMS servers.

oauth2 parameters

| eFTL OAuth 2.0 specific parameter meter | Description |
|---|---|
| <code>oauth2_server_url</code> | This is the URL of the OAuth 2.0 server |
| <code>oauth2_client_id</code> | OAuth 2.0 specific client id use to get an OAuth 2.0 access token |

| eFTL OAuth 2.0 specific parameter meter | Description |
|---|--|
| oauth2_client_secret | OAuth 2.0 specific client id use to get an OAuth 2.0 access token |
| oauth2_server_trust_file | If the connection to the OAuth 2.0 server is secure, specify this trust file |
| oauth2_expected_hostname | The expected hostname of the server from the server certificate. |
| oauth2_disable_verify_hostname | Option to disable verify hostname |

FTL Administration Utility

Administrators use the FTL administration utility, `tibftladmin`, to:

- Stop an FTL server process
- Update or dump the realm configuration
- Backup and restore the realm configuration
- Backup and restore persistence data (disk persistence only)
- Compact the realm configuration on disk
- Compact persistence data on disk (online or offline)

Most command line parameters and options have both a short and a long form. The command line parser accepts either form.

Help

| Parameter | Arguments | Description |
|---------------------|-----------|--|
| <code>--help</code> | | Display a help message describing the command line parameters and options. |
| <code>-h</code> | | |

Connecting to the FTL Server

| Parameter | Arguments | Description |
|----------------------|----------------------|---|
| --ftlserver -ftls | <url1> <url2> <url3> | <p>Required.</p> <p>Administer the FTL server at this location.</p> <p>You can provide more than one URLs.URLs are pipe seperated, each URL is of the form</p> <pre>http://<host1>:<port1></pre> <pre>https://<host1>: <port1></pre> <pre>http://<host2>:<port2></pre> <pre>https://<host2>: <port2></pre> <pre>http://<host3>:<port3></pre> <pre>https://<host3>: <port3></pre> |
| --user -u | user_name | Credentials. Required when the FTL server enables user authentication. |
| --password -pw | password | <p>Credentials. Required when the FTL server enables user authentication.</p> <p>To see the ways to specify a password and to hide the password from casual</p> |

| Parameter | Arguments | Description |
|---|-----------|---|
| | | observers, see Password Security . |
| <code>--tls.trust.file</code> <code>-tf</code> | <path> | <p>Optional. (Required for TLS communication with a secure FTL server.)</p> <p>The administration utility trusts an FTL server based on this trust file. Supply the file path of a local copy of the FTL server trust file.</p> <p>When both are present, this parameter overrides the parameter: <code>--tls.trust.everyone</code>.</p> |
| <code>--tls.trust.everyone</code> <code>-te</code> | | <p>Optional.</p> <p>The administration utility trusts any FTL server without verifying trust in the server's certificate.</p> <div> <p>Warning: Do not use this parameter except for convenience in development and testing. It is <i>not</i> secure.</p> </div> |
| <code>--oauth2.token</code> | <string> | <p>When the FTL server is setup with oAuth 2.0 authentication provider, then the administrative tool must use the <code>--oauth2.token</code> parameter to specify an oauth2 access token.</p> |

| Parameter | Arguments | Description |
|--|-------------------------------|--|
| <code>--tls.client.cert</code> | <code><path></code> | <p>Specify a TLS client certificate for mTLS authentication. The given file must be in PEM format. <code>--tls.client.private.key</code> must be specified as well.</p> <p>If the private key is encrypted, <code>--tls.client.private.key.password</code> must be specified.</p> |
| <code>--tls.client.private.key</code> | <code><path></code> | <p>Specify the private key corresponding to the TLS client certificate. The given file must be in PEM format.</p> <p>Use <code>--tls.client.private.key.password</code> to specify the decrypt passphrase, if needed.</p> |
| <code>--tls.client.private.key.password</code> | <code><password></code> | <p>Decrypt passphrase for the TLS client certificate's private key.</p> <p>It can be specified in one of four ways: <code>'file:<filename>'</code> The password is in a file specified by filename. The first line of text in the file is used as the password, with leading and trailing whitespace trimmed.</p> <p><code>'env:<variable>'</code> The named environment variable is read, and its contents used as the password.</p> <p><code>'pass:<password>'</code> The password is specified directly</p> |

| Parameter | Arguments | Description |
|-----------|-----------|--|
| | | by the parameter. |
| | | '<password>' The password is specified directly by the parameter. |
| | | Do not use passwords that begin with any of these reserved prefixes. To hide the password from casual observers use file or env. |

Specifying Files on Disk

| Parameter | Arguments | Description |
|---|-----------------------------|---|
| <code>--datadir</code> <code>-d</code> | <code><path></code> | <p>Operates on one of the following:</p> <ul style="list-style-type: none"> • The realm configuration data belonging to the named FTL server • The persistence data belonging to the named persistence service <p>Use in conjunction with the <code>--name</code> parameter (but not with the <code>--ftlserver</code> parameter).</p> |
| <code>--name</code> <code>-n</code> | <code><string></code> | <p>Operates on one of the following:</p> <ul style="list-style-type: none"> • The realm configuration data belonging to the named FTL server • The persistence data belonging to the named persistence service <p>Use in conjunction with the <code>--datadir</code> parameter (but not with the <code>--ftlserver</code> parameter).</p> |

Commands

Commands are mutually exclusive. You may use only one at a time.

Most commands affect an FTL server process that is already running. Specify that FTL server using the `--ftlserver` parameter.

The following options denote commands that affect a running FTL server process.

`--dumprealm`, `--updaterealm`, `--testupdate`,
`--backup_realm`, `--compact_realm`,
`--backup_persist`, `--compact_online`,
`--status`, `--server_status`, `--cluster_status`,
`--shutdown`, `--shutdown_cluster`, `--available`

Some commands operate on the realm configuration data of an FTL server or the persistence data of a persistence service. Specify the data directory using the `--datadir` parameter. Specify the name of the FTL server or persistence service using the `--name` parameter.

The following options denote commands that operate on the realm configuration data of an FTL server or the persistence data of a persistence service.

`--restore_realm`, `--restore_persist`, `--compact_offline`

Supply commands only on the command line. It is illegal to supply commands in a configuration file.

| Parameter | Arguments | Description |
|--|---------------------------|---|
| <code>--dumprealm</code> <code>-dr</code> | <code><path></code> | Write the realm definition to a JSON file at <code><path></code> . Modifiers: <code>--debug</code> |
| <code>--updaterealm</code> | <code><path></code> | Update the realm definition from the JSON file at <code><path></code> . |

| Parameter | Arguments | Description |
|---------------------|-----------|--|
| m -ur | | <p>If a realm definition already exists, the utility replaces the old realm definition with the new definition. This command is available only for a primary server.</p> <p>After updating one primary FTL server, it automatically propagates the new realm definition to all other affiliated servers.</p> <p>Modifiers: --force</p> |
| --testupdate -tu | <path> | <p>Test deployment using the realm definition in the file at <path>. The realm definition file is in JSON format.</p> <p>For information about the test, see The Deploy Transaction</p> |
| --backup_realm | | <p>Backup the FTL realm configuration.</p> <p>This command creates a backup of the current FTL realm configuration and associated state. A collection of files is written to the backups subdirectory of each FTL server's data directory.</p> <p>The file names incorporate a timestamp. To restore the realm configuration data for a particular FTL server from backup, see --restore_realm.</p> |
| --backup_persist | | <p>Backup the persistence data for the given persistence cluster.</p> <p>The persistence cluster must have disk persistence enabled.</p> <p>The --cluster <string> modifier is required.</p> <p>This command creates a backup of all persistence data, including message data and acknowledgments, for the given persistence cluster. The backup files are written to the backups directory of each persistence service in the cluster. The file names incorporate a timestamp.</p> <p>By default, the backup directory for a persistence service is the same as the service's data directory. A different backup directory may be specified setting the savedir parameter for</p> |

| Parameter | Arguments | Description |
|--|-----------|---|
| | | <p>the persistence service in the FTL server YAML configuration file.</p> <p>To restore the persistence data for a particular persistence service from backup, see <code>--restore_backup</code>.</p> <p>Required modifier: <code>--cluster <string></code></p> |
| <code>--compact_online</code> | | <p>Compact the persistence data for the given persistence cluster.</p> <p>The persistence cluster must have disk persistence enabled.</p> <p>The <code>--cluster <string></code> modifier is required.</p> <p>This command causes each persistence service in the given cluster to compact the service's disk persistence files. Active publishers and consumers are not interrupted, though some performance degradation is possible until the compaction completes.</p> <p>See <code>--compact_offline</code> for compaction while the persistence service is not running and if online compaction is unwanted or impractical.</p> <p>Modifier: <code>--cluster <string></code></p> |
| <code>--status</code> <code>-st</code> | | Get the status of the FTL server and its clients. Send output to stdout. |
| <code>--server_status</code> | | Get the status of the FTL server, but omit the status of its clients. Send the status output to stdout. |
| <code>--cluster_status</code> | | Get the status of the FTL server cluster, but omit the status of clients. Send the status output to stdout. |
| <code>--shutdown</code> <code>-x</code> | | <p>Stop the FTL server process and all the services it provides.</p> <p>This command is asynchronous, that is, the utility does not wait for confirmation that the server has stopped). To verify that the server has actually stopped, use the <code>--status</code></p> |

| Parameter | Arguments | Description |
|--|-----------|---|
| | | <p>command.</p> <p>Modifier: <code>--savestate</code> When present, the FTL Server saves the state to the disk before exiting.</p> <p>Note: This option is not needed if disk persistence is enabled.</p> |
| <p><code>--shutdown_cluster</code></p> <p><code>-xc</code></p> | | <p>Stop all FTL server processes in the cluster and all the services they provide.</p> <p>This command is asynchronous, that is, the utility does not wait for confirmation that the servers have stopped). To verify that the servers have actually stopped, use the <code>--status</code> command.</p> <p>Modifier: <code>--savestate</code> When present, the FTLServer saves the state to the disk before exiting.</p> <p>Note: This option is not needed if disk persistence is enabled.</p> |
| <p><code>--available</code></p> <p><code>-a</code></p> | | <p>Check to see if the FTL server is available and all services are running. Returns 200 OK or 503 Unavailable.</p> |
| <p><code>--restore_realm</code></p> | | <p>Restore the realm configuration data and associated state of an FTL server (given by <code>--name</code>) from backup.</p> <p>The <code>--backupdir <path></code> modifier is required.</p> <p>This command must be executed when the FTL server is not running.</p> <p>This command selects the latest available realm configuration data (as determined by the timestamp in the file names) and copies it to the data directory (given by <code>--datadir</code>). There must not be any data files related to the given FTL server in the data directory.</p> |

| Parameter | Arguments | Description |
|--------------------------------|--|---|
| | | Required modifier: <code>--backupdir <path></code> |
| <code>--restore_persist</code> | | <p>Restore the persistence data of a persistence service (given by <code>--name</code>) from backup. The <code>--backupdir <path></code> modifier is required.</p> <p>This command must be executed when the persistence service is not running.</p> <p>This command selects the latest available persistence data (as determined by the timestamp in the file name) and copies it to the data directory (given by <code>--datadir</code>).</p> <p>There must not be any data files related to the given persistence service in the data directory.</p> <p>Required modifier: <code>--backupdir <path></code></p> |
| <code>--compact_offline</code> | | <p>Compact the disk persistence files of a persistence service (given by <code>--name</code>) in its data directory (given by <code>--datadir</code>).</p> <p>This command must be executed when the persistence service is not running.</p> <p>Offline compaction may be used when online compaction is unwanted or impractical. (See <code>--compact_online</code> for compaction while the persistence service is running.)</p> |
| <code>--compact_realm</code> | | Compact the FTL realm configuration. This command compacts the database containing the FTL realm configuration. FTL clients are not interrupted. |
| <code>--hash</code> | | Accept a password on <code>stdin</code> , and send the hash output to <code>stdout</code> . |
| <code>--hash_all</code> | <code><filename or stdin></code> | <p>Accept a filename for a file (in TIBCO plain text user file format), and send the hash output (all passwords) to <code>stdout</code>.</p> <p>For argument <code><stdin></code>, accept input from standard input instead of a file.</p> |

| Parameter | Arguments | Description |
|--------------------------|-----------------------|---|
| <code>--hash_out</code> | <i>filename</i> | Use this parameter with the <code>--hash_all</code> parameter. Send the output of <code>hash_all</code> to a filename. |
| <code>--mask</code> | | Use this parameter alone to perform password mask encoding explicitly. Mask strings begin with <code>\$mask\$</code> . For example: <pre>\$ tibftladmin --mask ***** ***** TIBCO FTL FTLAdmin 7.0.1 V2 Copyright 2009-2024 Cloud Software Group, Inc. All Rights Reserved.</pre> |
| <code>--mask_file</code> | <i>inputfilename</i> | Perform password mask encoding on passwords contained in a configuration file and output the file to stdout. Masked passwords begin with <code>\$mask\$</code> and the rest of the file is unaltered. |
| <code>--mask_out</code> | <i>outputfilename</i> | Use with <code>--mask_file</code> to direct output to a specified file. For example: <pre>\$ tibftladmin --mask_file inputfilename --mask_out outputfilename</pre> |
| <code>--import</code> | <i><path></i> | Set a symlink <code>tibemsd</code> <i><path></i> to <code>tibems</code> server in <code>bin/modules</code> directory. <i><path></i> must be of the form <i><directory>/<filename></i> . |
| <code>--version</code> | | Prints the version of <code>tibftladmin</code> and exits. |

Command Modifiers

| Long | Arguments | Description |
|----------------------|-----------|--|
| <code>--force</code> | | When present along with <code>--updaterealm</code> , break the realm |

| Long | Arguments | Description |
|-------------|-----------------|--|
| | | lock, discarding any undeployed changes to the realm definition. |
| --debug | | When present along with --dumprealm, dump additional information for debugging. Use only when TIBCO support staff request this information. |
| --savestate | <true or false> | <p>When present along with -xc --shutdown_cluster or -x --shutdown, the FTL Server saves the state to the disk before exiting.</p> <p>Note: This option is not needed if disk persistence is enabled.</p> |
| --cluster | <string> | Name a persistence cluster for the --backup_persist or --compact_online commands. |
| --backupdir | <path> | Identify the location of backup files for the --restore_realm or --restore_persist commands. |

Realm Administration Tools

The *realm service* stores the realm definition, and makes it available to client application processes. Administrators can use a graphical user interface (GUI) and a REST-style web API to configure the realm definition and monitor client processes.

Administrators define the realm and its parts using the *configuration interface* (see [FTL Server GUI: Configuration](#)).

Administrators monitor and manage the realm's client processes using a *monitoring interface* (see [FTL Server GUI: Monitoring](#)).

These interfaces are accessible through a web browser that connects to the FTL server (see [FTL Server and Interfaces](#)).

Administrators can also define, monitor, and manage the realm and its clients through a REST-style web API interface (see [FTL Server Web API](#)).

Administrators can use external tools to monitor processes and analyze their metrics (see [Catalog of Metrics](#)).

Administrators can perform a realm backup and restore. Backup can be performed while the servers are running and without interruption to clients. See [FTL Administration Utility](#) for the commands to backup and restore the realm.

Realm Definition Terminology

Administrators configure a realm by defining its transports, applications, application instances, and formats.

Transport Definition

Configures a transport, including its name, transport protocol, and parameter values.

Application Definition

Determines an application's endpoints, the configuration of the application's process instances, and the subset of message formats available to the application.

Application Instance Definition

Implements endpoints by associating them with transports.

Format Definition

Determines the field names and field value types that can appear in messages that use a managed format.

For further information, see the following topics:

- [Formats](#)
- [Application and Endpoint Concepts](#)
- [Transport Concepts](#)

Application and Endpoint Concepts

The term *application* can refer to an executable program that communicates using TIBCO FTL software, or to an administrative configuration in the realm (an application definition). Application programs can use a provided default application definition, or be customized by administrators.

The term *application instance* can refer either to a process instance of an application program, or to an administrative configuration in the realm (that is, an application instance definition). Application processes can use the default instance definition. When needed, administrators can declare and configure specialized instance definitions.

Through a matching algorithm, one such configuration can apply to a set of process instances. When defining a configuration, administrators can assign different resources to specific sets of process instances, based on attribute matching, to achieve fine-grained control over communications traffic.

When defining an application instance configuration, the administrator must connect each application endpoint to one or more transports, which actually move message data.

Endpoints represent data exit and entry points within application programs. Transports carry message data between endpoints. Administrators configure endpoint implementation and transport usage in the realm.

Configuration Model

This overarching model serves as a framework for understanding the concepts and tasks of configuring a realm. Refer back to this model as you read the topics that follow.

The model is only a skeleton. It is not a complete description. It is not necessary to completely understand every aspect of the model the first time you read this section. To help integrate the new ideas, return to this model after reading subsequent sections.

- *Endpoints represent requirements.*
 - Endpoints constitute a program's communication interface. An application program with one or more endpoints requires communication through those

endpoints.

- For each of its endpoints, an application program requires communication using some or all of the endpoint's four abilities (see [Abilities](#)). That is, each ability that the program uses for communication is a separate and specific functional requirement.
- A *transport connector* represents a set of *abilities*. It binds a transport's abilities to an endpoint's required abilities. When a set of transport connectors covers all the required abilities of an endpoint, then the connectors satisfy the endpoint (that is, they satisfy the requirements that the endpoint represents).
- An *application instance definition* represents a *solution*.
 - It implements each endpoint (requirement) with a set of connectors (abilities), satisfying the requirement.
 - Moreover, it targets a set of process instances. A matching algorithm dynamically determines that target set.
 - You can define several such solutions, each satisfying the same requirements with different abilities, and targeting a different set of process instances.

The topics that follow describe endpoint implementation and its configuration: first at the micro level of endpoints, abilities, and connectors, and then at the macro level of applications and application instances. In practice, administrators generally configure from both ends inward:

- Endpoint, transport
- Application, application instance, transport connector

Endpoints

An endpoint is an abstraction that represents a set of publishers and subscribers in communicating programs.

Programming

Within an application program, an endpoint begins as a name. Program code uses the endpoint name to create endpoint instances, that is, publisher and subscriber objects. Then programs use the publishers to send messages, and the subscribers to receive messages.

Each subscriber object and each call to a publisher send method introduces an ability requirement (see [Abilities](#)). Application architects and developers record these requirements in endpoint coordination forms. See the "Endpoint Coordination Form" in the FTL [Product Guides](#) list.

✓ **Tip:** For many applications or communicating application suites, a single endpoint name suffices.

Configuration

Administrators configure transport connectors to satisfy those ability requirements. Connectors bind transports to endpoints. Those transports implement the endpoints, transferring message data from publishers to subscribers.

Administrators can also view an endpoint as a complex entity, embracing four separate communication abilities. A transport connector can separately enable any subset of those four abilities.

✓ **Tip:** The default application definition is a valid configuration for applications that use only one endpoint.

Abilities

An endpoint has four communication abilities. A separate aspect of the API embodies each of these abilities. The following table summarizes the four abilities.

Each of these four abilities and the API call that embodies it correspond to a distinct sub-stream of messages, called an *ability sub-stream*.

Endpoint Abilities: Definitions

| Ability | API | Description |
|---------------------|-------------------|--|
| One-to-Many Receive | Subscriber object | Carries one-to-many messages inbound to the endpoint's subscribers in the application. |

| Ability | API | Description |
|--------------------|--|---|
| One-to-One Receive | Inbox subscriber object | Carries one-to-one messages inbound to the endpoint's inbox subscribers in the application. |
| One-to-Many Send | Send call of a publisher object | Carries one-to-many messages outbound from the endpoint's publishers in the application. |
| One-to-One Send | Send-to-inbox call of a publisher object | Carries one-to-one messages outbound from the endpoint's publishers in the application. |

Notice that each ability combines two dimensions: *reach* and *direction*:

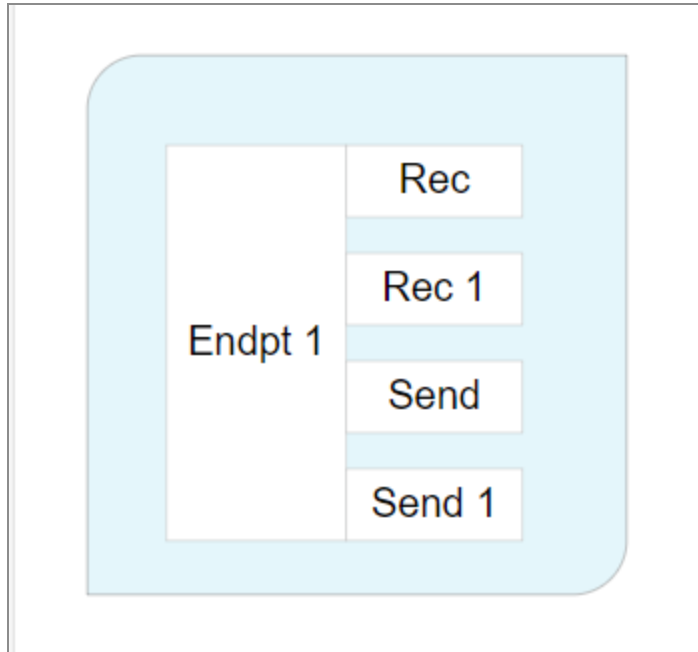
- *Reach*
 - *One-to-many abilities* carry messages that can reach *potentially many* subscribers.
 - *One-to-one abilities* carry messages that can reach *one* specific *inbox* subscriber.
- *Direction*
 - *Receive abilities* carry messages *inbound to* an application.
 - *Send abilities* carry messages *outbound from* an application.

Endpoint Abilities Matrix

| | Inbound | Outbound |
|-------------|---------------|------------|
| One-to-Many | Receive | Send |
| One-to-One | Receive Inbox | Send Inbox |

Administrators can configure these abilities in the FTL server GUI (see [Endpoint Details Panel](#)).

Endpoint and its abilities are represented with the following shape. Here, Endpoint 1 has four abilities: Receive, Receive 1, Send, and Send 1.

Figure 1: Four Endpoint Abilities

Implementation: Endpoints (Micro)

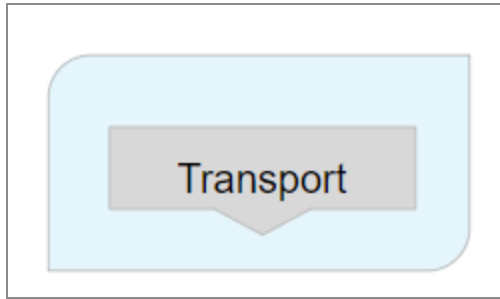
Administrators arrange transports to implement endpoints.

Transports

Transports transfer message data between process instances of application programs. Within a program process, a transport carries messages outbound from the program's publishers, and carries messages inbound to the program's subscribers.

Administrators are responsible for defining appropriate transports to transfer messages, and for binding those transports to the endpoints of the various program processes.

A transport is represented with a triangular point indicates that the transport acts *outside* of an application process (for example, in a network, or a shared memory segment).

Figure 2: Transport

Transport Connectors

A *transport connector* is a configuration within an application instance definition, which binds a transport to the abilities of an endpoint.

Structurally, a connector consists of a transport and a set of abilities, within the context of an application definition or application instance definition.

The meaning of a connector is that the transport carries the message streams that use that set of abilities.

Transport Connectors in the GUI

To create a transport connector, locate the endpoint in the Applications grid, then select **Add Transport** from the corresponding drop-down menu in its Endpoint column.

To view or configure the abilities of a transport connector, view the endpoint's details panel. Each row in the Endpoint Abilities area represents a transport connector, where you can configure abilities by selecting and clearing checkboxes.



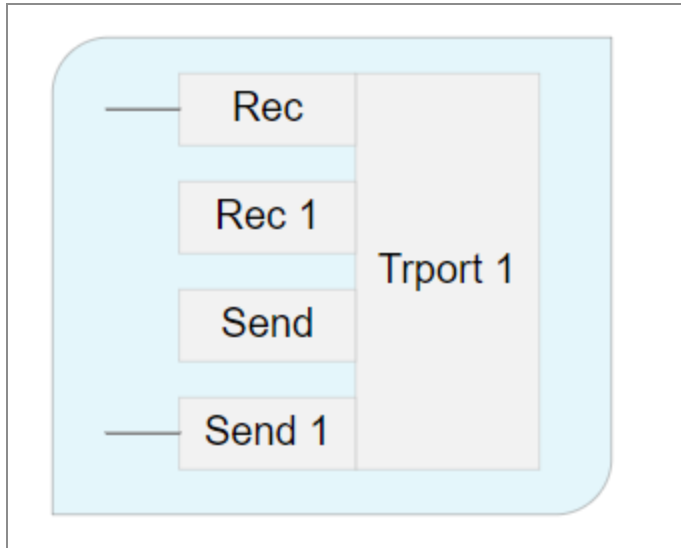
Tip: Default configurations bind all four abilities, which works well in many situations. While more complex bindings are possible they are usually not necessary, except to divide the corresponding message streams among different transports.

Transport and Connector Diagrams

A connector and its abilities are represented as a labeled rectangular body. The bulges on the left side represent the four potential abilities. Lines represent the endpoint abilities

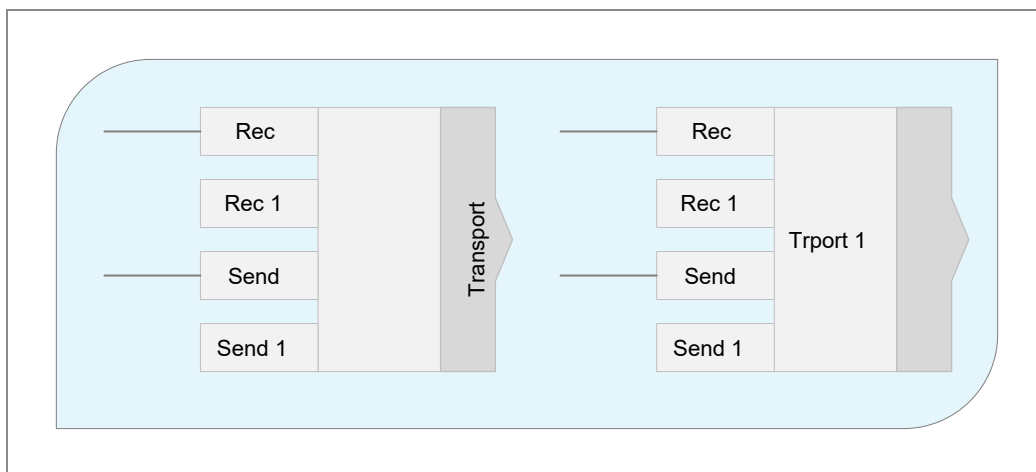
that the connector actually binds. The graphical configuration interface for a connector represents the four abilities as checkboxes. Each line in this connector diagram corresponds to a selected checkbox in the Endpoint Abilities section of the configuration interface (see [Endpoint Details Panel](#)).

Figure 3: Transport Connectors



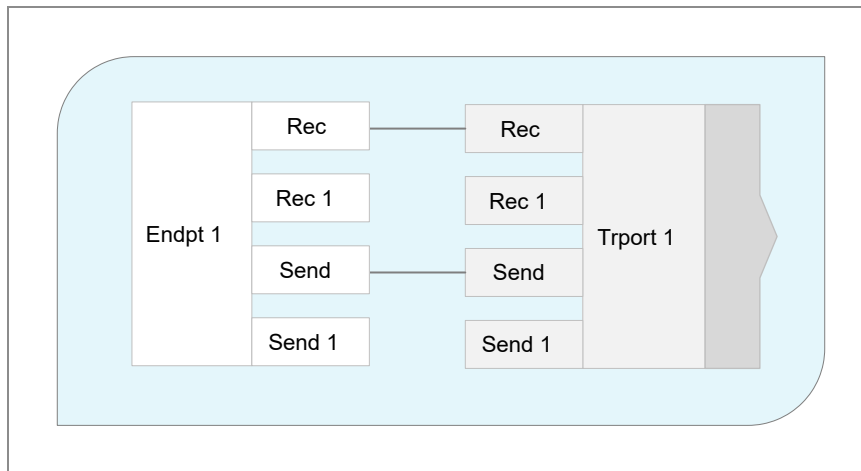
When a connector and its transport appear as adjacent shapes, diagrams can omit the transport name from one or the other.

Figure 4: One Transport Connector Name Omitted



Diagrams often depict the connector along with its endpoint, as it would appear in the context of an application instance.

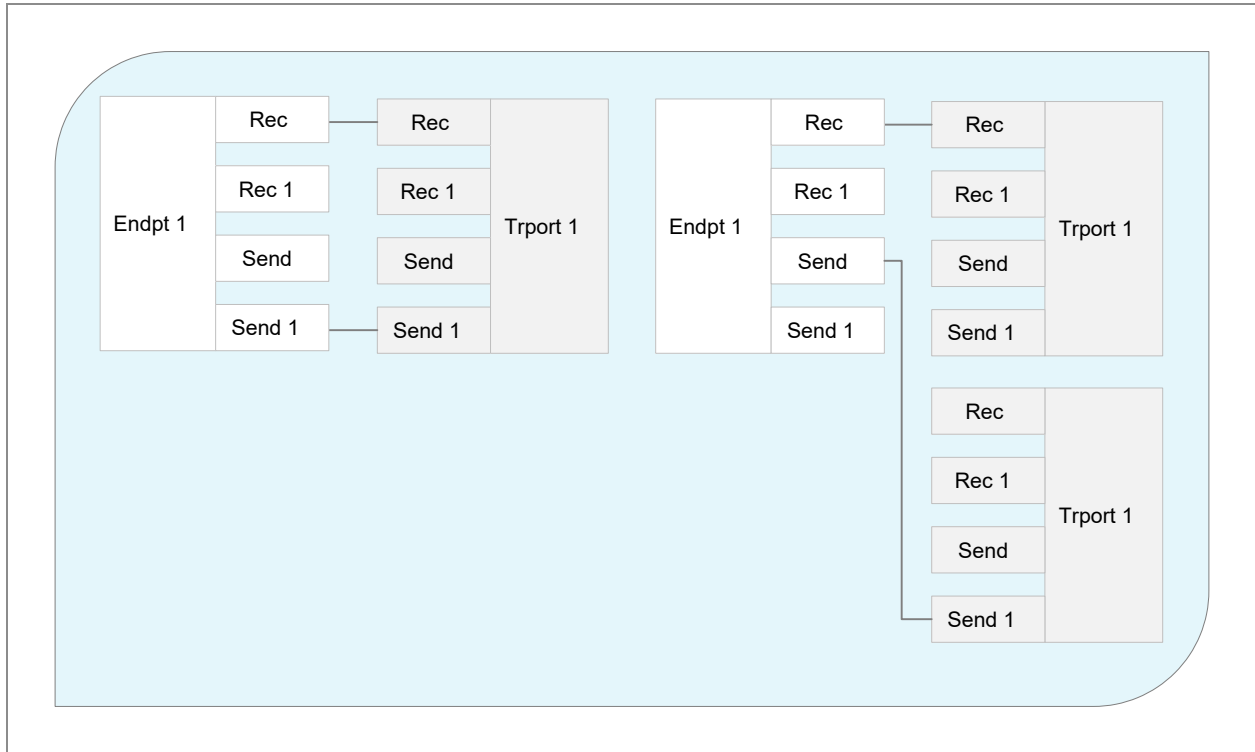
Figure 5: Connector and Endpoint Depiction



Connectors to Cover Required Abilities

When a connector binds a transport to an endpoint, in effect it binds the transport to a *subset* of the endpoint's required abilities. To be useful, the set of connectors in an application instance definition must *cover* the set of abilities that a program requires for communication.

For example, in the following diagram, a program uses an endpoint to send one-to-one messages and to receive one-to-many messages. An endpoint configuration that binds only the send inbox ability and the receive ability is sufficient to implement that endpoint for the program: that is, it covers the two abilities that the program uses. The application instance definition could cover the required abilities using either one transport connector, or two. The diagram illustrates both of these possibilities.

Figure 6: Connectors Cover Required Endpoint Abilities

The number of connectors in an application instance definition depends on the number of abilities that a program requires, and on the transports the administrator chooses to carry those abilities. In most cases, an application instance would contain only a few connectors, because it must cover at most four abilities. Even if an application instance were to use a separate transport to carry each of the four abilities, it would need at most four connectors to cover them. Since one transport can carry several abilities, the number of connectors in the application instance can be less than four.

The application definition binds all four abilities to the default transport, using a single connector.

Implementation: Application (Macro)

Administrators view an application program both as an executable program, and also as a set of *process instances* of that program, running on one or more host computers. Arranging process instances on various host computers is one of the administrator's responsibilities.

All the process instances of an application program use the same set of endpoints. Developers hard-code that requirement into the program by creating publishers and subscribers, and by calling their methods.

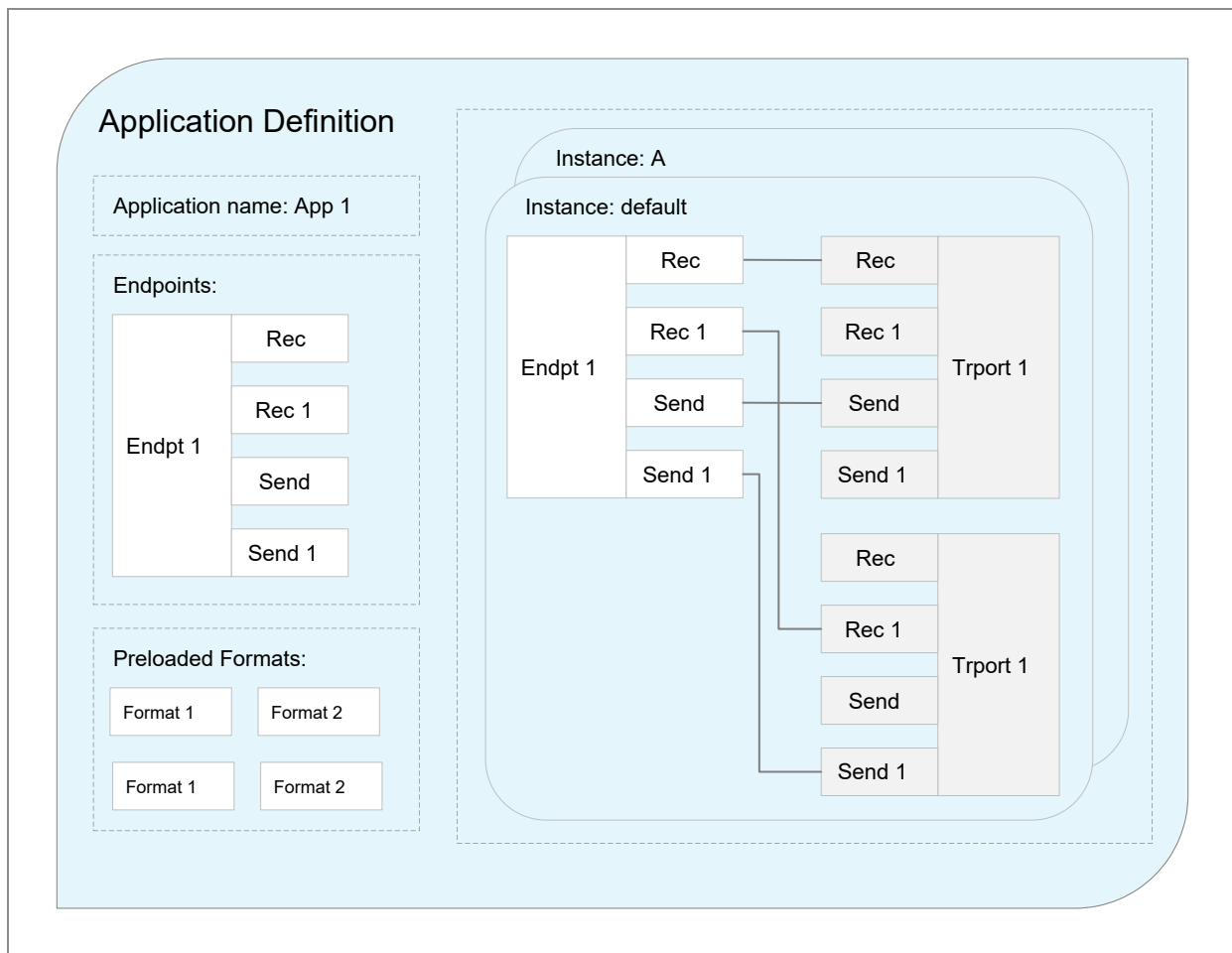
However, from an administrator's perspective, it could be necessary or advantageous for some of the process instances to implement those endpoints using a different set of transports. To arrange these variations, administrators define application definitions and application instance definitions within the realm.

Application Definitions

An *application definition* is a configuration within the realm that represents an application program, declares its endpoints and formats, and configures endpoint implementation for its process instances.

Structurally, an application definition consists of a name, a set of endpoints, a set of application instance definitions, and a set of preload formats. The following diagram depicts the structure of an application definition.

Figure 7: Application Definition



Tip: You can use the FTL server GUI to define an application definition; see [Applications Grid](#).

The set of endpoints in the application definition must be identical to the set of endpoints that the program code uses. To ensure agreement, developers and administrators can use the [TIBCO FTL Endpoint Coordination Form](#) as a contract between program code and realm.

An application definition always configures a default instance, and may also configure other application instances for use in specific situations.

Administrators also define the set of managed formats that the application program preloads from the realm service. (For more information, see [Defining Formats](#). Formats are orthogonal to endpoint implementation.)

Application Instance Definitions

Application instance definitions configure the implementation of endpoints for process instances of an application.

Structurally, an application instance definition consists of a name, a set of match parameters, and a set of connectors to implement each endpoint.

To add an application instance definition, see [Applications Grid](#).

Each application instance definition has two tasks with respect to implementing endpoints:

- Statically specify a configuration that satisfies all of the program's endpoint and ability requirements.
The set of connectors configures the implementation of endpoints, accomplishing this task.
- Dynamically determine a set of process instances that satisfy their requirements according to that configuration.
The match parameters determine the set of process instances, accomplishing this task.

Default Instance and Named Instances

Every application definition has a *default instance definition*. You may also define an optional ordered list of *named instance definitions*.

A named instance definition can include values for match parameters. The configuration interface prevents the default instance from including match parameters.

Instance Matching

A matching algorithm dynamically determines a set of process instances.

The matching algorithm compares attribute values of a process instance against the match parameter values in the named instance definitions. A process instance *matches* a named instance definition when *every* match parameter you configure in the named instance definition is *identical* to the corresponding attribute value in the process instance. A match

parameter without any value acts like a wildcard: that is, it matches any attribute value in every process instance.

The order of the named instance definitions within the application definition determines the order in which the matching algorithm attempts to match them. The first named instance definition that matches the process instance determines the endpoint implementation for the process.

If *none* of the named instances matches the process instance, then the default instance determines the endpoint implementation for the process. (If you have not defined any named instances, then the default instance always determines implementation for all process instances, without matching.)

To configure match parameters, set the Identifier and Host columns of the applications grid (see [Instance Level \(Optional\)](#)).

Instance Matching and Persistence

You can also use instance matching with persistence stores, to map from subscriber names to static durables. Define an instance that matches on an identifier, and configure the subscriber mapping on its endpoints. See also [Instance Matching for Subscriber Name Mapping](#).

Instances Determine Subscribers and Durables

When an application uses persistence stores and durables, application instances determine, for each endpoint, the mapping from subscriber names to static durables.

See [Configuration of Durable Subscribers in an Application or Instance](#), and [Instance Matching for Subscriber Name Mapping](#).



Note: This mapping is not relevant when an application program supplies a *durable name* rather than a *subscriber name*.



Note: This mapping does not apply to *dynamic* durables, only to *static* durables.

Administrative Requirements

Administrators must coordinate with application developers to properly configure and deploy applications.

Administrative requirements for endpoint implementation depend on the application programs. Each program instantiates a set of endpoints, embodying them through API objects and method calls, which in turn use a subset of the endpoints' abilities. Program developers and administrators must coordinate to ensure correct endpoint implementation:

- Developers must specify the *endpoints* and *communication abilities* that their applications use (and inform administrators). In the configuration model, these are the *requirements*.

(To facilitate coordination, developers complete the [TIBCO FTL Endpoint Coordination Form](#).)

- Administrators must configure *transports* and *connectors* to cover that set of endpoint requirements. In the configuration model, these are the *transport abilities*.
- Administrators may configure several *application instance definitions*, each implementing the endpoints in a different way. In the configuration model, these are the *solutions*.

It is an administrative error to leave an ability requirement unsatisfied. When the application program attempts to use that ability, the API call fails. For example, if no transport carries an endpoint's one-to-many receive ability, the subscriber creation call fails. If no transport carries an endpoint's one-to-one send ability, the publisher's send to inbox call fails.

See Also: [Configuration Model](#)

Administrative Options

When configuring endpoint implementation, administrative choices affect transmission characteristics for the application. Consider these factors when optimizing overall performance.

Transport Protocol and Abilities

Some transport protocols are often more appropriate to carry some communication abilities than others. For example, shared memory transports and multicast transports are well suited to carry one-to-many messages.

Transport Protocol and Host Computer

Some transport protocols are often more appropriate to application instances that are co-located on one multi-core host computer, while others are appropriate to communicate among separate host computers.

For example, shared memory transports are well suited to carry messages among co-located application instances. In contrast, TCP, and multicast transports are well suited to carry messages among application instances on separate host computers.

Segregating Traffic by Priority

When a stream of messages has high priority and low volume, it could be appropriate to segregate it on a separate transport, away from other streams with lower priority or high volume.

Multiple Transports and Serial Communications

It is possible to configure several transports to carry one endpoint ability. In such cases, communications are serial.

For example, connectors could specify that a multicast transport and a TCP transport both carry the send ability. The multicast transport reaches receivers within a LAN, and the TCP transport reaches remote receivers. (In this example, no intervening hardware router exists to forward multicast packets.)

When two or more transports carry an endpoint ability, the TIBCO FTL base library arranges their communications serially, within one thread, according to the order that the transports appear in the application instance's endpoint connectors.

Serial communications apply in two situations:

- Send abilities place outbound message data from a send call on each transport serially.

- Receive abilities associated with *inline* event queues accomplish all six phases of delivery in a single thread, attending to each transport in a round-robin arrangement. For background information, see [Inline Mode for Administrators](#).

(In contrast, receive abilities associated with regular event queues receive inbound messages from several transports in separate threads. The dispatch phase merges them into the event queue for dispatch in yet another thread.)

i Note: Serial processing can increase latency.

Sample Configuration 1: Default with `tibsend` and `tibrecv`

This first example illustrates the simplest configuration for communication. The sample applications `tibsend` and `tibrecv` use the default configuration definitions.

The default application definition and default transport definition provide an infrastructure for programs that use a single endpoint. This simplicity helps first-time users to quickly run sample programs.

Consider the sample applications `tibsend` and `tibrecv`, and the sample realm configuration (see [Sample Programs Reference](#)).

| | <code>tibsend</code> | <code>tibrecv</code> |
|---------------------|--|---|
| Program Description | The sample program <code>tibsend</code> creates a publisher object, and sends one message over its one-to-many send ability. | The sample program <code>tibrecv</code> creates a subscriber object, and receives one message over its one-to-many receive ability. |
| Application Name | The program connects to the FTL server with a null application name. Null selects the default application definition. | The program connects to the FTL server with a null application name. Null selects the default application definition. |
| Endpoints | The sample realm defines the application default with one | The sample realm defines the application default with one |

| | tibsend | tibrecv |
|-----------|--|--|
| | unnamed endpoint. The program creates a publisher with a null endpoint name, which selects the default endpoint from the application definition. | unnamed endpoint. The program creates a subscriber with a null endpoint name, which selects the default endpoint from the application definition. |
| Transport | The sample realm defines the transport default as a dynamic TCP protocol. The default application definition uses this transport to carry all four abilities of its default endpoint. | |
| Result | <p>Notice that the default transport implements the unnamed endpoints in both programs. The effect is to join the two endpoints so that messages flow from <code>tibsend</code> to <code>tibrecv</code> over a dynamic TCP bus.</p> <p>After exchanging one message, both programs exit.</p> | |

Program developers can also use the default definitions for a variety of applications with a similar communication structure (that is, one endpoint). To use the default definitions, programs supply null as the application name in the realm connect call, and supply null as the endpoint name in subscriber create and publisher create calls.

Sample Configuration 2: tibsendex and tibrecvex

This second example illustrates basic concepts in configuring transports and endpoint implementation, namely: binding a transport to carry an ability, configuring connectors, joining endpoints through a common transport, dividing message streams over separate transports.

To clarify the concepts of endpoints and transports, consider the sample applications `tibsendex` and `tibrecvex`, and the sample realm configuration (see [Sample Programs Reference](#)).

| | tibsendex | tibrecvex |
|---------------------|---|--|
| Program Description | The sample program <code>tibsendex</code> creates a publisher object, and sends messages over its one-to-many send ability. | The sample program <code>tibrecvex</code> creates a subscriber object, and receives messages over its one-to-many receive ability. |
| Endpoints | The sample realm defines the application <code>tibsendex</code> with one endpoint (named <code>tibsend-endpoint</code>). | The sample realm defines the application <code>tibrecvex</code> with one endpoint (named <code>tibrecv-endpoint</code>). |

Default Application Instance

First consider the default application instance, and the effect of its configuration.

| | tibsendex | tibrecvex |
|------------------------------|---|--|
| Default Application Instance | The sample realm configures the default application instance of <code>tibsendex</code> , implementing <code>tibsend-endpoint</code> with the transport <code>shm-sendrecv-tport</code> , which carries the endpoint's one-to-many send ability. | The sample realm configures the default application instance of <code>tibrecvex</code> , implementing <code>tibrecv-endpoint</code> with the transport <code>shm-sendrecv-tport</code> , which carries the endpoint's one-to-many receive ability. |
| End Result | Notice that the two application instances specify the same transport (<code>shm-sendrecv-tport</code>) to implement their respective endpoints. The effect is to join the two endpoints (<code>tibsend-endpoint</code> and <code>tibrecv-endpoint</code>) so that messages flow between them, from <code>tibsendex</code> to <code>tibrecvex</code> . | |

Another Application Instance

Next, consider another application instance, named `mcast`.

| | tibsendex | tibrecvex |
|----------------------------|--|---|
| mcast Application Instance | The sample realm configures the mcast application instance of tibsendex, to match the identifier mcast. For matching processes, it implements tibsend-endpoint with the transport mcast-send-tport, which carries the endpoint's one-to-many send ability. | The sample realm configures the mcast application instance of tibrecvex, to match the identifier mcast. For matching processes, it implements tibrecv-endpoint with the transport mcast-recv-tport, which carries the endpoint's one-to-many receive ability. |
| End Result | <p>Notice that the two application instances specify a pair of related transport definitions to implement their respective endpoints. Furthermore, those transport definitions are complementary, that is, the multicast send group of one is a multicast listen group of the other. The effect is to join the two endpoints (tibsend-endpoint and tibrecv-endpoint) with a single transport bus, so that messages flow between them, from tibsendex to tibrecvex.</p> <p>If several process instances of tibsendex and tibrecvex were to run on networked host computers, their common transport bus would enable each instance of tibsendex to communicate with each instance of tibrecvex.</p> <div> <p>Note: Messages do <i>not</i> flow between default application instances and mcast application instances. They use two different transport buses, which keep their message streams separate.</p> </div> | |

Sample Configuration 3: tibrequest and tibreply

This third example illustrates additional concepts in configuring transports and endpoint implementation, namely: binding a transport to carry several abilities; two-way message traffic; and connection-oriented transports with connecting and listening ends.

| | tibrequest | tibreply |
|---------------------|---|---|
| Program Description | The sample program tibrequest creates a publisher object, and sends messages over its one-to- | The sample program tibreply creates a subscriber object, and receives request messages over its one-to-many |

| | tibrequest | tibreply |
|-----------|---|--|
| | <p>many send ability.</p> <p>tibrequest also creates an inbox subscriber object, and receives reply messages over its one-to-one receive ability.</p> | <p>receive ability.</p> <p>It responds to each request message by sending back a reply message over its one-to-one send ability.</p> |
| Endpoints | The sample realm defines the application tibrequest with one endpoint (named tibrequest-endpoint). | The sample realm defines the application tibreply with one endpoint (named tibreply-endpoint). |

Transport Concepts

Transports are the underlying mechanisms that facilitate message communication among endpoint instances in programs.

TIBCO FTL software supports a variety of transport protocols. For descriptions, see [Transport Protocol Types](#).

Transport Roles

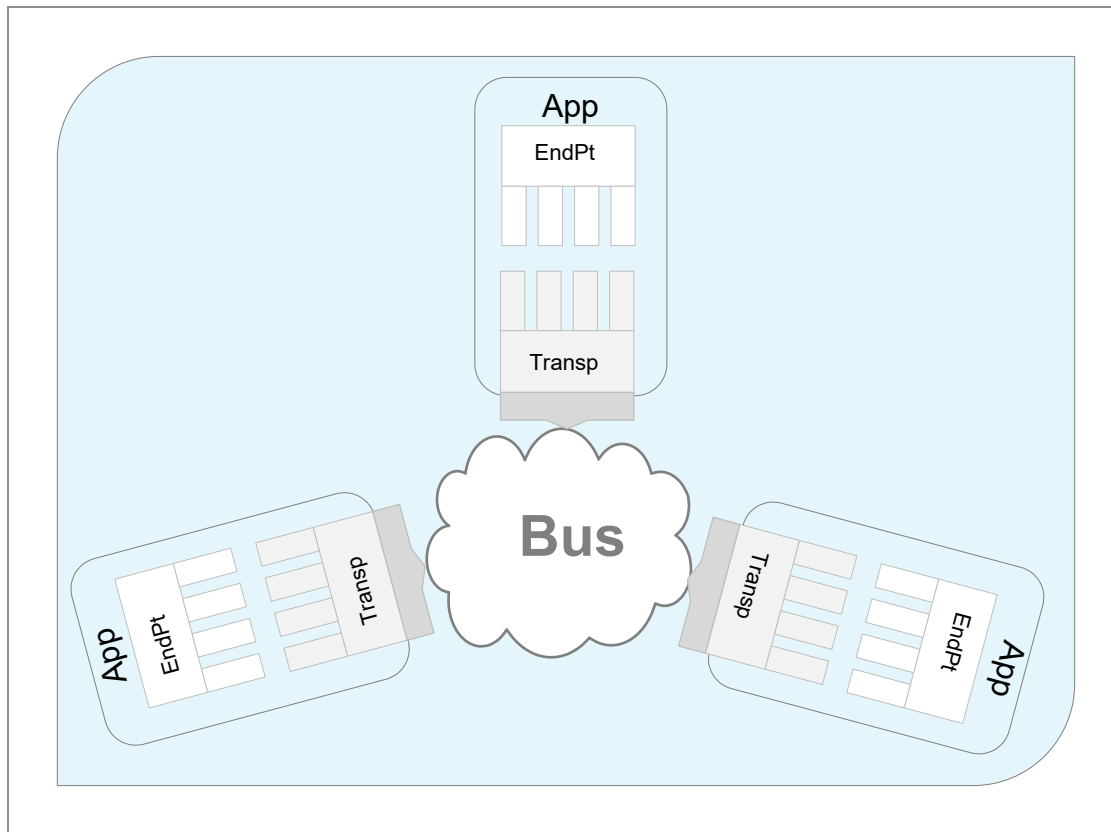
Transports have two important runtime tasks:

- Establish a *bus*, that is, a shared communication medium.
- Serve as a *communications interface*, mediating between endpoint instances (that is, publishers and subscribers in application processes) and the bus.

Inside a Transport

The following diagram presents a generalized view of transports and their operating roles.

Figure 8: Transports and Bus



Notice several aspects of this runtime perspective:

- Endpoint instances are objects within application processes. Application code uses API calls to create publisher and subscriber objects within application process instances.

- Transports are objects that straddle the edge of application processes.

Administrators configure transport definitions in the realm, and the TIBCO FTL base library expresses the transport definitions in application processes at run time. These runtime transports act within the process to establish a bus and serve as interfaces between endpoints and the bus.

- Transports and their operating details remain hidden from application code.
- The protocol type and details of the bus remain hidden from application code.
- The bus is conceptually *outside* the application processes. Transport objects mediate *between* the application program and the bus.

Transport Definitions Must be Unique

It is erroneous to define two or more transports with identical transport configuration values.

For example:

- Do not configure two transport definitions that access the same shared memory segment.
- Do not configure two transport definitions with the same multicast send group and listen groups.

The scope of this prohibition is even wider than a single realm. If two or more realms exist within the same network, then violating this rule can result in interference between the realms.

Do not bend this rule as a technique to bridge between two realms. Instead, develop a custom application to translate messages between realms. Such an application would straddle the realms, connecting to two FTL servers, which maintain the distinct realm definitions for their respective realms. (The translation application must pay special attention to the formats of messages as they cross between realms.)

Unitary and Fragmentary Transport Definitions

Transport definitions can be either unitary or fragmentary:

Unitary

Transports in several communicating process instances or process threads use the *same* transport definition to establish a bus.

Fragmentary

Transports in process instances use *different* transport definitions, which cooperate to establish a bus.

Inherently Unitary

Definitions of dynamic TCP mesh transports, shared memory transports, and process transports are *inherently unitary*. For example, one transport definition contains all the information that application processes require to establish and join a dynamic TCP mesh.

Similarly, one shared memory segment serves as a bus for endpoints in many application processes, and one transport definition contains all the information that the application processes require to establish and join that bus.

Inherently Fragmentary

In contrast, static TCP, dynamic TCP listen/connect pairs, are *inherently fragmentary* because they use connection-oriented protocols to establish a bus. For example, establishing each static TCP connection requires asymmetric transport definitions at the listening end and the connecting end. The two transport definitions specify complementary roles while establishing the bus: one listens, the other connects. Yet they must cooperate: they listen and connect at a common interface and port.

Neither Inherently Unitary nor Inherently Fragmentary

Multicast transport definitions can be *either unitary or fragmentary*. Multicast communication does not require any initial connection protocol, so the determining factor is the use case.

- In a unitary use case, a set of communicating peers could all use the same multicast group as their bus. One transport definition suffices, and it specifies an identical multicast group for listen and for send.
- In the more common fragmentary case, administrators control network data flow to achieve specific results, such as bandwidth allocation, performance, data segregation, or asymmetric multicast topologies. A set of shared multicast groups serve as the bus, as applications require different transport definitions, reflecting their different communication characteristics.

Related Transports

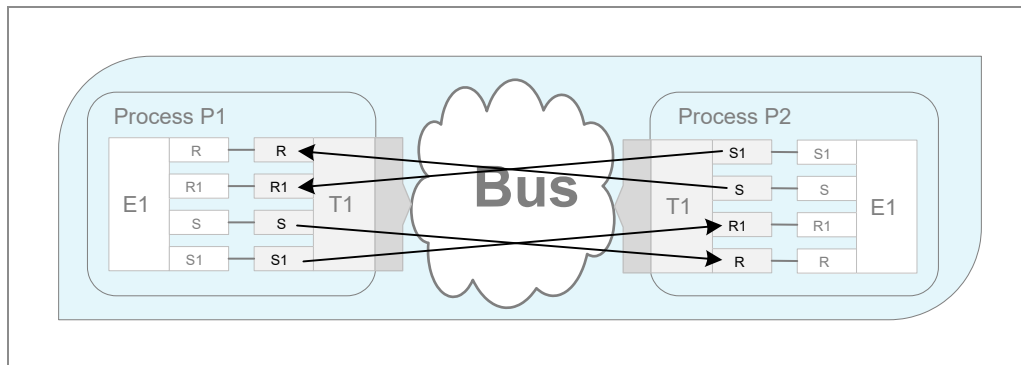
Two fragmentary transport definitions are called *related transports* when they cooperate to establish a bus.

For example, dynamic TCP transport definitions are related when they define a listen end and a connect end within the same transport group. Other connection-oriented transport definitions are related when they define a listen end and a connect end on the same port number. Asymmetric multicast transport definitions are related by the send and listen groups they share.

i Note: Administrators must properly configure related transports so that they can establish a bus. As a counterexample, consider a static TCP transport, T1, which listens on port 5678, and a static TCP transport, T2, which connects on port 7890. These transports are *not* related, and cannot establish a TCP connection.

When related transports establish a bus, data flows between them through the bus. The following diagram depicts two program processes, each with one endpoint. A pair of related transports implements the two endpoints. Arrows show the data flow among the abilities of the related transports. Notice that when process P1 sends a one-to-one message, it flows through the send inbox ability over transport T1, then into process P2 through the receive inbox ability of transport T2 and P2 receives it in its inbox (which instantiates endpoint E2).

Figure 9: Related Transports: Data Flow



Bus Topologies

You can configure a set of cooperating transports to establish a bus with a predetermined topology. The following topics present configuration techniques for some example topologies.

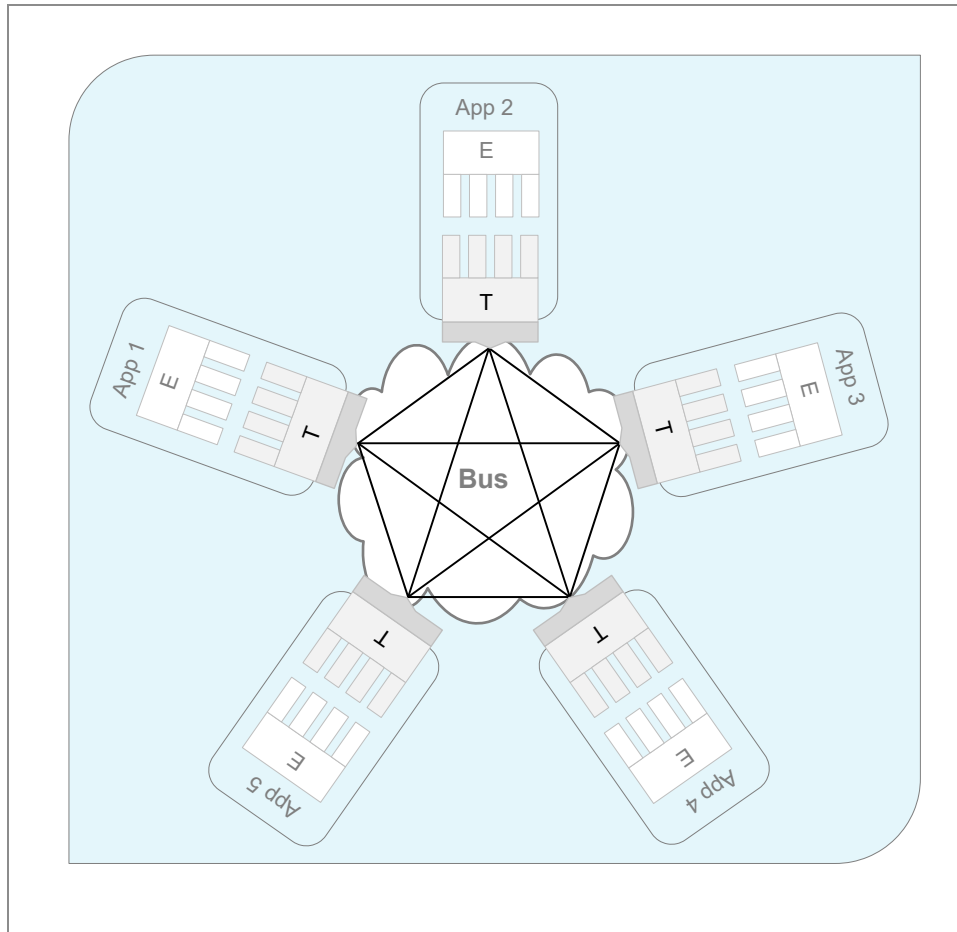
Unitary Mesh

The natural topology of a unitary transport definition is a mesh, that is, a complete graph, over all its runtime transport objects.

The diagram shows this topology with five process instances. Any process on this bus can send messages to all of its peers and receive messages from all of its peers.

Notice that all five processes necessarily use the same transport definition (T) to join the bus (this is the definition of a unitary transport definition).

Figure 10: Unitary Mesh



You can easily create mesh topologies using a single transport definition of type dynamic TCP, shared memory, or symmetric multicast.

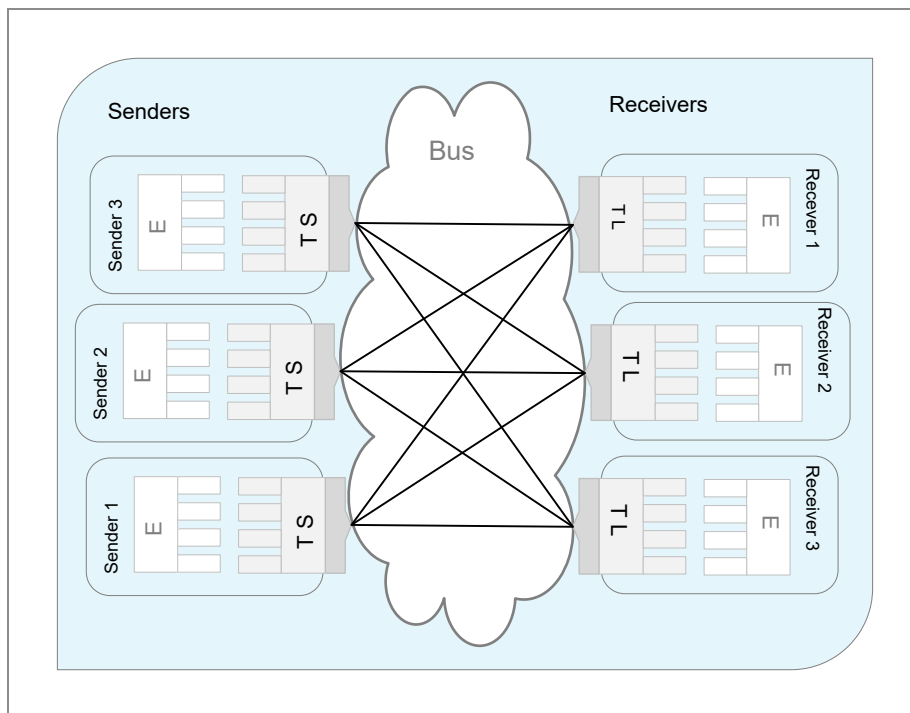
Asymmetric Multicast Topologies

In this example of an asymmetric multicast topology, three sending applications on the left side can send one-to-many messages to three receiving applications on the right side. You can easily generalize this example to any number of senders and receivers, and the number of senders and receivers need not be the same.

Configure these definitions:

- Transport TS specifies send group G1.
- Transport TL specifies listen group G1.
- Connectors in the senders (left) bind TS to carry the send ability.
- Connectors in the receivers (right) bind TL to carry the receive ability.

Figure 11: Asymmetric Multicast, Bipartite Graph



Notice that all these processes use only two different transport definitions, TS and TL, to join the bus, reflecting their opposite roles as either senders or receivers.

Assembling Larger Topologies from Pair Connections

The basic building block in connection-oriented transports (such as static TCP, or RUDP) is a pair connection, which carries two-way communication between two runtime transports across a network. You can assemble these basic parts into larger topologies, such as hub-and-spoke, or mesh.

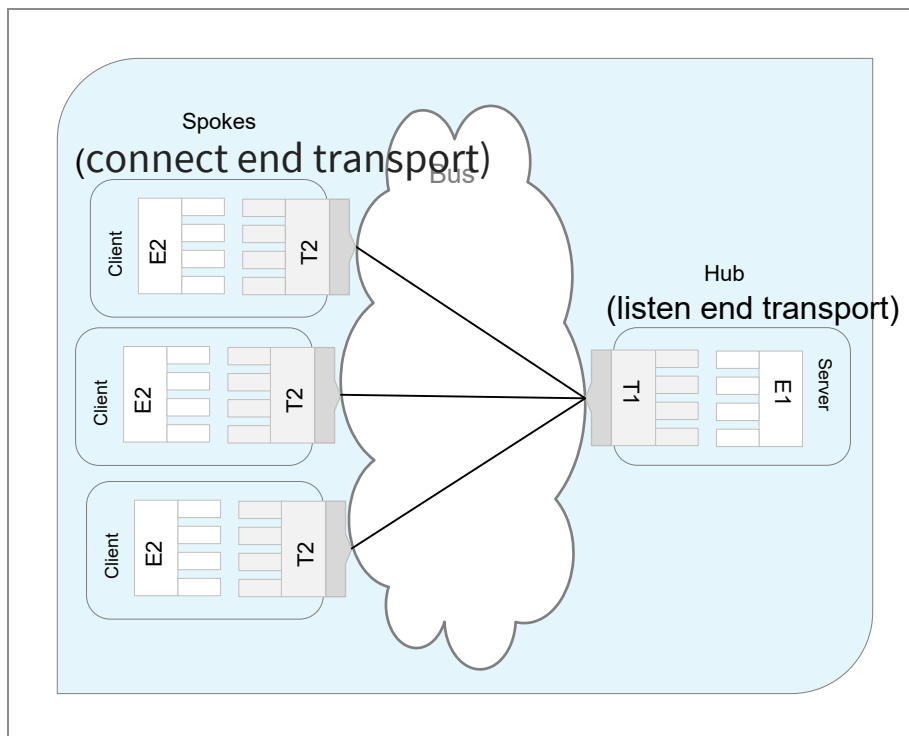
Hub-and-Spoke

Hub-and-spoke topology characterizes a server with many clients. Configure one listen end transport definition for the hub and one connect end transport definition for all the spokes. Start the hub application, then start the spoke applications.

Another name for hub-and-spoke topology is *star* topology.

For further information, see [Listen End and Connect End](#).

Figure 12: Hub-and-Spoke Topology



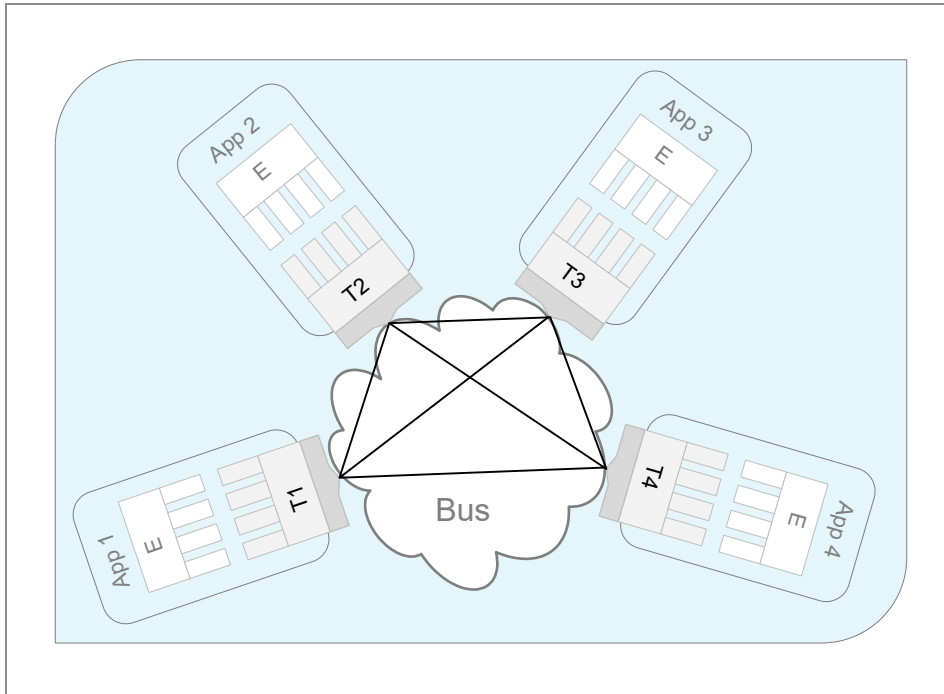
Mesh from Pair Connections

You can explicitly assemble a mesh with four vertices. Configure the six pair connections (edges) using four transport definitions:

- T1 listens on port P1.
- T2 listens on port P2 and connects to port P1 on A1's host computer.
- T3 listens on port P3 and connects to ports P1 on A1's host computer, and to port P2 on A2's host computer.

- T4 connects to ports P1 on A1's host computer, and to port P2 on A2's host computer, and to port P3 on A3's host computer.

Figure 13: Mesh Topology for Connection-Oriented Transport Protocols



Notice that all four processes necessarily use different transport definitions to join the bus. It cannot be otherwise, because connection-oriented transport definitions are fragmentary.

Also notice that the four transport definitions (T1 - T4) are each tied to specific host computers. In order to use them, the application process instances (A1 - A4) must run on those host computers, respectively. You can configure this constraint using application instance definitions that match the transport's host parameter.



Tip: It is easier to use a dynamic TCP mesh than to painstakingly assemble it from static TCP connections. The realm service automatically coordinates the host and port details for dynamic TCP transports.

Pair Connections

When defining connection-oriented transports (such as static TCP, and RUDP) the connect end and the listen end must cooperate to establish a connection.

Listen End and Connect End

In a pair connection, the assignment of listen end and connect end is not arbitrary. Instead, it depends on the type of interaction, or on the bus topology. To simplify transport configuration, follow these guidelines.

In general, a process that must start first is a good candidate for the listen end of pair connection. This rule helps avoid situations in which programs send messages before the transport establishes a pair connection bus.

Listen End and Connect End

| Situation | Listen End | Connect End |
|--|--------------------|---------------------|
| Request/Reply Interactions | Replying process | Requesting process |
| Server and Clients | Server | Clients |
| Long-Lived Process and Ephemeral Processes | Long-lived process | Ephemeral processes |
| Hub-and-Spoke Topology | Hub | Spokes |

Start Order

Administrators must arrange for processes to start in the correct order, according to application semantics. If a requesting process starts before the replying process, the replying process could miss the first request (or several requests).

The principles that determine which processes must start first are similar to the principles that determine the preference for listen end of a pair connection (see the preceding table). Namely, replying processes, servers, long-lived processes and hub processes must start first.

Connect Requests are Specific to IP Addresses

In a connection-oriented transport, the definition for a connect end hard-configures a specific IP address at the listening end.

Consider these consequences:

- A connect end transport definition can operate on several different host computers, as long as the listening end port is already open.
- A listen end transport must run at the location where its connect end counterparts expect to find it.
- If you move a listen end process to another host computer, you must reconfigure its counterpart connect end so it can find the listen end.
- If you move a connect end process to another host computer, you need not modify its counterpart listen end.
- A connect end must direct its connection requests to a listen port on *one specific* network interface of the host computer at the listen end. That is, you must specify either an IP address or a host name, but asterisk (*) is not valid. The listen port must be open on that network interface, otherwise the connection request fails.
- A listen end transport may open a listen port on all available network interfaces, or on only one. That is, you may specify asterisk (*) as the host interface, or you may specify a specific IP address or host name.

Subscriber Interest

Whenever possible, publishers send messages only to interested subscribers. That is, publishers send only those messages that match at least one subscriber content matcher. This behavior can reduce bandwidth consumption and can enhance subscriber performance in some situations.

This behavior extends across a transport bridge when the transports support it end-to-end (see [Transport Bridge](#)).

This support is automatic for TCP and shared memory transports.

For multicast transports, the configuration must explicitly require subscriber interest and you must also configure the transport's multicast groups to support *bi-directional* communication (that is, inbound and outbound) on the transport bus (see [Multicast Transport: Parameters Reference](#) and [Multicast Transport: Parameters Reference](#)). The two directions need not use the same multicast group. See [Multicast Transport: Parameters Reference](#).

Send: Blocking versus Non-Blocking

Administrators determine the behavior of send calls when a transport bus is full. Program developers may advise administrators concerning the correct behavior.

Send calls in an application program place outbound messages on a bus, which is external to the program process. A transport mediates this I/O operation. Under normal circumstances, the I/O operation completes quickly, and the send call returns.

If the bus is *full*, that is, it cannot accept any more messages, then two behaviors are possible. Administrators select one of these two behaviors, and configure it in the transport definition.

Blocking and Non-Blocking Send

| Behavior | Description |
|--------------------------------------|---|
| Blocking Send | When the bus is full, the send call blocks, and does not return until it has placed all the outbound message data on the bus. |
| Non-Blocking Send (Default Behavior) | <p>When the bus is full, the transport buffers outbound messages in a backlog buffer in process memory until the bus can accept them. The send call returns.</p> <p>When the bus can accept more messages, the transport automatically moves them from its backlog buffer to the bus.</p> <p>If the backlog buffer overflows, the transport discards the oldest messages in the buffer to make room for new messages. The discarded messages do not arrive at slow receivers. The base library reports dataloss advisories to slow receivers.</p> |

Use [TIBCO FTL Endpoint Coordination Form](#) to document the correct behavior.

To configure this behavior in a transport definition, see “Transport Definition Reference.”

Shared Memory Transports and One-to-Many Non-Blocking Send

Shared memory transports buffer message data from one-to-many send calls in the shared memory segment itself, rather than in a backlog buffer in process memory.

To counteract dataloss in this situation, enlarge the [Segment Size](#) of the shared memory segment in the transport definition, rather than increasing the backlog buffer size.

Blocking Send and Inline Mode

Programs that use inline event queues must ensure that callbacks return promptly and do not include operations that might block. If a program with inline event queues sends messages from within a callback, administrators must ensure that those outbound transports specify only non-blocking send. In this situation, blocking send could cause the send call within the callback to block the inline dispatch thread (even though the programmer expected the send call to return immediately).

See Also: [Inline Mode for Administrators](#)

Inline Mode for Administrators

Programs that receive time-sensitive messages can use inline mode to favor low latency over high throughput. Inline mode reduces inbound message latency by consolidating transport I/O and message callback processing into one thread.

For a complete description of inline mode, its usage, requirements and restrictions, see "Inline Mode" in TIBCO FTL [Development](#).

When specifying inline mode, programmers and administrators must coordinate to avoid illegal state exceptions. In particular, inline mode for event queues imposes restrictions on the transports you can use to implement endpoints. See [Transport Restrictions with Inline Mode](#), and on the FTL [Product Guides](#) list, "Application Coordination Form".

When specifying inline mode, programmers and administrators must also coordinate to avoid blocking send operations that could delay callbacks from returning promptly (see [Blocking Send and Inline Mode](#)).

Transport Restrictions with Inline Mode

Inline mode restricts underlying transports. Conformance with the following restriction must be ensured.



Restriction: When a transport is associated with an *inline* event queue, it cannot be associated with any other event queue.

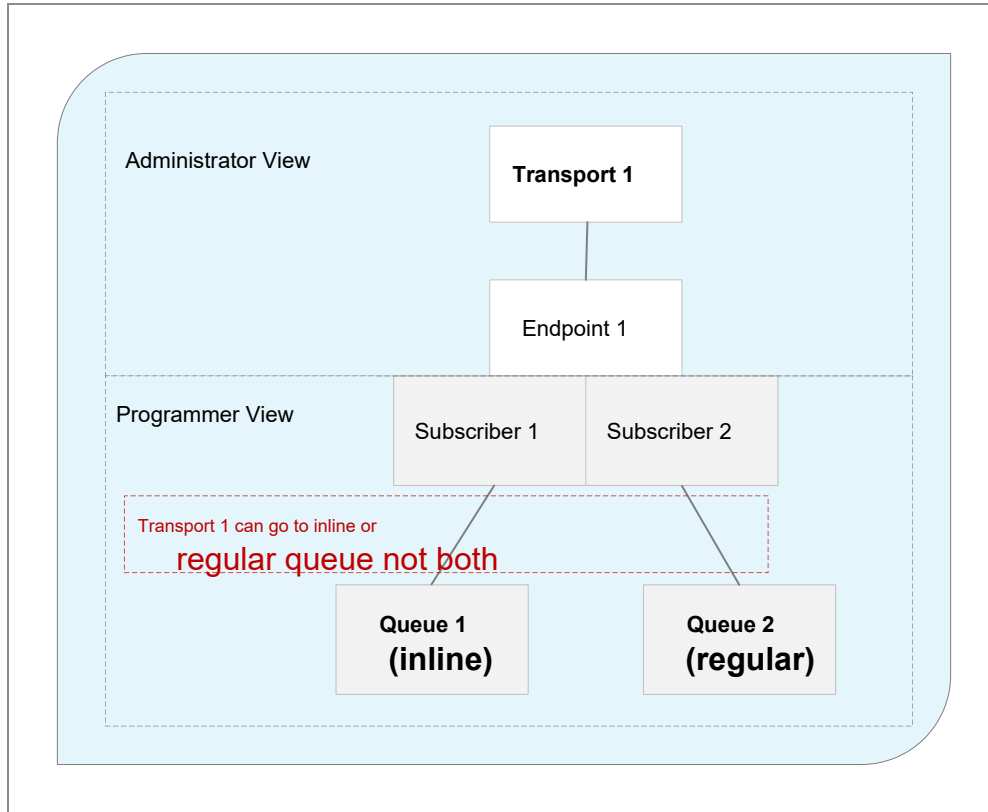
Violating the restriction results in an illegal state exception. The add subscriber API call throws the exception when the program attempts to add a subscriber that would violate the principle.

When specifying inline mode, programmers must coordinate with administrators to avoid illegal state exceptions.

Programmer Violation

The first diagram illustrates a violation attributable to programmer error. The programmer created two subscribers (S1 and S2) from a single endpoint (E1), and adds them to two event queues, at least one of which (Q1) is an inline queue. This situation would associate the underlying transport (T1) of the subscribers with two incompatible queues (Q1 and Q2), violating the restriction.

Figure 14: Illegal Inline Queue, Simple Case



To rectify the violation, either add both subscribers to a single inline event queue (Q1), or to two *ordinary* event queues, neither of which use inline mode.

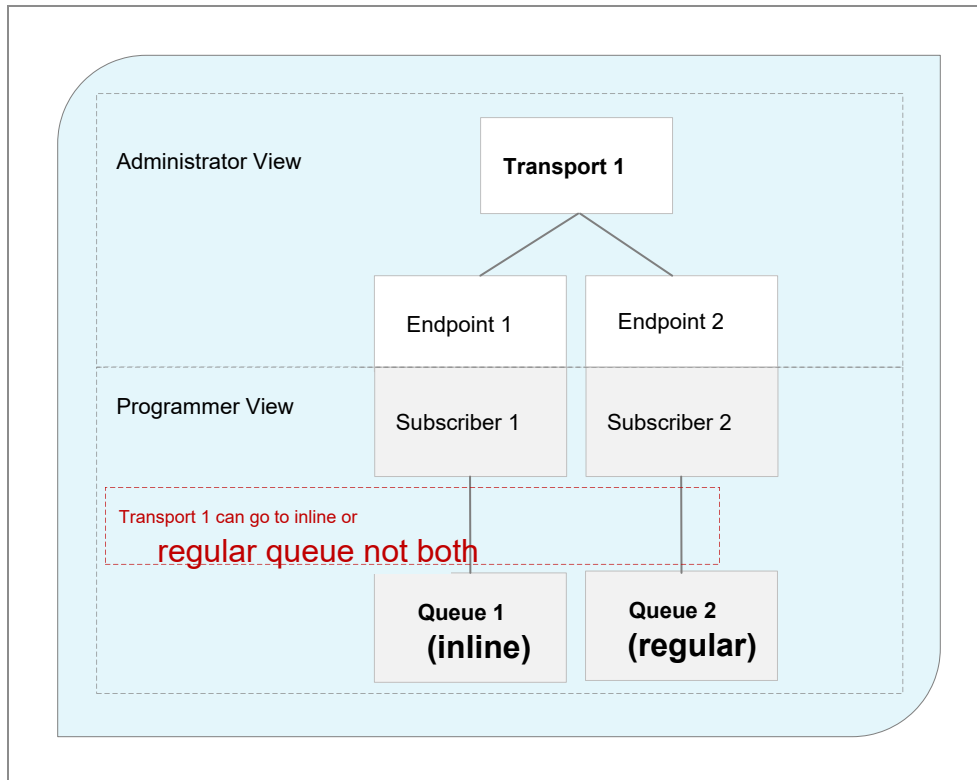
Hidden Violation

While it would appear straightforward for programmers to avoid the violation of the simple case of the first diagram, the situation is more complicated because responsibilities are divided between programmers and administrators.

From a programmer's perspective, transports are ordinarily hidden behind endpoint names. Conversely, administrators can ordinarily employ transports to implement endpoints without understanding the specifics of subscribers and queues in application programs. However, inline mode tightly couples transport I/O to program code, which in turn blurs the usual separation of responsibilities between programmer and administrator. As a result, the restriction requires that programmers and administrators coordinate whenever a program uses inline event queues.

Consider the following example. The situation in the second diagram might appear correct from the separate perspectives of the programmer and the administrator, but combining their perspectives reveals a hidden violation.

Figure 15: Illegal Inline Queue, Hidden Violation



The programmer instantiates two subscribers (S1 and S2) on two separate endpoints (E1 and E2), and associates them with separate queues (Q1 and Q2). Q1 is an inline queue.

Meanwhile, the administrator employs a single transport (T1) to implement the two endpoints (E1 and E2).

Because T1 is associated with Q1, which is an inline queue, this arrangement violates the principle. When the program adds S2 to Q2, the add subscriber call throws an exception, whether or not Q2 uses inline mode.

By coordinating their information, the programmer and administrator can cooperate to avoid the violation in any of three ways:

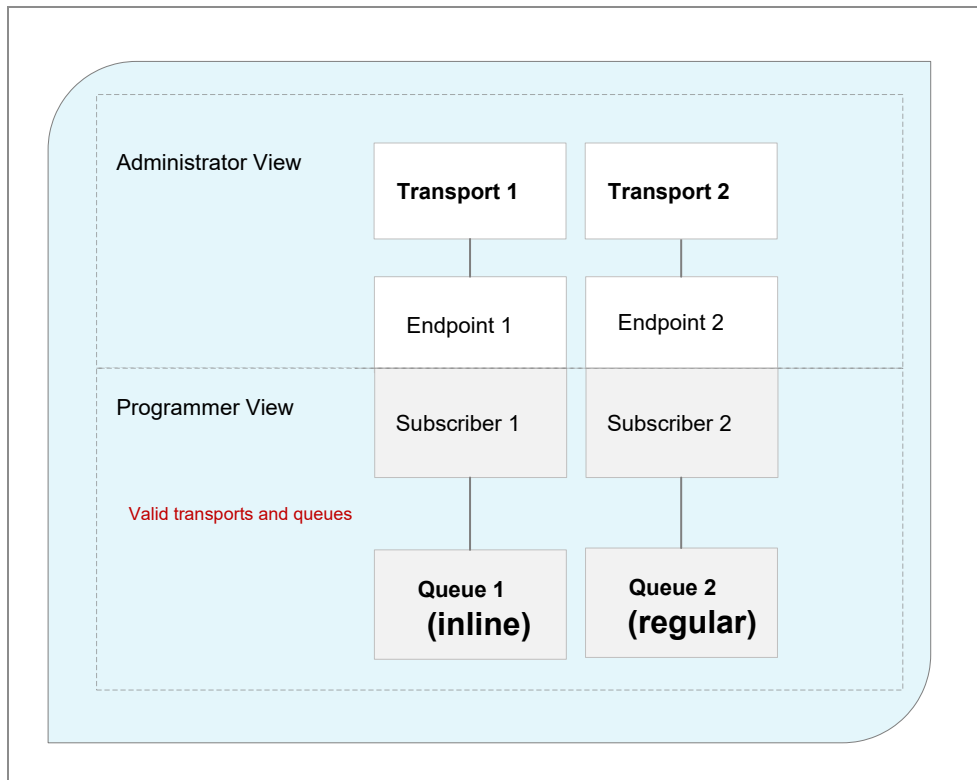
- Use ordinary queues instead of inline mode.
- Use separate transports to implement the two endpoints E1 and E2 (see [Separate Transports to Correct a Violation](#), which follows).

- Add subscribers S1 and S2 to only one inline queue (see [Single Queue to Correct a Violation](#), which follows).

Separate Transports to Correct a Violation

In the third diagram, transport T1 is associated with only one inline event queue, Q1. This arrangement does *not* violate the restriction.

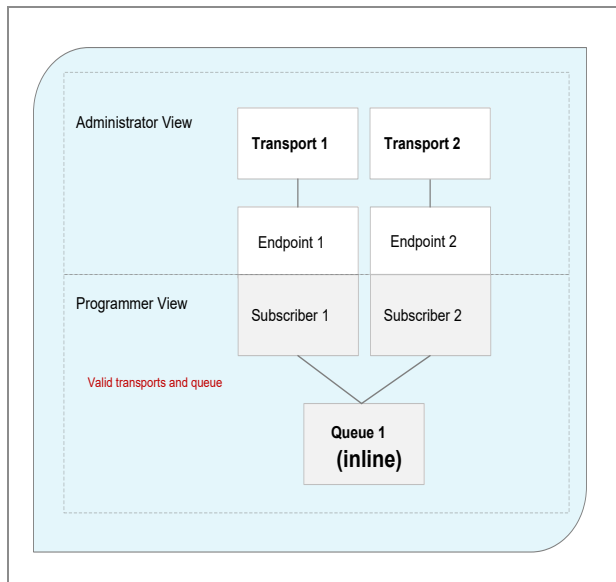
Figure 16: Legal Inline Queue, Separate Transports



Single Queue to Correct a Violation

In the fourth diagram, even though two subscribers (S1 and S2) share the same underlying transport (T1), they are associated with the same inline event queue (Q1). This arrangement does *not* violate the restriction.

Figure 17: Legal Inline Queue, Single Queue



Receive Spin Limit

Receive spin limit is an advanced parameter of transports. If you understand the role of threads in inbound message delivery, you can use it to eliminate a potential source of latency by increasing the CPU resources dedicated to inbound delivery.



Warning: Do not adjust the [Receive Spin Limit](#) except to achieve specific and well-defined performance objectives. Do not adjust unless you thoroughly understand *all* of the material about this topic.

For an introduction to message delivery, see “Inbound Message Delivery” in [TIBCO FTL Development](#).

Polling for Data

The TIBCO FTL library polls for available data during two phases of inbound message delivery:

- In the receive phase, it polls to receive data from a transport’s data source.

- In the dispatch phase, it polls to take messages from an event queue.

The paradigm is identical in both polling phases:

1. Determine the time limit for polling, that is, the *receive spin limit*.
2. Poll for available data.
 - If available, get the data and return it.
 - If not, continue polling until the time limit elapses.
3. When the time limit elapses, block until data becomes available. Blocking relinquishes the CPU resource to do other work.

Threads and Latency

Polling code operates within a thread.

- If the thread is already executing on a CPU core and polling for data, then when data becomes available, the thread immediately gets the data, without adding latency.
- However, if the thread has blocked, the context switching time until it resumes execution adds to message latency. If it is crucial that your application minimize latency, then consider avoiding this situation by adjusting receive spin limit.

Receive Spin Limit Tuning

The default values for receive spin limit yield good performance in many situations. When necessary, administrators can adjust receive spin limit as a parameter of individual transport definitions.

Goal: Minimal Latency

Blocking frees CPU resources for other threads. However, if minimal latency is crucial and CPU resources are plentiful, then you can avoid blocking by adjusting receive spin limit.

If unacceptable latency spikes occur during conditions of sparse message traffic, one possible cause could be that messages arrive shortly after the receive spin limit has elapsed, so the polling thread has recently blocked. In this situation, consider increasing the receive spin limit to a value that is greater than the maximum expected time between messages. Select a value that is large enough to prevent threads from blocking between consecutive messages. Test the results empirically.



Warning: Coordinate with developers to understand the thread behavior of application programs. When the application program creates many dispatch threads, use caution when increasing receive spin limits. Avoid starving threads by monopolizing CPU resources.

Goal: Optimization of CPU Resources

If CPU resources are scarce or expensive, then consider decreasing the receive spin limit so threads that are waiting for data relinquish their CPU resources to do other work.

Zero is an acceptable value. With this value, the TIBCO FTL library polls for available data only once. If data is not available, then the library code blocks immediately.

See Also

[Buffer and Performance Settings for Transports](#)

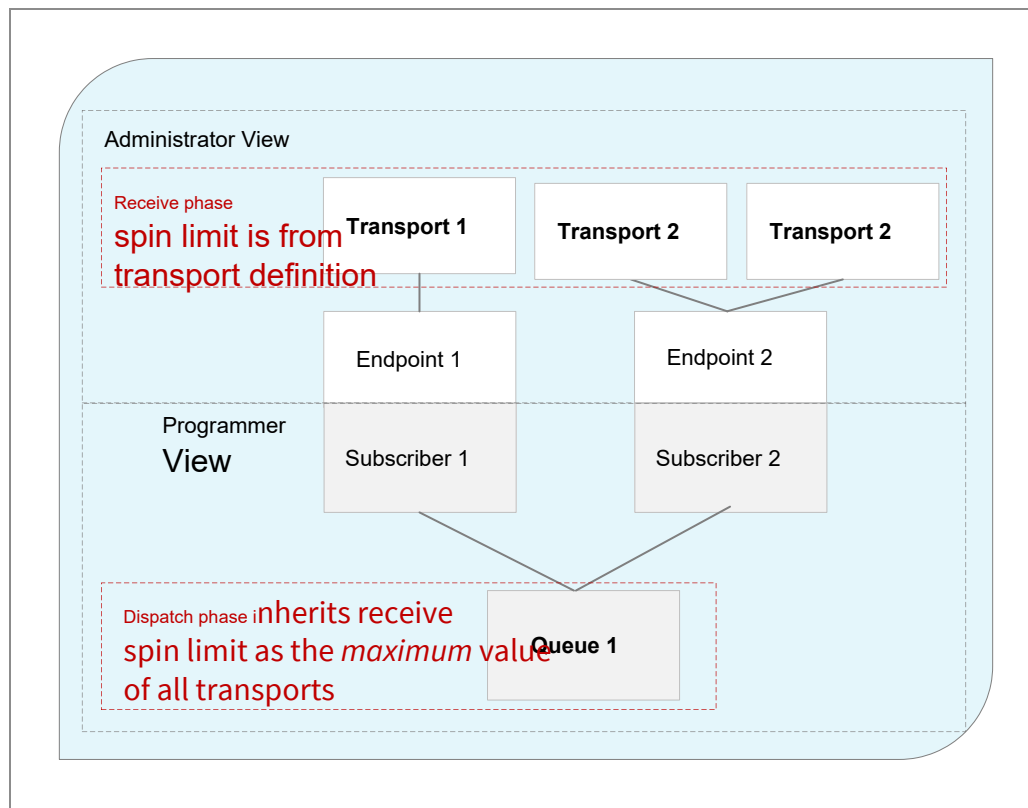
Inheritance of Receive Spin Limit From Transports to Dispatch

Receive spin limit is a parameter of transports, but its value governs the behavior of two delivery phases.

The receive phase operates at the transport level and takes its receive spin limit directly from the transport definition.

The dispatch phase operates at the event queue level and inherits its receive spin limit as the *maximum* value over all the transports that feed messages into the event queue (see the following diagram).

Figure 18: Receive Spin Limit, Example with Several Transports



Inline Mode

Regular operation separates the receive and dispatch phases into distinct threads. In contrast, inline mode merges the phases into a single thread, that is, the thread in which the program calls the dispatch method. Accordingly, for inline mode the governing spin limit in *both* polling phases is the maximum value over all the relevant transports (as mentioned regarding the dispatch phase).

UDP Packet Send Limit

With UDP-based transports such as multicast, fast sending programs can overwhelm slower receiving programs, which could result in retransmission-related latency, extra resource consumption of CPU cycles and network bandwidth, retransmission storms, or data loss. To prevent these scenarios, you can limit the rate at which sending programs transmit UDP packets to the network.

Administrators can empirically tune the flow of data packets from sending programs by adjusting the packet send limit, which is a required property of UDP-based transports.

Limiting the packet send rate also limits the data send rate (bytes per second), because the data send rate varies with the packet rate and the data sizes of the packets. However, even if the data send rate is low, a high packet send rate can still overwhelm a receiver. Hardware limits on the number of NIC I/O interrupts translate into limits on the receiver's inbound message rate. When an inbound packet arrives, the cost of the NIC interrupt is independent of the size of the data in the packet.

General Guidelines

- Tune the packet send limit parameter on the senders' transports. (This sending parameter has no effect at the receiving end.)
- When an application sends batches of several messages in its send calls, it is probably already using UDP bandwidth efficiently. For best results in most situations, set the packet send limit to zero. However, if slow receivers report dataloss, then consider tuning the packet send limit (see the next item).
- When an application rapidly sends one small message per send call, then adjust the packet send limit to increase throughput efficiency. Use these guidelines as you empirically tune this parameter:
 - The general goal is to determine the highest value that does not overwhelm any receiving hardware.
 - To begin, try a value within the range 50,000 to 150,000 packets per second. Select an initial value based on capacity of the receiving CPUs to handle NIC interrupts.
 - To reduce average latency, try raising the value.
 - If the number of packets overwhelms any receiver, try a lower value. A lower value raises average throughput by utilizing MTU capacity more efficiently.

To configure, see [Multicast Transport: Parameters Reference](#).

Transport Protocol Types

TIBCO FTL software supports several transport types, each characterized by the protocols it uses to establish a bus and to transfer message data.

TIBCO FTL software supports the following transport protocols.

- Dynamic TCP Transport (DTCP)
- Static TCP Transport
- Auto Transport
- Multicast Transport
- Process Transport
- Shared Memory Transport
- Direct Shared Memory Transport
- Reliable UDP Transport (RUDP)

Dynamic TCP Transport

A *dynamic TCP* (DTCP) transport bus establishes a set of TCP connections dynamically, decoupling the transport definitions from specific hosts and ports. You can use dynamic TCP transports to easily specify a variety of communication topologies. A DTCP transport can be server-based or peer-to-peer.

i Note: Dynamic TCP is an easy and flexible way to configure many topologies. (For backward compatibility, FTL software continues to support static TCP transport definitions; see [Static TCP Transport](#).)

Modes and Transport Groups

A dynamic TCP transport definition specifies one of the following modes:

- **Mesh** Endpoints form a full mesh.
- **Listen** Endpoints establish a bus by listening for connections within a transport group.
- **Connect** Endpoints establish a bus by connecting to all listening endpoints within a transport group.

Mesh

A dynamic TCP mesh uses a single transport definition to establish a bus composed of individual TCP pair connections. However, all the details of those pair connections are transparent to the administrator. The application and the realm service automatically and dynamically arrange those details: listening end, connecting end, host, and port.

Listen and Connect

You can build other dynamic TCP topologies by defining a listening end and a connecting end, and linking them with a common transport group name.

You can use these two definitions to implement any number of application endpoints. They establish a bus composed of TCP pair connections between every connecting endpoint and every listening endpoint. However, all the details of those individual pair connections are transparent to the administrator. The applications and the realm service automatically and dynamically arrange the hosts and port numbers.

Messages can potentially flow in either direction along each pair connection. The terms *listen* and *connect* refer only to the TCP connection protocol, and do not affect message communication.

You can use a pair of listen and connect definitions to assemble topologies such as such as pair connection, hub-and-spoke, and asymmetric multicast. (For diagrams of these topologies, see [Hub-and-Spoke](#) and [Asymmetric Multicast Topologies](#).)

Configuring only two transport definitions can dramatically simplify a bus with several endpoints. (In contrast, the same topology could require many static TCP transport definitions, which are tightly coupled to specific IP addresses.)

Scope of Dynamic TCP

A dynamic TCP bus is limited to applications that connect to a local cluster of FTL server *processes*. That is, client applications of a satellite FTL server cannot participate in a dynamic TCP bus with client applications of the primary FTL server.

However, such applications can still communicate through a transport bridge that connects the otherwise separate dynamic TCP buses that form at different locations. For an example, see [Bridges among Dynamic TCP Meshes](#).

Dynamic TCP Transport: Parameters Reference

The following tables describe the parameters specific to dynamic TCP transports and secure dynamic TCP transports in the configuration interface.

For web API access to transport definitions, see [Transport Definition Objects](#).

| GUI Parameter | JSON Attribute | Description |
|---------------|----------------|--|
| Mode | mode | <p>Required.</p> <p>Select a mode from the drop-down menu: mesh, listen, or connect.</p> <p>For the meaning of these values, see Dynamic TCP Transport.</p> |
| Group | virtual_name | <p>Required when the mode is either listen or connect.</p> <p>The group name ties together a listen end and a connect end, decoupling these definitions from the host and port.</p> |
| Subnet Mask | subnet_mask | <p>Optional.</p> <p>Limit the hardware interfaces that this TCP transport can use.</p> <p>When network administrators allocate subnets for specific purposes, use this parameter to comply.</p> <p>Specify a IPv4 or IPv6 CIDR mask. For example, in IPv4: 192.168.2.0/24. In IPv6: 2001:db8::/32.</p> |

| GUI Parameter | JSON Attribute | Description |
|---------------|----------------|---|
| Port Range | port_range | <p>Optional.</p> <p>Designate a range of port numbers to establish a TCP bus, e.g. 7777–7785.</p> <p>When absent, dynamic TCP transports use ephemeral listen ports to establish a bus. A firewall can thwart this mechanism. To circumvent the obstacle, designate a port range for this purpose and arrange for the firewall to open those ports.</p> <p>When present, all applications that use the transport definition bind one of the ports in the port range to establish a bus.</p> |

See also, [Buffer and Performance Settings for Transports](#).

Static TCP Transport

A static TCP transport communicates over a set of dedicated connections. A static TCP transport can be server-based or peer-to-peer.

i Note: In most cases it is simpler and more flexible to use dynamic TCP transports (see [Dynamic TCP Transport](#)).

Static TCP is a connection-oriented protocol. Its transport definitions are inherently fragmentary. You must define at least two complementary transport definitions to establish a bus. For more information, see [Pair Connections](#).

Although an individual static TCP connection links exactly two host computers, you can combine several connections into a bus with a more complex topology (see [Assembling Larger Topologies from Pair Connections](#)).

Static TCP Transport: Parameters Reference

The following tables describe the parameters specific to static TCP transports and secure static TCP transports in the configuration interface.

For web API access to transport definitions, see [Transport Definition Objects](#).

Connection List

You must specify at least one pair connection triplet: host, port, and initialization mode.

In the web API, the `hosts` attribute contains a collection of these triplet objects.

| GUI Parameter | JSON Attribute | Description |
|---------------|-------------------|---|
| Host | <code>host</code> | <p>Required.</p> <p>For the host, specify an interface either as a host name or as an IP address.</p> <p>Asterisk (*) is a special value, which designates all interfaces on the host computer (analogous to <code>INADDR_ANY</code>, but limited to TCP interfaces). This special value is available only with listen mode (that is, transports can listen for connections on all network interfaces, but they must connect to a <i>specific</i> interface).</p> |
| Port | <code>port</code> | <p>Required.</p> <p>Specify a port number.</p> |
| Mode | <code>mode</code> | <p>Required.</p> <p>You must select either Connect or Listen from the drop-down menu:</p> <p>connect</p> <p>The transport initiates TCP communication by sending a connection request to <code><host>:<port></code>.</p> <p>listen</p> <p>The transport listens for TCP connection requests at <code><host>:<port></code>.</p> <p>For more information, see Listen End and Connect End.</p> |

See also, [Buffer and Performance Settings for Transports](#).

Auto Transport

An *Auto* transport bus, like a DTCP transport, establishes a set of TCP connections dynamically, decoupling the transport definitions from specific hosts and ports. However, with an Auto transport, in the simplest configuration, you don't need to specify listen or connect ports; a transport name is all that is required. An Auto transport can only be server-based and is typically used for persistence service transports, eFTL service transports, and group server transports. One main difference between DTCP transport and Auto transport is that DTCP transports require additional ports to be open if running behind a load balancer or a firewall, whereas Auto transports redirect all communication via the FTL server port.

Scope of an Auto Transport

Auto transports may only be used to communicate with FTL server components. Auto transports cannot be configured on a client's endpoint. For example, clients never participate in an auto transport bus together (no "peer-to-peer"). Any communication occurs indirectly via FTL server components, for example, the persistence service ("server-based" only).

Multicast Transport (mcast)

A multicast transport is a peer-to-peer transport that carries reliable multicast communication among endpoints on many host computers. Multicast transports are efficient for high fan-out communication across a LAN.

Multicast transports excel at one-to-many communications.

Multicast transports use a negative acknowledgment protocol for reliable message transfer.

Multicast transports can also support one-to-one communications. For example, servers can send one-to-one replies to individual clients (that is, requestors). For configuration details, see [Configuring Multicast Inbox Communication](#).

Multicast transports can filter messages by subscriber interest.

Multicast Group as Shared Communication Medium

Multicast transport definitions can be either unitary or fragmentary. They do not require any initial connection protocol, so the determining factor is the use case.

- In a unitary use case, a set of communicating peers could all use the same transport. For example, all endpoints send on the same multicast group, and all endpoints listen to the same multicast group (or groups).
- In the more common fragmentary case, administrators control network data flow to achieve specific results, such as bandwidth allocation, performance, data segregation, or asymmetric multicast topologies. Sending peers and listening peers use related transports with the same port, but different send and listen groups.

Multicast Transport: Parameters Reference

The following table describes the parameters specific to multicast transports in the configuration interface.

For web API access to transport definitions, see [Transport Definition Objects](#).

| GUI Parameter | JSON Attribute | Description |
|---------------|----------------|---|
| Host | host | <p>Optional.</p> <p>Specify a network interface or an IP address. The transport resolves the value of the host parameter when the application establishes a bus.</p> <p>If you supply an IP address, the transport first treats the IP address as a host address, comparing it against the interface addresses of the host computer, and it uses the first matching interface. If it does not find a direct match, the transport next treats the IP address as a subnet address, comparing the IP address against the network addresses of the interfaces of the host computer, and uses the first interface address whose network address matches the IP address.</p> <p>Asterisk (*) is a special value, which designates all</p> |

| GUI Parameter | JSON Attribute | Description |
|----------------------|-------------------|--|
| | | <p>network interfaces on the host computer (that is, INADDR_ANY).</p> <p>For testing multicast on a single host, you may supply the loopback interface, 127.0.0.1.</p> <p>When absent, the transport uses the default interface address. First it calls <code>gethostname</code> to get the host name string. Then it calls <code>gethostbyname</code> with that host name to get the IP address of the default network interface.</p> |
| Port | port | <p>Required.</p> <p>Specify a multicast port.</p> |
| Listen Groups | lgrps | <p>The transport joins all the listen groups that you specify. You may configure between zero and 64 listen groups.</p> <p>A listen group enables inbound communication.</p> <p>When absent, the transport does not support receive nor receive inbox communication.</p> |
| Multicast Send Group | sgrp | <p>When present, the transport sends on this multicast group. You may supply at most one send group.</p> <p>A send group enables outbound communication.</p> <p>When absent, the transport does not support send nor send inbox communication.</p> |
| Packet Send Limit | packet_send-limit | <p>Required. Use this upper limit to prevent fast senders from overwhelming slower receivers.</p> <p>This value limits the maximum number of UDP packets per second that a program can <i>send</i> over the multicast transport. When a sending program exceeds this maximum, the transport buffers the excess packets in order to keep its outbound packet rate at</p> |

| GUI Parameter | JSON Attribute | Description |
|---|----------------------|---|
| | | <p>or below this limit.</p> <p>Zero is a special value, indicating no upper limit on sending programs.</p> <p>For a detailed explanation, see UDP Packet Send Limit.</p> |
| Retransmission Receiver Loss Suppression Level | rx_c_loss_tolerance | <p>When absent or zero, the transport always requests retransmission (within the constraints of the reliability window).</p> <p>When present and non-zero, the transport can suppress retransmission requests from receivers that miss packets in excess of this loss percentage, and can report this condition in advisory messages. Specify the suppression behavior with the following parameter.</p> <p>For details, see Multicast Retransmission Suppression.</p> |
| Retransmission Receiver Loss Suppression Behavior | rx_c_advise_only | <p>This parameter modifies the behavior of the previous parameter.</p> <p>When the loss suppression level is non-zero, you can select either of two behaviors:</p> <ul style="list-style-type: none"> • Stop NAKs Suppress retransmission requests and report advisory messages. • Warning Only Output warning log messages, but do not suppress retransmission requests. <p>For details, see Multicast Retransmission Suppression.</p> |
| Send Messages | enable_inbox_support | <p>Only on Subscriber Interest (or the JSON attribute is set to the <i>boolean</i> value <code>true</code>) the multicast transport carries only those messages that match subscriber interest, filtering outbound messages at sending endpoints (see Subscriber Interest).</p> |

| GUI Parameter | JSON Attribute | Description |
|------------------|-------------------------------|---|
| | | <p>However, enabling this parameter is not sufficient. You must also configure the transport's multicast groups to support <i>bi-directional</i> communication on the transport bus (see Listen Groups). The two directions, inbound and outbound, need not use the same multicast group.</p> <p>Always (or the JSON attribute is set to the <i>boolean</i> value <code>false</code>) the multicast transport carries all messages, without filtering. This value is the default, for backward compatibility.</p> |
| Loopback On Send | <code>enable_loopback</code> | <p>When enabled, the transport includes the loopback interface in its bus, so co-located application processes can communicate over the multicast bus. (Otherwise the multicast bus carries messages only over the network.)</p> <p>To properly configure loopback behavior, select this parameter for all the transports of the bus that use loopback (that is, on the sending transports <i>and</i> the receiving transports).</p> <div> <p>Note: The internal details of loopback semantics vary by operating system. For best results, configure for consistent behavior on every platform, rather than coupling the transport definition to a specific operating system.</p> </div> |
| Reliability Time | <code>reliability_time</code> | <p>Required.</p> <p>The transport retains multicast data during this time window (a floating point value, in seconds).</p> <p>Transports in sending applications retain recent multicast data in order to redeliver data to receiving applications upon request.</p> <p>The reliability time of the sending transport</p> |

| GUI Parameter | JSON Attribute | Description |
|----------------------|-----------------|--|
| | | <p>determines the effective reliability window of all applications receiving data from it. A receiving transport does not request data beyond the reliability window of the sending transport (even if the receiver configures a longer reliability window).</p> <p>For this value, the transport supports time granularity of 0.001 (1 millisecond) and a maximum value of 120.0 seconds (2 minutes). Within application programs, the base library throws an exception when it detects a non-zero value less than 1 millisecond or greater than 120 seconds.</p> <p>Zero is a special value. Sending transports do not store data for retransmission. Receiving applications never request retransmission.</p> |
| Reliability Memory | reliability_mem | <p>When present, the transport retains multicast data up to this maximum (in bytes). See also Size Units Reference.</p> <p>When absent, the default is zero, a special value indicating no reliability limit by size.</p> <div> <p>Note: reliability_mem is not effected if reliability_time == 0</p> </div> |
| UDP Send Buffer Size | udp_sndbuf | <p>When present, the library requests a buffer of this size, in bytes, for outbound multicast data. Operating system constraints can limit or override this request. The value must be a non-negative integer. (See also Size Units Reference.)</p> <p>When absent or zero, the library requests the default buffer size, 16MB.</p> <p>In most situations the default buffer size yields the best results. However, in some high-volume situations, larger buffers could yield higher</p> |

| GUI Parameter | JSON Attribute | Description |
|-----------------------------|-------------------|---|
| | | throughput (depending on other factors). |
| UDP Receive Buffer Size | udp_rcvbuf | <p>When present, the library requests a buffer of this size, in bytes, for inbound multicast data. (Operating system constraints can limit or override this request.) The value must be a non-negative integer. (See also Size Units Reference.)</p> <p>When absent or zero, the library requests the default buffer size, 16MB.</p> <p>In most situations the default buffer size yields the best results. However, in some high-volume situations, larger buffers could yield higher throughput (depending on other factors).</p> |
| UDP Sender Port Range | udp_sender_port | <p>Port Ranges</p> <p>When present, the transport binds available ports within these ranges.</p> <p>When absent, the transport uses ephemeral ports assigned by the operating system.</p> <p>For motivation and details, see Multicast Port Ranges.</p> <p>For syntax, see Multicast Port Range Syntax.</p> |
| UDP Receiver Port Range | udp_receiver_port | |
| UDP Maximum Packet MTU Size | mtu_size | <p>When networks support a larger MTU size than the standard, set this parameter to the maximum packet size, in bytes.</p> <p>When present, the multicast transport uses this size rather than the standard size.</p> |

See also, [Buffer and Performance Settings for Transports](#).

Multicast Port Ranges

To cooperate with firewalls, you can explicitly specify ranges of ports for multicast transports to use instead of ephemeral ports.

Background

Ordinarily, multicast transports use ephemeral UDP ports for data transmission and protocols. However, the operating system could assign ephemeral ports that are not open through a firewall, which interferes with transport operation.

To cooperate with firewalls, you can explicitly specify two ranges of UDP ports that are open through the firewall. In this configuration the transport binds ports from these ranges instead of requesting ephemeral ports from the operating system. Ensure that the ports you specify are *open in both directions* through the firewall.

Port Ranges Correspond to Transport Abilities

If a transport definition configures one or more `Listen Groups`, the runtime transport binds one of the available ports in the transport definition's `UDP Receiver Port Range`. Conversely, if the transport definition does not configure a listen group, then the runtime transport does not bind a receiver port.

If a transport definition configures a `Send Group`, the runtime transport binds one of the available ports in the transport definition's `UDP Sender Port Range`. Conversely, if the transport definition does not configure a send group, then the runtime transport does not bind a sender port.

Sizing

Ensure that the ranges you specify include enough ports.

Each application process binds at most one port from the transport definition's `UDP Receiver Port Range`, and at most one port from the transport definition's `UDP Sender Port Range`.

When many application processes run on a multi-core host computer, the port ranges must include enough distinct ports for all the processes on the host.

In contrast, when application processes run on separate host computers, they can bind the same UDP port numbers without collision.

Disjoint Ranges

For best results, ensure that the port numbers in the `UDP Receiver Port Range` and the `UDP Sender Port Range` do not overlap. Overlapping ports could slow transport creation.

Multicast Port Range Syntax

You can use the following syntax to specify ranges of UDP ports that are open through a firewall.

| Syntax | Example | Description |
|-----------------------|-----------------------------|---|
| <n> | 125 | A single port number |
| <n>, <n> | 101,106,111 | A set of single port numbers |
| <n>-<m> | 100-130 | An inclusive sequence of port numbers |
| <n>-<m>, <p>-<q>, <r> | 101,103,200-210,235-250,300 | A combination of single ports and sequences |

Configuring Multicast Inbox Communication

Multicast transports can carry one-to-one communication.

Procedure

1. Configure the transport's multicast groups to support *bi-directional* communication on the transport bus (see [Multicast Transport: Parameters Reference](#) and [Multicast Transport: Parameters Reference](#)). The two directions, inbound and outbound, need not use the same multicast group.
2. Select the send inbox and receive inbox abilities for endpoints that use the transport.

Multicast Retransmission Suppression

The high volume of retransmission requests from chronically slow or lossy receivers can overwhelm publishing applications and even overwhelm the entire network. Receiver loss can result from faulty network hardware, oversubscribed applications, or underpowered host computers.

The retransmission suppression feature can detect lossy receivers and suppress their retransmission requests. Administrators can set a receiver loss threshold. If the percentage

of lost packets between any pair of multicast subscriber and publisher exceeds this threshold, then the transport suppresses all retransmission requests from that subscriber to that publisher.

To configure this threshold, see [Multicast Transport: Parameters Reference](#).

Process Transport (PROC)

Endpoints within a single program process can communicate using a process (PROC) transport. Process transports are peer-to-peer transports that communicate at high speeds and exhibit very low latency.

Process transports are an excellent strategy for fast communication among threads within a process.

Process transport definitions are inherently unitary.

Process transports cannot participate in transport bridges.

Administrators arrange process transports *only* when program developers request it.

For best performance, do not bind an endpoint to both a process transport and a slower transport, as the slower transport limits the performance of the process transport. (See [Multiple Transports and Serial Communications](#).)

Process Transport: Parameters Reference

The following table describes the parameters specific to process transports in the configuration interface.

For web API access to transport definitions, see [Transport Definition Objects](#).

| GUI Parameter | JSON Attribute | Description |
|---------------|-------------------|---|
| Send Behavior | backlog_full_wait | This parameter governs the behavior of the transport when the bus is full (that is, it cannot accept any more outbound messages). |

| GUI Parameter | JSON Attribute | Description |
|------------------|-------------------|---|
| | | <p>Non-Blocking</p> <p>When the bus is full, the transport holds outbound messages in a backlog buffer in process memory until the bus can accept them.</p> <p>Blocking</p> <p>When the bus is full, then the transport blocks send calls in the program.</p> <p>For more information, see Send: Blocking versus Non-Blocking</p> |
| Buffer Size | size | <p>Required.</p> <p>A process transport passes messages between threads through a buffer in process memory. This parameter determines the maximum number of messages that the buffer can hold.</p> <p>When the buffer is full, a <i>non-blocking</i> process transport discards new messages from the sending endpoint (instead of placing them in the buffer). <i>Blocking</i> process transports block until the buffer can accept the new message.</p> |

Shared Memory Transport

Endpoints in programs that run on the same host computer can communicate using a shared memory transport. Shared memory transports are peer-to-peer transports that communicate at high speeds and exhibit very low latency.

Shared memory transports are an excellent strategy for fast communication among processes on multi-core hardware.

Shared memory transport definitions are inherently unitary.

Topology

The natural topology of a shared memory bus is the mesh (complete graph) of all its runtime transports. All runtime transports can use the bus for both sending and receiving

operations. When appropriate, you can configure connectors to artificially limit the abilities that an endpoint may use.

Non-Blocking Send

See [Shared Memory Transports and One-to-Many Non-Blocking Send](#).

Configuring a Shared Memory Transport for Access by Different Users: UNIX

On UNIX platforms, the shared memory segment underlying a shared memory transport belongs to a specific user. Nonetheless, you can configure the UNIX environment so that different users can access it.

Before you begin

The users must be in the same group.

Procedure

1. Set the umask as follows:

```
umask u=rwx,g=rwx,o=rx
```

After setting the umask globally, the application process automatically creates a segment that different users can access.

Configuring a Shared Memory Transport for Access by Different Users: Windows

On Windows platforms, the shared memory segment underlying a shared memory transport can either be a global segment, or belong to a specific user. A parameter of the transport definition configures this distinction.

Procedure

1. Set the `Global Scope` property in the transport definition.

See [Global Scope](#)

Shared Memory Transport: Parameters Reference

These parameters are specific to shared memory transports in the configuration interface.

For web API access to transport definitions, see [Transport Definition Objects](#).

| GUI Parameter | JSON Attribute | Description |
|----------------|---------------------------|--|
| Segment Name | <code>segment_name</code> | <p>Required.</p> <p>Name of the shared memory segment.</p> <p>Use only ASCII alphanumeric characters in the segment name.</p> <p>The maximum length of the segment name depends on the operating system:</p> <ul style="list-style-type: none">• Mac OS X: 16 characters• UNIX: <code>PATH_MAX</code>• Windows: <code>MAX_PATH</code> |
| Segment Size | <code>size</code> | Size of the shared memory segment. |
| Directory Name | <code>dirname</code> | <p>Optional.</p> <p>Each shared memory transport uses a scratch directory to store information about the runtime transports that use its shared memory segments.</p> <p>When present, this directory path specifies the scratch directory. (If the directory you specify doesn't exist in the file system, the transport attempts to create it.)</p> <p>When absent, the default scratch directory is <code>/tmp</code> (or its equivalent on non-UNIX platforms).</p> |

| GUI Parameter | JSON Attribute | Description |
|---------------|----------------|---|
| Global Scope | globalscope | <p>Windows only (on other platforms, the base library ignores this parameter).</p> <p>On Windows platforms, shared memory segments can have either of two scopes:</p> <ul style="list-style-type: none"> • <i>Local</i> to a specific user, that is, a login name. • <i>Global</i>, that is, available to all usernames. <p>If the process instances using the transport run with different usernames, then you must use a global segment. To specify a global segment, enable this parameter (or in the web API, set the JSON attribute to <code>global</code>).</p> <p>You may use a local segment <i>only</i> if the process instances using the transport all run with the same username. To specify a local segment, specific to the user running the application process, disable this parameter (or in the web API, set the JSON attribute to <code>local</code>).</p> <p>For further information, see documentation for the <code>CreateFileMapping</code> function on the Microsoft Developer Network (ISDN) web site.</p> |

See also, [Buffer and Performance Settings for Transports](#).

Direct Shared Memory Transport

Direct publishers and direct subscribers can use a direct shared memory transport to communicate within the same host computer. Direct shared memory transports are peer-to-peer transports similar to shared memory transports, with some additional restrictions. They communicate at high speeds and exhibit extremely low latency.

Direct publishers and direct subscribers *require* direct shared memory transports. Direct shared memory transports yield fast communication among processes that use direct publishers and direct subscribers within a multi-core hardware host computer.

Direct shared memory transport definitions are inherently unitary.

Parameters

Direct shared memory transports use the same set of parameters as shared memory transports. For details, see [Shared Memory Transport: Parameters Reference](#).

Topology

The natural topology of a direct shared memory bus is a hub-and-spoke topology, in which a single direct publisher object at the hub sends to potentially many direct subscriber objects.

All of the endpoints can create direct subscribers on the bus, and a process can even create many direct subscribers.

A suite of application processes that communicate over a direct shared memory bus may contain *only one* direct publisher object at a time. It is the shared responsibility of the application developers and the administrators to ensure this condition.

For example, suppose application process A1 creates a direct publisher, DP1. A1 may not ever create another direct publisher object. Furthermore, before a new process, A2, can create a direct publisher on the bus, A1 must close DP1, and A1 must exit.

Uniqueness

If you implement an endpoint with a direct shared memory transport, it must be the *only* transport definition that implements that endpoint.

Size

The default size of the shared memory segment is 1 megabyte. For best results, use the default size. Larger sizes can result in significant backlog, which can increase latency.

The minimum size is also 1 megabyte.

Blocking and Non-Blocking Send

Direct shared memory transports support either blocking or non-blocking send.

When one or more direct subscribers are slower than a direct publisher, the shared memory segment can become full:

- **Blocking** - When the administrator has configured blocking send, the direct

publisher's reserve call blocks until the direct subscribers consume the data, which frees up memory to reserve more buffers.

- **Non-Blocking** - When the administrator has configured non-blocking send, the direct publisher discards older data buffers, so it can reuse their memory to reserve new buffers. Slow consumers cannot access the discarded buffers. The base library reports data loss advisories to slow consumers.

Reliable UDP Transport (RUDP)

A reliable UDP (RUDP) transport is a peer-to-peer transport that carries reliable communication over a set of dedicated connections. Endpoints on an RUDP bus exchange UDP datagrams. RUDP transports use a positive acknowledgement protocol for reliable message transfer.

Consider using RUDP transports for network client-server communications that are not sensitive to latency, in situations of high fan-out or fan-in combined with low data volume. (TCP yields faster data transfer. However, RUDP features more efficient fan-out as RUDP consumes only one socket, which accepts many connections.)

RUDP is a connection-oriented protocol. Its transport definitions are inherently fragmentary. You must define at least two complementary transport definitions to establish a bus. For more information, see [Pair Connections](#).

Although an individual RUDP connection links exactly two host computers, you can combine several connections into a bus with a more complex topology (see [Assembling Larger Topologies from Pair Connections](#)).

RUDP Transport: Parameters Reference

The following tables describe the parameters specific to RUDP transports in the configuration interface.

For web API access to transport definitions, see [Transport Definition Objects](#).

Connection List

You must specify at least one triplet: host, port, and initialization mode.

In the web API, the `hosts` attribute contains a collection of these triplet objects.

| GUI Parameter | JSON Attribute | Description |
|---------------|----------------|--|
| Host | host | <p>Required.</p> <p>For the host, specify an RUDP interface either as a host name or as an IP address (IPv4 or IPv6).</p> <p>Asterisk (*) is a special value, which designates all RUDP interfaces on the host computer (that is, the union of INADDR_ANY and INADDR6_ANY). This special value is available only with Listen mode (that is, RUDP transports can listen for connections on all network interfaces, but they must connect to a <i>specific</i> interface).</p> |
| Port | port | <p>Required.</p> <p>Specify a port number.</p> |
| Mode | mode | <p>Required.</p> <p>You must select either Connect or Listen from the drop-down menu:</p> <p>connect</p> <p>The transport initiates TCP communication by sending a connection request to <host>:<port>.</p> <p>listen</p> <p>The transport listens for TCP connection requests at <host>:<port>.</p> <p>For more information, see Listen End and Connect End.</p> |

Buffer and Performance Settings Specific to RUDP

See also, [Buffer and Performance Settings for Transports](#).

| GUI Parameter | JSON Attribute | Description |
|-------------------------|----------------|---|
| UDP Send Buffer Size | udp_sndbuf | <p>When present, the library requests a buffer of this size, in bytes, for outbound RUDP data. (Operating system constraints can limit or override this request.) The value must be a non-negative integer. (See also Size Units Reference.)</p> <p>When absent or zero, the library requests the default buffer size, 16MB.</p> <p>In most situations the default buffer size yields the best results. However, in some high-volume situations, larger buffers could yield higher throughput (depending on other factors).</p> |
| UDP Receive Buffer Size | udp_rcvbuf | <p>When present, the library requests a buffer of this size, in bytes, for inbound RUDP data. (Operating system constraints can limit or override this request.) The value must be a non-negative integer. (See also Size Units Reference.)</p> <p>When absent or zero, the library requests the default buffer size, 16MB.</p> <p>In most situations the default buffer size yields the best results. However, in some high-volume situations, larger buffers could yield higher throughput (depending on other factors).</p> |

Buffer and Performance Settings for Transports

This topic explains buffer and performance parameters that are common to several transport protocols.

| GUI Parameter | JSON Attribute | Description |
|-----------------------|-------------------|---|
| Send Behavior | backlog_full_wait | <p>This parameter governs the behavior of the transport when the bus is full (that is, it cannot accept any more outbound messages).</p> <p>Non-Blocking</p> <p>When the bus is full, the transport holds outbound messages in a backlog buffer in process memory until the bus can accept them. Also set the Buffer Sizes: Backlog (below).</p> <p>Blocking</p> <p>When the bus is full, then the transport blocks send calls in the program.</p> <p>For more information, see Send: Blocking versus Non-Blocking</p> |
| Buffer Sizes: Backlog | backlog_size | <p>Required with Send Behavior, Non-Blocking described above.</p> <p>This parameter determines the size, in bytes, of the backlog buffer. The default value is 64 megabytes for peer-to-peer transports and 256 megabytes for persistence transports.</p> <p>With zero bytes of backlog buffer, a non-blocking send call immediately discards its message if the transport cannot accept outbound data.</p> |
| Receive Spin | recv_spin_limit | This parameter governs blocking behavior of threads when |

| GUI Parameter | JSON Attribute | Description |
|---------------|----------------|---|
| Limit | | <p>waiting for message data to arrive.</p> <p>The default value for most transport protocols is 0.0 seconds. However, for shared memory and direct shared memory transports the default value is 0.01 seconds.</p> <p>Warning: Do not adjust this parameter unless you thoroughly understand the material in Receive Spin Limit.</p> |

Formats

A *format* defines the set of fields that a message can contain: including field names and data types.

Formats: Managed, Built-In, and Dynamic

TIBCO FTL software supports three broad categories of format: managed, built-in, and dynamic formats. These categories have differing constraints and performance characteristics.

Dynamic Format

An application can define a format dynamically, as it constructs a message. That is, the sending program implicitly determines the set of fields and their data types as it sets field values.

When a program creates a message, and specifies an unrecognized format name (that is, neither built-in nor preloaded), then TIBCO FTL software automatically treats it as a dynamic format.

While easy to use, dynamic formats are inefficient because each message must be self-describing. That is, a dynamic format message necessarily includes its own format metadata. Expect larger message sizes, slower transmission, slower processing, and longer field access times.

Dynamic formats are useful for rapid prototyping projects. To improve performance, you can switch to defining the application's formats as managed formats when the project nears the production stage.

Unnamed Dynamic Format

A program can create a message with a dynamic format, but supply null as the format name. Such *unnamed dynamic formats* are single-use. Administrators cannot ever upgrade them to managed formats.

You can use unnamed dynamic formats as a convenience when you cannot determine the structure and content of a message in advance. Unnamed formats save you the effort of naming the formats and keeping track of the names.

The performance of unnamed dynamic formats is identical to named dynamic formats.

Managed Format

Application architects and developers determine the set of formats that an application needs. Administrators then define those managed formats globally.

Administrators make managed formats available to an application by including them as *preload formats* for the application.

Programs can use these managed formats to compose new messages, or to unpack inbound messages. A message with a managed format is small and fast. It does not carry format metadata; instead, applications get that metadata from the realm service.

For efficiency and for safety, administrators can use the `Manage All Formats` and `Dynamic Message Formats` parameters to restrict the formats available to some or all of the applications in a realm. That is, administrators can require that applications use only preload formats and built-in formats, and prohibit applications from using dynamic formats.

Built-In Format

TIBCO FTL software defines a set of highly optimized built-in formats, which are defined within the base library and always available to all applications. No administrative action is required to make them available.

The performance characteristics of built-in formats are similar to managed formats. When used properly, a built-in format can outperform a managed format.

For more information, see [Built-In Formats](#).

Defining Formats

Administrators define the set of formats in a realm. Administrators also regulate the subset of formats available to each application.

Procedure

1. Coordinate with developers.

The first step is to consult with developers to determine the format requirements. To begin the discussion, the developer completes the [TIBCO FTL Format Coordination Form](#). Then developer and administrator review the forms together, and modify them to reflect your agreement.

2. Define managed formats in the realm.

Define the managed formats as specified in the final consensus revision of the coordination forms. Use the configuration interface to define the formats (see [Formats Grid](#)).

3. Preload formats.

When you define the application, add managed formats to the [Preload Formats](#) column. When a suite of application programs must exchange messages, ensure that each application definition preloads the formats that the program sends and receives.

The term *preload* implies that the application loads these formats when it receives its realm definition. Furthermore, the only managed formats that an application can use are its preload formats.

The list of preload formats does not affect built-in formats nor dynamic formats. You need not include built-in formats in the list of preload formats, as built-in formats are always available to all applications.

4. Allow or disallow dynamic formats.

When the message structure and content are highly variable, and applications cannot determine or limit that content in advance, dynamic formats might be appropriate.

If developers and administrators agree that the application requires dynamic formats, and that this will not degrade the performance of other applications in the realm (for example, by consuming bandwidth) then administrators can permit dynamic formats. (See [Message Formats Administration](#).)

What to do next

To upgrade dynamic formats to managed formats while an application is running, see [Add formats](#).

Before modifying the definition of a managed format, see [Add fields](#).

Built-In Formats

Built-in formats are optimized for efficiency, and are always available. Administrators do not need to explicitly define them, nor specify them among an application's preload formats.

- Opaque
- Keyed Opaque

Affiliated FTL Servers

You can configure a family of *affiliated* FTL server processes that serve a single realm, cooperating to distribute fault-tolerant services at one or more geographic locations. A family can include clusters of primary servers, satellite servers, disaster recovery servers, and auxiliary servers.

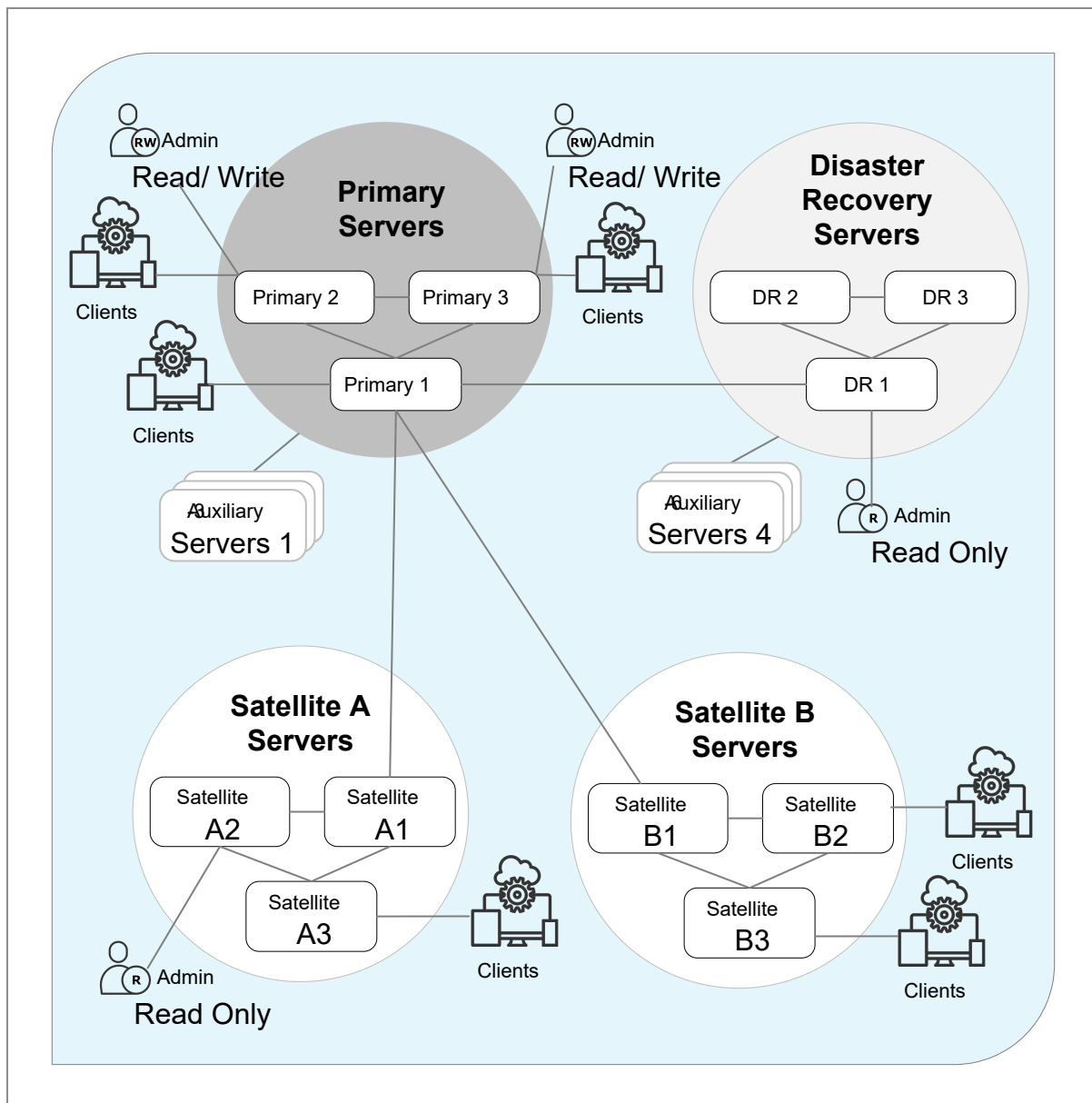
Server Roles

Affiliated FTL server processes operate in localized clusters. Each cluster plays a specific role.

Administrators configure the relationships among affiliated servers in the server configuration files.

The diagram illustrates a family of cooperating FTL servers.

Figure 19: Affiliated FTL Servers



Primary

The *primary servers* (dark gray cluster circle) form the central hub within a family of affiliated servers.

- Servers in the primary cluster function as interchangeable peers.
- Primary servers operate within a local network.

- A client application process can specify a list of server URLs, and it can attempt to connect to any of them. Nonetheless, each client process needs and maintains a connection to exactly one FTL server at a time.
- Each client sends monitoring and log data to its server (more precisely, to the realm service that the server provides).
- The servers in a localized cluster (any circle) present the same status view, as their realm services share client monitoring and log data. However, servers do not share client data outside their cluster circle.
- Administrators can update the realm definition at any of the primary servers. The primaries share that definition with one another so they all remain up to date. The primaries deploy updates to their clients and to affiliated satellite servers.

Satellite

Optional *satellite servers* (white circles) provide local service to clients that are geographically distant from the primaries, or clients that are insulated from the primaries (to fulfill enterprise requirements).

- Servers in a satellite cluster function as interchangeable peers within a local network.
- Separate satellite clusters (separate white circles) do not intercommunicate.
- Several remote satellite clusters can connect to any primary. A cluster cannot be a satellite of another satellite.
- A client application process can specify a list of server URLs, and it can attempt to connect to any of them. Nonetheless, each client process needs and maintains a connection to exactly one FTL server at a time.
- A satellite server does not accept client connection requests until it first receives a realm definition from a primary server.
- Each client sends monitoring and log data to its server.
- The servers in a localized cluster (any circle) present the same status view, as their realm services share client monitoring and log data. However, servers do not share client data outside their cluster circle.
- A satellite initiates a TCP connection to a primary server, and not the reverse.
- A satellite receives the realm definition from a primary. When an administrator deploys an updated realm definition, the primaries push the deployment to the satellites. (Administrators cannot deploy updates directly to a satellite server.)

- A satellite is loosely coupled to a primary. If one of them became inoperative, or the connection between them broke, the primary would not take any restorative action. Rather, the satellite would attempt to reconnect to a primary, continuing to serve its local clients in the interim. Primaries can deploy a new realm definition while satellites are disconnected (but see [Disconnected Satellite](#)).

Disaster Recovery

Optional *disaster recovery servers* (light grey cluster circle) provides a remote failover point in case a disaster disrupts service at the main site.

For background information and details, see [Disaster Recovery](#).

- The disaster recovery cluster functions as a hot standby to the primary cluster.
- Disaster recovery servers do not accept client connections until administrators explicitly promote them to primary status.

Auxiliary

Optional *auxiliary servers* (outside any cluster circle) are servers that you designate for a special purpose.

For details, see [Auxiliary Servers](#).

Modifications to the Realm Definition of Affiliated Servers

Administrators can modify the realm definition using the browser or web configuration interfaces.

- Only a primary server can accept realm modifications and updates directly from administrators, or from another primary server.
- A satellite server can accept updates only from a primary server or from another satellite in the same satellite cluster.
- A disaster recovery server can accept updates only from a primary server, or from another server in the disaster recovery cluster.

Disconnected Satellite

Although a primary server can accept an update even while disconnected from a satellite, avoid this practice, as it can cause administrative complexities. *Before* attempting to deploy an updated realm definition, check the health of all affiliated servers and correct any network issues.

Consider a situation in which a WAN fault disconnects a satellite cluster from its primary cluster. Nonetheless, an administrator deploys an updated realm definition at a primary. The deployment can proceed (unless some other condition prevents it, such as a No answer from a client).

Meanwhile, if some of the satellites are still operating, they continue to service client requests with information from the outdated realm definition.

When the satellites reconnect to the primaries, the primaries deploy the updated realm definition to the satellites. Because the update is already deployed at other affiliated servers, the primaries do not first test the update against the satellites and their clients. Instead, the primaries deploy the update to the satellites, and the satellites immediately deploy it to their clients, even though the update might cause difficulties for some clients of the satellites.

If you must deploy an updated realm definition in such circumstances, then as the satellites reconnect and deploy the update, it is prudent to carefully monitor the satellites' clients.

Configuring Satellite FTL Servers

You can designate a cluster of satellite servers to provide services at a geographically remote location.

Procedure

Task A Configure Primary Servers

1. If authentication is required, supply credentials that the primary servers can use to authenticate themselves to the satellite servers in the user and password parameters.

Ensure that this username is in the authorization group [FTL Server Authorization Groups](#).

Task B Configuring Satellite Servers

2. Supply the `satelliteof` parameter. The value is a URL list indicating the locations of the primary servers.
3. If authentication is required, supply credentials that the satellite servers can use to authenticate themselves to the primary servers in the `server.user` and `server.password` parameters.

Ensure that this username is in the authorization group `ftl-internal`.

Task C Configuring Clients

4. In the `realm connect` call, programmers supply the locations of the local cluster of FTL servers:

Supply the locations as a pipe-separated list of URLs as an argument.

Realm Service

The realm service contains the complete realm definition, which configures communications among all client applications.

The realm service distributes to each client a version of the realm definition that is tailored for that specific client. The client process stores this tailored realm definition in a realm object.

Client applications and services send their operating metrics to the realm service. To use metrics data, see [Catalog of Metrics](#) .

Also see: [Realm Service Configuration Parameters](#).

Developing Secure Applications

To implement security, application developers focus on the realm connect call and its arguments. Complete this task, or use its steps as a checklist.

Administrators determine whether authentication and, optionally, TLS are required for client connections.

If authentication is required, application developers must pass properties that identify the application to the realm connect call. There are three possibilities:

- **Basic auth (username and password):** The application must set the user name and user password properties. The administrator must configure at least one of these auth providers like the built-in flat-file authenticator, the built-in ldap authenticator, or the customizable HTTP(s) authenticator.
- **mTLS auth:** The application must set the `CLIENT_CERT` and `CLIENT_PRIVATE_KEY` properties. If the private key is encrypted, the application must set the `CLIENT_PRIVATE_KEY_PASSWORD` property. TLS is required (see below). The administrator must configure FTL server to use the mTLS auth provider.
- **Oauth2:** The application must provide an oauth2 access token (in signed JWT format), or the URL of an oauth2 server and the credentials needed to obtain an access token. The administrator must configure FTL server to use the oauth2 provider.

If TLS is required (in addition to authentication), application developers must specify the https scheme for all URLs passed to the realm connect call. Applications must also specify how to trust FTL server's TLS certificate. There are two possibilities:

- **Trust file:** The application must set the `TRUST_FILE` property. The trust file is supplied by the administrator. This is the only valid method when the administrator uses FTL-generated certificates.
- **System trust store:** If the application does not set a trust file, the FTL library loads the system trust store. The trust roots for the certificate used by the administrator must be installed in the system trust store. This method is only supported when the administrator configures user-defined certificates. (The FTL-generated trust file cannot be installed in the system trust store.)

For details, see [Enabling TLS for FTL Server](#)

When using TLS with user-defined certificates, the host name passed to the realm connect call must match the subject alternative name in FTL server's certificate, or the connection fails.

When using TLS, FTL application developers may specify the openssl security level. (The openssl library is used for connections to FTL server or other FTL clients.) In general, a higher security level requires stronger certificates and encryption.

For details, see the developer [API Documentation](#) in Web Help or in the FTL include directory.

After the realm connect call, additional secure transports used by the application will automatically enforce authentication and TLS requirements. The administrator must decide which transports to secure (for example, persistence service transports, eFTL service transports, group service transports, or peer-to-peer transports).

The administrator may optionally configure additional authorization checks (permissions) for persistence stores and eFTL channels.

For details, see [Authorization](#)

Ensuring FTL System Security: Tasks for Administrators

To ensure security among FTL applications, eFTL applications, FTL servers, and administrative tools, administrators complete the following tasks.

Procedure

1. Applications:

You must enable authentication and authorization at FTL server. If applications need to communicate directly with each other, then you must enable TLS using FTL-generated certificates. See [Enabling TLS for FTL Server](#). Also, all transports used for peer-to-peer communication must be marked as secure in the realm configuration. Use only these transport protocols.

- Secure Dynamic TCP
- Secure TCP

If applications communicate with services provided by FTL server (for example, persistence services), then you may use TLS with FTL-generated certificates or user-defined certificates. Then take additional steps:

- TLS may be enabled at FTL server (see step 3). Alternatively, administrators may provide TLS termination at an ingress point, or secure the network through other means.
- Authentication and authorization must be enabled for all relevant services (see step 4).

2. Authentication and Authorization:

- a. Configure authentication and authorization.
 - i. Your role includes configuring your enterprise authentication and authorization system (such as an LDAP service) with appropriate information to support TIBCO FTL components and application users.
 - ii. For details, see [Authentication](#).

3. Enabling TLS:

- a. When enabling TLS at FTL server, you must choose whether to use FTL-generated certificates or user-defined certificates. For details, see “Enabling TLS for FTL Server”.
- b. If providing TLS termination at an ingress point with a user-defined certificate, do not enable TLS at FTL server. Instead, instruct application developers to enable TLS in their applications, and provide the appropriate trust file.

4. TIBCO FTL Component Services:

- Secure all transport bridges. Verify that the transports interconnected by the bridges use only secure transport protocols.

For details, see [Securing Transport Bridges](#).

- Secure all persistence services. Configure the persistence clusters so that all relevant transports use only secure transport protocols.

For details, see [Securing Persistence Services](#).

- Secure all eFTL services.

TIBCO eFTL services must use secure transports to communicate with one another, and with eFTL applications. Your role includes these subtasks:

- Reconfigure the automatically-generated eFTL transport definitions so that all relevant transports use only secure transport protocols.
- Configure channels with appropriate authorization groups.
- Coordinate with application developers to ensure that eFTL clients connect to the eFTL services using the secure web sockets protocol (WSS).

5. Secure the FTL server data directories and files against unwanted access by other users.

Enabling TLS for FTL Server

FTL supports two different mechanisms for enabling TLS at FTL server:

- User-defined certificates, where administrators provide TLS certificates to FTL server (and possibly clients, for mTLS authentication). The administrator is responsible for

choosing a certificate authority and obtaining the certificates. FTL server loads its certificate on startup.

- FTL-generated certificates, where administrators run FTL tools to initialize TLS keys and trust roots for an FTL realm. At runtime, FTL server issues TLS certificates as needed.

FTL and eFTL components can use TLS 1.2 or TLS 1.3. TLS 1.3 is used wherever possible.

The openssl library is used for FTL transports between FTL components (on both client and server side). The openssl library is also used for the server side of any HTTPS and eFTL client connections accepted by FTL server.

Administrators may configure the openssl security level. In general, a higher security level requires stronger certificates and encryption. See [FTL Server Configuration Parameters](#)

Enabling TLS with User-defined Certificates

To enable TLS with user-defined certificates, specify the following parameters in the FTL server yaml configuration file:

- `tls.server.cert`
- `tls.server.private.key`
- `tls.server.private.key.password` (if needed)
- `tls.client.trust.file` (if needed)

For details, see [FTL Server Configuration Parameters](#)

On starting, FTL server loads the certificate and private key. The private key password is used to decrypt the private key. When making outgoing TLS connections to other FTL servers, FTL server verifies the remote server's certificate using the trust file, or the system trust store if no trust file is specified.

When clients and other FTL servers connect to an FTL server, they ensure that the hostname used to connect to FTL server matches the subject alternative name (SAN) in FTL server's certificate. The certificate must have a SAN. In the certificate's SAN, the leftmost component of a DNS name may be a wildcard.

i Note: The certificate must look valid to both clients and FTL servers. If clients and servers use different hostnames to connect to FTL server, the certificate's SAN must contain multiple entries and/or a wildcard entry.

If you are terminating TLS at an ingress point for client connections, do not configure FTL server with a TLS certificate. This means that FTL server does not use TLS for communication with clients or other FTL servers.

i Note: Secure peer-to-peer transports (used for direct communication between applications) are not permitted when user-defined certificates are configured. If applications must communicate directly (without FTL server in the middle), see [Enabling TLS for FTL Server](#)

See `samples/yaml/tls-user` for details on how to setup FTL Server with user-defined certificates.

Enabling TLS with FTL-Generated Certificates

Instead of taking a certificate from the user, FTL server can generate and manage TLS certificates. This is required in the following cases:

- Applications communicate directly with each other, using secure peer-to-peer transports.
- Applications communicate with each other via transport bridges, using secure transports

Before you begin

- Choose a keystore file password, and determine the appropriate level of security for that password.
 - Ensure that the clocks on all servers in a cluster are synchronized.
1. Remove any obsolete TLS data files from the FTL servers' data directories.
 2. Generate TLS data files.

To generate full-security files, enter:

```
tibftlserver --init-security file:<pw_file_name> -c <my_config_file_path>
-n <svr_name>
```

This command instructs the FTL server to generate new TLS data files, encrypting the new keystore file with the password.

(If the FTL server detects existing TLS files, it does not generate them anew. However, the FTL server does not decrypt or inspect existing files.)The server generates TLS files in the data directory (specified in the configuration file). If the data directory is unavailable, the server writes these files to the current directory. After writing the files, the FTL server exits.

3. Distribute the TLS files.The keystore file and trust file must be distributed to all FTL servers which include all core servers and auxiliary servers at all sites (including primary, satellite, and DR sites).Every server uses the same private key to identify itself. Every server uses the same trust file to verify the identity of FTL servers.
 - Supply copies of the keystore file and trust file to every FTL server. Place these files in the data directory of the servers.

i Note: Specify the data directory in the configuration file for each FTL server

- Supply a copy of the trust file to every client including application programs and browsers that access the FTL server GUI.

i Note: When a server generates new TLS data files, you must redistribute these files.

i Note: See the ftlstart script in the samples directory. The --secure option illustrates a basic way to start a secure FTL server.

Securing Transport Bridges

To secure a transport bridge, complete this task.

Before you begin

All FTL servers must enable authentication. All FTL servers must enable TLS using FTL-generated certificates.

Procedure

1. Verify that the transports interconnect by the bridge. Configure only secure network transport protocols. This ensures that authentication and TLS are enabled for all connections.

Use only these transport protocols:

- Secure Dynamic TCP
- Secure TCP

Securing Persistence Services

To secure a persistence service, complete this task.

Before you begin

All FTL servers must enable authentication. Administrators may optionally enable TLS and fine-grained permissions.

1. Verify that the persistence cluster definition specifies secure transport protocols. This ensures that authentication (and, optionally, TLS) is enabled for all connections. The client protocol, cluster protocol, disaster recovery (DR) protocol, and inter-cluster protocol must be secure.

Use only these transport protocols:

- Secure Dynamic TCP
 - Secure TCP
 - Secure Auto
2. For further details, see "Clusters Grid" in TIBCO FTL [Administration](#)
 3. For details, see [Authorization](#)

Securing eFTL Services

To secure an eFTL service, complete this task.

Before you begin

All FTL servers must enable authentication. Administrators may optionally enable TLS and fine-grained permissions.

If any channels use EMS servers or FTL persistence services, those services must also be secure.

Procedure

1. Verify secure transport protocols.

This ensures that authentication (and, optionally, TLS) is enabled for all FTL transport connections.

The cluster-facing transport and all the channel application-facing transports must be secure. Check their protocols in the transports grid

Use only these transport protocols:

- Secure Dynamic TCP
- Secure TCP
- Secure Auto

2. Include authenticated usernames.

Specify the parameter `publish.user` in the eFTL service section of the FTL server configuration file.

With this option, the eFTL service appends a field to messages published by eFTL client apps when it forwards them to FTL and EMS subscribers. That field contains the authenticated username of the eFTL publisher. FTL and EMS application code can use this username to authorize requests.

3. Enable authentication for eFTL client connections.
4. Optional. Enable fine-grained permissions for eFTL channels.
5. Optional. Enable TLS for FTL server. This will also enable TLS for eFTL client connections. For details, see [Enabling TLS for FTL Server](#)
6. [Enabling TLS for FTL Server](#)

Enabling TLS for FTL Server

7. For details about the content of that file, see [SSL Parameters for EMS Connections](#) in [TIBCO eFTL Administration](#).

Logging

FTL server has the ability to log events related to security. The `loglevel` should be specified in the `ftlserver.properties` section of the yaml configuration file. For the configuration reference, see "loglevel" in [FTL Server Configuration Parameters](#)

To log authentication results for incoming connections from clients or other FTL servers, set the `loglevel` to `auth:verbose`.

To log TLS configuration parameters and the establishment of TLS connections, set the `loglevel` to `tls:verbose`.

Logging components may be combined with a semicolon as delimiter.

For example, the `loglevel` can be set to `"auth:verbose;tls:verbose"`.

Elevated logging is not recommended for deployments with many short-lived, recurring connections. The best practice is to develop applications with long-lived connections to FTL server.

Authentication and Authorization

Authentication in FTL is enabled by specifying one or more authentication providers through the `auth.providers` field of the FTL server yaml configuration. For the configuration reference, see the [FTL Server Configuration Parameters](#) and For more details, see [Authentication](#)

When `auth.providers` is set, authentication is required for the following interfaces:

- Realm connections made through the FTL client API.
- The FTL server user interface.
- The FTL server REST API.

FTL offers optional features in addition to the above. To enable authentication for these features, do the following:

- Peer-to-peer FTL transports: Ensure that all transports used for direct communication between FTL clients are secure. Use transport protocol `Secure Dynamic TCP` or `Secure Static TCP`. For more information, see [Transports Grid](#)

i Note: TLS with FTL-generated certificates is required for secure peer-to-peer transports. For more information, see [Enabling TLS for FTL Server](#)

- Transport bridges: Follow the steps for securing peer-to-peer FTL transports.
- Persistence services: All transports used by the persistence cluster must use transport protocol `Secure Dynamic TCP`, `Secure Static TCP`, or `Secure Auto`. TLS is optional.
- Group service: All group service transports must use protocol `Secure Auto` or `Secure Dynamic TCP`.
- Eftl channels: Ensure that all transports used by the eftl cluster use protocol "Secure Auto" or "Secure Dynamic TCP". Ensure that authentication is enabled for the eftl cluster. See "**eFTL Clusters Grid**", in **eftl administration**].

In addition to successfully authenticating, users must belong to specific authorization groups (or roles) to access these interfaces. For more details, see [FTL Server Authorization Groups](#).

Administrators may optionally configure fine-grained permissions for the following features:

- Persistence services: See [Authorization](#)
- Eftl channels: See client Authentication and Authorization in eftl administration

Authentication

When authentication is enabled, FTL clients, eFTL clients, administrative tools, and other FTL servers must authenticate to the FTL server. They can authenticate to the FTL server in one of three ways:

- **Basic authentication:** The user provides a username and password.
- **mTLS authentication:** The user provides a TLS certificate and its corresponding private key. The FTL server verifies the client's certificate during the TLS handshake.



Note: mTLS authentication is not supported for eFTL clients or the eFTL REST API.

- **oauth2 authentication:** The user provides a signed JWT token, or the URL of an oauth2 server that can issue a signed JWT token and also provide credentials for accessing that server.

For more details, see [Authentication](#)

FTL server supports various authentication providers. Each authentication provider has its own configuration and is used for exactly one of the authentication modes above (basic, mTLS or oauth2). The purpose of the authentication provider is to determine if the client has authenticated successfully and if authenticated to determine the client's username and authorization roles. In the case of basic authentication, FTL server can try each basic authentication provider until one succeeds, or they all fail.

The following are the supported authentication providers. More than one provider can be configured. However, duplicate providers are not allowed. For example, it is illegal to configure multiple flat file authentication providers, but it is legal to configure a flat file provider and an ldap provider.

- **Flat File: Basic authentication only:** A built-in provider within FTL server that reads

a file of usernames and passwords. For details, see [Using the Built in Flat-File Authentication Service](#)

- **LDAP:Basic authentication only.** A built-in provider within FTL server that uses an LDAP server to authenticate clients. For details, see [Using the Built in LDAP Authentication Service](#)
- **HTTP/HTTPS (customizable):** Basic authentication only. FTL server makes HTTP/HTTPS calls to an external authentication service to authenticate clients. The authentication service may be completely customized by the administrator. For details, see [Using the external custom HTTP / HTTPS based authentication service](#)
- **mTLS: mTLS authentication only:** Enables verification of client certificates during a TLS handshake. For details, see [Using the Built in mTLS Based Authentication Service](#)
- **oauth2: oauth2 authentication only:** Enables verification of signed JWT tokens, plus enforcement of the token's expiration time under certain circumstances. For details, see [Using the built in OAuth 2.0 based authentication service](#)

Using the Built in Flat-File Authentication Service

The FTL flat-file authentication service provides authentication functionality for the FTL server, reading username and password data from a flat file. It runs inside the FTL server. Set **auth.providers to file:<file-path>** in order to use the built-in flat-file authentication service. For details see, [FTL Server Configuration Parameters](#)

Procedure

1. Configure the flat file with username and password data.

Passwords must be clear text, not obfuscated nor checksummed, but they can be hashed.

i Note: Syntax Summary for flat file

- Each line defines one user.
- Each line must specify a username and password, and may also specify optional authorization roles or groups.
- Delimit the username with a required colon.
- You may add optional space characters after the colon. The password begins with the first non-whitespace character after the colon.
- Delimit the password with a comma-space pair. If a line contains more than one comma-space pair, the *rightmost* pair delimits the password. Earlier pairs become part of the password, as do individual comma and space characters.
- Hashed passwords are allowed.
- Separate authorization roles or groups with a comma *only* (spaces are not valid).

For example:

```
admin: my_admin_pw, ftl,ftl-admin
ftl_svr: my_ftl_svr_pw, ftl-internal,ftl-admin,ftl,auth
app_user_1:my_pw, ftl
app_user_2:      her_pw, ftl
app_user_3:  my pw, more pw,, and still more pw , role-1,ftl
```

In the last example, the boldface type illustrates a complicated password containing spaces, commas, and even comma-space pairs.

2. Supply the location of the flat file through the configuration parameter `auth.providers`

For example:

```
auth.providers: file:/opt/tibco/ftl/samples/yaml/basic-auth/users.txt
```

See also: `samples/yaml/basic-auth/tibftlserver_basic_auth.yaml` in the FTL installation.

Using the Built in LDAP Authentication Service

When LDAP authentication is enabled, FTL server delegates authentication requests to the LDAP server.

To enable LDAP authentication, ensure that the "auth.providers" parameter in the FTL server configuration file contains an LDAP URL. The URL can be either `ldap://<host>:<port>`, or `ldaps://<host>:<port>`.

In addition, configure the `ldap.config` parameter in the FTL server yaml configuration file. This file contains configuration for connecting to the LDAP server.

For details, see [FTL Server Configuration Parameters](#).

For an example, see `samples/yaml/ldap` in the FTL installation directory.

Table here shows various ldap authentication service related parameters to be specified in the file specified in **ldap.config**

| LDAP Configuration name | Type | Examples | Description |
|-------------------------|--------|------------------|--|
| ldap.user.basedn | String | ou=People,dc=ftl | Based DN used to search for ldap users |
| ldap.user.scope | String | ldap_user_scope | The scope of the search. Valid values include: <ul style="list-style-type: none">• onelevel• subtree• object The default is to use a one level search. |
| ldap.user.class | String | ldap_user_class | Criteria used for user search. What class indicates a |

| | | | |
|---------------------|--------|------------------|--|
| | | | user, what attribute contains a user's unique identifier Unless ldap.group.filter is specified, default search filter is auto generated from relevant configuration parameters |
| ldap.user.attribute | String | uid | The attribute that is compared to the user name for the search. The default is uid. |
| ldap.user.filter | String | | <p>The filter used when searching for a user.</p> <p>If a more complex filter is needed, use this property to override the default.</p> <p>Any occurrence of {0} in the search string is the user attribute, and {1} is replaced with the user name.</p> <p>The default is {0}={1}</p> |
| ldap.group.basedn | String | ou=Groups,dc=ftl | The base path for |

| | | | |
|------------------------------------|--------|--------------------|--|
| | | | the LDAP static group search. If null or not set, static groups are not searched. |
| ldap.group.scope | String | subtree | <p>The scope of the static group search. Valid values include onelevel, subtree, and object.</p> <p>Default is to use a subtree search.</p> |
| ldap.group.filter | String | | Criteria used for static group search, similar to that used for user search Unless ldap.group.filter is specified, default search filter is autogenerated from relevant configuration parameters |
| ldap.group.static.class | String | groupofuniqueNames | |
| ldap.group.static.attribute | String | cn | The attribute of a static LDAP group that contains the group name. Default is cn. |
| ldap.group.static.member.attribute | String | uniqueMember | The attribute ID of a dynamic |

LDAP group object that specifies the name of members of the group. Default is uniqueMember.

Using the HTTP/HTTPS Authentication Service

You can write the http/https based authentication service similar to JAAS based Authentication service.

Using the External JAAS Authentication Service

The FTL JAAS authentication service is a containerized service that provides JAAS functionality for the FTL server. The service connects to your enterprise LDAP server.

If you use an authentication service that is external to the FTL server, you must start it *before* starting the FTL server.

These steps start the sample external JAAS authentication service in a Jetty container. To modify the sample service, see the file `<TIBCO_HOME>/ftl/<version>/samples/jaas/FTL-JAAS.readme.md`.

1. Configure the JAAS file.

The JAAS file specifies Java classes that implement authentication. (You can reuse the same JAAS file as you used in Release 5.x and earlier.)

For details, see *JAAS Login Configuration File* in Oracle Java documentation.

2. Deploy the service in a Jetty application container. The WAR file `FTL-JAAS.war` implements the service. Copy the example WAR file from `<TIBCO_HOME>/ftl/<version>/samples/jaas/FTL-JAAS.war` into Jetty's `demo-base/webapps`

directory.

3. Configure the JAAS realm within Jetty.

Edit `demo-base/etc/login.conf` and add a `tibftlserver` section. See the sample file `<TIBCO_HOME>/ftl/<version>/samples/jaas/ldap.jaas` for an example.

Note: The term "realm" denotes two separate concepts in JAAS and FTL.

4. Optional. Configure TLS within Jetty.

It is good practice to use TLS security for authentication service communications.

5. Start the authentication service in a Jetty container.

Clients can reach the authentication service at `<protocol>://<host>:<port>/FTL-JAAS/login`. (Supply this URL to the FTL server in the next step.)

6. Configure the FTL server to use the authentication service.

The FTL server is a client of the authentication service.

Supply the `yaml` parameter `auth.providers` to specify the URL of the JAAS authentication service. For example:

```
globals:
  # ...
  auth.url: <protocol>://<host>:<port>/FTL-JAAS/login
```

If the authentication service enables TLS, then its URL specifies the `https://` protocol, and you must supply these additional parameters to the FTL server:

```
auth.trust: <auth_service_public_cert_location>
auth.user: <ftl_server_user_name>
auth.password: <ftl_server_password>
```

JAAS Login Modules

TIBCO FTL software supports these sample login modules.

- `org.eclipse.jetty.jaas.spi.JDBCLoginModule`
- `org.eclipse.jetty.jaas.spi.PropertyFileLoginModule`
- `org.eclipse.jetty.jaas.spi.DataSourceLoginModule`

- `org.eclipse.jetty.jaas.ldap.LdapLoginModule`

For more information about login modules and how to configure them, see “JAAS Support - Eclipse” in *Jetty: The Definitive Reference*.

To implement your own login module, see “Custom Login Modules” in *Oracle Application Server Containers for J2EE Security Guide*.

Using the external custom HTTP / HTTPS based authentication service

You can customize the **http/https** based authentication service similar to JAAS based Authentication service.

- Custom authentication Service has to respond to `/auth` rest request, and return the roles associated with the user.
- See samples `/src//golang/advanced/src/tibco.com/ftl-sample/`
 - `tibffauthsvc`
 - `security`

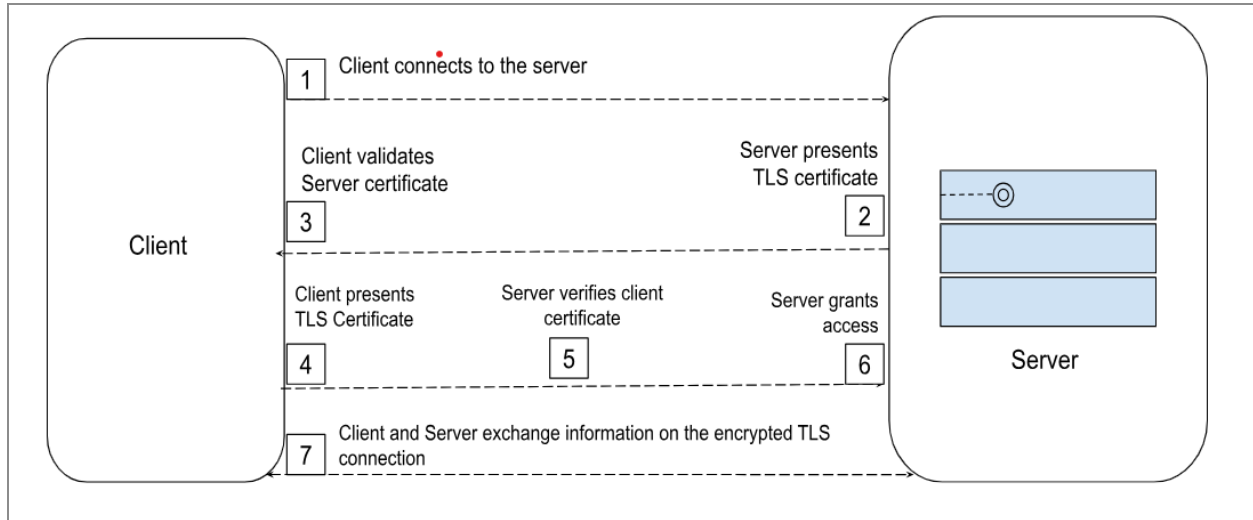
Using the Built in mTLS Based Authentication Service

mTLS (Mutual transport layer Security) allows client-to-server or server-to-server connections to authenticate each other during the TLS handshake. mTLS in FTL requires user-defined certificates.

For details, see: [Enabling TLS for FTL Server](#)

The picture below shows interactions between client and server during mTLS based authentication.

Figure 20: Built in mTLS based authentication



In order to enable mTLS for incoming connections, the 'auth.providers' in the FTLServer yaml configuration must include '**mtls**' as one of the providers. In addition, configure the "tls.server.trust.file parameter in the FTL server yaml configuration file.

For details, see [FTL Server Configuration Parameters](#)

Ensure that TLS with user-defined certificates is also configured, see [Enabling TLS for FTL Server](#)

When a client or another FTL server connects to FTL server and sends its client certificate, FTL server will verify the client certificate using `tls.server.trust.file`.

If the certificate is valid, FTL server will then parse the common name (CN) of the certificate. The CN must be a string in the following format:

```
<username>:<role1>[,<role2>,...]
```

For example, here are some valid common names:

```
admin:ftl-admin
internal:ftl-internal
subscribe-user:ftl,sub
```

The username and roles in the CN string become the username and roles for the incoming client or FTL server.

See `samples/yaml/mtls` in the FTL installation for example yaml configurations and certificates. See `samples/yaml/certs` in the FTL installation for an example of how to generate certificates for mTLS.

Some key points to consider while using mTLS

- When accessing the REST API, the REST client may authenticate itself with a client certificate
- mTLS is not supported for the UI, so another authentication provider must be configured to support UI users.
- mTLS is not supported for eFTL clients or the eFTL REST API.

Using the built in OAuth 2.0 based authentication service

FTL supports authenticating clients using OAuth 2.0 access tokens. When connecting to an FTL server configured with OAuth 2.0 authentication, an FTL client must authenticate itself to the server by presenting an access token issued by an OAuth 2.0 authorization server.

This access token must be a signed JSON Web Token (JWT) that includes claims that the FTLServer can extract to validate that the client has the right FTL role. See <https://datatracker.ietf.org/doc/html/rfc7519> for more information on what constitutes a signed JWT token.

The FTL server validates the access token's signature and claims and accepts or rejects the connection request accordingly. If this authentication process is successful, the FTL client is allowed to connect and access grants based on the FTL role specified in the access token.

In order for an OAuth 2.0 authorization server to issue an access token with the expected claims, the relevant FTL user and group information must be made available to it. Depending on your OAuth 2.0 provider, there may be a number of options available for achieving this. For example, you may be able to define FTL users and groups directly in your provider, or you may be able to integrate your provider with an external authentication service such as LDAP. Refer to your OAuth 2.0 provider's documentation for instruction.

Obtaining an Access Token

FTL and eFTL clients require an access token to connect to an FTL server configured with OAuth 2.0 authentication. Additionally, an FTL server itself may require an access token to connect to another FTL server or a TIBCO messaging product that is configured for OAuth 2.0 authentication.



Note: FTL only supports access tokens in the form of signed JWTs with asymmetric validation keys. Unsigned JWTs, or JWTs with symmetric validation keys are not supported.

The FTL client APIs, eFTL Client APIs and the FTL server can be configured to use either the Client Credentials grant or Resource Owner Password Credentials grant for obtaining OAuth 2.0 access tokens.

FTL and eFTL clients can set up client side properties that then allow FTL/eFTL client library to retrieve OAuth 2.0 access tokens from the specified OAuth 2.0 provider. Clients may also provide the access tokens that are externally obtained.. These OAuth 2.0 access tokens are then validated by the FTLServer.

Client Credentials Grant

In the Client Credentials grant, the FTL or eFTL client (or FTL server) presents a client ID and client secret to the OAuth 2.0 authorization server. The authorization server uses these credentials to authenticate the FTL client or eFTL client (or FTL server) and issues it an access token.

This is the preferred grant type for both the FTL client and FTL server.

Resource Owner Password Credentials Grant

In the resource owner password credentials grant, the FTL client (or eFTL Client, or EMS server) first authenticates itself to the OAuth 2.0 authorization server by presenting a client ID and client secret, then provides an FTL user and password for validation by the authorization server. If both authentication operations are successful, the authorization server issues an access token to the client.

Refresh tokens are supported with this grant type. If the authorization server issues a refresh token along with the requested access token, the FTL client (or eFTL Client or FTL

server) uses the refresh token instead of the grant for requesting the next access token. If it fails to obtain a new access token using the refresh token, it try's again using the grant.

This grant type has been deemed legacy and is expected to be removed in a future update to the OAuth 2.0 framework. It should only be used in situations where the client credentials grant type is not feasible.

Using Externally-Obtained Access Tokens

In addition to the above grant types, the FTL and eFTL client APIs and the FTL server can be configured to use access tokens obtained via external means. If access tokens are directly configured in this manner, the FTL server and client APIs does not attempt to obtain access tokens using the OAuth 2.0 grants.

i Note: The FTL client APIs additionally support user-defined callbacks for obtaining access tokens. This method of obtaining access tokens requires the use of new APIs. Existing FTL client applications need to be modified to use the new APIs in order to make use of this method. Refer to the corresponding client API documentation for details.

User-defined callbacks are only available in the client APIs. This method of obtaining access tokens is not supported in the FTL server.

Access Token Expiration

OAuth 2.0 JWT access tokens can have an expiration time specified through the 'exp' claim. The FTL server enforces access token expiration by disconnecting the associated FTL client (or eFTL client or FTL server).

i Note: FTL server cannot enforce access token expirations in the following scenarios:

- Authentication is enabled, but TLS is not enabled, and the FTL-generated keystore is present.
- TLS is enabled with FTL-generated certificates.

In both scenarios, the access token is verified when the client first connects, but FTL server does not disconnect the client when the token expires. It is possible to migrate away from

both of the above configurations. See [Eliminating the FTL Generated Keystore or Eliminating FTL generated certificates](#). Once the relevant procedure is complete, access token expirations can be enforced by FTL server.

FTLServer OAuth 2.0 Configuration

To enable OAuth 2.0 authentication, the `auth.providers` parameter in the FTL server yaml configuration file must contain `oauth2` as one of the providers.

In addition, set the following parameters in the FTL server yaml configuration file:

For details, see [FTL Server Configuration Parameters](#)

For an example, see `samples/yaml/oauth2` in the FTL installation directory.

Single Sign-On with OAuth 2.0

If OAuth 2.0 authentication is enabled, FTL server can redirect the administrative UI to the user's oauth server for authentication. The browser will take the user through the oauth authorization code flow.

To enable single sign-on, set the following parameters in the FTL server yaml configuration file:

- `oauth2.ui.endpoint.auth`
- `oauth2.ui.endpoint.token`
- `oauth2.ui.client.id`
- `oauth2.ui.client.secret`
- `oauth2.ui.endpoint.logout`

For details, see [FTL Server Configuration Parameters](#)

Authorization

FTL Server Authorization Groups

A username may belong to several authorization groups (also known as *roles*). The following table specifies authorization group requirements.

Authorization Groups

| Authorization Group | Usage |
|---------------------|--|
| ftl | FTL servers require client programs to authenticate with usernames in the authorization group <code>ftl</code> . |
| ftl-admin | Authenticated users in the authorization group <code>ftl-admin</code> can execute administrative operations, modify the realm definition, and view monitoring pages. |
| ftl-internal | FTL servers require affiliated FTL servers to authenticate with a username in the authorization group <code>ftl-internal</code> . |



Note: As of Release 6.0, the authorization groups `ftl-primary`, `ftl-satellite`, `ftl-backup`, `ftl-dr` are obsolete. For each of these, use `ftl-internal` instead.

Mapping Authorization Groups

FTL allows administrators to map their own roles to FTL roles. This can be done via the `auth.rolemap` file that can contain a mapping specific to the authentication provider.

Rules on role mapping file

- Role mapping within 'ldap' section applies to ldap or ldaps
e.g in the example below
 - [ldap]:

- FTL-Admin: ftl-admin
- Only one mapping per authentication provider is allowed, for example you cannot have two separate authentication sections for ldap/ldaps or mtls or oauth2

Note: Syntax Summary for the mapping file

- A role mapping will have sub sections per authentication type
- Each subsection starts with a square bracket [, **followed by the authentication provider name and a closing square bracket]**.
 - Allowed authentication provider names
 - ldap or ldaps
 - mtls
 - oauth2
- A line with a header with this syntax [**<authentication provider>**] defines the subsection of the authentication type and subsequent lines until the next [**<authentication provider>**] section would be role mapping associated with that authentication provider.
- There can be only one subsection per authentication provider.
- A role mapping within the authentication provider section defines the mapping from the user defined role to FTL built in role that's pertinent to the authentication provide
- Delimit the user-defined role with a required colon. (e.g FTL-Admin: ftl-admin)
- You may add optional space characters after the colon. The FTL built-in role begins with the first non-whitespace character after the colon.
- Delimit the FTL role with a comma to add additional FTL built -in roles that map to the same user defined role to FTL roles.
- Separate authorization roles or groups with a comma only (spaces are not valid).

Here is an example of role mapping file named rolemap.txt

```
[ldap]
FTL-Admin: ftl-admin
[mtls]
group1: ftl-admin, ftl-internal
```

```
group2: ftl
[oauth2]
oauth2-admin: ftl-admin
oauth2-apps: ftl
```

Permissions

Administrators can secure the messaging infrastructure by setting permissions at the cluster and store level to grant access to objects for a given user/role.

Administrators enable permissions by setting the realm property `enable_permissions` to `true`.

The requirements to use permissions follow:

- All persistence transports must be marked secure.
- Clients must be using FTL 6.8.0 or later.
- Authentication must be enabled (via the `auth.providers` parameter in the `ftlserver` configuration file).
- Once authentication is enabled:
 - Assign permissions via the Users Grid and Roles Grid. See [Configuring Permissions](#).
 - On the Realm Properties Details Panel, **Permissions** is set as **Disabled** by default and must be set as **Enabled**. See [Realm Properties Details Panel](#).
- When `enable_permissions` is `true`, endpoint store inboxes are always enabled. See [Endpoint Store Inboxes](#).

For details on the preferred method to upgrade to FTL 6.8.0 , see [Migrating to FTL 6.8.0 when Using Permissions](#) .

Enable Permissions

To enable permissions, set the `enable_permissions` flag in the realm properties to `true`.

When the `enable_permissions` flag is set to `true`, permissions are enforced at every persistence cluster defined in the realm. By default, a user has no permissions; permissions must be explicitly granted to users or roles as described in the next section, [Assign Permissions](#). There is one exception: users with the predefined `ftl-internal` role always

have all permissions. For this reason the `ftl-internal` role should not be assigned to ordinary users.

Endpoint store inboxes are enabled when either of the following are true. (See [Endpoint Store Inboxes](#).)

- `use_endpoint_store_for_inbox` is true
- `enable_permissions` is true

Setting `enable_permissions` to true does not change the value of `use_endpoint_store_for_inbox`, but endpoint store inboxes are enabled. If `enable_permissions` is set back to false, then endpoint store inbox behavior depends on the existing value of `use_endpoint_store_for_inbox`. The value is not changed.

When using permissions with the request-reply API, the requestor has an implicit inbox subscriber on the reply endpoint's store. (By default the reply endpoint is the same as the publisher's endpoint.) Because endpoint store inboxes are automatically enabled, this subscription is created in the reply endpoint's store. Therefore:

- Requests (using `tibPublisher_Request`) require publish permissions on the endpoint's store and subscribe permissions on the reply endpoint's store (which could be a different endpoint)
- Replies (using `tibPublisher_Reply`) require 'publish' permissions

Assign Permissions

The administrator assigns permissions at the store or persistence cluster level.

Each store and persistence cluster in the realm configuration has a permission map (the `acl` field) that associates permissions with usernames and/or roles.

In the permissions map, a username or role can be associated with a list of string values representing permissions. Any FTL client with the specified username or role will have the associated permissions when interacting with the given persistence store or persistence cluster. If an FTL client matches multiple entries in the permissions map (a username and/or any number of roles), the FTL client has the union of all specified permissions.

Persistence cluster and persistence store permissions do not overlap. Persistence stores do not inherit permissions defined at the persistence cluster level.

The following permissions may be defined at a persistence cluster:

- `lock`

The following permissions may be defined at a persistence store:

- publish
- subscribe
- map

See [Required Permissions for API Calls](#).

Adding permissions to a username or role does not require a restart of any FTL components.

Revoke Permissions

Permissions associated with a username or role at any persistence cluster or persistence store may be removed at any time. Removing permissions does not require a restart of any FTL components.

Permission revocation is not enforced synchronously. At some later time FTL will enforce the new restrictions.

See [Required Permissions for API Calls](#).

Even when the new restrictions are applied, a subscriber whose subscribe permission was revoked may receive up to prefetch number of messages.

Monitoring Gateway Service

[TIBCO® Messaging Monitor for TIBCO FTL®](#) runs as an ordinary FTL client. The subscribe permission needs to be granted to the appropriate user and/or role on the built-in monitoring and logging stores (`ftl.system.mon.store` and `ftl.system.log.store`). The same is true of any application that subscribes on the monitoring or logging endpoints. It is not possible to grant the publish or map permissions on the monitoring and logging stores.

Configuring Permissions

Permissions for users and roles can be configured from the user interface on the Users grid and Roles grid.

In edit mode, you can add and delete users and roles.

For information on enabling, assigning, and revoking permissions as well as monitoring permissions, see [Authorization](#).

For the consequences of modifying the permission configurations, see [Required Permissions for API Calls](#).

Users Grid

| Column | Description |
|---------------------|---|
| User | The FTL client's username. |
| Resource | The identifier of the defined resource. Example: Resource = ftl.nonpersistent.store |
| Resource Type | The type of resource including persistence stores and eFTL channels. Example: Resource Type = Persistence Store |
| Resource Belongs To | The name of the object. The resource belongs to a persistence cluster or eFTL cluster, so in that sense it belongs to a service that is servicing this cluster. Example: Resource Belongs To = ftl.default.cluster |
| Permissions | <p>The permissions granted.</p> <p>Requirements:</p> <ul style="list-style-type: none">• The FTL server must have TLS and authentication enabled. See Authorization.• The Realm Properties Details Panel, Permissions must be set as Enabled. See Realm Properties Details Panel. <p>At the persistence cluster level, the value is <code>lock</code>.</p> <p>At the persistence store level, the value can be:</p> <ul style="list-style-type: none">• <code>publish</code>• <code>subscribe</code>• <code>map</code> |

Roles Grid

| Column | Description |
|---------------------|---|
| Roles | The FTL client's role. |
| Resource | The identifier of the defined resource. Example: Resource = ftl.nonpersistent.store |
| Resource Type | The type of resource including persistence stores and eFTL channels. Example: Resource Type = Persistence Store |
| Resource Belongs To | The name of the object. Example: Resource Belongs To = ftl.default.cluster |
| Permissions | The permissions granted. At the persistence cluster level, the value is lock. At the persistence store level, the value can be: <ul style="list-style-type: none">• publish• subscribe• map |

Required Permissions for API Calls

Permissions are granted to the FTL client's user name and/or roles. To set user and role permissions from the FTL user interface, see [Configuring Permissions](#).

When an application interacts with a persistence store or cluster, FTL determines if that interaction is allowed, based on the permissions granted to the FTL client's username and/or roles. Mostly commonly this interaction results from an API call, but all interactions are regulated regardless of the source.

Interactions involving locks are regulated by permissions set at the persistence cluster level.

Interactions involving publishers, subscribers, maps, and stored data are regulated by permissions set at the persistence store level.

FTL server logs all authorization failures at the loglevel `acl:verbose`.

Where possible, if an FTL client is not authorized to take a certain action, the API call will fail immediately regardless of any retry duration.

The following table shows the required permissions for various API calls.



Note: Only the C APIs are listed, but the same holds true for all client APIs.

Permissions for API Calls

| Operation | FTL API | FTL Permission |
|----------------|--|----------------|
| acquire lock | Implicit via map calls <code>withLock</code> | lock |
| return lock | <code>tibLock_Return</code> <code>tibLock_Destroy</code> | lock |
| close map | <code>tibMap_Close</code> | map |
| create map | <code>tibRealm_CreateMap</code> | map |
| delete map | <code>tibRealm_RemoveMap</code> | map |
| map get | <code>tibMap_Get</code> <code>tibMap_GetMultiple</code> | map |
| map get size | <code>tibMap_GetSize</code> | map |
| map iterate | <code>tibMap_CreateIterator</code> <code>tibMapIterator_Next</code> | map |
| map remove | <code>tibMap_Remove</code> | map |
| map remove all | <code>tibMap_RemoveAll</code> | map |

| Operation | FTL API | FTL Permission |
|-------------------------|--|-----------------------|
| map set | tibMap_Set tibMap_SetMultiple | map |
| close publisher | tibPublisher_Close | publish |
| create publisher | tibPublisher_Create | publish |
| publish | tibPublisher_Send tibPublisher_SendToInbox tibPublisher_SendMessages | publish |
| send reply | tibPublisher_SendReply | publish |
| send request | tibPublisher_SendRequest Note: Sending the request requires the publish permission on the endpoint's store. Receiving the reply requires the subscribe permission on the reply endpoint's store (which could be a different store) | publish AND subscribe |
| acknowledge | tibEventQueue_Dispatch (auto) tibMessage_Acknowledge (explicit) tibSubscriber_AcknowledgeMessages (explicit) | subscribe |
| close subscriber | tibSubscriber_Close | subscribe |
| durable create | Implicit via tibSubscriber_Create | subscribe |
| dynamic durable destroy | tibRealm_Unsubscribe tibRealm_UnsubscribeEx | subscribe |

| Operation | FTL API | FTL Permission |
|------------------------|---|----------------|
| start subscriber | <code>tibEventQueue_AddSubscriber</code> | subscribe |
| stop subscriber | <code>tibEventQueue_RemoveSubscriber</code> | subscribe |
| subscribe | <code>tibSubscriber_Create</code> <code>tibSubscriber_CreateOnInbox</code> | subscribe |
| rewind | <code>tibRealm_RewindSubscription()</code> | subscribe |
| create browser | <code>tibBrowser_Create()</code> | subscribe |
| browse message | <code>tibBrowser_Next()</code> | subscribe |
| delete browsed message | <code>tibBrowser_DeleteMessage()</code> | subscribe |
| close browser | <code>tibBrowser_Close()</code> | subscribe |

Migrating to FTL 6.8.0 when Using Permissions

If you are migrating from a version of FTL prior to 6.8.0 and want to set up permissions, complete the steps in the following order to minimize the number of application restarts.

1. Ensure that TLS and authentication are enabled.
2. Ensure that all persistence transports are secure. Changing this will require a restart of persistence services and clients.
3. Upgrade the servers to 6.8.0.
4. Upgrade the clients to 6.8.0.
5. Set `enable_permissions` to true and configure appropriate permissions for users and/or roles. This will require a restart of any publisher or subscriber using server-based inboxes (via a dedicated inbox store, `ftl.system.inbox.store` or `ftl.routing.inbox.store`) because endpoint store inboxes are now required. See

Endpoint Store Inboxes.

6. Verify that stores are using the permissions you have configured. If no permissions are set up, users will not have access to publish/subscribe to a store. The loglevel `acl:verbose` may be set at the FTL server (specifically, the persistence service) to identify authorization failures.

FTL Prometheus Metric Naming

FTL supports integration with Prometheus for application metrics monitoring. Prometheus is a monitoring tool that helps in analyzing the application metrics for flows and activities.

Prometheus servers scrape data from the HTTP /metrics endpoint of the FTL server.

Prometheus integrates with Grafana, which provides better visual analytics.

! **Important:** The Prometheus standards for [metric and label naming](#) are used to guide the renaming of the metrics.

Configuring Prometheus to monitor

| Type | FTL metric name | Prometheus name |
|------|--------------------|------------------------------------|
| 1 | messages_sent | tibco_ftl_sent_messages |
| 2 | messages_received | tibco_ftl_received_messages |
| 11 | bytes_sent | tibco_ftl_sent_bytes |
| 12 | bytes_received | tibco_ftl_received_bytes |
| 31 | data_lost | tibco_ftl_dataloss_events |
| 32 | format_unavailable | tibco_ftl_unavailable_formats |
| 41 | queue_backlog | tibco_ftl_queue_backlog_messages |
| 42 | queue_discards | tibco_ftl_queue_discarded_messages |

| Type | FTL metric name | Prometheus name |
|------|------------------------|-----------------------------------|
| 51 | dynamic_formats | tibco_ftl_dynamic_formats |
| 61 | packets_sent | tibco_ftl_sent_packets |
| 62 | packets_received | tibco_ftl_received_packets |
| 63 | packets_retransmitted | tibco_ftl_retransmitted_packets |
| 64 | packets_missed | tibco_ftl_missed_packets |
| 65 | packets_lost_outbound | tibco_ftl_discarded_packets |
| 66 | packets_lost_inbound | tibco_ftl_lost_packets |
| 67 | store_mismatch_message | tibco_ftl_store_mismatch_messages |
| 70 | process_rss_kb | tibco_ftl_process_rss_bytes |
| 71 | process_peak_rss_kb | tibco_ftl_process_peak_rss_bytes |
| 72 | process_vm_kb | tibco_ftl_process_vm_bytes |
| 73 | user_cpu_time | tibco_ftl_cpu_time_user_seconds |
| 74 | system_cpu_time | tibco_ftl_cpu_time_system_seconds |
| 75 | total_cpu_time | tibco_ftl_cpu_time_seconds |

| Type | FTL metric name | Prometheus name |
|------|----------------------------|---|
| 500 | queue_latency_msg_n | tibco_ftl_queue_latency_msg_samples |
| 501 | queue_latency_msg_max | tibco_ftl_queue_latency_msg_max_seconds |
| 502 | queue_latency_msg_mean | tibco_ftl_queue_latency_msg_mean_seconds |
| 503 | queue_latency_msg_stddev | tibco_ftl_queue_latency_msg_stddev_seconds |
| 504 | queue_latency_timer_n | tibco_ftl_queue_latency_timer_samples |
| 505 | queue_latency_timer_max | tibco_ftl_queue_latency_timer_max_seconds |
| 506 | queue_latency_timer_mean | tibco_ftl_queue_latency_timer_mean_seconds |
| 507 | queue_latency_timer_stddev | tibco_ftl_queue_latency_timer_stddev_seconds |
| 508 | reqreply_latency_n | tibco_ftl_reqreply_latency_samples |
| 509 | reqreply_latency_max | tibco_ftl_reqreply_latency_max_seconds |
| 510 | reqreply_latency_mean | tibco_ftl_reqreply_latency_mean_seconds |
| 511 | reqreply_latency_stddev | tibco_ftl_reqreply_latency_timer_stddev_seconds |

| Type | FTL metric name | Prometheus name |
|------|----------------------------|--|
| 1000 | message_count | tibco_ftl_store_messages |
| 1001 | message_size | tibco_ftl_store_message_bytes |
| 1002 | records_synced | tibco_ftl_store_synced_records |
| 1003 | records_avail_ for_sync | tibco_ftl_store_available_for_sync_records |
| 1004 | records_ caught_up | tibco_ftl_store_caught_up_records |
| 1005 | records_to_ catch_up | tibco_ftl_store_to_catch_up_records |
| 1006 | swap_message_ count | tibco_ftl_store_swap_messages |
| 1007 | swap_message_ size | tibco_ftl_store_swap_message_bytes |
| 1008 | store_byte_limit | tibco_ftl_store_limit_bytes |
| 1009 | store_message_ limit | tibco_ftl_store_limit_messages |
| 1010 | expiration_ count | tibco_ftl_store_expiration_events |
| 1100 | persistence_ server | Stored as label PersistenceServer |
| 1101 | persistence_ cluster | Stored as label Cluster |
| 2000 | durable_count | tibco_ftl_store_durables |

| Type | FTL metric name | Prometheus name |
|------|---------------------------------------|--|
| 3000 | quorum_state | tibco_ftl_quorum_state_info |
| 3001 | quorum_number | tibco_ftl_store_quorum_number_info {state="current"} |
| 3002 | history_quorum_number | tibco_ftl_store_quorum_number_info {state="history"} |
| 3003 | history_update_version | tibco_ftl_store_version_info{state="history_update"} |
| 3004 | history_timestamp | tibco_ftl_store_timestamp{state="history"} |
| 3005 | pending_updates | tibco_ftl_store_updates{state="pending"} |
| 3006 | committed_updates | tibco_ftl_store_updates {state="committed"} |
| 3007 | commit_latency | tibco_ftl_store_commit_latency_seconds |
| 3008 | history_update_version_non_replicated | tibco_ftl_store_version_info{state="history-non-replicated"} |
| 3009 | disk_state | tibco_ftl_store_disk_state |
| 3010 | consistent_quorum_number | tibco_ftl_store_quorum_number_info {state="consistent"} |
| 3011 | consistent_version | tibco_ftl_store_version_info {state="consistent"} |

| Type | FTL metric name | Prometheus name |
|------|----------------------|---|
| 3012 | consistent_timestamp | tibco_ftl_store_timestamp {state="consistent"} |
| 3013 | disk_size | tibco_ftl_pserver_disk_allocated_bytes |
| 3014 | backlog_limit | tibco_ftl_pserver_backlog_memory_limit_bytes |
| 3015 | backlog_size | tibco_ftl_pserver_backlog_memory_bytes |
| 3016 | disk_backlog_size | tibco_ftl_pserver_disk_backlog_memory_bytes |
| 3017 | inbound_byte_count | tibco_ftl_store_received_bytes |
| 3018 | outbound_byte_count | tibco_ftl_store_sent_bytes |
| 3019 | disk_inuse_size | tibco_ftl_pserver_disk_used_bytes |
| 3020 | disk_capacity | tibco_ftl_pserver_disk_capacity_bytes |
| 3021 | disk_usage_limit | tibco_ftl_pserver_disk_usage_limit_bytes |
| 3022 | disk_available | tibco_ftl_pserver_disk_available_bytes |
| 3023 | disk_used | tibco_ftl_pserver_disk_usage_bytes |
| 3024 | disk_size_swap | tibco_ftl_pserver_disk_swap_bytes |
| 3025 | disk_inuse_size_swap | tibco_ftl_pserver_disk_inuse_swap_bytes |
| 3025 | disk_capacity_swap | tibco_ftl_pserver_disk_capacity_swap_ |

| Type | FTL metric name | Prometheus name |
|------|-----------------------|---|
| | swap | bytes |
| 3027 | disk_usage_limit_swap | tibco_ftl_pserver_disk_usage_limit_swap_bytes |
| 3028 | disk_available_swap | tibco_ftl_pserver_disk_available_swap_bytes |
| 3029 | disk_used_swap | tibco_ftl_pserver_disk_used_swap_bytes |
| 3030 | disk_write_count | tibco_ftl_pserver_disk_write_count |
| 3031 | disk_write_bytes | tibco_ftl_pserver_disk_write_bytes |
| 3032 | disk_read_count | disk_read_count tibco_ftl_pserver_disk_read_count |
| 3033 | disk_read_bytes | tibco_ftl_pserver_disk_read_bytes |
| 3034 | disk_fdatasync_count | tibco_ftl_pserver_disk_fdatasync_count |
| 3035 | replica_rtt_n | tibco_ftl_pserver_replica_rtt_samples |
| 3036 | replica_rtt_max | tibco_ftl_pserver_replica_rtt_max |
| 3037 | replica_rtt_total | tibco_ftl_pserver_replica_rtt_total |
| 3038 | replica_rtt_bytes_n | tibco_ftl_pserver_replica_rtt_bytes_samples |
| 3039 | replica_rtt_bytes_max | tibco_ftl_pserver_replica_rtt_bytes_max |

| Type | FTL metric name | Prometheus name |
|------|-------------------------------|---|
| 3040 | replica_rtt_bytes_total | tibco_ftl_pserver_replica_rtt_bytes_total |
| 3041 | replica_rtt_async_n | tibco_ftl_pserver_replica_rtt_async_samples |
| 3042 | replica_rtt_async_max | tibco_ftl_pserver_replica_rtt_async_max |
| 3043 | replica_rtt_async_total | tibco_ftl_pserver_replica_rtt_async_total |
| 3044 | replica_rtt_bytes_async_n | tibco_ftl_pserver_replica_rtt_bytes_async_samples |
| 3045 | replica_rtt_bytes_async_max | tibco_ftl_pserver_replica_rtt_bytes_async_max |
| 3046 | replica_rtt_bytes_async_total | tibco_ftl_pserver_replica_rtt_bytes_async_total |
| 3047 | disk_write_latency_n | tibco_ftl_pserver_disk_write_latency_samples |
| 3048 | disk_write_latency_max | tibco_ftl_pserver_disk_write_latency_max |
| 3049 | disk_write_latency_total | tibco_ftl_pserver_disk_write_latency_total |
| 3050 | disk_write_batch_bytes_n | tibco_ftl_pserver_disk_write_batch_bytes_samples |
| 3051 | disk_write_batch_bytes_max | tibco_ftl_pserver_disk_write_batch_bytes_max |

| Type | FTL metric name | Prometheus name |
|------|--------------------------------|--|
| | batch_bytes_max | max |
| 3052 | disk_write_batch_bytes_total | tibco_ftl_pserver_disk_write_batch_bytes_total |
| 4000 | connection_count | tibco_ftl_eftl_connections{state="current"} |
| 4001 | subscription_count | tibco_ftl_eftl_subscriptions |
| 4002 | inbound_eftl_message_count | tibco_ftl_eftl_received_messages |
| 4003 | outbound_eftl_message_count | tibco_ftl_eftl_sent_messages |
| 4004 | inbound_cluster_message_count | tibco_ftl_received_cluster_messages |
| 4005 | outbound_cluster_message_count | tibco_ftl_sent_cluster_messages |
| 4006 | inbound_ftl_message_count | tibco_ftl_received_ftl_messages |
| 4007 | outbound_ftl_message_count | tibco_ftl_sent_ftl_messages |
| 4008 | protocol_message_count | tibco_ftl_protocol_messages |

| Type | FTL metric name | Prometheus name |
|------|-----------------------------|---|
| 4009 | filtered_message_count | tibco_ftl_filtered_messages |
| 4010 | discarded_message_count | tibco_ftl_eftl_discarded_messages |
| 4011 | dataloss_message_count | tibco_ftl_dataloss_messages |
| 4012 | pending_connection_count | tibco_ftl_eftl_connections{state="pending"} |
| 4013 | suspended_connection_count | tibco_ftl_eftl_connections{state="suspended"} |
| 4014 | channel_enabled | tibco_ftl_eftl_channel_enabled |
| 4015 | max_connection_count | tibco_ftl_eftl_connections{state="max"} |
| 4016 | session_count | tibco_ftl_eftl_sessions{state="current"} |
| 4017 | max_session_count | tibco_ftl_eftl_sessions{state="max"} |
| 4018 | cumulative_connection_count | tibco_ftl_eftl_successful_connections |
| 5000 | client_count | tibco_ftl_clients_total |
| 5001 | satellite_count | tibco_ftl_satellites_total |

| Type | FTL metric name | Prometheus name |
|------|-----------------------------|--|
| 5002 | client_running_count | tibco_ftl_clients{state="running"} |
| 5003 | client_needs_restart_count | tibco_ftl_clients{state="needs_restart"} |
| 5004 | client_timed_out_count | tibco_ftl_clients{state="timed_out"} |
| 5005 | client_exception_count | tibco_ftl_clients{state="exception"} |
| 5006 | client_out_of_sync_count | tibco_ftl_clients{state="out_of_sync"} |
| 5007 | client_off_line_count | tibco_ftl_clients{state="offline"} |
| 5008 | client_destroyed_count | tibco_ftl_clients{state="destroyed"} |
| 5009 | client_connect_count | tibco_ftl_client_connects |
| 5010 | client_reconnect_count | tibco_ftl_client_reconnects |
| 5011 | send_to_inbox_failure_count | tibco_ftl_send_to_inbox_failures |
| 5012 | satellite_running_count | tibco_ftl_satellites{state="running"} |

| Type | FTL metric name | Prometheus name |
|------|----------------------------------|---|
| 5013 | satellite_needs_restart_count | tibco_ftl_satellites{state="needs_restart"} |
| 5014 | satellite_timed_out_count | tibco_ftl_satellites{state="timed_out"} |
| 5015 | satellite_exception_count | tibco_ftl_satellites{state="exception"} |
| 5016 | satellite_out_of_sync_count | tibco_ftl_satellites{state="out_of_sync"} |
| 5017 | satellite_off_line_count | tibco_ftl_satellites{state="offline"} |
| 5018 | satellite_destroyed_count | tibco_ftl_satellites{state="destroyed"} |
| 5019 | group_server_count | tibco_ftl_groupservers_total |
| 5020 | group_server_running_count | tibco_ftl_groupservers{state="running"} |
| 5021 | group_server_needs_restart_count | tibco_ftl_groupservers{state="needs_restart"} |
| 5022 | group_server_timed_out_count | tibco_ftl_groupservers{state="timed_out"} |
| 5023 | group_server_exception_count | tibco_ftl_groupservers{state="exception"} |

| Type | FTL metric name | Prometheus name |
|------|----------------------------------|---|
| | count | |
| 5024 | group_server_out_of_sync_count | tibco_ftl_groupservers{state="out_of_sync"} |
| 5025 | group_server_off_line_count | tibco_ftl_groupservers{state="offline"} |
| 5026 | group_server_destroyed_count | tibco_ftl_groupservers{state="destroyed"} |
| 5027 | group_client_count | tibco_ftl_groupclients_total |
| 5028 | group_client_running_count | tibco_ftl_groupclients{state="running"} |
| 5029 | group_client_needs_restart_count | tibco_ftl_groupclients{state="needs_restart"} |
| 5030 | group_client_timed_out_count | tibco_ftl_groupclients{state="timed_out"} |
| 5031 | group_client_exception_count | tibco_ftl_groupclients{state="exception"} |
| 5032 | group_client_out_of_sync_count | tibco_ftl_groupclients{state="out_of_sync"} |
| 5033 | group_client_ | tibco_ftl_groupclients{state="offline"} |

| Type | FTL metric name | Prometheus name |
|------|----------------------------------|---|
| | off_line_count | |
| 5034 | group_client_destroyed_count | tibco_ftl_groupclients{state="destroyed"} |
| 5035 | bridge_count | tibco_ftl_bridges_total |
| 5036 | bridge_running_count | tibco_ftl_bridges{state="running"} |
| 5037 | bridge_needs_restart_count | tibco_ftl_bridges{state="needs_restart"} |
| 5038 | bridge_timed_out_count | tibco_ftl_bridges{state="timed_out"} |
| 5039 | bridge_exception_count | tibco_ftl_bridges{state="exception"} |
| 5040 | bridge_out_of_sync_count | tibco_ftl_bridges{state="out_of_sync"} |
| 5041 | bridge_off_line_count | tibco_ftl_bridges{state="offline"} |
| 5042 | bridge_destroyed_count | tibco_ftl_bridges{state="destroyed"} |
| 5043 | persistence_server_count | tibco_ftl_persistenceservers_total |
| 5044 | persistence_server_running_count | tibco_ftl_persistenceservers{state="running"} |

| Type | FTL metric name | Prometheus name |
|------|--|---|
| | count | |
| 5045 | persistence_server_needs_restart_count | tibco_ftl_persistenceservers{state="needs_restart"} |
| 5046 | persistence_server_timed_out_count | persistence_server_timed_out_count |
| 5047 | persistence_server_exception_count | tibco_ftl_persistenceservers{state="exception"} |
| 5048 | persistence_server_out_of_sync_count | tibco_ftl_persistenceservers{state="out_of_sync"} |
| 5049 | persistence_server_off_line_count | tibco_ftl_persistenceservers{state="offline"} |
| 5050 | persistence_server_destroyed_count | tibco_ftl_persistenceservers{state="destroyed"} |
| 5051 | eftl_cluster_count | tibco_ftl_eftlclusters_total |
| 5052 | eftl_cluster_running_count | tibco_ftl_eftlclusters{state="running"} |
| 5053 | eftl_cluster_needs_restart_count | tibco_ftl_eftlclusters{state="needs_restart"} |

| Type | FTL metric name | Prometheus name |
|------|--------------------------------|---|
| | count | |
| 5054 | eftl_cluster_timed_out_count | tibco_ftl_eftlclusters{state="timed_out"} |
| 5055 | eftl_cluster_exception_count | tibco_ftl_eftlclusters{state="exception"} |
| 5056 | eftl_cluster_out_of_sync_count | tibco_ftl_eftlclusters{state="out_of_sync"} |
| 5057 | eftl_cluster_offline_count | tibco_ftl_eftlclusters{state="offline"} |
| 5058 | eftl_cluster_destroyed_count | tibco_ftl_eftlclusters{state="destroyed"} |
| 5059 | ftl_server_count | tibco_ftl_ftlservers_total |
| 5060 | ftl_server_running_count | tibco_ftl_ftlservers{state="running"} |
| 5061 | ftl_server_needs_restart_count | tibco_ftl_ftlservers{state="needs_restart"} |
| 5062 | ftl_server_timed_out_count | tibco_ftl_ftlservers{state="timed_out"} |
| 5063 | ftl_server_exception_count | tibco_ftl_ftlservers{state="exception"} |

| Type | FTL metric name | Prometheus name |
|------|-----------------------------------|--|
| | count | |
| 5064 | ftl_server_out_of_sync_count | tibco_ftl_ftlservers{state="out_of_sync"} |
| 5065 | ftl_server_off_line_count | tibco_ftl_ftlservers{state="offline"} |
| 5066 | ftl_server_destroyed_count | tibco_ftl_ftlservers{state="destroyed"} |
| 5067 | other_service_count | tibco_ftl_otherservices_total |
| 5068 | other_service_running_count | tibco_ftl_otherservices{state="running"} |
| 5069 | other_service_needs_restart_count | tibco_ftl_otherservices{state="needs_restart"} |
| 5070 | other_service_timed_out_count | tibco_ftl_otherservices{state="timed_out"} |
| 5071 | other_service_exception_count | tibco_ftl_otherservices{state="exception"} |
| 5072 | other_service_out_of_sync_count | tibco_ftl_otherservices{state="out_of_sync"} |
| 5073 | other_service_off_line_count | tibco_ftl_otherservices{state="offline"} |

| Type | FTL metric name | Prometheus name |
|------|---------------------------------------|--|
| 5074 | other_service_ destroyed_ count | tibco_ftl_otherservices{state="destroyed"} |
| 6000 | bridge_active | tibco_ftl_bridge_active |

FTL Server and Interfaces

Administrators can interact with the FTL server using its graphical user interfaces (GUI) and its REST API.

You can use the FTL server GUI to modify the realm definition and to deploy changes to client application processes and affiliated FTL servers. You can also monitor and manage clients as they run, and as the server updates them to use a modified realm definition.

You can use the FTL server *REST API* to do all of the previously mentioned administration tasks programmatically (see [FTL Server Web API](#)).

TIBCO eFTL

Administrators also use the FTL server GUI and web API to configure and monitor the TIBCO eFTL™ service. For information about these GUI pages and REST API, see the TIBCO eFTL documentation.

FTL Server GUI Address

To access the FTL server graphical user interface, use a URL based on one of these templates:

```
https://<ftl_server_host>:<port>
```

```
http://<ftl_server_host>:<port>
```

Swagger REST API Interface

In addition to using curl to interactively access the FTL REST API, you can also use the integrated Swagger interface, which is available via the FTL UI. In the UI, at the command bar at the top, click **REST API Reference**.

GUI Overview

The FTL server GUI uses several visual layouts to display information.

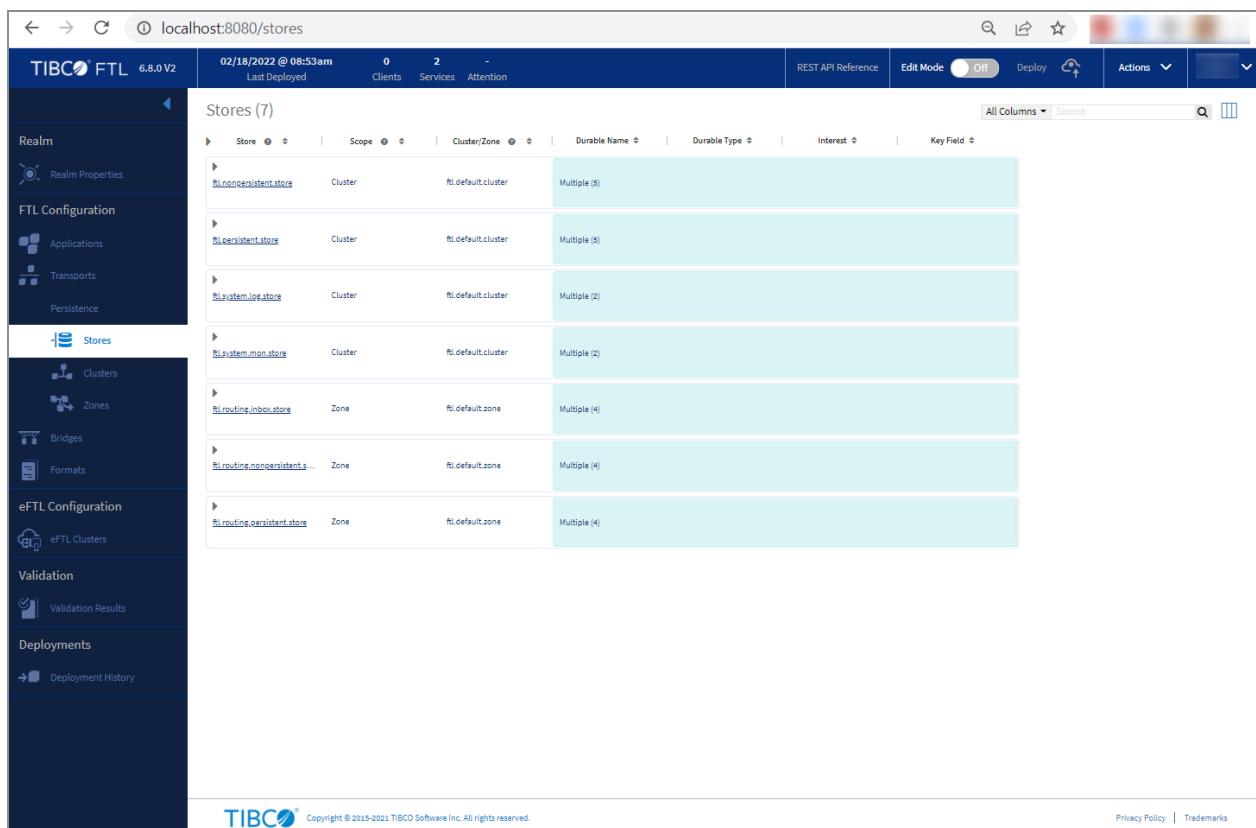
The most important layouts are grids, details panels, and status tables.

Requirements

For best results, view the FTL server GUI in a browser window at least 1280 pixels wide and 800 pixels high, or larger.

For a list of minimum versions of popular browsers, see the product's `readme.txt` file.

Figure 21: FTL GUI



GUI Top Bar

The top bar of the GUI displays the state of FTL services and clients at a glance. You can access some of the most important GUI functionality by clicking in top bar fields.

Last Deployed

The date and time of the most recent realm definition deployment.

Clients, Services, Attention

The number of application *clients* and TIBCO FTL component *services* connected to the local cluster of FTL servers. A numeric badge signals the administrator that some of those clients or services require *attention*.

Clicking anywhere in this area expands more the summary view to subtotal the clients and services by the following categories:

- Clients
- Stores
- Clusters
- eFTL
- Bridges
- FTL Servers
- Groups
- Realm Services
- Other Services

Clicking a category heading opens a status table of items in that category.

REST API Reference

Click **REST API Reference** to interactively access the FTL REST API (web API) using the integrated Swagger interface.

Edit Mode

When **Edit Mode** is off, the GUI displays a read-only view of the realm definition.

When you enable **Edit Mode**, the FTL server obtains the workspace so you can edit the realm definition.

Disabling **Edit Mode** discards your changes to the realm definition.

Clicking **Deploy** saves your changes and deploys the updated realm definition to all clients and services. For complete details, see [The Deploy Transaction](#).

Edit mode is not available for read-only FTL servers, such as satellite servers.

Actions

This menu includes action commands:

- Load or save the realm definition.
- View product documentation.
- Show hidden internal objects. (Internal objects have names starting with the underscore character, and are otherwise hidden.)

User

This field displays the username with which you signed in to the FTL server GUI. Click this area to sign out.

GUI Status Tables

A status table summarizes the current state of FTL clients and services.

Each client and service reports its operating metrics to the FTL server at regular intervals. The FTL server GUI displays the most recent metrics for each client and service.

Rows and Expansions

Each row summarizes the state of an object, with details appearing in columns.

When more detailed information is available about an object, or related sub-objects, you can click on its row to view those details. The row expands downward, revealing the details in a sub-table. To hide the sub-table, click again on the row above it.

For example, the Clusters status table lists each persistence cluster in a row. A cluster row expands to show information about each persistence store in the cluster.

Values less than zero indicate that information is not available.

Action Icons

Some table rows include action icons to the right of their columns. Clicking these icons performs a command action on the object that the row denotes.

GUI Left Column Menu

The left column menu of the GUI is a table of contents into the realm definition and its history. Each item in this menu accesses a page where you can view and modify an aspect of the realm.

While the left column menu is always present, you can minimize the area that it occupies, to yield more horizontal space to main display pane.

GUI Grids

A grid summarizes information about a category of objects in the realm. Rows present objects as rectangles. Columns present details of an object.

Levels

A row of the grid can contain a series of nested rectangles, called *levels*. Horizontally, levels visually group a set of details across several columns. Vertically, levels visually group a set of related items within an object.

For example, the Applications grid shows application definitions at level 1, their endpoints at level 2, and the transport definitions that implement those endpoints at level 3 (along with details about the parameters of those transports).

You can vertically collapse and expand a level by clicking the triangle icon in the upper left corner of its rectangle.

Columns

A column heading in boldface type indicates that the column is always visible in the grid. A column heading in lighter type indicates that you can hide or show the column.

Clicking on a column heading or its sort-order icon sorts the grid according to the contents of that column.

Menu

To open a menu of commands you can apply to an object, click the ... icon at the upper left of an object rectangle. (**Edit Mode** must be **On**.) The menu can contain commands like the following:

- **View** the details of the object.
- **Add** a sub-object at the next level.
- **Delete** the object.
- **Remove** the object from its enclosing object so it becomes an independent object.

The menu contains only those commands that actually apply to the object.

Adding Objects

In edit mode, you can add new objects to a grid.

To add an object at level 1, click the plus icon at the upper left corner of the grid.

To add a child object, select the **Add** command in the menu of the enclosing rectangle.

Deleting Objects

In edit mode, you can delete objects from the grid. Select the **Delete** command in the menu of the object's rectangle.

Pagination

When a *grid* contains many object rows at level 1, it automatically divides them into pages. Page selector buttons at the bottom of the grid indicate the current page. To view another page, click the corresponding button.

When a deeper *level* contains many object rows, the grid automatically divides them into groups. Arrow buttons at the top of the level's column indicate the object rows that level currently displays. To view another group of rows, click these arrow buttons.

When the number of object rows in a level fills more than two groups, you can search for objects using the item name in the level's column, and you can sort the rows in that level by object name.


Grid Search

You can search for objects in a GUI grid, or in any column of a grid. You can search using simple strings or regular expressions.

Understanding a grid can be difficult when a grid presents a large number of objects. Similarly, finding a specific object can be difficult when a grid contains many objects. You can use the grid search feature to reduce the number of objects that are visible in a grid.

The grid search box is at the upper right corner of a grid.

You can apply the search criterion to all columns of the grid, or to a specific column. Use the dropdown menu to select the column scope of the search, or to toggle between string search and regular expression search.

 **Note:** Typing regular expression syntax characters is *not* sufficient to indicate regular expression search. You must explicitly enable regular expression search using the dropdown menu.

You can type a search string or a regular expression in the search box. To submit a search, either type the **Enter** key, click the magnifier icon, or pause during typing. The search box changes color to indicate that you have applied the search criterion and the grid displays a filtered set of objects.

GUI Details Panels

A details panel shows all the configurable parameters of an object in the realm definition. In edit mode, you can modify the parameter values.

Within a details panel, smaller panes visually group together related parameters.

Many configuration definitions have a description field, an optional text field you can use as a comment to describe the application and attach administrative notes. When available, the description appears near the top of the panel, under the definition name.

Restrictions on Names

Names and other literal strings must conform to the restrictions in the following topics.

Reserved Names

All names that begin with an underscore character (_) are reserved. It is illegal for programs to define fields, endpoints, applications, transports, formats, durables, stores, clusters, or zones with these reserved names.

You may use the underscore character elsewhere, that is, where it is *not* the *first* character of a name.

Length Limit

All names are limited to a maximum length of 256 bytes.

This limit includes names of fields, endpoints, applications, transports, formats, durables, stores, clusters, or zones. It also extends to string values in content matchers.

If you use UTF-8 characters that are larger than one byte, remember that the limit restricts the number of bytes in the name, rather than the number of characters.

Size Units Reference

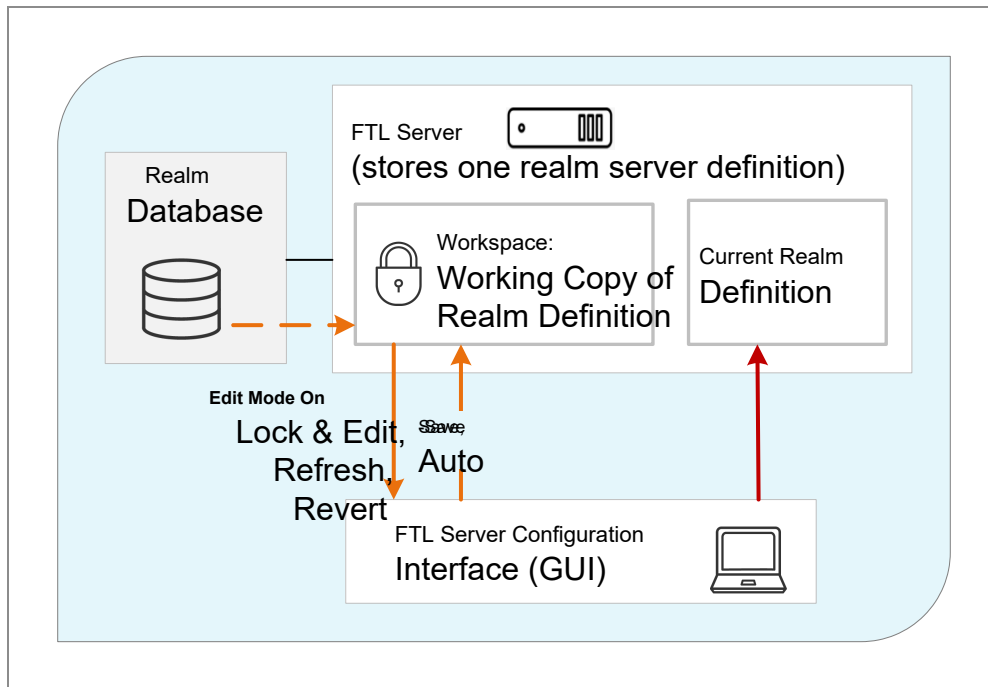
Sizing parameters that accept numeric values also accept unit designators as part of their arguments.

| Unit | Designates | Example |
|-------------|------------|---------|
| <i>none</i> | bytes | 2048 |
| KB, K | kilobytes | 100KB |
| MB, M | megabytes | 500MB |
| GB, G | gigabytes | 1GB |

Realm Definition Storage

Each FTL server stores the definition of exactly one realm stored as the current realm definition or a working copy of the realm definition.

Figure 22: Realm Definition Storage and Transitions



The FTL server configuration interfaces access either the working copy or the current realm definition based on the edit mode:

- **Edit Mode Off (red arrow):** When edit mode is off, the interface accesses the current realm definition which is used by clients and services to guide their behavior.
- **Edit Mode On (orange arrows):** Entering edit mode (orange dotted arrow) obtains the modification lock and opens a workspace as a copy of the current realm definition. The interface displays the workspace and uses it as intermediate storage while the administrator modifies the realm definition. The interface automatically saves your modifications to the workspace.

Modification Lock (Edit Mode On)

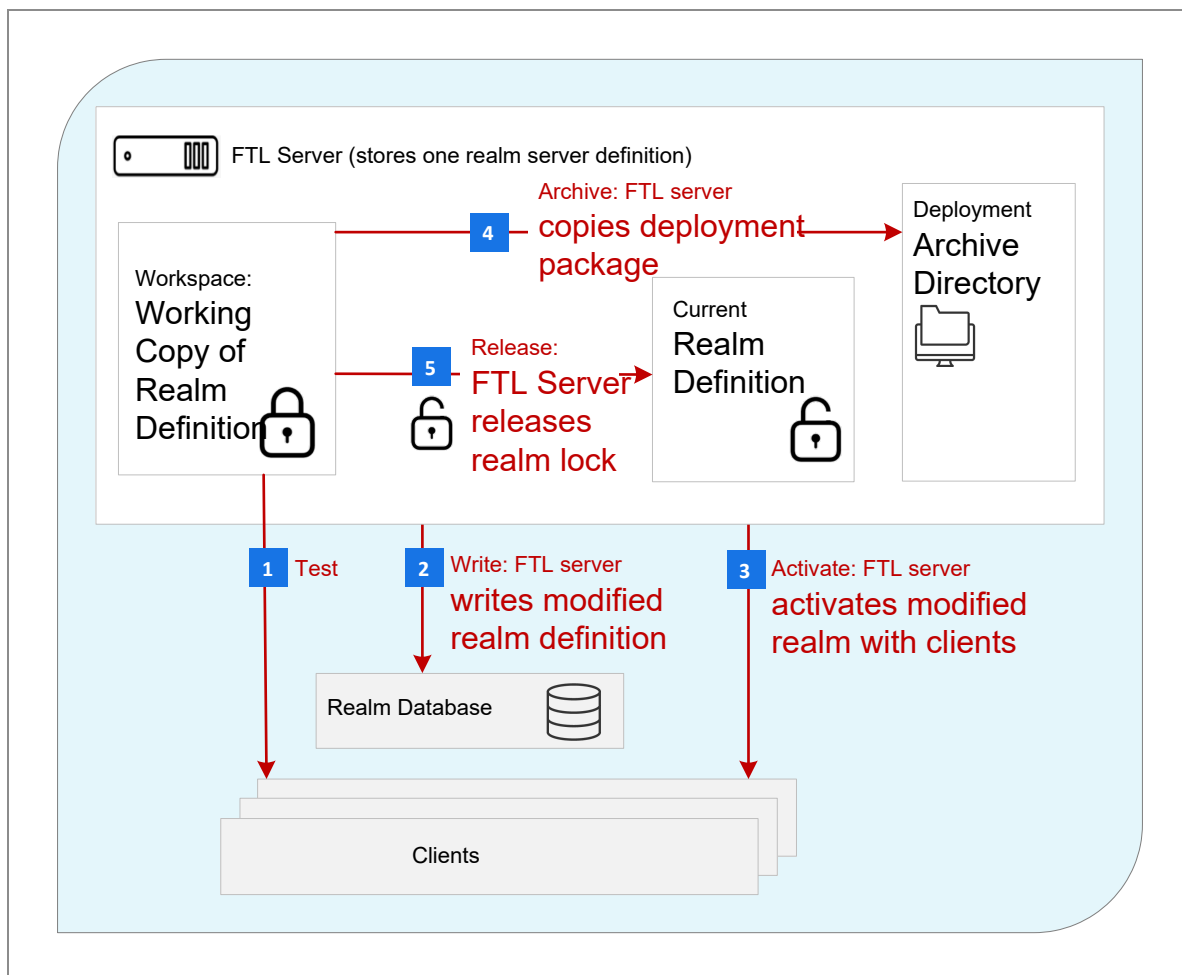
An administrator must be in the `ftl-admin` authorization group to set **Edit Mode On** to obtain a lock to edit the realm definition. When locked, only one administrator at a time

can enter modifications. If you do not hold the lock, the interface is read-only. The lock is not required for management operations, such as purging durables from a persistence store, or saving the state of a persistence service.

The Deploy Transaction

Deploying the workspace for the current realm definition involves several steps. The deploy command initiates a transaction so all clients begin using the modified realm definition.

Figure 23: Steps of the Deploy Transaction



1. *Test.* The server tests the workspace realm definition against all connected client application processes of all affiliated FTL servers (see [Affiliated Servers Test Their](#)

Clients).

Each client tests the modified realm definition to determine whether it can accept the modifications and continue running. Each client reports its response to the FTL server (see [Responses to Deployment Tests](#)). The server combines the answers from its clients, and reports a single consolidated answer (see [Consolidation of Responses to Deployment Tests](#)).

If the consolidated response is OK (test succeeds) or an administrator forces the deployment to continue, the server proceeds to the next step. Otherwise the server rolls back the transaction.

2. *Write*. The FTL server writes the modified workspace realm definition to its database.
3. *Activate*. The FTL server instructs client application processes to begin using the modified realm definition.
4. *Archive*. The FTL server copies the deployment package from the workspace directory to a deployment archive directory.
5. *Release*. The server releases the realm modification lock.

Dialog Options

At each juncture, the deploy transaction dialog offers a subset of these options, as appropriate:

Cancel

Cancel the deployment or test. The transaction rolls back. The FTL server and its clients discard the modified realm definition, and continue as they were, using the realm definition of the previous successful deployment.

Test

Attempt to execute only step 1 of the deploy transaction, and display the results. When the test completes, a dialog again offers a subset of these options.

Deploy

Attempt to execute all the transaction steps, and display the results.

If a test completes step 1 with all clients accepting, you may immediately continue to the remaining steps of the transaction by selecting this option.

Force Deploy

If a test or deploy attempt completed with any clients unable to accept the deployment, and you determine that it is appropriate to continue, then you may force the deployment to the clients that can accept it.

The transaction proceeds to step 2 [Write. The FTL server writes the modified workspace realm definition to its database.](#) . When the transaction commits, the FTL server stores the modified realm definition in the database, and instructs the clients that can accept it to begin using it.

Clients that cannot accept it continue running as they were, however, problems might arise as those clients interact with updated clients.

Responses to Deployment Tests

This table lists the possible responses after testing a realm update deployment, from *most favorable to least favorable*.

For information about the various types of modifications, and the reasons that clients accept and reject them, see [Realm Modifications Reference](#).

Responses after Testing a Modified Realm Definition

| Response | Description |
|-----------------|--|
| OK | The client can accept the modifications, and continue running. |
| Require Restart | The current client process cannot accept the modifications, but restarting the client process would resolve the issue. |
| Rejected | The client program cannot accept the modifications. (Even restarting the client process would not suffice to resolve the issue.) |
| No Response | The FTL server does not receive a response from the client program. (This symptom could indicate that the client has exited, or a network problem prevents communication.) |

Affiliated Servers Test Their Clients

Affiliated FTL servers broaden step 1 of the deploy transaction. A server not only tests the deployment against its own clients, it also tests the deployment against each of its

affiliated servers, which, in turn, test the modified realm definition against their clients.

Each of those clients determines whether it can accept the modifications and continue running. Each client reports its answer to its server. Each affiliated server combines the answers from its clients, and reports a single consolidated answer to the server that requested the recursive test.

The starting server consolidates the answers from its clients and satellites, using the same rules in [Consolidation of Responses to Deployment Tests](#).

Consolidation of Responses to Deployment Tests

The starting server consolidates the responses from all of its clients and all of its affiliated servers.

The consolidated response is the *least favorable* response reported by any client or affiliated server. For example, if 77 clients report OK, and one client reports Require Restart, then the consolidated response is Require Restart. If even one client were to report Rejected, then the consolidated response would be Rejected.

Successful Test

If the consolidated response is OK, then the deploy transaction proceeds to step 2.

Unsuccessful Test

If the consolidated response is anything *other* than OK, then a dialog again offers a set of options, as described previously.

Realm Modifications Reference

The following tables describe the ways in which you can modify the realm definition, and the deployment consequences of each modification.

The details in these tables determine the validity of a deployment (see [Responses to Deployment Tests](#)).

Realm Property Modifications

| Modification | Details |
|-------------------------|--|
| Modify realm properties | <p>You can modify any realm property.</p> <p>If you modify the realm property Realm Properties Details Panel, then some clients could require restart. In particular, if the modification changes an application's effective value of this parameter, then all instances of that application require restart.</p> <p>Modifying <code>use_endpoint_store_for_inbox</code> may require a restart of affected client applications. For details, see Endpoint Store Inboxes.</p> |

Application Definition Modifications

| Modification | Details |
|-----------------------------|---|
| Add applications | You can add a new application definition. |
| Delete applications | <p>You can delete an application definition from the realm definition. All client instances of that application implicitly become invalid (though they continue to run). You must explicitly stop or disable those client processes (see Conditions for Disabling Clients).</p> |
| Add endpoints | You can add new endpoints to an application definition. |
| Delete endpoints | <p>You can delete an endpoint from an application definition, but only if the client does not have any publishers or subscribers on the endpoint.</p> <p>If a client has publisher or subscriber objects on that endpoint, then the application throws a protocol exception.</p> |
| Add transports to endpoints | You can add a transport to implement an existing endpoint in an application instance. Upon deployment, existing process instances immediately bind the new transport to implement the endpoint, and attempt to establish a bus. |

| Modification | Details |
|----------------------------------|--|
| Delete transports from endpoints | <p>You can delete a transport from an endpoint (in an application instance). Client processes immediately stop using that transport to implement that endpoint:</p> <ul style="list-style-type: none"> • Send to inbox calls raise an exception (when the inbox is no longer accessible). • Send one-to-many calls return normally, but the deleted transport does not carry messages. (If the send ability still has other operating transports, then they continue to carry messages. Otherwise the endpoint silently discards messages.) • Subscribers silently stop delivering messages from the deleted transport. |
| Enable or disable abilities | <p>You can enable or disable transport abilities of an endpoint in an application instance.</p> <p>Communications on the endpoint immediately expand to use the additional ability.</p> <p>A disabled ability immediately stops carrying messages:</p> <ul style="list-style-type: none"> • Send to inbox calls raise an exception (when the inbox is no longer accessible). • Send one-to-many calls return normally, but the transport's send ability does not carry messages. (The endpoint silently discards messages.) • The transport's receive and receive inbox abilities silently stop delivering messages. |
| Manage All Formats | <p>If you modify the application parameter Manage All Formats , then clients could require restart. In particular, if the modification changes an application's effective value of this parameter, then all instances of that application require restart.</p> |

Transport Modifications

| Modification | Details |
|-----------------------------|--|
| Add transports | You can add new transports to an application definition. |
| Delete transports | <p>You can delete an existing transport definition. Client processes immediately stop using the bus on the affected transport:</p> <ul style="list-style-type: none"> • Send to inbox calls raise an exception (when the inbox is no longer accessible). • Send one-to-many calls return normally, but the deleted transport does not carry messages. (If an endpoint still has other operating transports, then they continue to carry messages. Otherwise the endpoint silently discards messages.) • Subscribers silently stop delivering messages from the deleted transport. |
| Modify transport parameters | You can modify the parameter values of a transport. If an application instance binds an affected transport, then it requires restart. |

Format Modifications

| Modification | Details |
|--------------|--|
| Add formats | <p>You can add a new managed format to an application definition.</p> <p>You can also upgrade a dynamic format to a managed format, in order to gain efficiency. When upgrading, the managed format must exactly match the existing dynamic format (that is, format name, field names, and field types). However, the managed format may also include additional fields that are not in the existing dynamic format.</p> <p>For a client to use a new managed format, you must also add that format to the application's preload formats list.</p> |
| Add preload | You can add preload formats to an application definition. |

| Modification | Details |
|---------------|--|
| format | |
| Add fields | <p>You can add fields to an existing format definition.</p> <p>Add fields <i>only</i> to the end of an existing format (that is, after all its existing fields).</p> <ul style="list-style-type: none"> • Conforming to this rule facilitates graceful migration of client processes from the old format definition to its new definition. • Violating this rule is disadvantageous. All clients that use the format require restart, and they all must restart in synchrony. Clients that use one version of the format definition cannot recognize messages that use another version, so they silently discard them. |
| Delete fields | <p>You can delete a field from an existing format definition. All clients that use the format require restart, and they all must restart in synchrony.</p> <p>Clients can recognize older format versions only if they received those versions during their initial preload from the FTL server. Clients silently discard messages with unrecognized formats.</p> |

Persistence Modifications: Store and Durable


| Modification | Details |
|------------------------------------|---|
| Add or delete persistence clusters | <p>You can add, rename, or delete a persistence cluster.</p> <p>You cannot delete a cluster definition while any of its persistence services are running.</p> <p>Renaming a cluster is equivalent to deleting and re-adding the cluster definition.</p> |
| Add or delete persistence stores | <p>You can add, rename, or delete a persistence store.</p> <p>Deleting a store discards all messages in the store.</p> <p>Renaming a store is like deleting it and creating a new, <i>empty</i> store.</p> |

| Modification | Details |
|--|--|
| | <p>Before deleting or renaming a store, you must first review every application definition in the realm, ensuring that none of the endpoints still refer to the store you plan to delete.</p> <p>After deleting or renaming a store, all the persistence services in the cluster require restart.</p> <ol style="list-style-type: none"> 1. Use the FTL server interface to stop one persistence service. The FTL server will automatically restart that persistence service. 2. Wait for the new persistence service to synchronize with the existing services. 3. Repeat for each persistence service in the cluster. |
| Add or delete durables | <p>You can add durables to a store, or delete durables from a store.</p> <p>(When deleting a static durable from a store, you must also remove subscriber name mappings to that durable.)</p> |
| Modify store or durable parameters | <p>You can modify parameter values of stores and durables.</p> <p>When modifying the publisher mode of a store, or the acknowledgment mode of a durable, all affected clients disconnect from the cluster (that is, the leader) and automatically reconnect.</p> <p>From the moment that a subscriber object first interacts with a durable, you may no longer change a durable's message interest, nor its matcher fields. If these configuration values must subsequently change, you can instead map the subscriber to a new durable with the correct message interest and matcher fields.</p> |
| Modify the mapping from subscriber names to durables | <p>You can modify the mapping from subscriber names to static durable names for an endpoint.</p> |
| Add dynamic durable | <p>You can add dynamic durable templates and associate endpoints with the new templates.</p> |

| Modification | Details |
|--|---|
| templates | |
| Modify or delete dynamic durable templates | You can modify or delete dynamic durable templates. You can associate an endpoint with a different dynamic durable template. All clients that have durable subscribers on an affected endpoint require restart. |
| Add or delete persistence services | <p>You can add a persistence service. You can delete or rename a persistence service only if none of the services in the cluster are running.</p> <p>Changes to persistence services in the realm definition usually entail parallel changes in the FTL server configuration file. Restart the FTL server to read its updated configuration file.</p> <p>When adding a service, assign it a low weight, so that it does not immediately become the leader. You can raise its weight later, after it replicates the cluster's message data.</p> <p>While reforming a quorum the cluster temporarily becomes unavailable. Ensure that adding or deleting a persistence service does not prevent forming a quorum.</p> |
| Modify cluster transport parameters of a persistence service | It is illegal to modify cluster transports of persistence services. |
| Modify client transport parameters of a persistence service | <p>You can modify client transports of persistence services.</p> <p>When modifying a primary client transport or any of its parameters, that persistence service and all its clients require restart.</p> <ol style="list-style-type: none"> 1. Stop the persistence service. The FTL server automatically restarts it. 2. Wait for the restarted service to synchronize its data state with the remaining services. 3. Restart all the clients of the restarted service. 4. Repeat for each persistence service in the cluster for which you have |

| Modification | Details |
|---|--|
| | modified a client transport. |
| Modify alternate client transport parameters of a persistence service | <p data-bbox="444 375 1281 403">You can modify alternate client transports of persistence services.</p> <p data-bbox="444 436 1349 548">Adding an alternate client transport that was not previously configured does not require restart. Clients can connect using that transport immediately following successful deployment.</p> <p data-bbox="444 581 1328 690">Deleting an alternate client transport does not require restart. Clients cannot connect nor reconnect using the deleted transport. However, existing clients can remain connected on the deleted transport.</p> <p data-bbox="444 724 1408 833">Modifying the transport parameters of an alternate client transport requires restart of the persistence service and the clients that connect using the modified transport.</p> <ol data-bbox="483 867 1398 1150" style="list-style-type: none"> 1. Stop the persistence service. The FTL server automatically restarts it. 2. Wait for the restarted service to synchronize its data state with the remaining services. 3. Restart all the clients of the restarted service. 4. Repeat for each persistence service in the cluster for which you have modified a client transport. <p data-bbox="444 1184 1408 1371">You can add and delete associated client host names. The persistence service does not require restart. Clients on added hosts can connect using the alternate client transport immediately following successful deployment. A client on a deleted host remains connected until the client restarts. For example:</p> <ul data-bbox="493 1404 1398 1728" style="list-style-type: none"> • If the alternate client transport already names host1 and you add host2, then upon deployment clients on host2 can connect using the alternate client transport. • If the alternate client transport names host1 and host3, and you delete host3, then upon deployment clients on host3 remain connected using the alternate client transport. However, a client on host3 that stops and subsequently restarts can no longer connect using the alternate client transport. |

FTL Server GUI: Configuration

You can define and configure the realm using the FTL server's graphical web interface. Click the arrow () in the upper left of the GUI to see these selections:



Realm Properties: See [Realm Properties Details Panel](#).



Application Grid (including instances and endpoints): See [Applications Grid](#).



Transports: See [Transports Grid](#).

Persistence: To configure persistence clusters, stores, and durables, see [Configuring Persistence](#).



Stores: See [Stores Grid](#) and [Store Detail Panel](#).



Clusters: See [Clusters Grid](#).



Zones: See [Zones Grid](#) and [Zone Settings Panel](#).

Permissions: To configure permissions for users and roles, see [Authorization](#).



Users: See [Configuring Permissions](#).



Roles: See [Configuring Permissions](#).



Bridges: See [Transport Bridge Configuration](#) and [Bridges Grid](#).



Formats: See [Formats Grid](#)



eFTL Clusters: See TIBCO eFTLTM [Administration](#), "Cluster and Service Definition" and "eFTL Clusters Grid" topics.




Validation Results: See [Validation Results](#).



Deployment History: See [Deployment History](#).

Realm Properties Details Panel

The Realm Properties details panel presents the global values of realm properties. In edit mode, you can modify the property values. Click the **Realm Properties** icon .

Client - FTL Server Intervals

Several intervals affect the operation of the FTL server and its interaction with clients.

| GUI Parameter | Description |
|---------------------------------------|---|
| Client to FTL Server Heartbeat | <p>The interval at which a client application sends a heartbeat to the FTL server process when no data flow is present (which is also necessary to carry monitoring data). The default is 60 seconds.</p> <p>Zero is a special value, instructing the clients not to send heartbeats.</p> |
| Client to FTL Server Timeout | <p>The amount of time a client waits for a response to the heartbeat sent before closing the existing connection to the FTL Server and initiating a client reconnection. The default is 180 seconds.</p> |

FTL Server - Client Intervals

Several intervals affect the operation of the FTL server and its interaction with clients.

| GUI Parameter | Description |
|---------------------------------------|--|
| FTL Server to client Heartbeat | <p>The FTL server sends heartbeats to its clients at this interval, in seconds. The default is 60 seconds.</p> <p>Zero is a special value, instructing the FTL server not to send heartbeats.</p> |
| FTL Server to client Timeout | <p>The amount of time a FTL server waits for a response to the heartbeat from the client application process. If no response is seen, the FTL server closes the existing connection. The Default is 3600 seconds (1 hour) to allow the FTL server to maintain connections with transient clients as needed. The server</p> |

| GUI Parameter | Description |
|---------------|---|
| | <p>can miss at most 60 heartbeats before triggering the client to close the connection.</p> <p>When a client does not receive server heartbeats for this interval, the client actively attempts to reconnect to a server. If the server is part of a fault tolerant cluster, the client can failover to another core server in the local cluster.</p> <p>The FTL Server uses this timeout to reap connections that are no longer valid.</p> |

Statistics Settings

| GUI Parameter | Description |
|--|---|
| Client sampling interval | Clients take samples of their operating metrics at this interval, in seconds. Zero is a special value, instructing all clients to disable the monitoring feature. |
| Collect subscription statistics | <p>Enabled: When enabled, clients gather monitoring metrics about subscriptions and message substreams, and send them to the FTL server.</p> <p>Disabled: When disabled, clients do not gather these metrics.</p> |

Persistence Settings

| GUI Parameter | Description |
|------------------------------------|--|
| Unlimited persistence retry | <p>Persistent publisher operations, subscriber operations, and map operations require access to the persistence service.</p> <p>When they cannot access the persistence service (usually because of network failure or quorum unavailability), they can automatically retry the interaction.</p> |

| GUI Parameter | Description |
|--|---|
| | <p>By default the Unlimited persistence retry toggle switch is enabled for Persistence operation calls to retry the interactions indefinitely. To disable, click the Unlimited persistence retry toggle switch to disable retry for persistence operations.</p> <p>This value only affects newly created subscribers, publishers, or map objects. Applications may override the default persistence retry duration when creating any of these object</p> |
| Default to non-inline persistence sends | <p>When enabled, by default publishers on endpoints with a persistence store will send messages non-inline. This offers the potential for increased throughput at the cost of higher latency. It also changes the semantics of the send call, since the application may return from the send call before the message is stored.</p> <p>This value only affects newly created publishers. Application code may override the default send policy when creating a publisher.</p> <p>For more information, see Publisher Mode</p> |
| Prevent local message delivery | <p>When enabled, newly created subscriptions enable no-local message delivery by default. The client's application code may override.</p> <p>For more information, see No-Local Message Delivery in Development Guide.</p> |
| Allow permissions | <p>Disabled: When the Allow permissions toggle switch is disabled, the permissions are not enabled for the FTL realm. If authorization is required for any eFTL channels, then the legacy method may be used:</p> <ul style="list-style-type: none"> • Set Publish Group and Subscribe Group to the appropriate values for each eFTL channel on the channel details page. • Select the Authentication checkbox on the eFTL clusters grid. See eFTL Clusters Grid. <p>Enabled: When the Allow permissions toggle switch is enabled, permissions are enabled for the FTL realm. Access to persistence clusters, persistence stores, and eFTL channels can be controlled by assigning permissions on the Users and Roles grids. Ensure that all persistence transports are secure.</p> |

| GUI Parameter | Description |
|---------------|---|
| | Ensure that the Authentication checkbox is checked for eFTL clusters (if used). See eFTL Clusters Grid. |

General Settings

| GUI Parameter | Description |
|---|---|
| Allow dynamic message formats | <p>When the Allow dynamic message formats toggle switch is enabled, applications may use any formats, including dynamic formats.</p> <p>When the Allow dynamic message formats toggle switch is disabled, it restricts the formats available to applications. The only available formats are preload formats and built-in formats. For complete information, see Message Formats Administration</p> |
| Default to no-inline peer-to-peer sends | When enabled, by default publishers on endpoints without any store will send messages non-inline. This offers the potential for increased throughput at the cost of higher latency. Applications may override this setting. |
| Warn about insecure transports between clients | When the Warn about insecure transports between clients toggle switch is enabled, then defining any transports that use nonsecure protocols triggers a validation warning. (This validation warning can trigger even if the FTL server does not enforce TLS and communications.) |
| Service Connection Policy | <p>Set the service connection route.</p> <ul style="list-style-type: none"> • Default: The FTL client attempts to determine the best connection path to the service automatically. • Force Direct: The FTL client connects directly to |

| GUI Parameter | Description |
|---------------|---|
| | <p>the FTL server hosting the service. If no direct path exists, the connection fails. This option is suitable for situations where consistent latency or a consistent network path is desired. <i>Force Direct</i> can reduce the cost and latency associated with an indirect path from client to server, provided that the FTL client can connect directly to each FTL server.</p> <ul style="list-style-type: none"> • Routed: Routed causes the FTL client to initiate all connections using the URL(s) provided in the realm connect call. This is appropriate when it is known that the FTL client is unable to connect directly to each FTL server (for example, the FTL servers are behind a load balancer). |

Message Formats Administration

For efficiency and for safety, administrators can restrict the message formats available to some or all of the applications in a realm. That is, administrators can require that applications use only preload formats and built-in formats, and prohibit applications from using dynamic formats.

Configuration

Two parameters determine whether this prohibition is in effect, and they apply with different scope and opposite polarity.

- The realm property `Dynamic Message Formats` applies to *all* applications throughout the *realm*.
- The application parameter `Manage All Formats` applies only to the instances of a specific *application*.

While administrators can configure these parameters independently, the effective value governing an application depends on both parameters. That is, if either the realm or the application restricts formats, then the restriction is in force for the application.

- To restrict formats in *all* applications, *disallowDynamic Message Formats* on the realm properties page.
- To restrict formats only in a specific application, *enableManage All Formats* on the details panel of the application definition.
- To permit formats without restriction in an application, you must *bothallowDynamic Message Formats* globally, and also *disableManage All Formats* for that application.

See Also:

- Background information: [Formats: Managed, Built-In, and Dynamic](#)
- Realm property: [Realm Properties Details Panel](#)
- Application parameter: [Manage All Formats](#)
- Changes require restart: [Realm Modifications Reference](#)

Applications Grid

The Applications grid presents application definitions in the realm. In edit mode, you can create new application definitions and modify existing application definitions. Click the

Application Grid icon .

Levels

- Application
- Instance (Optional)
- Endpoint
- Transport Connector

Application Level

| Column | Description |
|-------------|--|
| Application | Every application definition has a name. |

| Column | Description |
|---------------|--|
| | This column presents application names. |
| Last Modified | This timestamp indicates the date and time of the most recent change to this application definition. |

Endpoint Level

| Column | Description |
|------------------|---|
| Endpoint | <p>Every application definition has at least one endpoint.</p> <p>This column presents the names of endpoints.</p> <p>This JSON attribute contains a list of endpoint objects, each defined by its name and persistence store.</p> |
| Cluster | <p>Optional.</p> <p>To associate an endpoint with a cluster, select the cluster name from the drop-down menu in this column.</p> <p>Note that the cluster name <code>ftl.forwarding.cluster</code> is associated with a wide-area store. Selecting this option affects the selections in the Store column.</p> |
| Store | <p>Optional.</p> <p>To associate an endpoint with a persistence store, select the store name from the drop-down menu in this column. For wide-area stores, first go to the Cluster column and select <code>ftl.forwarding.cluster</code>.</p> |
| Durable Template | <p>Optional.</p> <p>To associate an endpoint and its store with a dynamic durable template, select the template name from the drop-down menu in this column.</p> <p>The application process uses this template to create all of its dynamic durables.</p> |

Transport Level

Each endpoint may have zero or more transport connectors. See [Address Level](#).

For message broker operation, do not add a transport to an endpoint. Instead, select the "Server" transport, which represents the absence of a direct-path transport.

See [Stores for Message Broker](#).

For background information, see [Implementation: Endpoints \(Micro\)](#).

| Column | Description |
|-----------|--|
| Transport | <p>Required.</p> <p>To apply a specific transport definition to an endpoint, select the name of that transport definition from the drop-down menu in this column. The value <code>Server</code> indicates message broker behavior over a server-defined Dynamic TCP or Auto TCP transport.</p> <p>The remaining columns present details of that transport definition.</p> <p>After adding transport connectors to an endpoint, edit the endpoint details to configure the abilities of the transport connectors.</p> |
| Protocol | The protocol of the transport. (Read only.) |
| Address | The values in the Address, Port, and Mode columns depend on the transport's protocol. |
| Port | |
| Mode | |

Instance Level (Optional)

Every application definition has at least one instance definition, named *default*. If any application defines more than one instance, then the grid automatically expands to show the instance level as level 2. (Otherwise, the grid automatically hides this level.)

For background information, see [Implementation: Application \(Macro\)](#).

| Column | Description |
|------------|---|
| Instance | <p>Every instance has a name.</p> <p>This column presents the names of instances.</p> <p>Instance names must be unique within each application. (You may use the same instance name in several different applications.)</p> <p>All names are limited to a maximum length of 256 characters.</p> <p>For an overall conceptual explanation of instances and transport connectors, see Implementation: Application (Macro). Required.</p> <p>Define the instances of the application.</p> <p>You must configure the default application instance (and you cannot delete it). You may also define other application instances (optional).</p> |
| Identifier | <p>Optional.</p> <p>Identifier string for matching.</p> <p>When present, the transport level of the instance definition applies only to application processes that supply this identifier when connecting to the FTL server. The identifier must match exactly.</p> <p>For more information, see Instance Matching.</p> |
| Host | <p>Optional.</p> <p>Host name or IP address for matching.</p> <p>When present, the transport level of the instance definition applies only to application processes running on this host.</p> <p>To match, this string value must exactly match either the output of the <code>hostname</code> operating system call on the computer where the application process is running, or the IP address bound to that host name.</p> <p>For more information, see Instance Matching.</p> |

Application Definition Details Panel

In the Application field on the Applications Grid, click ... then click View Details. The Application definition details panel presents the parameters that apply to an individual application definition. In edit mode, you can modify the parameter values.

| GUI Parameter | JSON Attribute | Description |
|--------------------|----------------------|--|
| Manage All Formats | manage_all_formats | <p>Restrict the formats available to an application.</p> <ul style="list-style-type: none"> To allow only the preload formats and built-in formats, <i>enable</i> this parameter, or set this JSON attribute to the Boolean value <code>true</code>. To allow dynamic formats too, <i>disable</i> this parameter, or set this JSON attribute to the Boolean value <code>false</code>. <p>For complete information, see Message Formats Administration.</p> |
| Preload Formats | preload_format_names | <p>Determines the set of format definitions that the application receives from the FTL server when the process instance starts, which is also the set of defined formats that an application can use. The process caches in its realm object.</p> <p>The GUI presents a horizontal list of the application's <i>preload</i> formats, and a vertical list of formats <i>defined</i> in the realm. You can filter the list of defined formats.</p> <p>The GUI presents two lists of formats:</p> <ul style="list-style-type: none"> <i>Preload</i> formats, as a horizontal list. <p>Clicking a preload format removes it from this horizontal list (that is, it is no longer a preload format).</p> <ul style="list-style-type: none"> <i>Defined</i> formats, as a vertical list. <p>At first, this list contains all formats defined in the realm. You can filter this list of defined formats.</p> <p>You can add or remove a preload format by clicking its name in the vertical list of defined formats.</p> |

| GUI Parameter | JSON Attribute | Description |
|-------------------------|----------------|--|
| Instance Matching Order | | <p>Instance matching tests the identifier and host values in a specific order. Arrange the rows of this table to specify that order.</p> <p>For background information, see Instance Matching.</p> |

Endpoint Details Panel

In the Endpoint field on the Applications Grid, click ... then click View Details. The Endpoint details panel presents the parameters that apply to an individual endpoint in an application or in an application instance. In edit mode, you can modify the parameter values.

The title of the panel indicates the location of the endpoint definition:

- Application / Endpoint
- Application / Instance / Endpoint

| GUI Parameter | Description |
|--------------------------|--|
| Store | <p>Optional.</p> <p>You can associate at most one persistence store with an <i>application</i> endpoint. Select the store from the dropdown menu.</p> <p>(Instance endpoints inherit the persistence store associated with the corresponding application endpoint.)</p> <p>For background information, see Persistence: Stores and Durables.</p> |
| Dynamic Durable Template | <p>Optional.</p> <p>You can associate at most one dynamic durable template with an application endpoint or an instance endpoint. Select the store from the dropdown menu. The menu offers the templates defined in the persistence store associated with the endpoint.</p> |

| GUI Parameter | Description |
|---|---|
| Inbox Template | The default inbox template is a standard durable template with prefetch and is configured for async acknowledgments. |
| Subscriber Name Mapping for Static Disables | <p>Optional.</p> <p>Note: Subscriber name mapping is an advanced topic.</p> <p>You can map any number of subscriber names to static durable. Select static durables from the dropdown menu. The menu offers the static durables defined in the persistence store associated with the endpoint.</p> <p>Although dynamic durables are simpler to use, this mapping continues to support existing projects.</p> <p>Note: If you change the name of a static durable while client are connected and messages are pending, messages are lost and the client must reconnect to the renamed static durable.</p> <p>For background information, see Configuration of Durable Subscribers in an Application or Instance.</p> |
| Endpoint Abilities | <p>Receive, Send, Receive Inbox, Send Inbox</p> <p>Each row denotes a transport connector, and specifies the ability that the transport carries for the endpoint. You must enable <i>at least one</i> ability for each transport. (A new transport connector is automatically configured with all abilities disabled.)</p> <p>For details, see Abilities, and Implementation: Endpoints (Micro).</p> |

Transports Grid

The Transports grid presents transport definitions in the realm. In edit mode, you can create new transport definitions and modify existing transport definitions. Click the

Transports icon .

For background information, see [Transport Concepts](#).

Levels

- Group
- Transport
- Address

Group Level

| Column | Description |
|--------|--|
| Group | <p>Optional.</p> <p>A group is like a folder or label. You can assign transport definitions to groups to help you organize them by purpose. To see each transport in its group, sort the grid on this column.</p> <p>However, a group is <i>required</i> in one special circumstance. When you define a pair of dynamic TCP transports, you must assign them to the same named group. Without a common group name, the listen and connect ends cannot establish a bus. (This requirement applies only to a listen and connect pair. It does not apply to a mesh.) For background information, see Dynamic TCP Transport.</p> |

Transport Level

| Column | Description |
|-----------|---|
| Transport | <p>Every transport definition has a name.</p> <p>This column presents transport names.</p> <div>Note: You cannot configure two transport definitions with identical names.</div> |
| Protocol | <p>Required.</p> <p>Every transport uses a specific protocol to establish a communication bus and to carry messages. See Transport Protocol Types</p> |

| Column | Description |
|---------------|--|
| | Select a transport protocol type from the drop-down menu. |
| Last Modified | This timestamp indicates the date and time of the most recent change to this transport definition. |

Address Level

The values in the Address/Subnet, Port Range, and Mode columns depend on the transport's protocol. They present the transport parameters that are most important for connecting to the transport, diagnosing misconfigurations, and distinguishing among transport definitions.

- **Dynamic TCP Protocol**

- **Address:** Empty, or a subnet mask.
- **Port:** *Auto-Assign*, or a TCP port for establishing a bus through a firewall.
- **Mode:** Mesh, listen, or connect.

- **Connection-Oriented Protocols**

Static TCP, Secure Static TCP, and Reliable UDP are connection-oriented protocols.

- **Address:** The host name for establishing a connection.
- **Port:** The port number for establishing a connection.
- **Mode:** To establish a bus, the transport either *listens* or *connects* on the host and port.

- **Multicast Protocol**

- **Address:** A multicast group for communication.
- **Port:** The multicast port number for communication.
- **Mode:** The transport uses the multicast group in the address column as either a *send group* or a *listen group*.

- **Shared Memory Protocols**

Shared Memory and Direct Shared Memory protocols use shared memory segments for communication.

- **Address:** The name of the shared segment.
- **Port:** Empty.
- **Mode:** Empty.

Transport Details Panel

In the **Transport** field on the **Transports** grid, click ... then click **View Details**. The Transport details panel presents the configuration details of a transport definition. In edit mode, you can modify the definition.



Note: It is erroneous to configure two transport definitions with identical parameter values.

The contents of this panel depend upon the transport protocol. For descriptions of the parameters, see the parameters reference topics for each protocol type (those topics are sub-topics under [Transport Protocol Types](#)). For buffer and performance parameters that are common to several transport protocols, see [Buffer and Performance Settings for Transports](#).

See Also: [Transport Concepts](#)

Formats Grid

The Formats grid presents message format definitions in the realm. In edit mode, you can create new format definitions and modify existing format definitions. Click the **Formats**

icon .

See also, [Defining Formats](#).

Levels

- Format
- Field


Format Level

| Column | Description |
|--------------------|---|
| Format | <p>This column presents the names of formats.</p> <p>Format names must be globally unique within the realm.</p> <p>All names are limited to a maximum length of 256 characters.</p> |
| Historical Version | <p>When you modify a deployed format definition by changing the number or names of fields, the FTL server creates a new version of the definition, and retains the old version. The FTL server retains older versions so that durable subscribers can still receive messages from the persistence store, even though those messages use older format versions.</p> <p>You may explicitly delete an old format version if you are certain that it is no longer relevant: that is, the older version is no longer used by any publishing applications, nor by any messages within persistence stores.</p> |
| Last Modified | <p>This timestamp indicates the date and time of the most recent change to this transport definition.</p> |
| Description | <p>Optional.</p> <p>This column presents a text field. You can annotate a format with a comment describing its usage.</p> |

Field Level

| Column | Description |
|--------|---|
| Field | <p>This column presents the names of fields.</p> <p>Field names must be unique within each format. (You may use the same field name in several different formats.)</p> |
| Type | <p>Every field holds data of a specific data type.</p> <p>Select a data type from the drop-down menu. For descriptions of the data types, see "Field Data Type Reference" in TIBCO FTL Development.</p> |

Validation Results

The FTL server validates the realm configuration. A GUI page reports configuration errors and warnings detected. Click the **Validation Results** icon ; this example icon identifies there are four errors which will prevent deployment.

Validation

The FTL server configuration interface validates modifications as you make them (that is, at every save and automatic save operation). The FTL server also validates when you deploy.

See Also:

- [The Deploy Transaction](#)

Columns


| Column | Description |
|-------------|---|
| Severity | Error, Warning, or Info For more information, see the following section, Severity of Validation Issues . |
| Component | The type of configuration definition which triggered a validation exception. This value is also a link to the GUI page where you can reconfigure the problematic definition. |
| Name | The name of the problematic definition. |
| Field | The problematic parameter of the definition. |
| Description | A brief description of the issue that triggered the validation exception. |

Severity of Validation Issues

The FTL server detects configuration issues and categorizes them by severity:

- **Error** The definition is not deployable.
For example, required parameters of a definition are missing, invalid, or undefined, as in an endpoint that lacks a transport.
- **Warning** The definition is deployable, but inconsistent, and could result in a run time issue.
For example, a dynamic TCP transport listen end is defined, without a corresponding connect end.
- **Info** The definition is deployable, but could be misconfigured.
For example, a format definition without any fields, or a persistence store without any static durables nor dynamic durable templates.

Deployment History

The Deployment History table presents details about the current and past deployments of the realm definition. Click the **Deployment History** icon .

| GUI Field | JSON Attribute | Description |
|-------------|---|---|
| Name | name | Name of the deployment. The deploy transaction dialog supplies an automatically generated name. Administrators can supply a more descriptive name. |
| Description | description | Optional. Administrators can add a string in the deploy transaction dialog, as a comment to describe the deployment and attach administrative notes. |
| Revision | realm_ revision (Numeric value type.) | Revision number of the realm definition as stored in the FTL server. This number increases each time you deploy modified definitions to the FTL server. This number does not |

| GUI Field | JSON Attribute | Description |
|-----------------|------------------|--|
| | | change if the changes do not affect realm semantics (for example, modifying only a description string). |
| Deployment Date | last_deployed_on | <p>Deployment request timestamp.</p> <p>This field displays the date and time that an administrator requested deployment. (After redeploying, this value can differ from Creation Date.)</p> |
| Deployer | last_deployed_by | <p>username of the administrator who requested the deployment.</p> <p>If the FTL server enables user authentication, this field displays the username that requested the deployment.</p> <p>If user authentication is disabled, this field displays the string anyone.</p> |
| Creation Date | created_on | <p>Deployment creation timestamp.</p> <p>This field displays the date and time that the deployment package was created.</p> |
| Creator | created_by | <p>username of the administrator who created the deployment package.</p> <p>If the FTL server enables user authentication, this field displays the username that created the deployment.</p> <p>If user authentication is disabled, this field displays the string anyone.</p> |

Commands

Icons at the right of each deployment row manipulate deployments.

- Redeploy the realm definition from the deployment of this row.
- Delete this deployment from the history list.

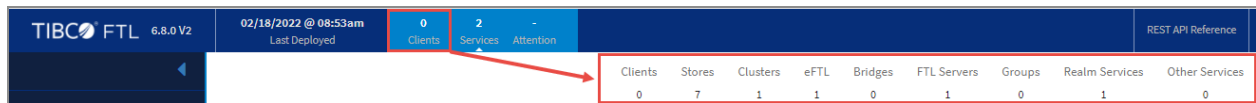
New Clients during a Deployment

When a deployment problem requires administrator intervention, the FTL server does *not* respond to requests from *new* clients that need an initial realm definition. Those new clients cannot initialize until you resolve the deployment problem.

The resolution you choose determines the official realm definition (either the new deployment or the previous deployment). Then the FTL server can send that realm definition in response to requests from new clients.

FTL Server GUI: Monitoring

You can use the GUI to monitor and manage the clients and services in a realm. Click **Clients** in the top bar to see the selections.



Make a selection to monitor:

- Application **Clients**: See [Clients Status Table](#).
- Persistence **Stores**: See [Persistence Stores Status Table](#).
- Persistence **Cluster** services: See [Persistence Clusters Status Table](#).
- **eFTL**: See the TIBCO eFTL [Administration](#) guide, "eFTL Clusters Status Table and Servers List" topic.
- Transport **Bridges**: See [Bridges Status Table](#).
- **FTL Servers**: See [FTL Servers Status Page](#).
- A **Group** of fault-tolerant applications: See [Groups Status Table](#).
- **Realm Services**: See [Realm Services Status Page](#).
- **Other Services** related to your realm, including Client [Status](#), Client Label, ID, Host, and Application.

Clients Status Table

The Clients status table presents the current state of the FTL server's application clients. In the top bar, click **Clients** then **Clients**.

Each row summarizes the state of one client. To see more detail about a client, click its row (see [Client Status Details](#)).

Columns

| Column | Description |
|--------------|---|
| Status | For descriptions of these values, see Client Status . |
| Client Label | The functional role of the client process. For background information, see Client Label . |
| ID | The FTL server assigns each client a unique identifier. |
| Host | The host name string of the client's host computer (if available). |
| Application | Application name (as configured in the realm definition). |
| Instance | Application instance name that the client matches. |
| Connect To | Name of the core server that serves the client. |

Commands

Above the top row at the far right is a the **Purge all timed out clients** button to purge the list of clients that have timed out.

At the far right of each client row, icon buttons trigger the following commands:

- Change logging level of a client.
 - Off
 - Severe
 - Warning
 - Info (default)
 - Verbose
 - Debug
 - Custom

For descriptions of log level values, see the TIBCO FTL [Development](#) guide, "Log Level

Reference". The Custom selection provides a text box to set different levels for different categories (elements) of log entries. For example:

```
api:debug; msg:off; transport:debug
```

For a list of elements, see the TIBCO FTL [Development](#) guide, "Log Element Tags Reference" topic.

- Change the logging mode of a client.

Collecting logs from many clients can produce a large volume of data. For each individual client you can enable it to send logs to the FTL server, or disable it from sending its log messages. All clients initially disable sending logs.

- Change the monitoring mode of a client.

Collecting subscription statistics can produce a large volume of data. You can enable and disable this feature for each individual client. When disabled, a client does not collect monitoring data.

- Disable a client.

For details, see [Conditions for Disabling Clients](#).

Client Status

These values report the status of FTL client processes.

Running

The client is running, and its local realm definition is up to date (that is, it is the latest realm deployment).

Needs Restart

The client is running, but its local realm definition is out-of-date. Restart the client to update its realm definition to the latest deployment.

Timed Out

The FTL server has lost the client's heartbeat signal. Either the client has stopped, or a network disconnect obstructs the heartbeat signal.

Exception

The client is running, but its realm definition is out-of-date. The deployment caused an error in the client, so the client continues to use its old realm definition.

Restarting the client might *not* be sufficient to resolve the issue. For example, the deployment specifies a new static TCP transport, but the port is already bound in some other process.

Out-of-Sync

The client's realm definition is a different revision than the server's.

For example, if an administrator deploys a modified realm definition while a network disconnect separates some clients from the FTL server, then those clients are out of sync with the new realm definition. When they reconnect, the FTL server displays this status until it can automatically deploy the latest realm definition to them. (After that catch-up deployment, their status shifts either to running or needs restart.)

Conditions for Disabling Clients

Administrators can disable individual client processes. A disabled client cannot use TIBCO FTL communications resources, and the FTL server no longer recognizes it as a client. Consider disabling clients in the following situations.

- If a client sends invalid messages that disrupt other clients, disable the misbehaving client so that other clients can continue to communicate.
- If a client consumes more message bandwidth than is appropriate, depriving other clients of communications resources, consider disabling the misbehaving client so that other clients can continue to communicate.
- When you delete or rename an application from the realm definition, all existing client instances of that application implicitly become invalid: that is, the realm no longer supports them. Nonetheless, they continue running. You must explicitly disable those clients.
- When you modify the realm definition, clients that cannot accept the new revision might need to restart. If immediately stopping their messages is critical, you can disable them as a group (and then restart them in some other way).

Client Status Details

To see more details about a client, click its row in the [Clients Status Table](#).

Client Information

This section presents information about the client as a process. To access this information using the web API, see [Clients Status Objects](#).

| GUI Field | Description |
|-------------------------------|---|
| Client ID | Client ID (assigned by the FTL server) identifies a client instance. |
| Type Label | A string denoting the client's type. |
| Realm Revision | Revision number of the realm definition as stored in the FTL server. This number increases each time you deploy modified definitions to the FTL server. This number does not change if the changes do not affect realm semantics (for example, modifying only a description string). |
| Status | An integer representing the client's status. |
| Status Label | A string describing the client's status. For descriptions of these values, see Client Status . |
| Client Label | Identifies a logical client and it can be identical across multiple instances. The functional role of the client process. For background information, see Client Label . |
| Connect (ms), Time (ms) | Timestamp when the client first contacted the FTL server. |
| Uptime (ms) | Time that the client has been running. |
| Last Contact Time | Time of the most recent contact between the FTL server and this client. |

| GUI Field | Description |
|-------------------------|--|
| Last Contact Delta (ms) | Approximate elapsed time since the most recent contact between the FTL server and this client. |
| Other Info | Additional information about the client, such as hardware, operating system, and process ID. |
| User | The client supplied this value for the username property in its realm connect call. |
| Version | Version string of the client's TIBCO FTL library. |
| Host | Host name of the client's host computer. |
| IP Address | IP address of the client's host computer. |
| Identifier | The application identifier of the client. The client passes this string as a property value when connecting to the FTL server. |
| App Name | Application name (as configured in the realm definition). |
| App Instance | Application instance name that the client matches. |
| Startup Time | Time that the client process started running. |

Application, Endpoints, and Queues

These sections present recent metrics from the client application. For explanations of these values, see [Catalog of Metrics](#).

Recent Advisories and Recent Logs

These sections present the most recent advisories and logs from the client. The FTL server retains up to 25 items per client in each table. For access to older items, see "Central Logging" in *TIBCO FTL Monitoring*.

Client Label

The *client label* is a human-friendly string that names the functional role of a client process. This label can help administrators recognize client application processes by their role within the enterprise. Client ID is given to the client by the realm (FTL server) but the client label is set by the client itself via `TIB_REALM_PROPERTY_STRING_CLIENT_LABEL`.

Programmers

It is good programming practice to supply a client label property in the realm connect call.

For flexibility, design programs to accept a client label value as a command line or configuration argument, which overrides a default value.

Administrators

The FTL server includes the client label in GUI displays, web API responses, and monitoring data.

Administrators can use identical client labels to maintain name recognition when several instances of an application run on different host computers. Conversely, you can use distinct client labels to distinguish among several instances of an application that run on the same host computer.

Administrators can use the client label to maintain continuity, so you can recognize a client process by its role even after it restarts. (In contrast, the FTL server assigns a unique client ID each time an application process restarts.)

FTL Servers Status Page

The FTL Servers status page presents the status of FTL servers.

FTL Servers Status Heat Map

The left pane of this page presents a graphic overview of the status of FTL servers and the services they provide.

Each square in the top overview map denotes an FTL server. Each rectangle below also denotes an FTL server, with enclosed rectangles denoting its services. Color indicates the health of the server, based on the aggregated status of its services:

- **Green** The server is running and all its services are running. The service is running.
- **Yellow** The server is running, but one or more of its services are not.
- **Red** The server or service is not running.

To view more details about an FTL server, click either of its graphic representations. The right pane of the page presents those details.

FTL Server Status Details

The right pane on the status page presents the state of an FTL server in detail. This includes connection counts:

- **Clients Connect Count (Max)** The cumulative number of different clients that have connected to this FTL server since restart.
- **Clients Connect Count (Current)** The number of clients currently connected to this FTL server.
- **Rejected Client Connections** The cumulative number of rejected client connect attempts since restart.
- **Client Lookup Failures** The cumulative number client lookup failures since restart.

To stop this FTL server and all the services it provides, click the **Shutdown this FTL Server** button.

Realm Services Status Page

The Realm Services status page presents the status of realm services.

Realm Services Pane

The left pane summarizes that status of the realm.

The bar graph indicates the distribution of clients and services among the realm services of the core servers cluster.

To view more details about a specific realm service, click its name in the bar graph. The right pane of the page presents those details.

Realm Services Status Details

The right pane presents the state of a realm service in detail.

Satellites Table

The next block is a table that summarizes the status of satellite servers, if any exist.

| Column | Description |
|----------------|---|
| Status | Aggregated client status of the satellite's clients. For explanations of these status values, see Client Status . |
| Server | This string identifies the satellite server process by its label argument. If the server command line omitted a label argument, this column instead displays the server's host name and HTTP port. |
| Host | The host name of the satellite's host computer. |
| Last Contact | Date and time of the most recent protocol message from the satellite. |
| Realm Revision | Revision number of the realm definition as stored in the FTL server. This number increases each time you deploy modified definitions to the FTL server. This number does not change if the changes do not affect realm semantics (for example, modifying only a description string). |

When a satellite server no longer exists, the GUI purges it from the satellites table after a timeout interval.

FTL Server Web API

You can use the FTL server web API to configure the realm definition, to monitor and manage the operation of clients and services, and to manage aspects of the FTL server itself.

The web API combines selected capabilities from the FTL server GUI and from the `tibftladmin` executable utility.

Realm Definition

You can use the web API to configure and deploy the realm definition.

The scope of the configuration API includes applications, formats, transports, transport bridges, persistence, eFTL, and FTL server properties.

Realm Monitoring and Management

You can use the web API to monitor and manage clients and other components that interact with the FTL server. For example, you can get status information, get client metrics, and update aspects of components to affect their operation.

The scope of the monitoring API includes clients, transport bridges, persistence, and eFTL.

FTL Server

You can use the web API to get the status of the FTL server, to back up the FTL server database, and to shut down the FTL server.

HTTP Request Addressing

Request addresses depend on the host and port of the FTL server.

When sending REST-style HTTP requests using the FTL server web API, address them to URIs with one of these prefixes:


```
http://<host>:<port>/api/v1/
```

```
https://<host>:<port>/api/v1/
```

- <host>:<port> is the FTL server's port (from its configuration file).
- v1 denotes the version of the FTL server web API. This version number could change in subsequent releases.

For example, using the `curl` utility:

```
curl -X GET http://<host>:<port>/api/v1/clients
```

```
curl -X POST http://<host>:<port>/api/v1/server --data '
{"cmd":"shutdown"}'
```

For brevity, this document omits the prefix, but you must supply it. For example:

- The documentation for the request `GET clients` implies that you would send a GET request to this URI:

```
http://<host>:<port>/api/v1/clients
```

- The documentation for the request `POST realm/transport/<name>` implies that you would send a POST request to this URI:

```
http://<host>:<port>/api/v1/realm/transport/<name>
```

(Substitute the actual transport name for the variable <name>.)

JSON Attribute Values

The web API uses JSON representations to express configuration and monitoring information. Information in the documentation topics for the API objects guide you in composing valid JSON request values and interpreting JSON response values.

HTTP Authentication

The FTL server web API can authenticate requests using HTTP basic authentication.

If the FTL server enables user authentication, then the web API authenticates requests using HTTP basic authentication.

Authenticated requests must include username and password credentials in the HTTP authentication header.

If the FTL server disables user authentication, then the web API does not require authentication for requests.



Note: Secure FTL servers use HTTPS and TLS.

See Also

[FTL Server Authorization Groups](#)

HTTP Authentication

HTTP Headers

The HTTP header Content-Type ensures correct processing of requests that contain JSON data.

When a web API request contains data in JSON format, include this HTTP header if needed:

```
Content-Type: application/json; charset=utf-8
```

For example, many POST and PUT methods in the API specify realm definition objects using JSON data.

Response Status Codes

HTTP response codes indicate success or failure of an FTL server web API request.

Codes in the 200 range indicate that the request succeeded. Codes in the 400 and 500 range indicate an error.

| Status Code | Reason | Description |
|-------------|-----------------------|--|
| 200 | OK | The request succeeded. |
| 201 | Created | The request succeeded in creating a resource. |
| 202 | Accepted | The request was accepted, but has not yet completed. |
| 204 | No content | The request succeeded, but did not return any JSON response. |
| 400 | Bad Request | The request is invalid because of syntax errors. |
| 401 | Unauthorized | The request did not include valid credentials. |
| 403 | Forbidden | The request is syntactically correct, but not allowed. For example, you cannot directly configure a satellite FTL server. |
| 404 | Not Found | The requested resource does not exist. |
| 405 | Method Not Allowed | The resource does not allow the requested method. |
| 409 | Conflict | The request conflicts with the current state of the resource. |
| 500 | Internal Server Error | An error in the FTL server prevented completion of the request. |

Responses

Responses to requests can contain a JSON representation.

Success Responses

Responses to successful GET, POST, and PUT requests contain JSON that represent the objects retrieved, created, or modified.

Responses to successful DELETE requests indicate status 204.

Error Responses

JSON error responses can include fields that contain further information about the error.

| Field | Description |
|---------|--|
| status | number The HTTP response code. |
| message | string Specific information about the error. |
| details | list of strings Each string describes a validation error that the FTL server detected while processing a POST or PUT request. |

Pagination

You can supply pagination parameters with GET requests to retrieve long lists of objects in smaller batches.

Parameters

| Parameter | Description |
|-------------|--|
| limit=<max> | Optional. Retrieve at most <max> objects. |

| Parameter | Description |
|--------------------|--|
| | When absent, retrieve the entire list. |
| offset =<start> | Optional. Available only for realm configuration requests. Retrieve objects starting at position <start>. When absent, the default offset is zero, indicating the first object in the larger list. |

Example

To get the first ten application objects:

```
realm/applications?limit=10
```

To get the next ten objects:

```
realm/applications?limit=10&offset=10
```

See Also

[Parameters for Client Filtering](#)

Semantics of Web API Objects and Methods

Each category of FTL server web API objects has a distinct semantic.

Realm Definition Objects

Realm definition objects correspond to aspects of the realm definition.

The main realm definition objects are applications, bridges, eftl, formats, persistence, and transports. Objects nested within these objects also belong to this category and have similar method semantics; for example, formats/<format_name> and eftl/clusters/<cluster_name>/channels.

For objects in this category, POST, PUT, and DELETE methods require the modification lock. These methods always affect the realm definition in the workspace, not in the FTL server database.

In contrast, the results of GET methods depend on the state of the modification lock:

- If you hold the modification lock, GET methods retrieve information from your workspace.
- If you do not hold the modification lock, and even if another user holds the lock, GET methods retrieve information from the FTL server database.
- While a deploy transaction is active, GET methods return an error response.

POST methods define new objects.

PUT methods update definition objects.

- If the object already exists, PUT updates its definition, completely replacing the old definition with method's input data.

To change the name of a definition object, supply the old name in the URI, and the new name in the JSON input data.

- If the object does not exist, PUT creates an object using the definition in the JSON input data (like POST).

Workspace

The workspace represents a modified realm definition that has not yet been deployed.

The main workspace object is `realm/workspace`.

GET methods retrieve or validate the workspace.

POST methods create a workspace.

DELETE methods delete the workspace.

Deployments

Deployments represent the current state of the realm definition. The FTL server maintains a history of successful deployments (this history excludes attempted deploy transactions that failed and rolled back).

The main deployment object is `realm/deployments`.

GET methods retrieve a collection of deployments or an individual deployment.

POST methods start a deploy transaction or test.

PUT methods can redeploy an earlier deployment.

DELETE methods delete a deployment from the history list.

Status Objects

Status objects correspond to executable processes and to objects within those processes. For example, executable processes include your own client processes, as well as operating components of TIBCO FTL software, such as transport bridge services, eFTL services, and persistence services. Some components contain nested objects; for example, persistence services contain stores and durables.

The main status objects are `server`, `clients`, `bridges`, `eftl`, `groupservice`, and `persistence`. Objects nested within these objects also belong to this category, and have similar method semantics; for example, `persistence/<cluster_name>/stores/<store_name>/durables`.

For objects in this category, GET methods report the status or metrics related to a process or its nested objects. GET methods can also retrieve a collection of nested objects from a process.

POST methods update the operating parameters of a process, such as logging behavior. POST methods can also deliver commands to a process, for example, to suspend it.

DELETE methods can delete a client status object from the FTL server, so the FTL server no longer tracks that client. Delete methods can also remove a nested object from within a client, for example, removing a durable subscriber from within a client.

The Monitoring Object

The main monitoring object, `monitoring`, provides access to a current snapshot of client metrics.

Configuring the Realm Definition Using the Web API

Administrators can use the FTL server web API to configure and deploy the realm definition.

Procedure

1. Create a realm workspace.

This step simultaneously requests the realm modification lock (see [Modification Lock \(Edit Mode On\)](#)) and creates a workspace.

```
POST realm/workspace
```

The realm modification lock ensures that at most one workspace can be open at a time. If another user has an open workspace, this method returns an error code.

While the workspace is open, the FTL server processes all requests in the context of that workspace. For example, methods that create, modify, or delete definitions affect the workspace, not the FTL server database.

Perhaps less intuitively, while you hold the modification lock, GET methods get definitions from your workspace, rather than from the database, reflecting changes you have made but not yet deployed. For users who do not hold the lock, GET methods still get definitions from the database. (For background information, see [Realm Definition Storage](#).)

2. Modify the workspace by adding definition objects and updating existing definition objects.

For example:

```
POST realm/transport <transport_json_data>
```

Repeat this step until your realm configuration is complete.

3. Validate the candidate realm definition.

Successful validation indicates that all the object definitions are complete, and mutually consistent.


```
GET realm/workspace/validation
```

If the validation results indicate errors, you must correct them before proceeding to the next step.

4. Start the deploy transaction.

This action corresponds to the full deploy transaction. (See [The Deploy Transaction](#).)

```
POST realm/deployments <deployment_json_data>
```

The success response code 202 indicates that the FTL server has accepted the deployment request. The workspace becomes the active deployment candidate. The deploy transaction continues asynchronously.

5. Check the results of the deploy transaction.

You can check the results of named deployment using the web API:

```
GET realm/deployments/<name>
```

You can also see the deployment results on the Deployment History page of the FTL server GUI (see [Deployment History](#)).

Creating or Modifying a Definition

You can create new definitions within the realm, or modify existing definitions, using the FTL server web API. The procedures for creating or modifying definitions are very similar, although their web requests use slightly different URIs.

This task occurs within the super-task [Configuring the Realm Definition Using the Web API](#).

Before you begin

You have already created and locked a deployment workspace for editing.

Procedure

1. Compose the JSON object representing the new definition.
2. Compose and send the web request.
 - To create a new definition, use the POST method:

```
POST <object_type><defn_json>
```

- To update an existing definition, use the PUT method:

```
PUT <object_type>/<defn_nam><defn_json>
```

3. Verify the response.
 - a. Ensure that the request succeeds, by checking the HTTP response code.
 - b. Check for validation warnings.

Workspace

Life Cycle

Each workspace begins as a working copy of the realm definition. An administrator can modify the realm definition in the workspace without affecting realm operations.

When modifications to the workspace are complete, the administrator can deploy the workspace definition, using the deploy transaction. See [POST realm/deployments](#).

For more information, see these topics:

- [Realm Definition Storage](#)
- [The Deploy Transaction](#)

Tasks and Methods

| Task | Method |
|---|--|
| Get the username of the administrator who has locked the workspace. | GET realm/workspace |
| Get validation results for the workspace. | GET realm/workspace/validation |

| Task | Method |
|--|--|
| Request the modification lock, and create a workspace for editing. | POST realm/workspace |
| Delete the workspace and release the modification lock. | DELETE realm/workspace |

JSON Attributes

GET requests return JSON data.

| Attribute | Description |
|-----------|---|
| user | The value of this attribute is the username of the administrator. When user authentication is disabled, the special username anyone represents any user. |
| results | The value of this attribute is a collection of strings. Each string describes a validation error in the workspace. |

GET realm/workspace

The web method `GET realm/workspace` retrieves the username of the workspace owner (that is, the administrator who holds the modification lock).

Example Requests

```
curl -X GET http://<host>:<port>/api/v1/realm/workspace
```

Example Response

```
{
  "user": "anyone"
}
```

The special value, anyone, indicates that the user did not supply a username credential. This situation is possible only when user authentication is disabled.

GET realm/workspace/validation

The web method GET `realm/workspace/validation` validates the realm definition in the workspace, and returns the validation results.

Example Requests

```
curl -X GET http://<host>:<port>/api/v1/realm/workspace/validation
```

Example JSON Response

```
{
  "results":[
    "TCP Transport disaster recovery transport for server named server-1457388321896 contains an empty host.",
    "TCP Transport disaster recovery transport for server named server-1457388321896 contains an empty port."
  ]
}
```

POST realm/workspace

The web method POST `realm/workspace` requests the modification lock, and opens the workspace.

If the modification lock is available, you can use this method without the `force` parameter.

If you request the realm modification lock while *another* user holds it, this request returns a 403 error response. Nonetheless, you can force this operation, grabbing the lock:

```
POST realm/workspace?force=true
```

Forcing this operation deletes the other user's workspace. Your new workspace is a fresh copy of the most recent deployment.

HTTP Parameters

| Syntax | Description |
|-----------------|---|
| force=<boolean> | Optional. When true, grab the modification lock, and create a new workspace. |

Example Requests

```
curl -X PUT http://<host>:<port>/api/v1/realm/workspace
```

Example Response

```
{  
  "user": "anyone"  
}
```

The special value, anyone, indicates that the user did not supply a username credential. This situation is possible only when user authentication is disabled.

DELETE realm/workspace

The web method DELETE `deployments/workspace` deletes the workspace, and releases the modification lock.

If you request this operation while *another* user holds the modification lock, this request returns a 403 error response. Nonetheless, you can force this operation:

```
DELETE deployments/workspace?force=true
```

HTTP Parameters

| Syntax | Description |
|-----------------|--|
| force=<boolean> | Optional. When true, break the modification lock, and delete the workspace. |

Example Requests

```
curl -X DELETE http://<host>:<port>/api/v1/deployments/workspace
```

Deployments

Deployments represent and track the state of the realm definition over time.

Life Cycle

Each deployment begins as a workspace, that is, a working copy of the realm definition, which an administrator can modify.

When modifications are complete, the administrator can deploy the workspace definition.

When the deploy transaction completes, the new definition has been stored in the FTL server database, pushed to all clients, and archived in the deployment directory (which preserves the deployment history).

For more information, see these topics:

- [Realm Definition Storage](#)
- [The Deploy Transaction](#)
- [Deployment History](#) and the topics that follow it.

Tasks and Methods

| Task | Method |
|---|---|
| Get a list of deployment objects. | GET realm/deployments |
| Get a specific deployment object. | GET realm/deployments/<name> |
| Deploy the workspace to the realm. | POST realm/deployments |
| Delete a specific deployment from the history list. | DELETE realm/deployments/<name> |
| Re-deploy a specific deployment. | PUT realm/deployments/<name> |

JSON Attributes

POST requests may include JSON data.

For response attributes, see [Deployment History](#).

GET realm/deployments

The web method `GET realm/deployments` retrieves the history list of deployments.

HTTP Parameters

| Syntax | Description |
|---------------------------------|--|
| <code>since=<date></code> | Optional. When present, get only deployments on or after the date. Specify the date in this format: MM-DD-YYYY. |

Example Requests

```
curl -X GET http://<host>:<port>/api/v1/realm/deployments
```

GET realm/deployments/<name>

This web method retrieves a deployment with a specific name.

Example Requests

```
curl -X GET http://<host>:<port>/api/v1/realm/deployments/<name>
```

POST realm/deployments

The web method POST realm/deployments starts a transaction to deploy the workspace.

For more information, see [The Deploy Transaction](#).

JSON Attributes

You may supply JSON data with the request:

```
{
  "name": "MyDeployment",
  "description": "This deployment is not like the others."
}
```

| Attribute | Description |
|-----------|---|
| name | Optional. When present, the FTL server names the deployment with the name string you supply. When absent, the FTL server names the deployment with a time stamp string. |

| Attribute | Description |
|-------------|--|
| description | <p>Optional.</p> <p>When present, the FTL server adds the description string you supply to the deployment object.</p> <p>When absent, the deployment has no description.</p> |

HTTP Parameters

| Syntax | Description |
|-----------------|--|
| force=<boolean> | <p>Optional.</p> <p>When <code>true</code>, force deployment of the workspace, even if the current user does not hold the modification lock.</p> <p>For example, you can force deployment of modifications made by another user who is unavailable to deploy them in person.</p> |
| test=<boolean> | <p>Optional.</p> <p>When <code>true</code>, test performs only the test step of a deployment.</p> |

Example Requests

```
curl -X POST http://<host>:<port>/api/v1/realm/deployments -d <json_data>
```

DELETE realm/deployments/<name>

This web method deletes a specific deployment from the history list.

Example Requests

```
curl -X DELETE http://<host>:<port>/api/v1/realm/deployments/<name>
```

PUT realm/deployments/<name>

This web method finds a deployment by name in the history list, and redeploys it.

That is, this method starts a deploy transaction. However, it uses the realm definition from a previous deployment, rather than from the workspace.

Example Requests

```
curl -X PUT http://<host>:<port>/api/v1/realm/deployments/<name>
```

Definition Objects

The FTL server web API uses definition objects to represent the configuration of the realm.

The following topics detail the various definition objects and their methods.

TIBCO eFTL

Administrators also use the FTL server GUI and web API to configure and monitor the TIBCO eFTL™ service. For information about these GUI pages and REST API, see the TIBCO eFTL documentation.

Application Definition Objects

The FTL server web API represents each application definition, with its endpoints and application instances, as a JSON object.

For more information about applications, see [Implementation: Application \(Macro\)](#).

For detailed information about application definitions, see these topics:

- [Applications Grid](#)
- [Application Definition Details Panel](#)

Example JSON Representation

```
{
  "description": "",
  "endpoints": [{
    "cluster": "Cluster",
    "dynamic_durable": {},
    "name": "tibsend-endpoint",
    "subscribers": [],
    "store": "sstore",
    "subscribers": [],
    "transports": [{
      "name": "dtcp",
      "receive": true,
      "receive_inbox": true,
      "send": true,
      "send_inbox": true
    }],
  }],
  "instances": [{
    "description": "",
    "endpoints": [{
      "dynamic_durable": {},
      "name": "tibsend-endpoint",
      "subscribers": [],
      "transports": [{
        "name": "shm",
        "receive": true,
        "receive_inbox": true,
        "send": true,
        "send_inbox": true
      }],
    }],
    "matcher": {
      "host": "reed",
      "identifier": ""
    },
    "name": "reed"
  }],
  "manage_all_formats": false,
  "name": "tibsend-shared",
}
```

```
"preload_format_names":["Format-1"]
}
```

JSON Attributes of Applications

For attributes and semantics see [Application Definition Details Panel](#).

GET realm/applications

The web method GET realm/applications retrieves a collection of application definitions.

Example Requests

```
curl -X GET http://<host>:<port>/api/v1/realm/applications
```

GET realm/applications/<app_name>

This web method retrieves the definition of an application by its name.

Example Requests

```
curl -X GET http://<host>:<port>/api/v1/realm/applications/<app_name>
```

GET realm/applications/<app_name>/endpoints

This web method retrieves the collection of endpoint objects from an application definition.

Example Requests

```
curl -X GET http://<host>:<port>/api/v1/realm/applications/<app_name>/endpoints
```

GET realm/applications/<app_name>/endpoints/<ep_name>

This web method retrieves a specific endpoint object from an application definition.

- <app> in the URI is the application name.
- <ep_name> in the URI is the endpoint name.

Example Requests

```
curl -X GET http://<host>:<port>/api/v1/realm/applications/<app_name>/endpoints/<ep_name>
```

GET realm/applications/<app_name>/instances

This web method retrieves the collection of application instance objects from an application definition.

Example Requests

```
curl -X GET http://<host>:<port>/api/v1/realm/applications/<app_name>/instances
```

GET realm/applications/<app_name>/instances/<inst_name>

This web method retrieves a specific application instance object from an application definition.

- <app_name> in the URI is the application name.
- <inst_name> in the URI is the application instance name.

Example Requests

```
curl -X GET http://<host>:<port>/api/v1/realm/applications/<app_name>/instances/<inst_name>
```

GET realm/applications/<app_name>/instances/<inst_name>/endpoints

This web method retrieves the instance endpoint configurations of a specific application instance object from an application definition.

- <app_name> in the URI is the application name.
- <inst_name> in the URI is the application instance name.

Example Requests

```
curl -X GET http://<host>:<port>/api/v1/realm/applications/<app_name>/instances/<inst_name>/endpoints
```

GET realm/applications/<app_name>/instances/<inst_name>/endpoints/<ep_name>

This web method retrieves a specific instance endpoint configuration from an application instance object within an application definition.

- <app_name> in the URI is the application name.
- <inst_name> in the URI is the application instance name.
- <ep_name> in the URI is the endpoint name.

Example Requests

```
curl -X GET http://<host>:<port>/api/v1/realm/applications/<app_name>/instances/<inst_name>/endpoints/<ep_name>
```

POST realm/applications

The web method POST realm/applications creates a new application definition.

Input Data

Supply a JSON representation of the definition in the message body.

Example Requests

```
curl -X POST http://<host>:<port>/api/v1/realm/applications -d <json_defn>
```

POST realm/applications/<app_name>/endpoints

This web method creates a new endpoint within an application definition.

Input Data

Supply a JSON representation of the endpoint object in the message body. At minimum, that representation must include the name attribute, specifying the name of the new endpoint.

Example Requests

```
curl -X POST http://<host>:<port>/api/v1/realm/applications/<app_name>/endpoints -d <json_defn>
```

POST realm/applications/<app_name>/instances

This web method creates an application instance within an application definition.

Input Data

Supply a JSON representation of the application instance object in the message body. At minimum, that representation must include the name attribute, specifying the name of the new instance.

Example Requests

```
curl -X POST http://<host>:<port>/api/v1/realm/applications/<app_name>/instances -d <json_defn>
```

DELETE realm/applications/<app_name>

This web method deletes an application definition from the realm.

Example Requests

```
curl -X DELETE http://<host>:<port>/api/v1/realm/applications/<app_name>
```


DELETE realm/applications/<app_name>/endpoints/<ep_name>

This web method deletes an endpoint from an application definition.

- <app_name> in the URI is the application name.
- <ep_name> in the URI is the endpoint name.

After deleting an endpoint from an application, it is good practice to also update the application instances to delete references to that endpoint.

Example Requests

```
curl -X DELETE http://<host>:<port>/api/v1/realm/applications/<app_name>/endpoints/<ep_name>
```

DELETE realm/applications/<app_name>/instances/<inst_name>

This web method deletes an application instance from an application definition.

- <app_name> in the URI is the application name.
- <inst_name> in the URI is the application instance name.

Example Requests

```
curl -X DELETE http://<host>:<port>/api/v1/realm/applications/<app_name>/instances/<inst_name>
```

PUT realm/applications/<app_name>

This web method updates an application definition.

Input Data

Supply a JSON representation of the new definition in the message body.

The new definition completely replaces the old definition in the workspace.

Example Requests

```
curl -X PUT http://<host>:<port>/api/v1/realm/applications/<app_name> -d  
<json_defn>
```

PUT realm/applications/<app_name>/endpoints/<ep_name>

This web method updates an endpoint definition.

- <app_name> in the URI is the application name.
- <ep_name> in the URI is the endpoint name.

Input Data

Supply a JSON representation of the endpoint object in the message body. At minimum, that representation must include the name attribute, specifying the endpoint name.

Example Requests

```
curl -X PUT http://<host>:<port>/api/v1/realm/applications/<app_  
name>/endpoints/<ep_name>
```

PUT realm/applications/<app_name>/instances/<inst_name>

This web method updates an application instance within an application definition.

- <app_name> in the URI is the application name.
- <inst_name> in the URI is the application instance name.

Input Data

Supply a JSON representation of the instance object in the message body. At minimum, that representation must include the name attribute, specifying the instance name.

Example Requests

```
curl -X PUT http://<host>:<port>/api/v1/realm/applications/<app_name>/instances/<inst_name> -d <json_defn>
```

PUT realm/applications/<app_name>/instances/<inst_name>/endpoints/<ep_name>

This web method updates the configuration of a specific instance endpoint within an application instance. You can use this method to modify the transport connectors of the instance endpoint.

- <app_name> in the URI is the application name.
- <inst_name> in the URI is the application instance name.
- <ep_name> in the URI is the endpoint name.

Input Data

Supply a JSON representation of the instance endpoint configuration object in the message body. That representation must include the name attribute, specifying the endpoint name.

Example Requests

```
curl -X PUT http://<host>:<port>/api/v1/realm/applications/<app_name>/instances/<inst_name>/endpoints/<ep_name> -d <json_defn>
```

Transport Definition Objects

The FTL server web API represents each transport definition as a JSON object.

For more information about transports, see [Transport Concepts](#) and its subtopics.

For detailed information about transport definitions, see these topics:

- [Transports Grid](#)
- [Transport Details Panel](#)
- [Transport Protocol Types](#)

Example JSON Representation

```
{
  "config":{
    "backlog_full_wait":"0",
    "backlog_size":"64mb",
    "port":"0",
    "recv_spin_limit":"0.0",
    "subnet_mask":"",
    "transport_type":"dtcp"
  },
  "description":"",
  "name":"dtcp-tport"
}
```

GET realm/transports

The web method GET `realm/transports` retrieves a collection of transport definitions.

Example Requests

```
curl -X GET http://<host>:<port>/api/v1/realm/transports
```

GET realm/transport/<name>

This web method retrieves the definition of a transport by its name.

Example Requests

```
curl -X GET http://<host>:<port>/api/v1/realm/transport/<name>
```

POST realm/transport

The web method POST realm/transport creates a new transport definition.

Input Data

Supply a JSON representation of the definition in the message body.

If you omit the transport protocol, the default protocol is a dynamic TCP mesh.

Example Requests

```
curl -X POST http://<host>:<port>/api/v1/realm/transport -d '{  
  "name": "MyTransport"}'
```

DELETE realm/transport/<name>

This web method deletes a transport definition from the realm.

Example Requests

```
curl -X DELETE http://<host>:<port>/api/v1/realm/transport/<name>
```

PUT realm/transport/<name>

The web method PUT realm/transport/<name> updates a transport definition.

Input Data

Supply a JSON representation of the new definition in the message body.

Example Requests

```
curl -X PUT http://<host>:<port>/api/v1/realm/transport -d '{
  "name": "MyTransport", "port": "7890"}'
```

Persistence Definition Objects

The FTL server web API represents each persistence definition as a JSON object.

For more information about persistence, see [Persistence: Stores and Durables](#).

For detailed information about persistence definitions, see these topics:

- [Stores Grid](#)
- [Store Detail Panel](#)
- [Durable Details Panel](#)
- [Clusters Grid](#)
- [Cluster Details Panel](#)
- [Persistence Service Details Panel](#)

Example JSON Representation

```
{
  "clusters": [{
    "client_pserver_heartbeat": 2.0,
    "client_timeout_pserver": 5.0,
    "description": "",
```

```

    "disk_state": 4,
    "disk_persistence": "async",
    "disk_swap": true,
    "dr_enabled": false,
    "name": "pcluster",
    "primary_set": "_setA",
    "pserver_pserver_heartbeat": 0.5,
    "pserver_timeout_pserver": 3.0,
    "servers": [{
      "description": "",
      "name": "pserver",
      "weight": 10
    }],
    "stores": [{
      "bytelimit": "",
      "description": "",
      "durable_templates": [{
        "ack_settings": {
          "mode": "sync"
        },
        "description": "",
        "name": "durable_template",
        "type": "standard"
      }],
      "durables": [{
        "swap_bytelimit": "200"
      }],
      "name": "pstore",
      "publisher_settings": "store_confirm_send",
      "replicated": true,
      "swap_bytelimit": "-1"
    }],
  }
}

```

GET realm/persistence

The web method GET realm/persistence retrieves a collection of persistence cluster definitions.

Example Requests

```
curl -X GET http://<host>:<port>/api/v1/realm/persistence
```

GET realm/persistence/<clus_name>

This web method retrieves the definition of a persistence cluster by its name.

Example Requests

```
curl -X GET http://<host>:<port>/api/v1/realm/persistence/<clus_name>
```

GET realm/persistence/<clus_name>/servers

This web method retrieves a collection of service objects from a persistence cluster definition.

<clus_name> in the URI is the name of the persistence cluster.

Example Requests

```
curl -X GET http://<host>:<port>/api/v1/realm/persistence/<clus_name>/servers
```

GET realm/persistence/<clus_name>/servers/<svc_name>

This web method retrieves a specific service object from a persistence cluster definition.

- <clus_name> in the URI is the cluster name.
- <svc_name> in the URI is the service name.

Example Requests

```
curl -X GET http://<host>:<port>/api/v1/realm/persistence/<clus_name>/servers/<svc_name>
```

GET realm/persistence/<clus_name>/sets

This web method retrieves a collection of service set definitions from a persistence cluster definition.

<clus_name> in the URI is the name of the persistence cluster.

Example Requests

```
curl -X GET http://<host>:<port>/api/v1/realm/persistence/<clus_name>/sets
```

GET realm/persistence/<clus_name>/sets/<set_name>

This web method retrieves a specific service set object from a persistence cluster definition.

- <clus_name> in the URI is the cluster name.
- <set_name> in the URI is the set name.

Example Requests

```
curl -X GET http://<host>:<port>/api/v1/realm/persistence/<clus_name>/sets/<set_name>
```

GET realm/persistence/<clus_name>/sets/<set_name>/servers

This web method retrieves a collection of service objects from a specific service set in a persistence cluster definition.

- <clus_name> in the URI is the name of the persistence cluster.
- <set_name> in the URI is the name of the service set.

Example Requests

```
curl -X GET http://<host>:<port>/api/v1/realm/persistence/<clus_name>/sets/<set_name>/servers
```

GET realm/persistence/<clus_name>/sets/<set_name>/servers/<svc_name>

This web method retrieves a specific service object from a specific service set in a persistence cluster definition.

- <clus_name> in the URI is the cluster name.
- <set_name> in the URI is the name of the service set.
- <svc_name> in the URI is the service name.

Example Requests

```
curl -X GET http://<host>:<port>/api/v1/realm/persistence/<clus_name>/sets/<set_name>/servers/<svc_name>
```

GET realm/persistence/<clus_name>/stores

This web method retrieves a collection of store objects from a persistence cluster definition.

<clus_name> in the URI is the cluster name.

Example Requests

```
curl -X GET http://<host>:<port>/api/v1/realm/persistence/<clus_name>/stores
```

GET realm/persistence/<clus_name>/stores/<stor_name>

This web method retrieves a specific store object from a persistence cluster definition.

- <clus_name> in the URI is the cluster name.
- <stor_name> in the URI is the store name.

Example Requests

```
curl -X GET http://<host>:<port>/api/v1/realm/persistence/<clus_name>/stores/<stor_name>
```

GET realm/persistence/<clus_name>/stores/<stor_name>/durables

This web method retrieves the static durables defined for a specific store object within a persistence cluster definition.

- <clus_name> in the URI is the cluster name.
- <stor_name> in the URI is the store name.

Example Requests

```
curl -X GET http://<host>:<port>/api/v1/realm/persistence/<clus_name>/stores/<stor_name>/durables
```

GET realm/persistence/<clus_name>/stores/<stor_name>/durables/<dur_name>

This web method retrieves a specific static durable definition from a store object in a persistence cluster definition.

- <clus_name> in the URI is the cluster name.
- <stor_name> in the URI is the store name.
- <dur_name> in the URI is the durable name.

Example Requests

```
curl -X GET http://<host>:<port>/api/v1/realm/persistence/<clus_name>/stores/<stor_name>/durables/<dur_name>
```

GET realm/persistence/<clus_name>/stores/<stor_name>/templates

This web method retrieves the dynamic durable templates defined for a specific store object in a persistence cluster definition.

- <clus_name> in the URI is the cluster name.
- <stor_name> in the URI is the store name.

Example Requests

```
curl -X GET http://<host>:<port>/api/v1/realm/persistence/<clus_name>/stores/<stor_name>/templates
```

GET realm/persistence/<clus_name>/stores/<stor_name>/templates/<tem_name>

This web method retrieves a specific dynamic durable template from a store object in a persistence cluster definition.

- <clus_name> in the URI is the cluster name.
- <stor_name> in the URI is the store name.
- <tem_name> in the URI is the template name.

Example Requests

```
curl -X GET http://<host>:<port>/api/v1/realm/persistence/<clus_name>/stores/<stor_name>/durables/<tem_name>
```

GET realm/persistence/<clus_name>/acl

This web method retrieves a specific cluster object's list of permissions from a persistence cluster definition.

- <clus_name> in the URI is the cluster name.

JSON Example

```
{
  "users": [
    {
      "name": "user1",
```

```

    "permissions": [
      "lock"
    ]
  },
  {
    "name": "user2",
    "permissions": []
  }
],
"roles": [
  {
    "name": "role1",
    "permissions": [
      "lock"
    ]
  },
  {
    "name": "role2",
    "permissions": []
  }
]
}

```

Example Requests

```
curl -X GET http://<host>:<port>/api/v1/realm/persistence/<cluster>/acl
```

GET realm/persistence/<clus_name>/acl/roles

This web method retrieves a specific cluster object's list of permissions by role.

- <clus_name> in the URI is the cluster name.

Example Requests

```
curl -X GET http://<host>:<port>/api/v1/realm/persistence/<clus_
name>/acl/roles
```

GET realm/persistence/<clus_name>/acl/roles/<role_name>

This web method retrieves a specific role from a cluster object's list of permissions granted to a specific role.

- <clus_name> in the URI is the cluster name.
- <role_name> in the URI is the role name with associated permissions.

Example Requests

```
curl -X GET http://<host>:<port>/api/v1/realm/persistence/<clus_name>/acl/roles/<role_name>
```

GET realm/persistence/<clus_name>/acl/users

This web method retrieves a specific cluster object's list of permissions by user.

- <clus_name> in the URI is the cluster name.

Example Requests

```
curl -X GET http://<host>:<port>/api/v1/realm/persistence/<clus_name>/acl/users
```

GET realm/persistence/<clus_name>/acl/users/<user_name>

This web method retrieves a specific cluster object's user based on the list of permissions granted to a specific user.

- <clus_name> in the URI is the cluster name.
- <user_name> in the URI is the username with associated permissions.

Example Requests

```
curl -X GET http://<host>:<port>/api/v1/realm/persistence/<clus_name>/acl/users/<user_name>
```

GET realm/persistence/<clus_name>/stores/<stor_name>/acl

This web method retrieves a specific store object's acl from a persistence cluster definition.

- <clus_name> in the URI is the cluster name.
- <store_name> in the URI is the store name.

Example Requests

```
curl -X GET http://<host>:<port>/api/v1/realm/persistence/<clus_name>/stores/<stor_name>/acl
```

GET realm/persistence/<clus_name>/stores/<stor_name>/acl/roles

This web method retrieves the roles of a specific store acl in a persistence cluster definition.

- <clus_name> in the URI is the cluster name.
- <stor_name> in the URI is the store name.

Example Requests

```
curl -X GET http://<host>:<port>/api/v1/realm/persistence/<clus_name>/stores/<stor_name>/acl/roles
```


GET realm/persistence/<clus_name>/stores/<stor_name>/acl/roles/<role_name>

This web method retrieves the role of a specific store acl roles definition from a persistence cluster.

- <clus_name> in the URI is the cluster name.
- <stor_name> in the URI is the store name.
- <role_name> in the URI is the role name with associated permissions.

Example Requests

```
curl -X GET http://<host>:<port>/api/v1/realm/persistence/<clus_name>/stores/<stor_name>/acl/roles/<role_name>
```

GET realm/persistence/<clus_name>/stores/<stor_name>/acl/users

This web method retrieves the users of a specific store acl in a persistence cluster definition.

- <clus_name> in the URI is the cluster name.
- <stor_name> in the URI is the store name.

Example Requests

```
curl -X GET http://<host>:<port>/api/v1/realm/persistence/clus_name/stores/<stor_name>/acl/users
```

GET realm/persistence/clus_name/stores/<stor_name>/acl/users/<user_name>

This web method retrieves a specific user of a specific store acl users definition in a persistence cluster.

- <clus_name> in the URI is the cluster name.
- <stor_name> in the URI is the store name.
- <user_name> in the URI is the username with associated permissions.

Example Requests

```
curl -X GET http://<host>:<port>/api/v1/realm/persistence/clus_name/stores/<stor_name>/acl/users/<user_name>
```

POST realm/persistence

The web method POST realm/persistence creates a new persistence cluster definition.

Input Data

Supply a JSON representation of the definition in the message body.

Example Requests

```
curl -X POST http://<host>:<port>/api/v1/realm/persistence -d <json_defn>
```

POST realm/persistence/<clus_name>/servers

This web method creates a new persistence service within a persistence cluster definition.

<clus_name> in the URI is the name of the persistence cluster.

Input Data

Supply a JSON representation of the service definition in the message body.

Example Requests

```
curl -X POST http://<host>:<port>/api/v1/realm/persistence/<clus_
name>/servers -d <json_defn>
```

POST realm/persistence/<clus_name>/sets

The web method POST realm/persistence/<clus_name>/sets adds a new service set within a persistence cluster definition.

<clus_name> in the URI is the name of the persistence cluster.

Input Data

Supply a JSON representation of the service set definition in the message body. At minimum, supply the set name:

```
{"name": "<my_set_name>"}
```

Example Requests

```
curl -X POST http://<host>:<port>/api/v1/realm/persistence/<clus_
name>/sets -d <json_defn>
```

POST realm/persistence/<clus_name>/sets/<set_name>/servers

This web method creates a new persistence service within a specific service set of a persistence cluster definition.

- <clus_name> in the URI is the name of the persistence cluster.
- <set_name> in the URI is the name of the service set.

Input Data

Supply a JSON representation of the service definition in the message body.

Example Requests

```
curl -X POST http://<host>:<port>/api/v1/realm/persistence/<clus_name>/sets/<set_name>/servers -d <json_defn>
```

POST realm/persistence/<clus_name>/stores

This web method creates a new persistence store within a persistence cluster definition.

<clus_name> in the URI is the name of the persistence cluster.

Input Data

Supply a JSON representation of the server definition in the message body.

Example Requests

```
curl -X POST http://<host>:<port>/api/v1/realm/persistence/<clus_name>/stores -d <json_defn>
```

POST realm/persistence/<clus_name>/stores/<stor_name>/durables

This web method defines one new durable in a persistence store within a persistence cluster definition.

- <clus_name> in the URI is the cluster name.
- <stor_name> in the URI is the store name.

Input Data

Supply a JSON representation of the new durables definition in the message body.

Example Requests

```
curl -X POST http://<host>:<port>/api/v1/realm/persistence/<clus_name>/stores/<stor_name>/durables -d <json_defn>
```

POST realm/persistence/<clus_name>/stores/<stor_name>/templates

This web method defines one new dynamic durable template in a persistence store within a persistence cluster definition.

- <clus_name> in the URI is the cluster name.
- <stor_name> in the URI is the store name.

Input Data

Supply a JSON representation of the new dynamic durable template definition in the message body.

Example Requests

```
curl -X POST http://<host>:<port>/api/v1/realm/persistence/<clus_name>/stores/<stor_name>/templates -d <json_defn>
```

POST realm/persistence/<clus_name>/acl/roles

The web method POST realm/persistence/clus_name/acl/roles grants permissions to a specific role.

- <clus_name> in the URI is the cluster name.

JSON Example

```
{
  "name": "role1",
  "permissions": [
    "lock"
  ]
}
```

Example Requests

```
curl -X POST http://<host>:<port>/api/v1/realm/persistence/<clus_
name>/acl/roles -d <json_defn>
```

POST realm/persistence/<clus_name>/acl/users

This web method grants permissions to a specific role.

- <clus_name> in the URI is the cluster name.

Example Requests

```
curl -X POST http://<host>:<port>/api/v1/realm/persistence/<clus_
name>/acl/users -d <json_defn>
```

POST realm/persistence/<clus_name>/stores/<stor_name>/acl/roles

The web method POST realm/persistence/<clus_name>/stores/<stor_name>/acl/roles defines the roles of a specific store list of permission from a persistence cluster definition.

- *clus_name* in the URI is the cluster name.
- <stor_name> in the URI is the store name.

Example Requests

```
curl -X POST http://<host>:<port>/api/v1/realm/persistence/<clus_name>/stores/<stor_name>/acl/roles -d <json_defn>
```

POST realm/persistence/<clus_name>/stores/<stor_name>/acl/users

This web method defines the users of a specific store list of permissions by role in a persistence cluster definition.

- <clus_name> in the URI is the cluster name.
- <stor_name> in the URI is the store name.

Example Requests

```
curl -X POST http://<host>:<port>/api/v1/realm/persistence/<clus_name>/stores/<stor_name>/acl/users -d <json_defn>
```

DELETE realm/persistence/<clus_name>

The web method DELETE realm/persistence/<clus_name> deletes a persistence cluster definition from the realm.

<clus_name> in the URI is the name of the persistence cluster.

Example Requests

```
curl -X DELETE http://<host>:<port>/api/v1/realm/persistence/<clus_name>
```

DELETE realm/persistence/<clus_name>/servers/<svc_name>

This web method deletes a persistence service from a persistence cluster definition.

- <clus_name> in the URI is the cluster name.
- <svc_name> in the URI is the service name.

Example Requests

```
curl -X DELETE http://<host>:<port>/api/v1/realm/persistence/<clus_name>/servers/<svc_name>
```

DELETE realm/persistence/<clus_name>/sets/<set_name>

This web method deletes a service set from a persistence cluster definition.

- <clus_name> in the URI is the cluster name.
- <set_name> in the URI is the set name.

Example Requests

```
curl -X DELETE http://<host>:<port>/api/v1/realm/persistence/<clus_name>/sets/<set_name>
```


DELETE realm/persistence/<clus_name>/sets/<set_name>/servers/<svc_name>

This web method deletes a persistence service from a specific service set in a persistence cluster definition.

- <clus_name> in the URI is the cluster name.
- <set_name> in the URI is the name of the service set.
- <svc_name> in the URI is the service name.

Example Requests

```
curl -X DELETE http://<host>:<port>/api/v1/realm/persistence/<clus_name>/sets/<set_name>/servers/<svc_name>
```

DELETE realm/persistence/<clus_name>/stores/<stor_name>

This web method deletes a persistence store from a persistence cluster definition.

- <clus_name> in the URI is the cluster name.
- <stor_name> in the URI is the store name.

After deleting a store, it is good practice to also update application endpoints to delete reference to that store.

Example Requests

```
curl -X DELETE http://<host>:<port>/api/v1/realm/persistence/<clus_name>/stores/<stor_name>
```

DELETE realm/persistence/<clus_name>/stores/<stor_name>/durables/<dur_name>

This web method deletes a specific static durable from a persistence store within a persistence cluster definition.

- <clus_name> in the URI is the cluster name.
- <stor_name> in the URI is the store name.
- <dur_name> in the URI is the durable name.

Example Requests

```
curl -X DELETE http://<host>:<port>/api/v1/realm/persistence/<clus_name>/stores/<stor_name>/durables/<dur_name>
```

DELETE realm/persistence/<clus_name>/stores/<stor_name>/templates/<tem_name>

This web method deletes a specific dynamic durable template definition from a persistence store within a persistence cluster definition.

- <clus_name> in the URI is the cluster name.
- <stor_name> in the URI is the store name.
- <tem_name> in the URI is the template name.

Example Requests

```
curl -X DELETE http://<host>:<port>/api/v1/realm/persistence/<clus_name>/stores/<stor_name>/templates/<tem_name>
```

DELETE realm/persistence/<clus_name>/acl/roles/<role_name>

This web method deletes a role and all associated permissions from the list of permissions.

- <clus_name> in the URI is the cluster name.
- <role_name> in the URI is the role name with associated permissions.

Example Requests

```
curl -X DELETE http://<host>:<port>/api/v1/realm/persistence/clus_name/acl/roles/<role_name>
```

DELETE realm/persistence/<clus_name>/acl/users/<user_name>

This web method deletes a user and all associated permissions from the list of permissions.

- <clus_name> in the URI is the cluster name.
- <user_name> in the URI is the username with associated permissions.

Example Requests

```
curl -X DELETE http://<host>:<port>/api/v1/realm/persistence/<clus_name>/acl/users/<user_name>
```

DELETE realm/persistence/<clus_name>/stores/<stor_name>/acl/roles/<role_name>

This web method deletes the permissions associated with a specific role.

- <clus_name> in the URI is the cluster name.

- <stor_name> in the URI is the store name.
- <role_name> in the URI is the role name with associated permissions.

Example Requests

```
curl -X DELETE http://<host>:<port>/api/v1/realm/persistence/<clus_
name>/stores/<stor_name>/acl/roles/<role_name>
```

DELETE realm/persistence/<clus_name>/stores/<stor_name>/acl/users/<user_name>

This web method deletes the permissions associated with a specific user.

- <clus_name> in the URI is the cluster name.
- <stor_name> in the URI is the store name.
- <user_name> in the URI is the username with associated permissions.

Example Requests

```
curl -X DELETE http://<host>:<port>/api/v1/realm/persistence/<clus_
name>/stores/<stor_name>/acl/users/<user_name>
```

PUT realm/persistence/<clus_name>

This web method updates a persistence cluster definition.

Input Data

Supply a JSON representation of the new definition in the message body.

Example Requests

```
curl -X PUT http://<host>:<port>/api/v1/realm/persistence/<name> -d  
<json_defn>
```

PUT realm/persistence/<clus_name>/servers/<svc_name>

This web method updates a persistence service definition within a persistence cluster definition.

- <clus_name> in the URI is the cluster name.
- <svc_name> in the URI is the service name.

Input Data

Supply a JSON representation of the new service definition in the message body.

Example Requests

```
curl -X PUT http://<host>:<port>/api/v1/realm/persistence/<clus_  
name>/servers/<svc_name> -d <json_defn>
```

PUT realm/persistence/<clus_name>/sets/<set_name>

This web method updates a persistence set definition within a persistence cluster definition.

- <clus_name> in the URI is the cluster name.
- <set_name> in the URI is the set name.

Input Data

Supply a JSON representation of the new service set definition in the message body.

Example Requests

```
curl -X PUT http://<host>:<port>/api/v1/realm/persistence/<clus_name>/sets/<set_name> -d <json_defn>
```

PUT realm/persistence/<clus_name>/sets/<set_name>/servers/<svc_name>

This web method updates a persistence service definition within a specific service set of persistence cluster definition.

- <clus_name> in the URI is the cluster name.
- <set_name> in the URI is the name of the service set.
- <svc_name> in the URI is the service name.

Input Data

Supply a JSON representation of the new service definition in the message body.

Example Requests

```
curl -X PUT http://<host>:<port>/api/v1/realm/persistence/<clus_name>/sets/<set_name>/servers/<svc_name> -d <json_defn>
```

PUT realm/persistence/<clus_name>/stores/<stor_name>

This web method updates a persistence store definition within a persistence cluster definition.

- <clus_name> in the URI is the cluster name.
- <stor_name> in the URI is the store name.

Input Data

Supply a JSON representation of the new store definition in the message body.

Example Requests

```
curl -X PUT http://<host>:<port>/api/v1/realm/persistence/<clus_name>/stores/<stor_name> -d <json_defn>
```

PUT realm/persistence/<clus_name>/stores/<stor_name>/durables/<dur_name>

This web method updates a static durable definition in a store object within a persistence cluster definition.

- <clus_name> in the URI is the cluster name.
- <stor_name> in the URI is the store name.
- <dur_name> in the URI is the durable name.

Input Data

Supply a JSON representation of the new durable definition in the message body.

Example Requests

```
curl -X PUT http://<host>:<port>/api/v1/realm/persistence/<clus_name>/stores/<stor_name>/durables/<dur_name> -d <json_defn>
```

PUT realm/persistence/<clus_name>/stores/<stor_name>/templates/<tem_name>

This web method updates a dynamic durable template definition in a store object within a persistence cluster definition.

- <clus_name> in the URI is the cluster name.
- <stor_name> in the URI is the store name.
- <tem_name> in the URI is the template name.

Input Data

Supply a JSON representation of the new dynamic durable template definition in the message body.

Example Requests

```
curl -X PUT http://<host>:<port>/api/v1/realm/persistence/<clus_name>/stores/<stor_name>/templates/<tem_name> -d <json_defn>
```

PUT realm/persistence/<clus_name>/acl

This web method updates a specific cluster object's acl from a persistence cluster definition.

- <clus_name> in the URI is the cluster name.

Example Requests

```
curl -X PUT http://<host>:<port>/api/v1/realm/persistence/<clus_name>/acl -d <json_defn>
```

PUT realm/persistence/<clus_name>/acl/roles/<role_name>

This web method updates a specific role from a cluster object's acl roles.

- <clus_name> in the URI is the cluster name.
- <role_name> in the URI is the role name with associated permissions.

Example Requests

```
curl -X PUT http://<host>:<port>/api/v1/realm/persistence/<clus_name>/acl/roles/<role_name> -d <json_defn>
```

PUT realm/persistence/<clus_name>/acl/users/<user_name>

This web method updates a specific cluster object's user based on the acl and users.

- <clus_name> in the URI is the cluster name.
- <user_name> in the URI is the username with associated permissions.

Example Requests

```
curl -X PUT http://<host>:<port>/api/v1/realm/persistence/<clus_name>/acl/users/<user_name> -d <json_defn>
```

PUT realm/persistence/<clus_name>/stores/<stor_name>/acl

The web method PUT realm/persistence/<clus_name>/stores/<stor_name>/acl updates a specific store object's acl in a specified cluster.

- <clus_name> in the URI is the cluster name.
- <stor_name> in the URI is the store name.

Example Requests

```
curl -X PUT http://<host>:<port>/api/v1/realm/persistence/<clus_name>/stores/<stor_name>/acl -d <json_defn>
```

PUT realm/persistence/<clus_name>/stores/<stor_name>/acl/roles/<role_name>

This web method updates the permissions associated with a specific role.

- <clus_name> in the URI is the cluster name.
- <stor_name> in the URI is the store name.
- <role_name> in the URI is the role name with associated permissions.

Example Requests

```
curl -X PUT http://<host>:<port>/api/v1/realm/persistence/<clus_name>/stores/<stor_name>/acl/roles/<role_name> -d <json_defn>
```

PUT realm/persistence/<clus_name>/stores/<stor_name>/acl/users/<user_name>

This web method updates the permissions associated with a specific user.

- <clus_name> in the URI is the cluster name.
- <stor_name> in the URI is the store name.
- <user_name> in the URI is the username with associated permissions.

Example Requests

```
curl -X PUT<host>:<port>/api/v1/realm/persistence/<clus_name>/stores/<stor_name>/acl/users/<user_name> -d <json_defn>
```

GET realm/zones

The web method GET realm/zones retrieves a collection of zone definitions.

Example Requests

```
curl -X GET http://<host>:<port>/api/v1/realm/zones
```

Example JSON Representation

```
{
  "zones": [
    {
      "last_modified": "2019-03-12T16:44:42.413025708-05:00",
      "last_modified_millis": 1552427082413,
      "last_modified_by": "anyone",
      "name": "z1",
      "description": "",
      "active": true,
      "type": "hub_spoke",
      "clusters": [
        "c1"
      ],
      "settings": {
        "hub": "c1",
        "hub-settings": {
          "prefetch": 1024,
          "message_ttl": "0",
          "batch_count": 1000
        },
        "spoke-settings": {
          "prefetch": 1024,
          "message_ttl": "0",
          "batch_count": 1000
        }
      }
    },
    {
      "last_modified": "2019-03-12T16:44:49.986076891-05:00",
      "last_modified_millis": 1552427089986,
      "last_modified_by": "anyone",
      "name": "z2",
      "description": "",
      "active": false,
      "type": "hub_spoke",
      "clusters": [
        "c1"
      ],
    },
  ]
}
```

```

    "settings": {
      "hub": "c1",
      "hub-settings": {
        "prefetch": 32,
        "message_ttl": "0",
        "batch_count": 1000
      },
      "spoke-settings": {
        "prefetch": 1024,
        "message_ttl": "0",
        "batch_count": 1000
      }
    }
  }
]
}

```

GET realm/zones/<zone_name>

The web method GET realm/zones/<zone_name> retrieves the definition of a zone by its name.

Example Requests

```
curl -X GET http://<host>:<port>/api/v1/realm/zones/<zone_name>
```

Example JSON Representation

```

{
  "last_modified": "2019-03-12T16:44:42.413025708-05:00",
  "last_modified_millis": 1552427082413,
  "last_modified_by": "anyone",
  "name": "z1",
  "description": "",
  "active": true,
  "type": "hub_spoke",
  "clusters": [
    "c1"
  ]
}

```

```

    ],
    "settings": {
      "hub": "c1",
      "hub-settings": {
        "prefetch": 1024,
        "message_ttl": "0",
        "batch_count": 1000
      },
      "spoke-settings": {
        "prefetch": 1024,
        "message_ttl": "0",
        "batch_count": 1000
      }
    }
  }
}

```

GET realm/zones/<zone_name>/stores/<stor_name>/acl

This web method retrieves a specific store object's acl from a zone.

- <zone_name> in the URI is the zone name.
- <stor_name> in the URI is the store name.

Example Requests

```
curl -X GET http://<host>:<port>/api/v1/realm/zones/<zone_name>/stores/<stor_name>/acl
```

GET realm/zones/<zone_name>/stores/<stor_name>/acl/roles

This web method retrieves the roles of a specific store acl definition in a zone.

- <zone_name> in the URI is the zone name.
- <stor_name> in the URI is the store name.

Example Requests

```
curl -X GET http://<host>:<port>/api/v1/realm/zones/<zone_name>/stores/<stor_name>/acl/roles
```

GET realm/zones/<zone_name>/stores/<stor_name>/acl/roles/<role_name>

This web method retrieves a specific role for a store acl roles definition in a zone.

- <zone_name> in the URI is the zone name.
- <stor_name> in the URI is the store name.
- <role_name> in the URI is the role name with associated permissions.

Example Requests

```
curl -X GET http://<host>:<port>/api/v1/realm/zones/<zone_name>/stores/<stor_name>/acl/users
```

GET realm/zones/<zone_name>/stores/<stor_name>/acl/users

This web method retrieves a specific store's acl users from a zone.

- <zone_name> in the URI is the zone name.
- <stor_name> in the URI is the store name.

Example Requests

```
curl -X GET http://<host>:<port>/api/v1/realm/zones/<zone_name>/stores/<stor_name>/acl/users
```

GET realm/zones/<zone_name>/stores/<stor_name>/acl/users/<user_name>

This web method retrieves a specific user from a specific store's acl users in a zone.

- <zone_name> in the URI is the zone name.
- <stor_name> in the URI is the store name.
- <user_name> in the URI is the username with associated permissions.

Example Requests

```
curl -X GET http://<host>:<port>/api/v1/realm/zones/<zone_name>/stores/<stor_name>/acl/users/<user_name>
```

POST realm/zones

The web method POST realm/zones defines a new persistence zone, and adds it to the existing zone definitions.

Input Data

Supply a JSON representation of a zone definition in the message body.

Example Requests

```
curl -X POST http://<host>:<port>/api/v1/realm/zones -d <json_defn>
```

Example JSON Representation

Form the <json_defn> argument like this example.

```
{  
  "name": "Zone-A",
```

```

    "description": "",
    "active": true,
    "clusters": ["c1", "c2", "c3"],
    "type": "hub_spoke",
    "settings":
    {
        "hub": "c3",
        "hub-settings":
        {
            "prefetch": 10,
            "message_ttl": "10",
            "batch_count": 10
        },
        "spoke-settings":
        {
            "prefetch": 5,
            "message_ttl": "5",
            "batch_count": 5,
            "max_delivery": "5"
        }
    }
}

```

POST realm/zones/<zone_name>/stores/<stor_name>

The web method POST realm/zones/<zone>_name/stores/<stor_name> defines a specific store in a zone.

- <zone>_name in the URI is the zone name.
- <store>_name in the URI is the store name.

Example Requests

```

curl -X POST http://<host>:<port>/api/v1/realm/zones/<zone_
name>/stores/<stor_name> -d '{"name": "st1", "zones": ["zone1"]}'

```


POST realm/zones/<zone_name>/stores/<stor_name>/acl/roles

This web method defines a specific store acl's roles in a zone.

- <zone_name> in the URI is the zone name.
- <stor_name> in the URI is the store name.

Example Requests

```
curl -X POST http://<host>:<port>/api/v1/realm/zones/<zone_name>/stores/<stor_name>/acl/roles -d <json_defn>
```

POST realm/zones/<zone_name>/stores/<stor_name>/acl/users

This web method defines a specific store's acl users for a zone.

- <zone_name> in the URI is the zone name.
- <stor_name> in the URI is the store name.

Example Requests

```
curl -X POST http://<host>:<port>/api/v1/realm/zones/<zone_name>/stores/<stor_name>/acl/users -d <json_defn>
```

POST realm/zones/<zone_name>

The web method PUT realm/zones/<zone_name> updates a zone definition, replacing an existing definition for the zone name.

<zone_name> in the URI is the zone name.

Input Data

Supply a JSON representation of the new definition in the message body.

Example Requests

```
curl -X PUT http://<host>:<port>/api/v1/realm/zones/<zone_name> -d  
<json_defn>
```

PUT realm/zones/<zone_name>/stores/<stor_name>/acl

This web method updates a specific store object's acl in a specified zone.

- <zone_name> in the URI is the zone name.
- <stor_name> in the URI is the store name.

Example Requests

```
curl -X PUT http://<host>:<port>/api/v1/realm/zones/<zone_  
name>/stores/<stor_name>/acl -d <json_defn>
```

PUT realm/zones/<zone_name>/stores/<stor_name>/acl/roles/<role_name>

This web method updates a specific role for a store acl roles definition in a zone.

- <zone_name> in the URI is the zone name.
- <stor_name> in the URI is the store name.
- <role_name> in the URI is the role name with associated permissions.

Example Requests

```
curl -X PUT http://<host>:<port>/api/v1/realm/zones/<zone_name>/stores/<stor_name>/acl/roles/<role_name> -d <json_defn>
```

PUT realm/zones/<zone_name>/stores/<stor_name>/acl/users/<user_name>

This web method updates a specific user from a specific store's acl users in a zone.

- <zone_name> in the URI is the zone name.
- <stor_name> in the URI is the store name.
- <user_name> in the URI is the username with associated permissions.

Example Requests

```
curl -X PUT http://<host>:<port>/api/v1/realm/zones/<zone_name>/stores/<stor_name>/acl/users/<user_name> -d <json_defn>
```

DELETE realm/zones/<zone_name>

This web method deletes a zone definition from the realm.

- <zone_name> in the URI is the zone name.

Example Requests

```
curl -X DELETE http://<host>:<port>/api/v1/realm/zones/<zone_name>
```

DELETE realm/zones/<zone_name>/stores/<stor_name>/acl/roles/<role_name>

This web method deletes a specific role for a store acl roles definition in a zone.

- <zone_name> in the URI is the zone name.
- <stor_name> in the URI is the store name.
- <role_name> in the URI is the role name with associated permissions.

Example Requests

```
curl -X DELETE http://<host>:<port>/api/v1/realm/zones/<zone>_
name/stores/<stor_name>/acl/roles/<role_name>
```

DELETE realm/zones/<zone_name>/stores/<stor_name>/acl/users/<user_name>

This web method deletes a specific user from a specific store's acl users in a zone.

- <zone_name> in the URI is the zone name.
- <stor_name> in the URI is the store name.
- <user_name> in the URI is the username with associated permissions.

Example Requests

```
curl -X DELETE http://<host>:<port>/api/v1/realm/zones/<zone>_
name/stores/<stor_name>/acl/users/<user_name>
```

Bridge Definition Objects

The FTL server web API represents each bridge definition as a JSON object.

For more information about bridges, see [Transport Bridge](#).

For detailed information about bridge definitions, see [Bridges Grid](#).

Example JSON Representation

```
{
  "bridges": [{
    "description": "",
    "name": "my_bridge",
    "sets": [
      {
        "transports": ["shm-requestreply-tport"]
      },
      {
        "transports": [
          "dtcp-tport",
          "stcp-tport"
        ]
      }
    ]
  }]
}
```

GET realm/bridges

This web method retrieves a collection of bridge definitions.

Example Requests

```
curl -X GET http://<host>:<port>/api/v1/realm/bridges
```

GET realm/bridges/<br_name>

This web method retrieves the definition of a bridge by its name.

Example Requests

```
curl -X GET http://<host>:<port>/api/v1/realm/bridges/<br_name>
```

POST realm/bridges

This web method creates a new bridge definition.

Input Data

Supply a JSON representation of the definition in the message body.

Example Requests

```
curl -X POST http://<host>:<port>/api/v1/realm/bridges -d <json_defn>
```

DELETE realm/bridges/<br_name>

This web method deletes a bridge definition from the realm.

Example Requests

```
curl -X DELETE http://<host>:<port>/api/v1/realm/bridges/<br_name>
```

PUT realm/bridges/<br_name>

This web method updates a bridge definition.

Input Data

Supply a JSON representation of the new definition in the message body.

Example Requests

```
curl -X PUT http://<host>:<port>/api/v1/realm/bridges/<br_name> -d  
<json_defn>
```

Format Definition Objects

The FTL server web API represents each format definition as a JSON object.

For more information about formats, see [Formats](#).

For detailed information about format definitions, see [Formats Grid](#).

Example JSON Representation

```
{  
  "description": "",  
  "fields": [  
    {  
      "name": "FullName",  
      "type": "string"  
    },  
    {  
      "name": "MemberID",  
      "type": "long"  
    },  
    {  
      "name": "Points",  
      "type": "long"  
    }  
  ],  
  "name": "LoyaltyPoints"  
}
```

GET realm/formats

This web method retrieves a collection of format definitions.

Example Requests

```
curl -X GET http://<host>:<port>/api/v1/realm/formats
```

GET realm/formats/<name>

This web method retrieves the definition of a format by its name.

Example Requests

```
curl -X GET http://<host>:<port>/api/v1/realm/formats/<name>
```

POST realm/formats

This web method creates a new format definition.

Input Data

Supply a JSON representation of the definition in the message body.

Example Requests

```
curl -X POST http://<host>:<port>/api/v1/realm/formats -d <json_defn>
```

Example Definitions

```
POST realm/formats
{"name": "MyFormat",
 "fields":
  [{"name": "MyField",
   "type": "String"}]}
```



```
]
}
```

DELETE realm/formats/<name>

This web method deletes a format definition from the realm.

Example Requests

```
curl -X DELETE http://<host>:<port>/api/v1/realm/formats/<name>
```

PUT realm/formats/<name>

This web method updates a format definition.

Input Data

Supply a JSON representation of the new definition in the message body.

Example Requests

```
curl -X PUT http://<host>:<port>/api/v1/realm/formats/<name> -d <json_
defn>
```

Realm Definition and Properties

The FTL server web API represents the realm definition and the set of global realm properties as JSON objects. You can use the FTL server web API to retrieve and update the entire realm definition, and to access the global realm properties.

Example JSON Representation of Global Realm Properties

```
{
  "client_monitor_sample_interval": 60,
  "client_server_heartbeat": 60,
  "client_timeout_server": 180,
  "manage_all_formats": false,
  "realm_is_secure": false,
  "server_client_heartbeat": 60,
  "server_timeout_client": 3600,
  "max_deployment_history": 25,
  "com.tibco.ftl.client.publisher.persistence.retry.duration": -1,
  "com.tibco.ftl.client.subscriber.persistence.retry.duration": -1,
  "com.tibco.ftl.client.map.persistence.retry.duration": -1,
  "com.tibco.ftl.client.publisher.send.policy": 1,
  "com.tibco.ftl.client.publisher.persistence.send.policy": 0,
  "service_connection_policy": "default",
  "use_endpoint_store_for_inbox": true,
  "enable_permissions": false
}
```

GET realm

This web method retrieves the complete realm definition.

Example Requests

```
curl -X GET http://<host>:<port>/api/v1/realm
```

POST realm

This web method replaces the existing realm definition with a new definition.

Modification Lock

Behavior depends on state of the modification lock.

- If the lock is not held by any user, you can use this request. Forcing is not needed.

- If you hold the lock, this request is allowed. It releases the modification lock, deletes your workspace, and updates the realm definition.
- If another user holds the lock, this request is not allowed.

However, you can force this request. Forcing breaks the modification lock, deletes the other user's workspace, and updates the realm definition.

Input Data

Supply a JSON representation of the definition in the message body.

HTTP Parameters

| Syntax | Description |
|-----------------|--|
| force=<boolean> | Optional. When <code>true</code> , force an update to the new realm definition, breaking the modification lock. |

Example Requests

```
curl -X POST http://<host>:<port>/api/v1/realm -d <json_defn>
```

```
curl -X POST http://<host>:<port>/api/v1/realm?force=true -d <json_defn>
```

GET realm/properties

This web method retrieves the global properties of the realm.

Properties

| JSON Property | Description |
|---|---|
| <code>com.tibco.ftl.client.publisher.persistence.retry.duration</code> | The retry duration for API calls made using persistence publishers, including sends to the persistence service when the publisher mode is <code>store_confirm_send</code> . The default value is <code>-1</code> (forever). Applications may override this value when creating a publisher. |
| <code>com.tibco.ftl.client.subscriber.persistence.retry.duration</code> | The retry duration for API calls made using persistence subscribers, including acknowledgments when the ack mode is synchronous. The default value is <code>-1</code> (forever). Applications may override this value when creating a subscriber. |
| <code>com.tibco.ftl.client.map.persistence.retry.duration</code> | The retry duration for API calls made using map objects. The default value is <code>-1</code> (forever). Applications may override this value when creating a map. |

| JSON Property | Description |
|---|--|
| <code>com.tibco.ftl.client.publisher.send.policy</code> | The send policy for publishers without persistence. The default value is 1 (non-inline). Applications may override this value when creating a publisher. |
| <code>com.tibco.ftl.client.publisher.persistence.send.policy</code> | The send policy for persistence publishers. The default value is 0 (inline). Applications may override this value when creating a publisher. |
| <code>use_endpoint_store_for_inbox</code> | Get the value of the use endpoint store for inbox flag. See Endpoint Store Inboxes . |
| <code>enable_permissions</code> | Get the value of the permissions flag. See Authorization . |
| <code>com.tibco.ftl.client.subscriber.nolocal.message.delivery</code> | The default no-local setting for newly created subscriptions. See No-Local Message Delivery in Development. |

```
{
  "client_monitor_sample_interval": 60,
  "client_server_heartbeat": 60,
  "client_timeout_server": 180,
  "manage_all_formats": false,
  "realm_is_secure": false,
  "server_client_heartbeat": 60,
  "server_timeout_client": 3600,
  "max_deployment_history": 25,
  "com.tibco.ftl.client.publisher.persistence.retry.duration": -1,
  "com.tibco.ftl.client.subscriber.persistence.retry.duration": -1,
  "com.tibco.ftl.client.map.persistence.retry.duration": -1,
  "com.tibco.ftl.client.publisher.send.policy": 1,
  "com.tibco.ftl.client.publisher.persistence.send.policy": 0,
  "service_connection_policy": "default",
  "use_endpoint_store_for_inbox": true,
  "enable_permissions": false
}
```

Example Requests

```
curl -X GET http://<host>:<port>/api/v1/realm/properties
```

POST realm/properties

This web method sets the values of the global realm properties.

Properties

| JSON Property | Description |
|---|--|
| com.tibco.ftl.client.publisher.persistence.retry.duration | The retry duration for API calls made using persistence publishers, including sends to the persistence service |

| JSON Property | Description |
|---|--|
| | when the publisher mode is store_confirm_send. The default value is -1 (forever). Applications may override this value when creating a publisher. |
| <code>com.tibco.ftl.client.subscriber.persistence.retry.duration</code> | The retry duration for API calls made using persistence subscribers, including acknowledgments when the ack mode is synchronous. The default value is -1 (forever). Applications may override this value when creating a subscriber. |
| <code>com.tibco.ftl.client.map.persistence.retry.duration</code> | The retry duration for API calls made using map objects. The default value is -1 (forever). Applications may override this value when creating a map. |
| <code>com.tibco.ftl.client.publisher.send.policy</code> | The send policy for publishers without persistence. The default value is 1 |

| JSON Property | Description |
|---|--|
| | (non-inline). Applications may override this value when creating a publisher. |
| <code>com.tibco.ftl.client.publisher.persistence.send.policy</code> | The send policy for persistence publishers. The default value is 0 (inline). Applications may override this value when creating a publisher. |
| <code>max_deployment_history</code> | Limit the size of the FTL server database by limiting the number of past deployments. |
| <code>enable_permissions</code> | Set the value of the permissions flag. See Authorization . |
| <code>use_endpoint_store_for_inbox</code> | Set the value of the use endpoint store for inbox flag. See Endpoint Store Inboxes . |
| <code>enable_format_serialization_optimization</code> | Set the value On when all the clients and servers are FTL 6.10 or greater. The default is off. |

| JSON Property | Description |
|---|---|
| <code>com.tibco.ftl.client.subscriber.nolocal.message.delivery</code> | The default no-local setting for newly created subscriptions. See No-Local Message Delivery in “Development”. |

Example Requests

```
curl -X POST http://<host>:<port>/api/v1/realm/properties -d <json_props>
```

```
curl -X POST http://<host>:<port>/api/v1/realm/properties -d '{"max_deployment_history":50}'
```

Status Objects

The FTL server web API uses status objects to represent the operating state of clients and components. You can use the API to get the current status of these objects, and to send commands.

TIBCO eFTL

Administrators also use the FTL server GUI and web API to configure and monitor the TIBCO eFTL™ service. For information about these GUI pages and REST API, see the TIBCO eFTL documentation.

FTL Server Status Objects

The FTL server web API represents the operating state of the FTL server and the core servers cluster as JSON objects.

GET available

This web method checks the health status of the FTL server and returns 200 if the FTL cluster is available, the initial deployment has completed, and the default cluster is running.

Example Requests

```
curl GET http://host:port/api/v1/available
```

GET ftlservers/<server_name>

This web method retrieves operating information about a specific FTL server or about all the FTL servers.

The <server_name> in the URI is the name of a specific FTL server.

Operating information is similar to the kind of information available about client processes. For details, see [GET clients/<ID>](#).

Example Requests

```
curl -X GET http://<host>:<port>/api/v1/ftlservers/<server_name>
```

```
curl -X GET http://<host>:<port>/api/v1/ftlservers
```

Example Response:

```
{
  "id": 11026,
  "record_id": "11026",
  "stale": false,
  "type": 9,
  "type_label": "FTLServer",
  "realm_revision": 300,
```

```

    "status": 1,
    "status_label": "Running",
    "client_label": "FTL.ftls1",
    "connect_time": 1540404282727,
    "uptime": 0,
    "last_contact": "2018-10-24T11:04:42.734766757-07:00",
    "last_contact_delta": 65413,
    "other_info": "Cores=8, User=bpeterse, Group=msgsrc, Effective
User=bpeterse, Effective Group=msgsrc, OS Spec=Linux 3.10.0-
327.18.2.el7.x86_64 x86_64, Pid=28910",
    "client_name": "",
    "user": "",
    "version": "6.0.0 V5",
    "host": "bender6.na.tibco.com",
    "ip": "10.101.2.22",
    "identifier": "",
    "app_name": "_dynamic_FTLServer",
    "reporting_server_label": "",
    "reporting_server_id": "403d24cf-acb5-46ba-ae36-064dfb5f96ed",
    "realm_name": "_default_realm",
    "app_instance": "default",
    "startup": "2018-10-24T11:04:23.427427-07:00",
    "sub_app_instance": "default",
    "log_statement": [],
    "advisory": []
}

```

POST ftlservers/<server_name>

This web method can send a shutdown command to a specific FTL server or to all FTL servers.

The <server_name> in the URI is the name of a specific FTL server.

Input Data

Supply the command in JSON format.

Commands

The only command available is shutdown.

| Syntax | Description |
|---|---|
| <code>{"cmd":"shutdown"}</code> | Shut down the FTL server and all the services it provides. |
| <code>{"cmd":"shutdown","args":[{"servers":"core.servers"}]}</code> | Shut down all the FTL core servers, along with all the services they provide. (Auxiliary servers are not affected.) |

Example Requests

```
curl -X POST http://<host>:<port>/api/v1/ftlservers/<server_name> -d '{
  "cmd":"shutdown"
}'
```

```
curl -X POST http://<host>:<port>/api/v1/ftlservers/ -d '{
  "cmd":"shutdown"
}'
```

```
curl -X POST http://<host>:<port>/api/v1/ftlservers/ -d '{
  "cmd":"shutdown","args":[{"servers":"core.servers"}]}
}'
```

You can also shut down all FTL servers with [POST cluster](#) .

GET ftlservers/<server_name>/status

This web method retrieves the status of a specific FTL server and the services it provides.

The <server_name> in the URI is the name of a specific FTL server.

Example Requests

```
curl -X GET http://<host>:<port>/api/v1/ftlservers/<server_name>/status
```

Example Response:

```
{
  "status": "RUNNING",
  "name": "ftls1",
  "starttime": 1540404270491,
  "uptime": 79794,
  "services": [
    {
      "name": "realm",
      "executable": "/tibco/ftl/6.4/bin/tibrserver",
      "parameters": [
        "--http",
        "127.0.0.1:35864",
        "--ftl",
        "0.0.0.0:44205",
        "--gui",
        "127.0.0.1:48640",
        "--client.url",
        "http://bender6:9199",
        "--ftlserver.urls",
        "ftls1@bender6:9199|ftls2@bender6:9299|ftls3@bender6:9399",
        "--ftlserver.name",
        "ftls1"
      ],
      "required_restarts": 0,
      "starttime": 1540404263454,
      "uptime": 86873,
      "state": "running",
      "status": "
{\\\"ftlserver\\\":\\\"ftls1\\\",\\\"group.service.mode\\\":\\\"Primary\\\",\\\"group.serv
ice.state\\\":\\\"RUNNING\\\",\\\"realm.quorum\\\":\\\"ftls3\\\",\\\"realmstate\\\":\\\"STAN
DALONE/PRIMARY\\\",\\\"service\\\":\\\"realm\\\",\\\"serviceid\\\":\\\"1024\\\"}\"
    },
    {
      "name": "ftl",
      "executable": "/tibco/ftl/6.4/bin/tibmux",
      "parameters": [
        "--realmserver",
        "http://127.0.0.1:35864",
        "--realm-bootstrap-retries",
        "5",
        "--realm-bootstrap-interval",
        "1000",
        "--realm-connect-interval",
        "1000",

```

```

        "--external-address",
        "ftls1@bender6:9199",
        "--default-http-service",
        "default-http@127.0.0.1:35864",
        "--local-services",
        "boot@127.0.0.1:35864|FTL-RS@0.0.0.0:44205",
        "--peer-muxes",
        "ftls2@bender6:9299|ftls3@bender6:9399|ftls1@bender6:9199"
    ],
    "required_restarts": 0,
    "starttime": 1540404265454,
    "uptime": 84877,
    "state": "running",
    "status": "null"
},
{
    "name": "persistence",
    "executable": "/tibco/ftl/6.4/bin/tibpserver",
    "parameters": [
        "--name",
        "default_ftls1",
        "--realmserver",
        "http://127.0.0.1:9199"
    ],
    "required_restarts": 0,
    "starttime": 1540404270493,
    "uptime": 79842,
    "state": "running",
    "status": "
{"ftlserver\\":\\"ftls1\\",\\"service\\":\\"persist\\",\\"serviceid\\":\\"1026\\",
\\"status\\":\\"RUNNING\\"}"
    "
",
    "current_connections": 25,
    "max_connections": 25,
    "rejected_connections": 1,
    "lookup_failures": 17
}

```

GET cluster

This web method retrieves the status of the FTL server cluster.

The response includes information about the cluster as a whole, plus information about each FTL server in the cluster.

Example Requests

```
curl -X GET http://<host>:<port>/api/v1/cluster
```

Example Response

```
{
  "server_name": "ftls1",
  "leader": "ftls3",
  "component_count": {
    "bridge_server_count": 0,
    "eftl_cluster_count": 0,
    "group_client_count": 0,
    "group_server_count": 3,
    "persistence_server_count": 3,
    "satellite_count": 0,
    "ftl_server_count": 3,
    "other_service_count": 9,
    "service_count": 18,
    "client_count": 0,
    "total_client_count": 18,
    "cumulative_client_count": 18,
    "cumulative_total_client_count": 18,
    "timed_out_client_count": 0
  },
  "members": [
    {
      "server_name": "ftls1",
      "server_host": "host6",
      "server_port": 9199,
      "server_type": "core",
      "server_status": "online",
      "server_uuid": "403d24cf-acb5-46ba-ae36-064dfb5f96ed",
      "server_label": "127.0.0.1:35864",
      "realm_uuid": "76a2709f-201d-4541-a5d7-4c701e522a66",
      "realm_revision": 300,
      "client_count": 8,
      "leader": "ftls3",
      "url": "http://127.0.0.1:35864",
      "component_count": {
        "bridge_server_count": 0,
        "eftl_cluster_count": 0,
        "group_client_count": 0,
        "group_server_count": 1,
        "persistence_server_count": 1,

```

```

    "satellite_count": 0,
    "ftl_server_count": 1,
    "other_service_count": 3,
    "service_count": 6,
    "client_count": 0,
    "total_client_count": 6,
    "cumulative_client_count": 6,
    "cumulative_total_client_count": 6,
    "timed_out_client_count": 0
  },
  "satellites": [],
  "startup_time": "Tue, 02 Aug 2022 13:04:25 PDT",
  "startup_time_millis": 1659470665954,
  "mode": "STANDALONE/PRIMARY",
  "mode_label": "Running"
},
{
  "server_name": "ftls2",
  "server_host": "host6",
  "server_port": 9299,
  "server_type": "core",
  "server_status": "online",
  "server_uuid": "96510be0-d83a-47f0-a715-df1ab0c9fecb",
  "server_label": "127.0.0.1:36818",
  "realm_uuid": "76a2709f-201d-4541-a5d7-4c701e522a66",
  "realm_revision": 300,
  "client_count": 8,
  "leader": "ftls3",
  "url": "http://127.0.0.1:36818",
  "component_count": {
    "bridge_server_count": 0,
    "eftl_cluster_count": 0,
    "group_client_count": 0,
    "group_server_count": 1,
    "persistence_server_count": 1,
    "satellite_count": 0,
    "ftl_server_count": 1,
    "other_service_count": 3,
    "service_count": 6,
    "client_count": 0,
    "total_client_count": 6,
    "cumulative_client_count": 6,
    "cumulative_total_client_count": 6,
    "timed_out_client_count": 0
  },
  "satellites": [],
  "startup_time": "Tue, 02 Aug 2022 13:04:25 PDT",

```



```

    "startup_time_millis": 1659470665954,
    "mode": "STANDALONE/PRIMARY",
    "mode_label": "Running"
  },
  {
    "server_name": "ftls3",
    "server_host": "host6",
    "server_port": 9399,
    "server_type": "core",
    "server_status": "online",
    "server_uuid": "30669c79-55e6-44c1-96a1-e2dcc8b0388b",
    "server_label": "127.0.0.1:59792",
    "realm_uuid": "76a2709f-201d-4541-a5d7-4c701e522a66",
    "realm_revision": 300,
    "client_count": 8,
    "leader": "ftls3",
    "url": "http://127.0.0.1:59792",
    "component_count": {
      "bridge_server_count": 0,
      "eftl_cluster_count": 0,
      "group_client_count": 0,
      "group_server_count": 1,
      "persistence_server_count": 1,
      "satellite_count": 0,
      "ftl_server_count": 1,
      "other_service_count": 3,
      "service_count": 6,
      "client_count": 0,
      "total_client_count": 6,
      "cumulative_client_count": 0,
      "cumulative_total_client_count": 6
      "timed_out_client_count": 0
    },
    "satellites": [],
    "startup_time": "Tue, 02 Aug 2022 13:04:25 PDT",
    "startup_time_millis": 1659470665954,
    "mode": "STANDALONE/PRIMARY",
    "mode_label": "Running"
  }
]
}

```

POST cluster

This web method can send a shutdown command to all FTL servers in the cluster.

Input Data

Supply the command in JSON format.

Commands

| Syntax | Description |
|--|--|
| <code>{"cmd": "shutdown"}</code> | Shut down the FTL server cluster. |
| <code>{"cmd": "activate_dr"}</code> | Activate a DR site. |
| <code>{"cmd": "enable_dr", "args": [{"drto": "my_urls"}]}</code> | Enable DR at a primary site, or point a primary site to a different DR site. |

Example Requests

Simple shutdown:

```
curl -X POST http://<host>:<port>/api/v1/cluster/ -d '{  
  "cmd": "shutdown"}'
```

Save persistence state to disk, then shut down:

```
curl -X POST http://<host>:<port>/api/v1/cluster/ -d '{"cmd": "shutdown",  
  "args": [{"savestate": true}]}'
```



Note: The savestate option is not needed if disk persistence is enabled.

You can also shut down all FTL servers with [POST ftlservers/<server_name>](#).

GET server

This web method retrieves the operating state of the realm service provided by an individual FTL server.

For state information and counts summed over all FTL core servers, see [GET cluster](#).

Example Requests

```
curl -X GET http://<host>:<port>/api/v1/server
```

Example Response

```
{
  "backup_heartbeat":1.0,
  "backup_timeout":3.0,
  "client_count":5,
  "code_version":"5.0.0    V0",
  "component_count":{
    "bridge_server_count":1,
    "eftl_cluster_count":1,
    "group_client_count":2,
    "group_server_count":1,
    "persistence_server_count":1,
    "satellite_count":0,
    "client_count":5,
    "ftl_server_count":1,
    "other_service_count":3,
    "service_count":6,
    "total_client_count":11,
    "cumulative_client_count": 5,
    "cumulative_total_client_count": 11,
    "timed_out_client_count":0
  },
  "startup_time_millis": 100,
  "current_deployment":"2018-09-24-14-05-08-628",
  "db_gen":"18",
  "dr_in_sync": true,
  "eftl_cluster_count":1,
  "free_vm_memory":339214336,
  "ftl_port":"8083",
  "host_port":"reed:8080",
  "http_port":"8080",
  "id":0,
  "last_deployed_on":"Sep 24, 2018 2:05:08 PM",
  "last_deployed_on_ms":1456351508721,
  "long_code_version":"TIBCO FTL Server Version 6.4.0    V0",
  "max_vm_memory":5586812928,
  "mode":"STANDALONE\PRIMARY",
  "mode_label":"Running",
  "pending_requests":[],
  "realm_revision":"41065",
```

```

    "satellite_count":0,
    "satellite_heartbeat":1.0,
    "satellite_timeout":3.0,
    "satellites":[],
    "satellites_in_sync":true,
    "server_uuid":"56afe864-dba1-4bc3-90b4-82daf41f672b",
    "ssl_enabled":false,
    "startup_time":"Sep 25, 2018 8:47:20 AM",
    "status_time":"Sep 25, 2018 10:34:01 AM",
    "status_time_millis":1456425241130,
    "total_vm_memory":309251368,
    "uptime_seconds":6400,
    "user_friendly_name":"reed:8080"
  }

```

POST server

This web method sends commands affecting the realm service provided by an FTL server.

Input Data

Supply the command in JSON format.

Commands

Commands are parallel to commands that are available through the FTL server administration utility. You can use these commands during upgrade migration tasks.

| Syntax | Description |
|--|---|
| <code>{"cmd":"backup"}</code> | Backup the FTL server realm database. |
| <code>{"cmd":"compact"}</code> | Compact the FTL server realm database. |
| <code>{"cmd":"exit_compatibility_mode"}</code> | After an upgrade and if needed, exit the compatibility window manually, when it is deemed safe to do so. For more information, see Upgrading from Release 6.x . |

Example Requests

```
curl -X POST http://<host>:<port>/api/v1/server -d '{"cmd":"backup"}'
```

```
curl -X POST http://<host>:<port>/api/v1/server -d '{"cmd":"compact"}'
```

Clients Status Objects

The FTL server web API represents the operating states of its clients as a JSON object.

Client ID

The FTL server assigns a unique client ID to each client. The ID is unique with the realm and across time. IDs are also unique across the different types of clients: ordinary application clients, and TIBCO FTL service clients, such as bridge services, eFTL services, and persistence services.

Example JSON Representation

```
{
  "clients":[{
    "app_instance":"default",
    "app_name":"tibrecvex",
    "host":"reed",
    "id":11055,
    "ip":"10.101.2.226",
    "last_contact":"Feb 24, 2016 10:11:56 AM",
    "last_contact_delta":31552,
    "other_info":"Cores=24,
                  Group=ms,
                  Effective User=bp,
                  User=bp,
                  Effective Group=ms,
                  Pid=26932,
                  OS Spec=Linux 2.6.32-573.3.1.el6.x86_64 x86_64",
    "realm_revision":6131,
    "startup":"Feb 23, 2016 3:27:17 PM",
    "status":1,
```

```

    "status_label": "Running",
    "user": "guest",
    "version": "5.0.0    V0"
  }]
}

```

JSON Attributes of Clients

For attributes and semantics, see [Client Status Details](#).

Filters for Client Status Methods

You can filter client status methods by attribute and value. Methods first filter the clients, and then apply any action to the clients that meet the filtering criteria. Filtering applies to all client status methods.

Clients

Filtering is available for ordinary application clients and also for TIBCO FTL service clients, such as bridge services, eFTL services, and persistence services.

GET Methods

If a GET method returns a collection of client status objects as its top level value, then you can filter the collection based on JSON attributes of the client objects in the collection.

A request for monitoring metrics is an action that applies *after* filtering the collection.

For example, the GET `clients` method returns a collection of client objects. You can filter the collection to clients that match one or more attribute values. For example, the request `GET clients?app_name=tibrecvex&host=reed` gets only clients with the application name `tibrecvex` running on host `reed`:

```

{
  "clients": [
    {
      "app_instance": "default",
      "app_name": "tibrecvex",
      "host": "reed",

```

```
"id":11055,  
"ip":"10.101.2.226",  
...
```

Other Methods

If a POST, PUT, or DELETE method affects a *collection* of client objects, then you can target a *narrower* collection by adding filter parameters to the request. The action of the method applies *after* filtering the collection. That is, the action applies to each client object that meets the filtering criteria.

Parameters for Client Filtering

You can filter requests for client status objects to narrow the collection of clients.

Filtering is available as part of any client status object method. For background information, see [Filters for Client Status Methods](#).

- You can filter the list by length.
- You can filter the list by comparing the relative values of client id numbers.
- You can filter on any of the JSON attributes of client status objects.

The following table suggests some of the most useful HTTP parameters for filtering.

| Syntax | Description |
|--------------|--|
| limit=<max> | Optional. When present, return at most <max> objects. When absent, retrieve the entire list. |
| before=<num> | Optional. When present, return only those clients with id less than <num>. |
| after=<num> | Optional. When present, return only those clients with id greater than <num>. |

| Syntax | Description |
|----------------------|--|
| app_ name=<name> | Optional. When present, return only those clients with this application name. |
| user=<user_ name> | Optional. When present, return only those clients with this username. |
| host=<host_ name> | Optional. When present, return only those clients running on the computer with this host name. |
| ip=<ip_addr> | Optional. When present, return only those clients running on the computer with this IP address. |
| status=<status> | Optional. When present, return only those clients with this operating status. Specify one of the following integer <status> values: <ol style="list-style-type: none">1. Running2. Needs restart3. Timed out4. Exception5. Out of synch |

Parameters for Client Monitoring

You can request monitoring metrics as part of any request that returns a client status object or a collection of client status objects.

Metrics data is available for ordinary application clients and also for TIBCO FTL service clients, such as bridge services, eFTL services, and persistence services.

| Syntax | Description |
|-----------------|---|
| monitoring=true | Optional. When present, the response includes monitoring metrics data as the value of the JSON attribute monitoring. |

Example JSON Response

```
{
  "app_instance":"default",
  "app_name":"tibrecvex",
  "host":"ree",
  "id":11055,
  "ip":"10.101.2.226",
  "last_contact":"Feb 26, 2016 1:27:00 PM",
  "last_contact_delta":43658,
  "monitoring":{
    "id":11055,
    "series":[
      {
        "avg":"68,425,328.673",
        "context":"client_transport_0_1456246848935",
        "description":"Bytes Sent",
        "id":32,
        "last":"68,469,297",
        "max":"68,469,297",
        "min":"68,469,297",
        "samples":1,
        "semantic":"VALUE",
        "total_samples":6,
        "trend":1,
        "type":"bytes_sent",
        "units":""
      },
      ...
    ]
  }
}
```

GET clients

This web method retrieves a collection of client status objects.

Example Requests

```
curl -X GET http://<host>:<port>/api/v1/clients
```

```
curl -X GET http://<host>:<port>/api/v1/clients?monitoring=true
```

GET clients/<ID>

This web method retrieves the status of the client with a specific client ID.

The <ID> in the URI is the numeric client ID of the client.

Example Requests

```
curl -X GET http://<host>:<port>/api/v1/clients/<ID>
```

```
curl -X GET http://<host>:<port>/api/v1/clients/<ID>?monitoring=true
```

POST clients/<ID> and POST clients

This web method sends a command that affects a specific client. The broader form, POST clients, sends a command to a collection of clients.

Targeting Clients

- To target a specific client, supply the client ID as the <ID> in the URI.
- To target a collection of clients, omit the <ID> from the URI.
- To target a narrower collection of clients, omit the <ID> from the URI, and supply filtering arguments.

This example filters the clients to those that run on a specific host computer, and disables only those clients:

```
POST clients?host=reed {"cmd":"disable"}
```

For details, see [Filters for Client Status Methods](#).

Input Data

Supply the command in JSON format, for example:

```
{"cmd":"setloglevel","args":[{"level":"warn"}]}
```

Commands

| Command | Description |
|---|---|
| <pre>{"cmd":"setloglevel", "args":[{"level": "element:level"}]}</pre> | <p>Set the client's log level.</p> <p>For details, see "Tuning the Log Level" in TIBCO FTL Development.</p> |
| <pre>{"cmd":"sendlogs", "args":[{"mode": "<mode>"}]}</pre> | <p>Instruct the client to enable or disable sending its logs to the FTL server. The initial mode for all clients is <code>off</code>.</p> <p>Values:</p> <ul style="list-style-type: none"> • <code>on</code> The client sends its log messages to the FTL server. • <code>off</code> The client does <i>not</i> send its log messages to the FTL server. (The client still outputs log messages to its log output target.) <p>For more information, see Catalog of Metrics .</p> |
| <pre>{"cmd":"setmonmode", "args":[{"mode": "<mode>"}]}</pre> | <p>Set the client's monitoring mode.</p> <p>Values:</p> <ul style="list-style-type: none"> • <code>dynamic_monitor_info_on</code> The client gathers metrics related to message content matching, and generates monitoring data submessages with types 90009 and 90010. For more information, see Catalog of Metrics . |

| Command | Description |
|--------------------------------|---|
| | <ul style="list-style-type: none"> <code>dynamic_monitor_info_off</code> The client does <i>not</i> gather metrics related to content matching, and does <i>not</i> generate monitoring data submessages with types 90009 and 90010. |
| <code>{"cmd":"disable"}</code> | <p>Disable the client.</p> <p>For background information, see Conditions for Disabling Clients.</p> |
| <code>{"cmd":"purge"}</code> | <p>Purge clients that have timed out.</p> <p>For background information, see Client Status.</p> |

Example Requests

```
curl -X POST http://<host>:<port>/api/v1/clients/<ID> -d <json_cmd>
```

Bridges Status Objects

The FTL server web API represents the operating states of transport bridges as a JSON object.

JSON Attributes of Bridges

Bridges are client processes of the FTL server. The set of JSON attributes for bridges is identical to the set of attributes for clients in general. For details, see [Client Status Details](#). For an example JSON representation, see [Clients Status Objects](#).

Client ID

The FTL server assigns a unique client ID to each client. The ID is unique with the realm and across time. IDs are also unique across the different types of clients: ordinary application clients, and TIBCO FTL service clients, such as bridge services, eFTL services, and persistence services.

GET bridges

This web method retrieves a collection of transport bridge status objects.

Example Requests

```
curl -X GET http://<host>:<port>/api/v1/bridges
```

GET bridges/<br_name>

This web method retrieves the status of the transport bridge with a specific bridge name.

The <br_name> in the URI is the name of the bridge.

Example Requests

```
curl -X GET http://<host>:<port>/api/v1/bridges/<br_name>
```

POST bridges/<br_name>

This web method changes the log level of the transport bridge with a specific bridge name.

The <br_name> in the URI is the name of the bridge.

Input Data

Supply the log level command in JSON format, for example:

```
{ "cmd": "setloglevel",  
  "args": [ { "level": "element:level" } ] }
```

For details, see "Tuning the Log Level" in TIBCO FTL [Development](#).

Example Requests

```
curl -X POST http://<host>:<port>/api/v1/bridges/<br_name> -d <json_cmd>
```

Group Server Status Object

The FTL server web API represents the operating state of the group service as a JSON object.

Example JSON Representation

```
{
  "groups": [{
    "activation_interval": 5.0,
    "members": [{
      "client_id": "11033",
      "member_descriptor": null,
      "member_type": "OBSERVER_MEMBER",
      "ordinal": 0
    },
    {
      "client_id": "11037",
      "member_descriptor": null,
      "member_type": "ORDINAL_MEMBER",
      "ordinal": 2
    },
    {
      "client_id": "11041",
      "member_descriptor": "{string:my_descriptor_string=\"My
descriptor\"}",
      "member_type": "FULL_MEMBER",
      "ordinal": 1
    }
  ]},
  "name": "myGroup"
}],
"server_mode": "Primary",
"server_state": "RUNNING"
}
```

GET available groupserver

This web method checks the health status of the FTL group service and returns 200 if the group service is running.

Example Requests

```
curl GET http://host:port/api/v1/available?groupserver=true
```

GET groupserver

This web method retrieves the operating state of the group service. That state includes a list of groups, and for each group, the state of each member client.

Example Requests

```
curl -X GET http://<host>:<port>/api/v1/groupserver
```

POST groupserver

This web method sends a command that affects the group service.

Input Data

Supply the command in JSON format, for example:

```
{"cmd": "start"}
```

Commands

| Command | Description |
|---------------------------------|--|
| <code>{"cmd": "start"}</code> | Start the group service. |
| <code>{"cmd": "stop"}</code> | Stop the group service. |
| <code>{"cmd": "restart"}</code> | Stop and restart the group service. Use this command after modifying the group service transport configuration. |

Example Requests

```
curl -X POST http://<host>:<port>/api/v1/groupserver -d <json_cmd>
```

GET groupserver/transport

This web method retrieves the group services transport definitions.

Example Requests

```
curl -X GET http://<host>:<port>/api/v1/groupserver/transport
```

Example Response

```
{
  "server_transport": {
    "last_modified": "2018-02-15T12:20:44.56549408-08:00",
    "last_modified_millis": 1518726044565,
    "last_modified_by": "anyone",
    "name": "_GroupServerTport1",
    "description": ""
```



```

    "relationships": [
      {
        "transport": "_GroupClientTport"
      }
    ],
    "id": 1025,
    "config": {
      "transport_type": "dtcp",
      "backlog_full_wait": "0",
      "recv_spin_limit": "0",
      "backlog_size": "64mb",
      "port": "7777",
      "subnet_mask": "",
      "virtual_name": "_Group_Server_VName",
      "mode": "server"
    }
  },
  "client_transport": {
    "last_modified": "2018-02-15T12:20:44.56549874-08:00",
    "last_modified_millis": 1518726044565,
    "last_modified_by": "anyone",
    "name": "_GroupClientTport",
    "description": "",
    "relationships": [
      {
        "transport": "_GroupServerTport1"
      }
    ],
    "id": 1027,
    "config": {
      "transport_type": "wdtcp",
      "backlog_full_wait": "0",
      "recv_spin_limit": "0",
      "backlog_size": "64mb",
      "port": "7777",
      "subnet_mask": "",
      "virtual_name": "_Group_Server_VName",
      "mode": "client"
    }
  }
}

```

POST groupserver/transports

This web method modifies the definitions of the group service transports.

After modifying the transport definitions, the FTL servers automatically restart their group services. See [POST groupserver](#).

Example Requests

```
curl -X POST http://<host>:<port>/api/v1/groupserver/transports -d
@<json_file>
```

Input Data

Supply the transport definitions in JSON format, for example:

```
{
  "server_transport": {
    "name": "_GroupServerTport1",
    "relationships": [
      {
        "transport": "_GroupClientTport"
      }
    ],
    "config": {
      "transport_type": "wdtcp",
      "port": "7777",
      "virtual_name": "_Group_Server_VName",
      "mode": "server"
    }
  },
  "client_transport": {
    "name": "_GroupClientTport",
    "relationships": [
      {
        "transport": "_GroupServerTport1"
      }
    ],
    "config": {
      "transport_type": "wdtcp",
      "port": "7777",
      "virtual_name": "_Group_Server_VName",
      "mode": "client"
    }
  }
}
```

Persistence Status Objects

The FTL server web API represents the operating state of each persistence cluster as a JSON object. The API also transmits management commands to persistence clusters, services, and stores.

Filters for Durables

You can filter client durables by attribute and value. Methods first filter the durables, and then apply any action to the durables that meet the filtering criteria. Filtering applies to all persistence status methods at the durable level.

If a GET method returns a collection of durable status objects as its top level value, then you can filter the collection based on JSON attributes of the durable objects in the collection.

For example, GET `persistence/<clus_name>/stores/<stor_name>/durables` returns a collection of durable objects. You can filter the collection to durables that match one or more top level attribute values. For example, the filter `?type=standard&dynamic=true` gets only dynamic standard durables.

Similarly, if a POST method affects a collection of durable status objects, then you can target a narrower collection by adding filter parameters to the request. The action of the method applies *after* filtering the collection. You can use this technique to selectively purge a narrower collection of durables from a persistence store.

JSON Attributes for Filtering Durables

You can filter by any top-level JSON attribute of durables that has a scalar, string, or boolean value. Nonetheless, the following are the most useful attributes for filtering durables:

- `type=<type>`
- `name=<dur_name>`
- `client_id=<ID>`
- `dynamic=<bool>`
- `filter=<filter_string>`

- staticOnly=true
- dynamicOnly=true

GET persistence/clusters

This web method retrieves a collection of persistence cluster status objects.

Example Requests

```
curl -X GET http://<host>:<port>/api/v1/persistence/clusters
```

Example JSON Representation

```
{
  "clusters": [{
    "name": "Cluster",
    "servers": ["pserver"],
    "stores": ["pstore"]
    "quorum": {
      "set_name": "_setA",
      "have_quorum": true,
      "leader": "pserver",
      "max_member_count": 1,
      "min_member_count": 1,
      "current_member_count": 1,
      "errors": [],
      "members_in_quorum": [
        "pserver",
      ],
      "members_not_in_quorum": []
    }
  }]
}
```

GET persistence/clusters/<clus_name>

This web method retrieves the status of a persistence cluster by its name.

<clus_name> in the URI is the cluster name.

Example Requests

```
curl -X GET http://<host>:<port>/api/v1/persistence/clusters/<clus_name>
```

GET persistence/clusters/<clus_name>/servers

This web method retrieves a collection of persistence service status objects within a specific cluster.

<clus_name> in the URI is the cluster name.

Example Requests

```
curl -X GET http://<host>:<port>/api/v1/persistence/clusters/<clus_name>/servers
```

Example JSON Representation

```
{
  "servers": [
    {
      "id": 17828,
      "record_id": "17828",
      "type": 6,
      "type_label": "Persistence Server",
      "realm_revision": 6026,
      "status": 1,
      "status_label": "Running",
      "client_label": "Persistence Server p14",
      "connect_time": 1661984877969,
      "uptime": 112185,
      "last_contact": "2022-08-31T17:29:52.064167001-05:00",
      "last_contact_delta": 1045,
      "other_info": "Group=group1, Effective User=user1, Effective
Group=group1, OS Spec=Linux 5.15.0-46-generic x86_64, Pid=261220,
Cores=4, User=user1",
      "client_name": "p14",
      "user": "",
      "version": "6.9.0 V3",
    }
  ]
}
```

```

"host": "test1-t470s",
"ip": "127.0.1.1",
"identifier": "p14",
"app_name": "_Server_p14",
"compaction_in_progress": false,
"backup_in_progress": false,
"reporting_server_label": "",
"reporting_server_id": "b6ca6f72-ea72-49e5-b1ee-32dbb0e119e2",
"realm_name": "_default_realm",
"app_instance": "default",
"startup": "2022-08-31T17:27:57.897779-05:00",
"sub_app_instance": "default",
"log_label": "Client(Persistence Server p14)",
"log_level": "info",
"log_mode": "off",
"initial_server_version": "6.9.0 V3",
"pserver_set": "sset1",
"history": "2,24",
"history_timestamp": 1661979067085,
"history_quorum_number": 2,
"history_update_version": 24,
"history_update_version_committed": 24,
"history_update_version_non_replicated": 0,
"consistent_history": "-1,-1",
"consistent_timestamp": -1,
"consistent_quorum": -1,
"consistent_version": -1,
"consistent_version_committed": -1,
"quorum_state": 2,
"quorum_state_label": "Leader",
"server_version": "TIBCO FTL Persistence Service 6.9.0 V3",
"leader_name": "p14",
"cluster": "c1",
"disk_state": 4,
"disk_persistence": "sync",
"disk_swap": true,
"disk_size": 139264,
"disk_inuse_size": 12288,
"backlog_limit": 268435456,
"backlog_size": 0,
"disk_backlog_size": 0,
"message_count": 0,
"message_size": 0,
"disk_capacity": 246563770368,
"disk_usage_limit": 234235581849,
"disk_available": 193774510080,

```

```

    "disk_used": 40189992960,
    "last_primary_contact_millis": 1661984993108,
    "log_statement": [],
    "advisory": []
  }
]
}

```



Note: "disk_state":2 indicates the successful completion of a dump-to-disk operation.

GET persistence/clusters/<clus_name>/servers/<svc_name>

This web method retrieves the status of a specific persistence service within a cluster.

- <clus_name> in the URI is the cluster name.
- <svc_name> in the URI is the service name.

Example Requests

```
curl -X GET http://<host>:<port>/api/v1/persistence/clusters/<clus_name>/servers/<svc_name>
```

GET persistence/clusters/<clus_name>/stores

This web method retrieves a collection of persistence store status objects within a specific cluster.

<clus_name> in the URI is the cluster name.

Example Requests

```
curl -X GET http://<host>:<port>/api/v1/persistence/clusters/<clus_name>/stores
```

Example JSON Representation

```
{
  "stores": [{
    "name": "pstore",
    "cluster": "cluster",
    "store_leader": "pserver",
    "durable_count": 1,
    "message_count": 0,
    "message_size": 0,
    "swap_message_count": 0,
    "swap_message_size": 0,
    "bytelimit": "9999457"
  }]
}
```

GET persistence/clusters/<clus_name>/stores/<stor_name>

This web method retrieves the status of a specific persistence store within a cluster.

- <clus_name> in the URI is the cluster name.
- <stor_name> in the URI is the store name.

Example Requests

```
curl -X GET http://<host>:<port>/api/v1/persistence/clusters/<clus_name>/stores/<stor_name>
```


Example JSON Representation

```
{
  "name": "st1",
  "cluster": "RealmPersistenceCluster",
  "store_leader": "pserver1",
  "durable_count": 3,
  "message_count": 0,
  "message_size": 0,
  "swap_message_count": 0,
  "swap_message_size": 0,
  "expiration_count": 1001,
  "bytelimit": "",
  "message_limit": 0
}
```

GET persistence/clusters/<clus_name>/stores/<stor_name>/durables

This web method retrieves a collection of durable status objects from a specific persistence store within a cluster.

- <clus_name> in the URI is the cluster name.
- <stor_name> in the URI is the store name.

Example Requests

```
curl -X GET http://<host>:<port>/api/v1/persistence/clusters/GET
http://<host>:<port>/api/v1/persistence/clusters/<clus_
name>/stores/<stor_name>/durables
```

Example JSON Representation

```
{
  "durables": [
    {
      "name": "dur1",
```

```

    "store": "s1",
    "cluster": "c1",
    "matcher": "{}",
    "template_name": "tmpl-std-store-fwd",
    "client_id": 1066,
    "durable_id": 1,
    "type": "standard-store-fwd",
    "dynamic": true,
    "subscribers": [
      {
        "sub_id": 2,
        "client_id": 1066,
        "client_label": "./tibrecvex",
        "last_acked": 130262,
        "last_dispatched": 130262,
        "last_send": 130262,
        "num_unacked": 0,
        "is_browser": 0,
        "browse_matcher": ""
      }
    ],
    "message_count": 0,
    "message_size": 0,
    "swap_message_count": 0,
    "swap_message_size": 0,
    "total_redeliveries": 0,
    "browser_count": 0,
    "bytelimit": "",
    "message_limit": 0
  }
]
}

```

Filtering

See [Filters for Durables](#).

GET persistence/clusters/<clus_name>/stores/<stor_name>/durables/<dur_name>

This web method retrieves the status of a specific durable in a specific persistence store within a cluster.

- <clus_name> in the URI is the cluster name.
- <stor_name> in the URI is the store name.
- <dur_name> in the URI is the durable name.

Example Requests

```
curl -X GET http://<host>:<port>/api/v1/persistence/clusters/<clus_name>/stores/<stor_name>/durables/<dur_name>
```

Example JSON Representation

```
{  
  
  "name": "dur1",  
  
  "store": "s1",  
  
  "cluster": "c1",  
  
  "matcher": "{}",  
  
  "template_name": "tpl-std-store-fwd",  
  
  "client_id": 0,  
  
  "durable_id": 1,  
  
  "type": "standard-with-prefetch",  
  
  "dynamic": true,
```

```
"message_count": 0,
```

```
"message_size": 0,
```

```
"unacked_message_count": 0,
```

```
"unacked_message_size": 0,
```

```
"swap_message_count": 0,
```

```
"swap_message_size": 0,
```

```
"total_redeliveries": 0,
```

```
"expiration_count": 0,
```

```
"browser_count": 0,
```

```
"last_sub_remove_time": 4141,
```

```
"bytelimit": "",
```

```
"message_limit": 0,
```

```
"max_delivery": 0,
```

```
"durable_ttl": 0,
```

```
"message_ttl": 0,
```

```
"retention_time": 0
```

```
}
```

GET persistence/clusters/<clus_name>/quorum

This web method retrieves a collection of persistence quorum status objects within a specific cluster.

<clus_name> in the URI is the cluster name.

Example Requests

```
curl -X GET http://<host>:<port>/api/v1/persistence/clusters/<clus_name>/quorum
```

Example JSON Representation

```
{
  "set_name": "_setA",
  "have_quorum": true,
  "leader": "default_ftls14",
  "max_member_count": 3,
  "min_member_count": 2,
  "current_member_count": 3,
  "errors": [],
  "members_in_quorum": [
    "default_ftls12",
    "default_ftls13",
    "default_ftls14"
  ],
  "members_not_in_quorum": []
}
```

POST persistence/clusters/<clus_name>

This web method sends a command to a specific persistence cluster.

<clus_name> in the URI is the cluster name.

Input Data

Supply the command in JSON format. For example, {"cmd":"suspend"}

Commands

| Command | Description |
|--------------------------|--|
| {"cmd":"backupdisk"} | Back up the disk(s). When disk persistence is enabled, this command backs up the files associated with disk persistence. See also Saving and Loading Persistence State . |
| {"cmd":"compactdisk"} | All members of the persistence cluster will start a compaction. See Compact Disk Persistence Files with Persistence Service Online . |
| {"cmd":"forceformation"} | Force the cluster to form a quorum immediately with available servers, without waiting for all servers to become available or waiting for the force_quorum_delay period to elapse. |
| {"cmd":"suspend"} | See Suspending a Persistence Cluster . |

Example Requests

```
curl -X POST http://<host>:<port>/api/v1/persistence/clusters/<clus_name> -d <json_cmd>
```



Note: You cannot suspend the internal cluster `ftl.default.cluster`.

POST persistence/clusters/<clus_name>/servers/<svc_name>

This web method sends a command to a specific persistence service.

- <clus_name> in the URI is the cluster name.
- <svc_name> in the URI is the service name.

Input Data

Supply the command in JSON format. For example, {"cmd":"dump"}

Commands

| Command | Description |
|---|---|
| <code>{"cmd":"dump"}</code> | See Saving the State of a Persistence Service . In the persistence service status output, "disk_state":2 indicates the successful completion of a dump-to-disk operation. |
| <code>{"cmd":"setloglevel","args":[{"level":"<element>:<level>"}]}</code> | See "Tuning the Log Level" in TIBCO FTL Development . |

Example Requests

```
curl -X POST http://<host>:<port>/api/v1/persistence/clusters/<clus_name>/servers/<svc_name> -d <json_cmd>
```

POST persistence/clusters/<clus_name>/stores/<stor_name>

This web method sends a command to purge all durables in a specific store.

- <clus_name> in the URI is the cluster name.
- <stor_name> in the URI is the store name.

Commands

| Command | Description |
|-------------------------------|--|
| <code>{"cmd": "purge"}</code> | Purge messages from all durables in a store. |

Filtering

See [Filters for Durables](#).

Example Requests

```
curl http://<host>:<port>/api/v1/persistence/clusters/<clus_name>/stores/<stor_name> -X POST -d '{"cmd": "purge"}'
```

POST persistence/clusters/<clus_name>/stores/<stor_name>/durables

This web method sends a command to purge, rewind, or delete selected durables, or delete all durables from a specific store.

- <clus_name> in the URI is the cluster name.
- <stor_name> in the URI is the store name.

Commands

| Command | Description |
|--|--------------------------------------|
| <code>{"cmd": "purge_durables", "args"}</code> | Purge the durables supplied in args. |

| Command | Description |
|--|---------------------------------------|
| <code>{"cmd":"delete_durables","args"}</code> | Delete the durables supplied in args. |
| <code>{"cmd":"delete_all_durables"}</code> | Delete all durables from a store. |
| <code>{"cmd":"rewind_durables", "args"}</code> | Rewind the durables supplied in args. |

Filtering

See [Filters for Durables](#).

Example Requests

```
curl http://<host>:<port>/api/v1/persistence/clusters/<clus_name>/stores/<stor_name>/durables -X POST -d '{"cmd":"purge_durables", "args": [{"names": ["dur1","dur2"]}]]'
```

```
curl http://<host>:<port>/api/v1/persistence/clusters/<clus_name>/stores/<stor_name>/durables -X POST -d '{"cmd":"delete_durables", "args": [{"names": ["dur1", "dur2"]}]]'
```

```
curl http://<host>:<port>/api/v1/persistence/clusters/<clus_name>/stores/<stor_name>/durables -X POST -d '{"cmd":"delete_all_durables"}'
```

```
curl http://<host>:<port>/api/v1/persistence/clusters/<clus_name>/stores/<stor_name>/durables -X POST -d '{"cmd":"rewind_durables", "args": [{"names": ["dur1","dur2"]}]]'
```

POST persistence/clusters/<clus_name>/stores/<stor_name>/durables/<dur_name>

This web method sends a command to purge or rewind a specific durable. The broader form, POST persistence/clusters/<clus_name>/stores/<stor_name> sends a command to purge all durables in a specific store.

- <clus_name> in the URI is the cluster name.
- <stor_name> in the URI is the store name.
- <dur_name> in the URI is the durable name.

Commands

| Command | Description |
|--------------------------------|--------------------------------|
| <code>{"cmd": "purge"}</code> | Purge messages from a durable. |
| <code>{"cmd": "rewind"}</code> | Rewind a durable. |

Filtering

See [Filters for Durables](#).

Example Requests

```
curl -X POST http://<host>:<port>/api/v1/persistence/clusters/<clus_name>/stores/<stor_name>/durables/<dur_name> -d <json_cmd>
```

DELETE persistence/clusters/<clus_name>/stores/<stor_name>/durables/<dur_name>

This web method deletes a specific durable from a specific persistence store from within a cluster.

- <clus_name> in the URI is the cluster name.
- <stor_name> in the URI is the store name.
- <dur_name> in the URI is the durable name.

Example Requests

```
curl -X DELETE
http://<host>:<port>/api/v1/realm/persistence/clusters/<clus_
name>/stores/<stor_name>/durables/<dur_name>
```

Filtering

See [Filters for Durables](#).

GET persistence/zones

This web method retrieves a collection of persistence zone objects.

Example Requests

```
curl -X GET http://<host>:<port>/api/v1/persistence/zones
```

Example JSON Representation

```
{
  "zones": [
    {
      "zone": "Z1H",
      "stores": [
        "S1H"
      ]
    },
    {
      "zone": "ZFM",
      "stores": [
        "SFM"
      ]
    },
    {
      "zone": "ZHS",
      "stores": [
        "SHS"
      ]
    }
  ]
}
```

```

    ]
  }
]
}

```

GET persistence/zones/<zone_name>

This web method retrieves the status of a persistence zone by its name.

<zone_name> in the URI is the zone name.

Example Requests

```
curl -X GET http://<host>:<port>/api/v1/persistence/zones/<zone_name>
```

Example JSON Representation

```

{
  "zone": "ZFM",
  "stores": [
    "SFM"
  ]
}

```

GET persistence/zones/<zone_name>/stores

This web method retrieves a collection of persistence store status objects within a specific zone.

<zone_name> in the URI is the zone name.

Example Requests

```
curl -X GET http://<host>:<port>/api/v1/persistence/zones/<zone_name>/stores
```

Example JSON Representation

```
{
  "stores": [
    {
      "name": "SFM",
      "cluster": "C1",
      "store_leader": "pserver3",
      "durable_count": 3,
      "message_count": 200,
      "message_size": 47872,
      "routing_state": [
        {
          "connection": "_routing_dur_from_C3_for_C1_in_Zone_ZFM",
          "state": "Up"
        },
        {
          "connection": "_routing_dur_from_C2_for_C1_in_Zone_ZFM",
          "state": "Up"
        }
      ]
    },
    {
      "name": "SFM",
      "cluster": "C3",
      "store_leader": "",
      "durable_count": 0,
      "message_count": 0,
      "message_size": 0
    },
    {
      "name": "SFM",
      "cluster": "C2",
      "store_leader": "",
      "durable_count": 0,
      "message_count": 0,
      "message_size": 0
    }
  ]
}
```

GET persistence/zones/<zone_name>/stores/<stor_name>

This web method retrieves the status of a specific persistence store within a zone.

- <zone_name> in the URI is the zone name.
- <stor_name> in the URI is the store name.

Example Requests

```
curl -X GET http://<host>:<port>/api/v1/persistence/zones/<zone_name>/stores/<stor_name>
```

Example JSON Representation

```
[
  {
    "name": "SFM",
    "cluster": "C1",
    "store_leader": "pserver3",
    "durable_count": 3,
    "message_count": 200,
    "message_size": 47872,
    "routing_state": [
      {
        "connection": "_routing_dur_from_C3_for_C1_in_Zone_ZFM",
        "state": "Up"
      },
      {
        "connection": "_routing_dur_from_C2_for_C1_in_Zone_ZFM",
        "state": "Up"
      }
    ]
  },
  {
    "name": "SFM",
    "cluster": "C2",
    "store_leader": "",
    "durable_count": 0,
    "message_count": 0,
    "message_size": 0
  }
]
```

```

    },
    {
      "name": "SFM",
      "cluster": "C3",
      "store_leader": "",
      "durable_count": 0,
      "message_count": 0,
      "message_size": 0
    }
  ]

```

GET persistence/zones/<zone_name>/stores/<stor_name>/durables

This web method retrieves a collection of durable status objects from a specific persistence store within a zone.

- <zone_name> in the URI is the cluster name.
- <stor_name> in the URI is the store name.

Example Requests

Example JSON Representation

```
curl -X GET http://<host>:<port>/api/v1/persistence/zones/<zone_name>/stores/<stor_name>/durables
```

```

{
  "durables": [
    {
      "name": "_routing_dur_from_C1_for_C2_in_Zone_ZFM",
      "store": "SFM",
      "cluster": "C1",
      "matcher": "[{\"absent_C2\":false}]",
      "routing-home": {

```

```

        "cluster": "C2",
        "store": "SFM"
    },
    "client_id": 1302208523,
    "durable_id": 3,
    "type": "standard-store-fwd",
    "dynamic": false,
    "subscribers": [
        {
            "sub_id": 8,
            "client_id": 1302208523,
            "last_acked": 143708,
            "last_dispatched": 125788,
            "last_send": 125789,
            "num_unacked": 100
        }
    ],
    "message_count": 100,
    "message_size": 24736
},
{
    "name": "_routing_dur_from_C1_for_C3_in_Zone_ZFM",
    "store": "SFM",
    "cluster": "C1",
    "matcher": "[{\"absent_C3\":false}]",
    "routing-home": {
        "cluster": "C3",
        "store": "SFM"
    },
    "client_id": 562949953442346,
    "durable_id": 4,
    "type": "standard-store-fwd",
    "dynamic": false,
    "subscribers": [
        {
            "sub_id": 12,
            "client_id": 562949953442346,
            "last_acked": 143679,
            "last_dispatched": 125788,
            "last_send": 125789,
            "num_unacked": 100
        }
    ],
    "message_count": 100,
    "message_size": 24736
},
{

```



```

    "name": "dur1",
    "store": "SFM",
    "cluster": "C1",
    "matcher": "{\"absent_C1\":false}",
    "template_name": "shdTemplate",
    "client_id": 0,
    "durable_id": 7,
    "type": "shared",
    "dynamic": true,
    "message_count": 200,
    "message_size": 47872
  }
]
}

```

Filtering

See [Filters for Durables](#).

GET persistence/zones/<zone_name>/stores/<stor_name>/durables/<dur_name>

This web method retrieves the status of a specific durable in a specific persistence store within a zone.

- <zone_name> in the URI is the zone name.
- <stor_name> in the URI is the store name.
- <dur_name> in the URI is the durable name.

Example Requests

```
curl -X GET http://<host>:<port>/api/v1/persistence/zones/<zone_name>/stores/<stor_name>/durables/<dur_name>
```

Example JSON Representation

```
{
  "name": "dur1",
  "store": "SFM",
  "cluster": "C1",
  "matcher": "{\"absent_C1\":false}",
  "template_name": "shdTemplate",
  "client_id": 0,
  "durable_id": 7,
  "type": "shared",
  "dynamic": true,
  "message_count": 200,
  "message_size": 47872
}
```

POST persistence/zones/<zone_name>/stores/<stor_name>

This web method sends a command to purge all durables in a specific store.

- <zone_name> in the URI is the zone name.
- <stor_name> in the URI is the store name.

Commands

| Command | Description |
|------------------------------|--|
| <code>{"cmd":"purge"}</code> | Purge messages from all durables in a store. |

Filtering

See [Filters for Durables](#).

Example Requests

```
curl http://<host>:<port>/api/v1/persistence/zones/<zone_name>/stores/<stor_name> -X POST -d '{"cmd":"purge"}'
```

POST persistence/zones/<zone_name>/stores/<stor_name>/durables

This web method can send a command to purge or delete selected durables, or delete all durables for a specific store within a specific zone.

- <zone_name> in the URI is the zone name.
- <stor_name> in the URI is the store name.

Commands

| Command | Description |
|---|--|
| <code>{"cmd":"purge_durables","args"}</code> | Purge messages from the durables supplied in args. |
| <code>{"cmd":"delete_durables","args"}</code> | Delete the durables supplied in args. |
| <code>{"cmd":"delete_all_durables"}</code> | Delete all durables from a store within a specific zone. |

Filtering

See [Filters for Durables](#).

Example Requests

```
curl http://<host>:<port>/api/v1/persistence/zones/<zone_name>/stores/<stor_name>/durables -X POST -d '{"cmd":"purge_durables", "args": [{"names": ["dur1","dur2"]}]}'
```

```
curl http://<host>:<port>/api/v1/persistence/zones/<zone_name>/stores/<stor_name>/durables -X POST -d '{"cmd":"delete_durables", "args": [{"names": ["dur1", "dur2"]}]}'
```

```
curl http://<host>:<port>/api/v1/persistence/zones/<zone_name>/stores/<stor_name>/durables -X POST -d '{"cmd":"delete_all_durables"}'
```

POST persistence/zones/<zone_name>/stores/<stor_name>/durables/<dur_name>

This web method sends a command to purge a specific durable.

The following form sends a command to purge all durables in a specific store: [POST persistence/zones/<zone_name>/stores/<stor_name>](#).

- <zone_name> in the URI is the zone name.
- <stor_name> in the URI is the store name.
- <dur_name> in the URI is the durable name.

Commands

| Command | Description |
|-----------------|--------------------------------|
| {"cmd":"purge"} | Purge messages from a durable. |

Filtering

See [Filters for Durables](#).

Example Requests

```
curl -X POST http://<host>:<port>/api/v1/persistence/zones/<zone_name>/stores/<stor_name>/durables/<dur_name> -d <json_cmd>
```

DELETE persistence/zones/<zone_name>/stores/<stor_name>/durables/<dur_name>

This web method deletes a specific durable from a specific persistence store from within a zone.

- <zone_name> in the URI is the zone name.
- <stor_name> in the URI is the store name.
- <dur_name> in the URI is the durable name.

Example Requests

```
curl -X DELETE
http://<host>:<port>/api/v1/realm/persistence/zones/<zone_
name>/stores/<stor_name>/durables/<dur_name>
```

Filtering

See [Filters for Durables](#).

Metrics and Monitoring Objects

The FTL server web API represents client metrics as a JSON object.

Example JSON Representation

```
{
  "clients": [
    {
      "id": 1026,
      "client_label": "FTL.srv1",
      "connect_time": 1585338198374,
      "series": [
        {
          "context": "queue_7f5e9866d9b0",
```

```

    "context_type": "queue",
    "type": "queue_discards",
    "description": "Queue Discards",
    "data": [
      {
        "sample": 641,
        "up_time": 540105,
        "value": 0,
        "start_time": 0
      }
    ]
  },
  ...

```

JSON Attributes

For types of metrics and their meaning, see [Catalog of Metrics](#) .

GET monitoring

The web method GET `monitoring` retrieves a collection of raw client metrics.

HTTP Parameters for Filtering

The complete set of current metrics for all clients can be very large. (The actual size of the set depends on the number of clients and their complexity.) You can filter the metrics by client ID.

| Syntax | Description |
|---------------------------------|---|
| <code>id=<ID_list></code> | Get metrics for a specific set of clients. Supply a comma-separated list of client ID numbers. |

Example Requests

```
curl -X GET http://<host>:<port>/api/v1/monitoring
```

Authenticating to FTL Server

FTL server can be configured with one or more authentication providers, allowing FTL server to understand basic authentication, mTLS authentication, oauth2 authentication, or all three. When accepting incoming connections, FTL server can accept different authentication modes for different connections, as long as an appropriate auth provider is configured.

In addition, each client and FTL server must be configured to authenticate itself to FTL servers. When making a connection to an FTL server, the client or server must choose exactly one of the possible authentication modes (basic, mTLS, or oauth2). For FTL server, this configuration (for outgoing connections) is independent of the auth provider configuration (for incoming connections). For more information, see [FTL Server Configuration Parameters](#)

Authenticating with Basic Authentication

FTL clients: Pass the username and password as properties to the realm connect call.

For example, in C API, pass `TIB_REALM_PROPERTY_STRING_USERNAME` and `TIB_REALM_PROPERTY_STRING_USERPASSWORD` to `tibRealm_Connect`

eFTL clients can pass the username and password as properties to the connect call.

For example, in C API, set the `username` and `password` options when calling `tibftl_Connect`.

FTL servers: in the yaml configuration file, set `user` and `password` in the `ftlserver.properties` section for each FTL server. See [Authenticating to other FTL Servers](#) in [FTL Server Configuration Parameters](#)

Administrative tools:

- For the REST API, include an “Authorization” header in HTTP requests. `<credentials>` is the base64 encoding of the string “`<username>:<password>`”.

```
Authorization: Basic <credentials>
```


- If using `tibftladmin`, specify the “--user” and “--password” command line parameters. See [FTL Administration Utility](#).

Authenticating with mTLS

FTL clients: pass the client cert, private key, and private key password as properties to the realm connect call.

For example, in C API, pass `TIB_REALM_PROPERTY_STRING_CLIENT_CERT`, `TIB_REALM_PROPERTY_STRING_CLIENT_PRIVATE_KEY`, and `TIB_REALM_PROPERTY_STRING_CLIENT_PRIVATE_KEY_PASSWORD` to `tibRealm_Connect`

eFTL clients: mTLS is not supported.

FTL servers: in the yaml configuration file, set `tls.client.cert`, `tls.client.private.key`, and `tls.client.private.key.password` in the `ftlserver.properties` section for each FTL server. See Authenticating to other FTL Servers in [FTL Server Configuration Parameters](#)

Administrative tools:

- mTLS is not supported for the UI or the eFTL REST API.
- For the FTL REST API, configure the TLS provider to present a client certificate when connecting to FTL server.
- If using `tibftladmin`, specify the `--tls.client.cert`, `--tls.client.private.key`, and `--tls.client.private.key.password` command line parameters. See [FTL Administration Utility](#)

Authenticating with OAuth 2.0

Following are the four options for FTL clients

- Client credentials grant: pass the oauth2 token endpoint, client id, client secret, and (if needed) trust file to the realm connect call. The FTL library will fetch and refresh the token as needed. For example, in C API, pass the following to `tibRealm_Connect`
 - `TIB_REALM_PROPERTY_STRING_OAUTH2_SERVER_URL`,
 - `TIB_REALM_PROPERTY_STRING_OAUTH2_CLIENT_ID`

- `TIB_REALM_PROPERTY_STRING_OAUTH2_CLIENT_SECRET`,
- `TIB_REALM_PROPERTY_STRING_OAUTH2_SERVER_TRUST_FILE`, if needed.
- Password credentials grant: in addition to the parameters for client credentials grant, pass a username and password to the realm connect call. The FTL library will fetch and refresh the token as needed. For example, in C API, in addition to the above parameters, pass `TIB_REALM_PROPERTY_STRING_USERNAME` and `TIB_REALM_PROPERTY_STRING_USERPASSWORD` to `tibRealm_Connect`.
- Long-lived access token: pass an oauth2 access token (a signed JWT) directly to the realm connect call. The token must be valid for the lifetime of the application. For example, in C API, pass `TIB_REALM_PROPERTY_STRING_OAUTH2_ACCESS_TOKEN` to `tibRealm_Connect`.
- Access token callback: set a callback in the properties object. The callback will be invoked by the FTL library whenever a new token is needed. For example, in C API, call `tibProperties_SetOAuth2TokenFetchCallback` and pass the properties object to `tibRealm_Connect`.

eFTL clients: there are four choices. Authentication with oauth2 is not supported for the javascript or python API.

- Client credentials grant: pass the oauth2 token endpoint, client id, client secret, and (if needed) trust file to the connect call. The eFTL library will fetch and refresh the token as needed. For example, in C API, set the `oAuth2ServerUrl`, `oAuth2ClientId`, `oAuth2ClientSecret`, and `oAuth2TrustStore` options when calling `tibefTL_Connect`.
- Password credentials grant: in addition to the parameters for client credentials grant, pass a username and password to the connect call. The eFTL library will fetch and refresh the token as needed. For example, in C API, in addition to the above parameters, set “username” and “password” when calling “`tibefTL_Connect`”.
- Long-lived access token: pass an oauth2 access token (a signed JWT) directly to the connect call. The token must be valid for the lifetime of the application. For example, in C API, set the `oAuth2AccessToken` option when calling `tibefTL_Connect`.
- Access token callback: pass a callback to the connect call. The callback will be invoked by the eFTL library whenever a new token is needed. For example, in C API, set the “`oAuth2TokenFetchCallback`” and “`oAuth2TokenFetchCallbackArg`” options when calling `tibefTL_Connect`.

FTL servers: there are three choices. See “Authenticating to other FTL Servers” in [FTL Server Configuration Parameters](#)

- Client credentials grant: in the yaml configuration file, set `oauth2.svr.endpoint.token`, `oauth2.svr.client.id`, `oauth2.svr.client.secret` and (if needed) `oauth2.provider.trust.file` in the “ftlserver.properties” section for each FTL server.
- Password credentials grant: in addition to the parameters for client credentials grant, set “user” and “password” in the “ftlserver.properties” section for each FTL server.
- Long-lived access token: in the yaml configuration file, set “oauth2.access.token” (a signed JWT) in the “ftlserver.properties” section for each FTL server. The token must be valid for the lifetime of the FTL server.

Administrative tools:

- Single sign-on may be configured for the UI. See “Single Sign-On with OAuth2” in [FTL Server Configuration Parameters](#)
- For the REST API, include an “Authorization” header in HTTP requests. <token> must be a signed JWT.

```
Authorization: Bearer <token>
```

- If using “tibftladmin”, specify the “--oauth2.token” command line parameter (a signed JWT). See [FTL Administration Utility](#).

Catalog of Metrics

Administrators can use external tools to monitor processes and analyze their metrics.

TIBCO Messaging Monitor provides an out-of-the-box solution for monitoring TIBCO FTL® (FTL). The TIBCO Messaging Monitor provides the core components to gather monitoring statistics into a centralized back end and provides a means to visualize these statistics into meaningful dashboards.

Catalog of Metrics

The catalogs that follow present the available monitoring metrics.

ID

A unique ID denotes each type of counter. When subscribing to the monitoring stream, content matchers can refer to these ID values using API constants.

Metric Type

A human-readable string represents each type of counter. These strings can appear in databases and user interfaces.

Description

This column explains the meaning of each type counter and the quantity it measures.

Tracking

This column describes the object that each counter tracks. For example, a counter could collect data about the operation of an endpoint, a transport, an event queue, a client process, or a persistence store.

Catalog of Application Metrics

These tables describe metrics that you can use to assess the operation of a client application or service.

TIBCO FTL software gathers metrics for clients (applications), event queues, transports, and endpoints. It can also gather metrics for subscribers, publishers, and connections.

Unless otherwise specified, a metric reports a counter's value at the end of each sample interval, and the counter resets to zero for the next interval.

Each data sample submessage contains a type field. Its value is one of the type ID numbers in these tables, denoting a metric type.

Application Metrics

These metrics measure the operation of a client.

| ID | Metric Type | Description |
|----|-------------------------------|---|
| 51 | Dynamic Formats | <p>Number of distinct dynamic formats registered within the client.</p> <p>Creating a message with a dynamic format can increment this counter. Receiving an inbound message with a dynamic format can increment this counter.</p> <p>This counter can decrease as the client library automatically unregisters unused formats.</p> |
| 70 | Resident Memory Set Size | Current size in kilobytes of an application's working memory set. |
| 71 | Peak Resident Memory Set Size | Maximum size in kilobytes, since the client started, of an application's working memory set. |
| 72 | Process VM | Current size in kilobytes of an application's virtual memory set. |
| 73 | User CPU Time | Total user CPU time in microseconds, that is, time executing the client's object code since the client started. |
| 74 | System CPU Time | Total system CPU time in microseconds, that is, time executing operating system calls from the client since the client started. |

| ID | Metric Type | Description |
|----|----------------|--|
| 75 | Total CPU Time | Total CPU time in microseconds, that is, the sum of time executing the client's object code and its operating system calls since the client started. |

Endpoint Metrics

These metrics report activity on an endpoint of a client.

Endpoint

| ID | Metric Type | Description | Tracking |
|----|-------------------------|---|--------------|
| 67 | Store Mismatch Messages | <p>Number of message flows that result from persistence store mismatch.</p> <p>Any non-zero value indicates a store mismatch misconfiguration.</p> <p><i>Store mismatch</i> is a misconfiguration in which a direct path transport connects two endpoints that are associated with two <i>different</i> persistence stores.</p> | Per endpoint |

Transport

| ID | Metric Type | Description | Tracking |
|----|--------------------|--|---------------|
| 12 | Bytes Received | Inbound data bytes on a transport in the client. | Per transport |
| 11 | Bytes Sent | Outbound data bytes on a transport in the client. | Per transport |
| 31 | Data Lost | <p>Inbound data loss events on a transport in the client.</p> <p>It is not possible to measure the magnitude of the events in bytes.</p> | Per transport |
| 32 | Format Unavailable | Messages with an unrecognized format carried on a transport in the client. | Per transport |

i Note: Bytes sent and received need not correlate exactly with messages sent and received on corresponding endpoints. For example, filtering and optimizations can decouple these metrics.

Queues

These metrics report activity in each individual event queue in a client.

| ID | Metric Type | Description |
|----|----------------|--|
| 41 | Queue Backlog | Maximum number of messages in an event queue in the client during the sample interval. |
| 42 | Queue Discards | Inbound messages discarded by a queue in the client. |

Metrics for Multicast and RUDP

TIBCO FTL software gathers additional metrics for transports and connections that use UDP-based protocols, such as multicast and RUDP transports.

Unless otherwise specified, a metric reports a counter's value at the end of each sample interval. Transport metrics are cumulative since the transport established its bus. Connection metrics are cumulative since the connection's start time.

Multicast and RUDP Transports

| ID | Metric Type | Description | Tracking |
|----|------------------|--|---------------|
| 61 | Packets Sent | Outbound data packets on a multicast or RUDP transport. This counter includes <i>all</i> outbound data packets, including original data packets and retransmitted data packets. | Per transport |
| 62 | Packets Received | Inbound data packets on a multicast or RUDP transport. | Per transport |

| ID | Metric Type | Description | Tracking |
|----|-----------------------|--|---------------|
| | | This counter includes <i>all</i> inbound data packets, including original data packets and retransmitted data packets. | |
| 63 | Packets Retransmitted | Outbound packets retransmitted on a multicast or RUDP transport. If the transport retransmits a packet several times, it increments this counter for each retransmission. | Per transport |
| 64 | Packets Missed | Number of missed inbound packets on a multicast or RUDP transport. | Per transport |
| 65 | Packets Lost Outbound | Number of outbound packets discarded on a multicast or RUDP transport. | Per transport |
| 66 | Packets Lost Inbound | Number of inbound packets deemed lost on a multicast or RUDP transport. The transport accounts a packet as lost after its reliability constraints expire. | Per transport |

Multicast Connection

| ID | Metric Type | Description | Tracking |
|-------|---------------------------|--|----------------------|
| 90040 | Send Backlog Maximum Size | The largest data backlog (in bytes) during this monitoring interval (not cumulative). For background information see " Send: Blocking versus Non-Blocking " in the TIBCO FTL documentation. | Per multicast sender |

Catalog of Persistence Metrics

This table describes metrics that you can use to assess the operation of persistence stores.

Each persistence service is itself an application client, and tracks application metrics; see [Catalog of Application Metrics](#).

The sampling interval for persistence metrics is fixed at 2 seconds. You cannot change this interval.

See the [API Documentation](#), `monitor.h` and other metric sections for counters that are useful to you.

Store Metrics

| ID | Metric Type | Description | Tracking |
|------|--------------------|--|-----------|
| 1000 | Message Count | Number of messages in a store at the end of the sample interval. | Per Store |
| 1001 | Message Size | Storage (in bytes) that messages occupy in a store at the end of the sample interval. | Per Store |
| 1006 | Swap Message Count | When <code>disk_swap</code> is true, this is the number of messages in memory, else 0. | Per Store |
| 1007 | Swap Message Size | When <code>disk_swap</code> is true, this is the byte count of all messages in memory, else 0. | Per Store |
| 1008 | Store Byte Limit | Configured store memory byte limit. | Per Store |

| ID | Metric Type | Description | Tracking |
|------|---------------------|--|-----------|
| 1009 | Store Message Limit | Configured store message count limit. | Per Store |
| 1010 | Expiration Count | Counts the number of message expiration events in a store for the lifetime of the current leader. This includes messages discarded due to message TTL (Time-To-Live), durable message limit, durable byte limit, or durable max delivery. The count is incremented when any message is expired by any durable. The count will increment multiple times if a given message is expired by multiple durables. | Per Store |
| 2000 | Durable Count | Number of durables in a | Per Store |

| ID | Metric Type | Description | Tracking |
|------|---------------------|---|-------------------------|
| | | store at the end of the sample interval. | |
| 3013 | Disk Size | Allocated disk space for this server's persistence data. This is a high-water mark for disk space in use. | Per Persistence Service |
| 3014 | Backlog Limit | Soft limit on memory for pending messages. | Per Persistence Service |
| 3015 | Backlog Size | Memory in use for pending messages. | Per Persistence Service |
| 3016 | Disk Backlog Size | Memory in use for pending disk writes. | Per Persistence Service |
| 3017 | Inbound Byte Count | Byte count of messages received from client applications or other persistence clusters. | Per Store |
| 3018 | Outbound Byte Count | Byte count of messages | Per Store |

| ID | Metric Type | Description | Tracking |
|------|------------------|---|-------------------------|
| | | delivered to client applications or other persistence clusters. | |
| 3019 | Disk In Use Size | Disk space currently in use for this server's persistence data. | Per Persistence Service |
| 3020 | Disk Capacity | Capacity of the disk volume containing the persistence data directory. | Per Persistence Service |
| 3021 | Disk Usage Limit | Attempt to limit total space used (including non-FTL data) to this value, based on disk capacity and max disk fraction. | Per Persistence Service |
| 3022 | Disk Available | Available space on the disk volume containing the persistence data directory. | Per Persistence Service |
| 3023 | Disk Used | Total space used on the disk volume | Per Persistence Service |

| ID | Metric Type | Description | Tracking |
|------|---------------------------|--|-------------------------|
| | | containing the persistence data directory. | |
| 3024 | Disk Size (Swap) . | Allocated disk space for this server's swap data | Per Persistence Service |
| 3025 | Disk In Use Size (Swap) . | Disk space currently in use for this server's swap data | Per Persistence Service |
| 3026 | Disk Capacity (Swap) . | Capacity of the disk volume containing the swap data directory. | Per Persistence Service |
| 3027 | Disk Usage Limit (Swap) | Attempt to limit total space used for swap (including non-FTL data) to this value, based on disk capacity and max disk fraction. | Per Persistence Service |
| 3028 | Disk Available (Swap) | Available space on the disk volume containing the swap data directory. | Per Persistence Service |

| ID | Metric Type | Description | Tracking |
|------|--------------------------------|---|-------------------------|
| 3029 | Disk Used (Swap) | Total space used on the disk volume containing the swap data directory. | Per Persistence Service |
| 3030 | Disk Write Count | Total number of write calls issued to the persistence data file. | Per Persistence Service |
| 3031 | Disk Bytes Written | Total bytes written to the persistence data file. | Per Persistence Service |
| 3032 | Disk Read Count | Total number of read calls issued to the persistence data file. | Per Persistence Service |
| 3033 | Disk Bytes Read | Total bytes read from the persistence data file. | Per Persistence Service |
| 3034 | Disk Fdatasync Count | Total number of fdatasync calls issued to the persistence data file. | Per Persistence Service |
| 3035 | Replica Round-Trip Time: Count | Sample count of all round-trip | Per Persistence Service |

| ID | Metric Type | Description | Tracking |
|------|--------------------------------|--|-------------------------|
| | | times to a replica. The metric context identifies the replica. | |
| 3036 | Replica Round-Trip Time: Max | <p>Maximum round-trip time to a replica over the last sample interval, in microseconds. The metric context identifies the replica.</p> <p>If disk persistence is configured, the round-trip time includes the sync write to disk at the replica.</p> | Per Persistence Service |
| 3037 | Replica Round-Trip Time: Total | <p>Total of all round-trip times to a replica, in microseconds. The metric context identifies the replica.</p> <p>If disk persistence is configured, the</p> | Per Persistence Service |

| ID | Metric Type | Description | Tracking |
|------|---|---|-------------------------|
| | | round-trip time includes the sync write to disk at the replica. | |
| 3038 | Replica Round-Trip Bytes Per Op: Count | Sample count of all round-trips to a replica. The metric context identifies the replica. | Per Persistence Service |
| 3039 | Replica Round-Trip Bytes Per Op: Max | Maximum bytes per round-trip to a replica over the last sample interval. The metric context identifies the replica. | Per Persistence Service |
| 3040 | Replica Round-Trip Bytes Per Op: Total | Total byte count for all round-trips to a replica. The metric context identifies the replica. | Per Persistence Service |
| 3041 | Replica Round-Trip Time (Async Disk): Count | Sample count of all round-trip times to a replica. The metric context identifies the replica. For async disk | Per Persistence Service |

| ID | Metric Type | Description | Tracking |
|------|---|---|-------------------------|
| | | persistence only. | |
| 3042 | Replica Round-Trip Time (Async Disk): Max | Maximum round-trip time to a replica over the last sample interval, in microseconds. The metric context identifies the replica. For async disk persistence only. | Per Persistence Service |
| 3043 | Replica Round-Trip Time (Async Disk): Total | Total of all round-trip times to a replica, in microseconds. The metric context identifies the replica. For async disk persistence only. The round-trip time only includes an async write to disk at the replica. | Per Persistence Service |
| 3044 | Replica Round-Trip Bytes Per Op (Async Disk): Count | Sample count of all round-trips to a replica. The metric context identifies the | Per Persistence Service |

| ID | Metric Type | Description | Tracking |
|------|---|---|-------------------------|
| | | replica. | |
| 3045 | Replica Round-Trip Bytes Per Op (Async Disk): Max | Maximum bytes per round-trip to a replica over the last sample interval. The metric context identifies the replica. For async disk persistence only. | Per Persistence Service |
| 3046 | Replica Round-Trip Bytes Per Op (Async Disk): Total | Total byte count for all round-trips to a replica. The metric context identifies the replica. For async disk persistence only | Per Persistence Service |
| 3047 | Disk Write Latency: Count | Sample count of all sync writes to disk. | Per Persistence Service |
| 3048 | Disk Write Latency: Max | Maximum latency of a sync disk write over the last sample interval, in microseconds. | Per Persistence Service |
| 3049 | Disk Write Latency: Total | Total latency of all sync disk | Per Persistence Service |

| ID | Metric Type | Description | Tracking |
|------|-----------------------------------|---|-------------------------|
| | | writes, in microseconds. | |
| 3050 | Disk Write Bytes Per Batch: Count | Sample count of all sync writes to disk. | Per Persistence Service |
| 3051 | Disk Write Bytes Per Batch: Max | Maximum bytes in a sync disk write over the last sample interval. | Per Persistence Service |
| 3052 | Disk Write Bytes Per Batch: Total | Total bytes in all sync disk writes. | Per Persistence Service |

Catalog of eFTL Metrics

This table describes metrics that you can use to assess the operation of eFTL channels.

Each TIBCO eFTL service is itself an application client, and tracks application metrics: for example, metrics for the server's endpoints, and metrics for the transports that implement those endpoints. See [Catalog of Application Metrics](#).

The sampling interval for eFTL metrics is fixed at 2 seconds. You cannot change this interval.

eFTL Metrics

| ID | Metric Type | Description | Tracking |
|------|--------------------|--|-------------|
| 4000 | Connection Count | Number of eFTL client connections on a channel at the end of the sample interval. | Per channel |
| 4001 | Subscription Count | Open subscriptions from eFTL clients on a channel at the end of the sample interval. | Per channel |

| ID | Metric Type | Description | Tracking |
|------|-----------------------------|---|-------------|
| 4002 | Inbound eFTL Message Count | Messages inbound from eFTL clients on a channel. | Per channel |
| 4003 | Outbound eFTL Message Count | Messages outbound to eFTL clients on a channel. | Per channel |
| 4010 | Discarded Message Count | <p>Data loss from publishers: the number of messages that the server discarded on a channel.</p> <p>Discards can occur because a publisher sent a message that exceeded the maximum message size.</p> | Per channel |
| 4011 | Dataloss Message Count | <p>Data loss to subscribers: the number of messages that the server discarded on a channel.</p> <p>Discards can occur because a subscriber's queue exceeded the maximum outbound backlog.</p> | Per channel |
| 4012 | Connection Pending Count | Number of new eFTL client connections on a channel that are not yet fully initialized at the end of the sample interval. | Per channel |
| 4013 | Connection Suspended Count | Number of eFTL client connections disconnected by a temporary network outage that have not yet automatically reconnected nor fully disconnected at the end of the sample interval. | Per channel |
| 4018 | Cumulative Connection Count | Counts all eFTL client or REST API successful connections to an eFTL channel for the given eFTL server. | Per channel |

Catalog of FTL Server and Service Metrics

This table describes metrics that you can use to assess the operation of the FTL server and services.

FTL Server Metrics

The sampling interval for FTL server metrics is fixed at 2 seconds. You cannot change this interval.

| ID | Metric Type | Description |
|------|------------------------|---|
| 5009 | Client Connect Count | Cumulative number of client connections since the FTL server process started. |
| 5010 | Client Reconnect Count | <p>Number of clients that reconnected to the FTL server during the sample interval.</p> <p>A spike in this metric could indicate a network connectivity issue between the FTL server and its clients.</p> |
| 5011 | Send to Inbox Failures | <p>Number of undelivered protocol messages that the FTL server sent to a client.</p> <p>A spike in this metric usually indicates that several clients abruptly exited.</p> |

Transport Bridge Metrics

Each bridge is itself an application client, and tracks application metrics; see [Catalog of Application Metrics](#).

The sampling interval for bridge metrics is fixed at 2 seconds. You cannot change this interval.

| ID | Metric Type | Description | Tracking |
|------|---------------|--|------------|
| 6000 | Bridge Active | <p>1 indicates that the bridge actively forwards messages.</p> <p>0 indicates that the bridge is a backup in standby mode.</p> | Per bridge |

Groups of Applications

Applications can use the group facility to coordinate fault-tolerant operation, or to distribute operating roles among application processes.

These topics present the group facility for administrators. Before reading them, first read the corresponding material in the topics “Groups of Applications” in [TIBCO FTL Development](#).

Introduction to Groups

A set of application processes can join a group. Each member of a group receives an *ordinal*, that is, a number representing its current position within the group. An application can specify a weight in a group member process to influence the ordinal assigned. Application program logic uses the ordinal to select one of several possible operating roles.

A *group service* tracks the group members as processes join and leave the group, and as they disconnect from and reconnect to the FTL server. At each of these events, the group facility revises ordinals to restore a one-to-one mapping between n members and the positive integers $[1, n]$.

Developers are responsible for the names of groups and for the API calls that join groups. Administrators are responsible for configuring group communication.

Group Communication

Within each member process, the group library connects to the group service within the FTL server. While connected, the member and the group service exchange periodic protocol messages. When disconnected, the member initiates reconnection.

For best performance, run all group members in nearby geographical proximity to one another.

Group protocol communication depends on application definitions that define endpoints, on transports that connect members to the group service, and on the formats of group protocol messages.

Group Service

Beyond configuring group communication in the realm, the group service does *not* require any further explicit administrative action.

The FTL server always provides a group service. You can neither disable it, configure it, nor purge it.

The group service and all the group members that it coordinates must run within one local area network. Do *not* use the group facility across routers and WANs.

Fault Tolerance of the Group Service

The group service is automatically fault-tolerant, if and only if the FTL server is fault-tolerant.

FTL servers automatically start a group service module alongside each realm service.

Group services within a local set of FTL servers automatically coordinate for fault-tolerant operation, and redirect group members to the currently active group service.

A fault-tolerant group service operates with local scope. That is, if an application client of an FTL server could failover to another FTL server in the same local network, then a group member in that client could failover to the corresponding group service in that FTL server.

For example, a fault-tolerant set of primary FTL servers implies a fault-tolerant set of group services that serves the combined clients of those primaries. Similarly, a fault-tolerant set of satellite FTL servers implies a fault-tolerant set of group services that serves the combined clients of those satellites. However, group members cannot cooperate across different local networks.

Groups Status Table

The Groups status table presents the status of application groups and their members.

Columns

| Column | Description |
|---------------------|---|
| Status | For descriptions of these values, see Client Status . |
| Name | The group name string. |
| Activation Interval | The activation interval of the group. For details, see the Development guide, Activation Interval . |
| Ordinal Members | A list of client IDs that are ordinal members of the group. |
| Observer Members | A list of client IDs that are observer members of the group. |

Members Sub-Tables

Clicking a row of the groups status table opens two sub-tables, listing the group's ordinal members and observer members.

| Column | Description |
|-------------|--|
| Ordinal | The ordinal currently assigned to the member process. |
| ID | The member's client ID. |
| Member Type | <ul style="list-style-type: none"> • Full Member Functioning member of the group. Receives ordinal updates and status updates. A full member has a specified member descriptor. • Ordinal Member Functioning member of the group. Receives <i>only</i> ordinal updates. Ordinal members do not have a specified member descriptor. • Observer Member Not a functioning member of the group. Receives <i>only</i> status updates. |

| Column | Description |
|-------------------|---|
| Member Descriptor | Clients may supply a JSON descriptor when they join a group. The descriptor is an FTL message object that uniquely identifies the member process. |

Transport Bridge

A *transport bridge* is an FTL service that efficiently forwards messages among sets of transport buses.

Functionality

You can configure a bridge with two or more terminals, called *transport sets*. You can configure each transport set with one or more transports.

A transport bridge forwards *all* messages in *all* directions. Whenever a message arrives on any one of its transports, it immediately republishes that message on *all* of the transports in *all* of the *other* sets. However, it does *not* forward the message on other transports within the same set.

The transports can be of the same type, or they can be of different transport protocols. For example, you can bridge between shared memory and TCP transports.

A transport bridge operates completely within a single realm. It forwards messages among transports defined within the realm, but it *cannot* forward messages between two different realms.

Transport bridges can forward one-to-many messages and one-to-one messages.

Transport bridges support one-to-one messages on transports that support inbox abilities.

You can arrange two or more bridge processes for fault-tolerant operation. Although this feature ensures automatic and quick recovery, failover might leave some messages unforwarded.

Design Features

Low latency is a key feature of the transport bridge. The transport bridge implementation is extremely fast and efficient. It forwards packets immediately, even before all the packets of a message have arrived.

Transport bridges are transparent to your application programs. Clients do not need to do anything different to communicate across a bridge, nor can they detect whether a bridge mediates between them.

Latency

Although the transport bridge is efficient, it still adds an extra layer between sending and receiving processes, which necessarily increases message latency.

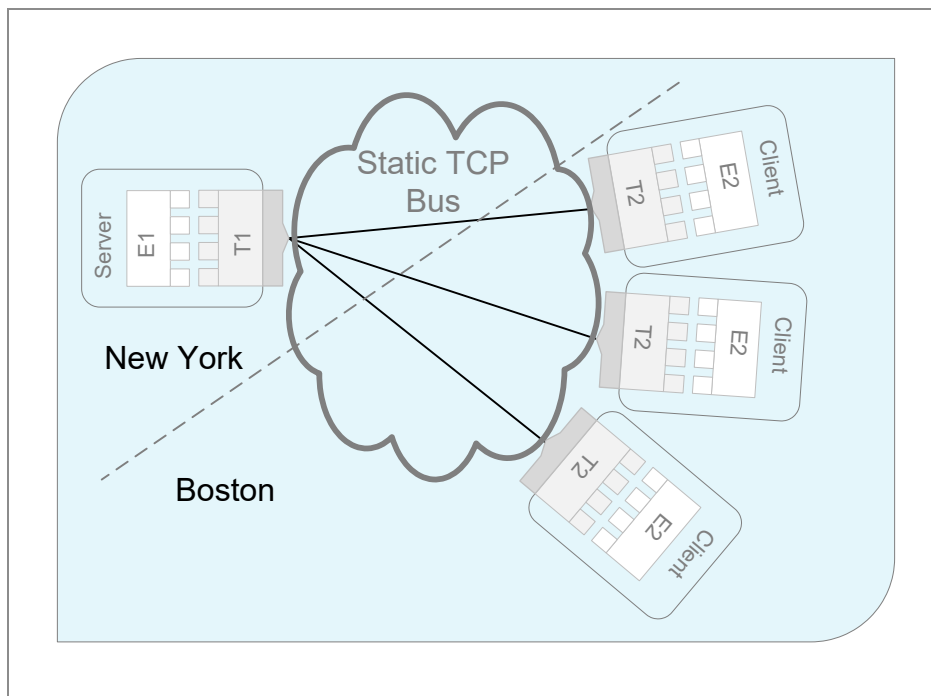
In some situations, sending on several transports could result in less additional latency than a bridge, at the cost of limiting maximum throughput. To optimize results for your specific application needs and resources, test empirically. For more information, see [Multiple Transports and Serial Communications](#).

Transport Bridge Use Cases

Transport Bridge to Shift Fanout for Efficiency

You can use a transport bridge to shift fanout closer to subscribers, for more efficient use of WAN bandwidth.

Figure 24: Fanout over WAN (without Transport Bridge)

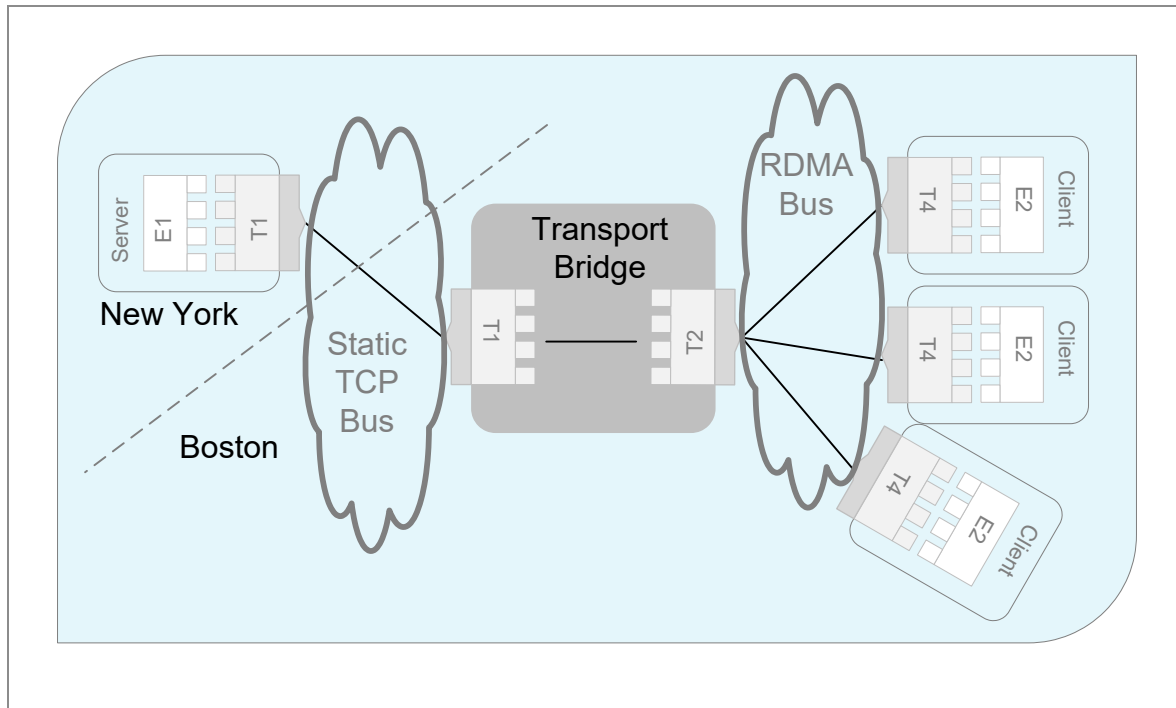


For example, in the first diagram (preceding), a publisher in New York communicates with several subscribers in Boston using a static TCP transport. In this situation, each subscriber

initiates a separate connection to the publisher, and each message crosses the WAN repeatedly (once per connection). This duplication of data over a slow or expensive link could be undesirable.

Instead consider the arrangement in the second diagram (following), in which a transport bridge in Boston connects to the New York publisher only once using a static TCP transport over a WAN, and forwards messages to many subscribers using LAN bandwidth.

Figure 25: Transport Bridge: Fanout after WAN

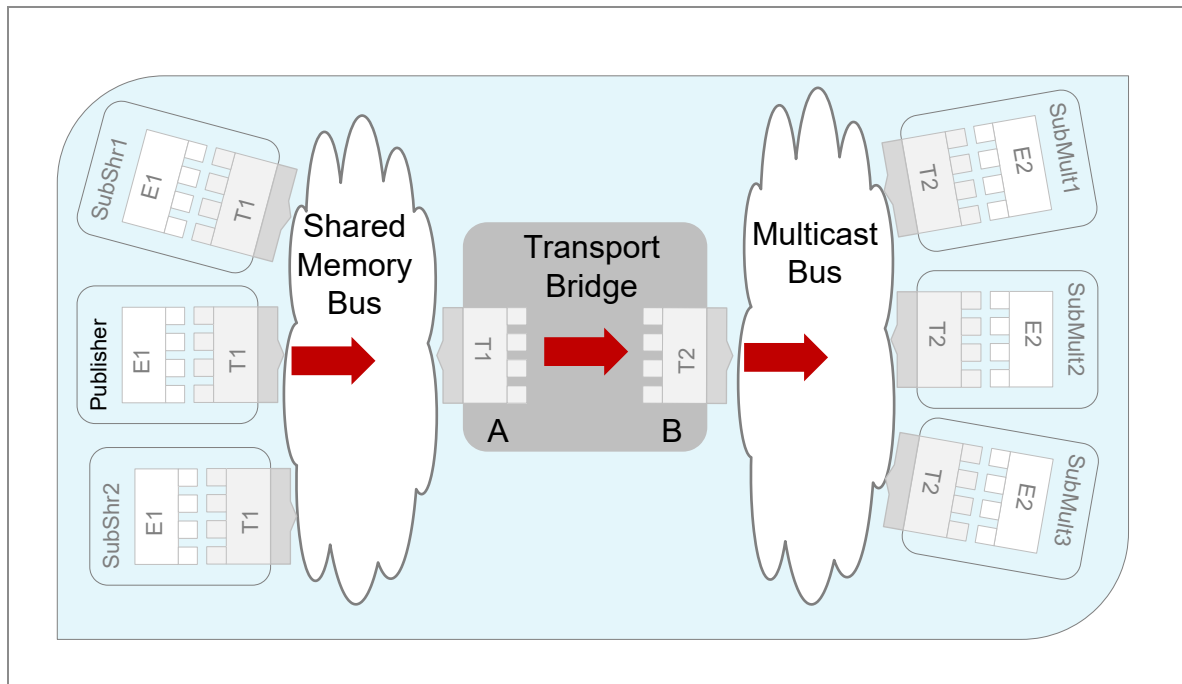


Transport Bridge to Extend beyond a Transport

You can use a transport bridge to extend the capabilities of a transport.

For example, in the following diagram, a publisher communicates with several subscribers on a single host computer using a shared memory transport T1. You can communicate with subscribers on other computers across a LAN by configuring a multicast transport T2, and a transport bridge between T1 and T2. The transport bridge must run on the computer that hosts the shared memory segment of T1.

Figure 26: Transport Bridge: Extending Capabilities

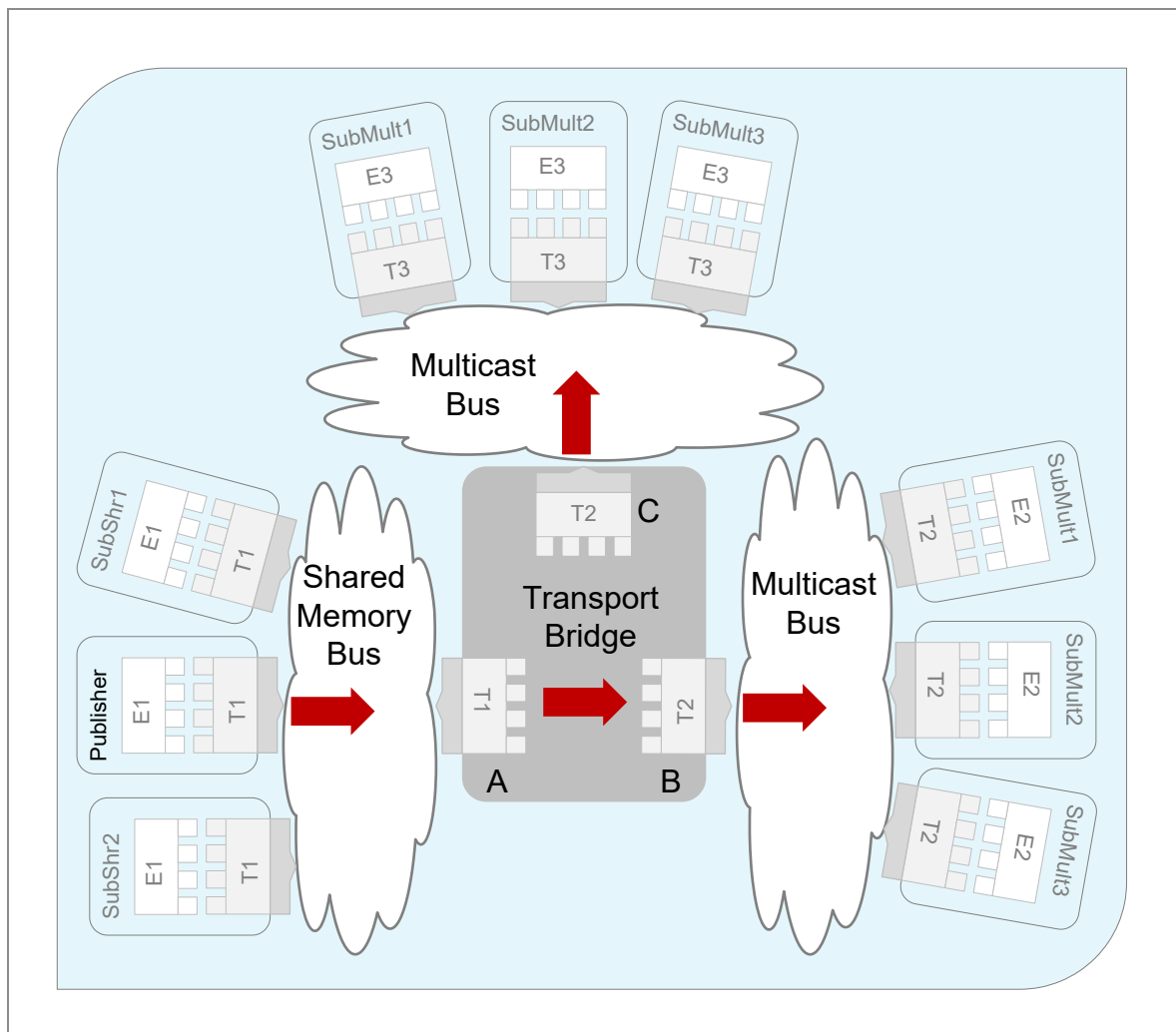


Transport Bridge: Multiple Transports

You can also configure a bridge with several transports in a transport set. (In contrast, previous bridge examples, the bridge forwards between two transport sets, each containing only one transport.)

For example, the following diagram shows only one multicast group in transport set B, but consider that an enterprise might communicate over *several* such multicast groups. To forward messages on all of them, the bridge requires a separate transport for each multicast send group. You can configure the bridge to include *all* of those multicast transports in transport set B.

Figure 27: Transport Bridge: Extending Capabilities



A bridge does *not* forward messages among the several transports in a transport set, only from the transport set where the message arrives, to all the transports in all the other transport sets. Consider the implications in the context of the current example.

- When a message arrives on T1, the bridge forwards it on all the transports in transport set B.
- When a message arrives on any one of the multicast transports in B, the bridge does *not* forward the message on any of the other multicast transports in B. It forwards only on T1, which is in transport set A.

The transports in a transport set need not be of the same transport protocol.

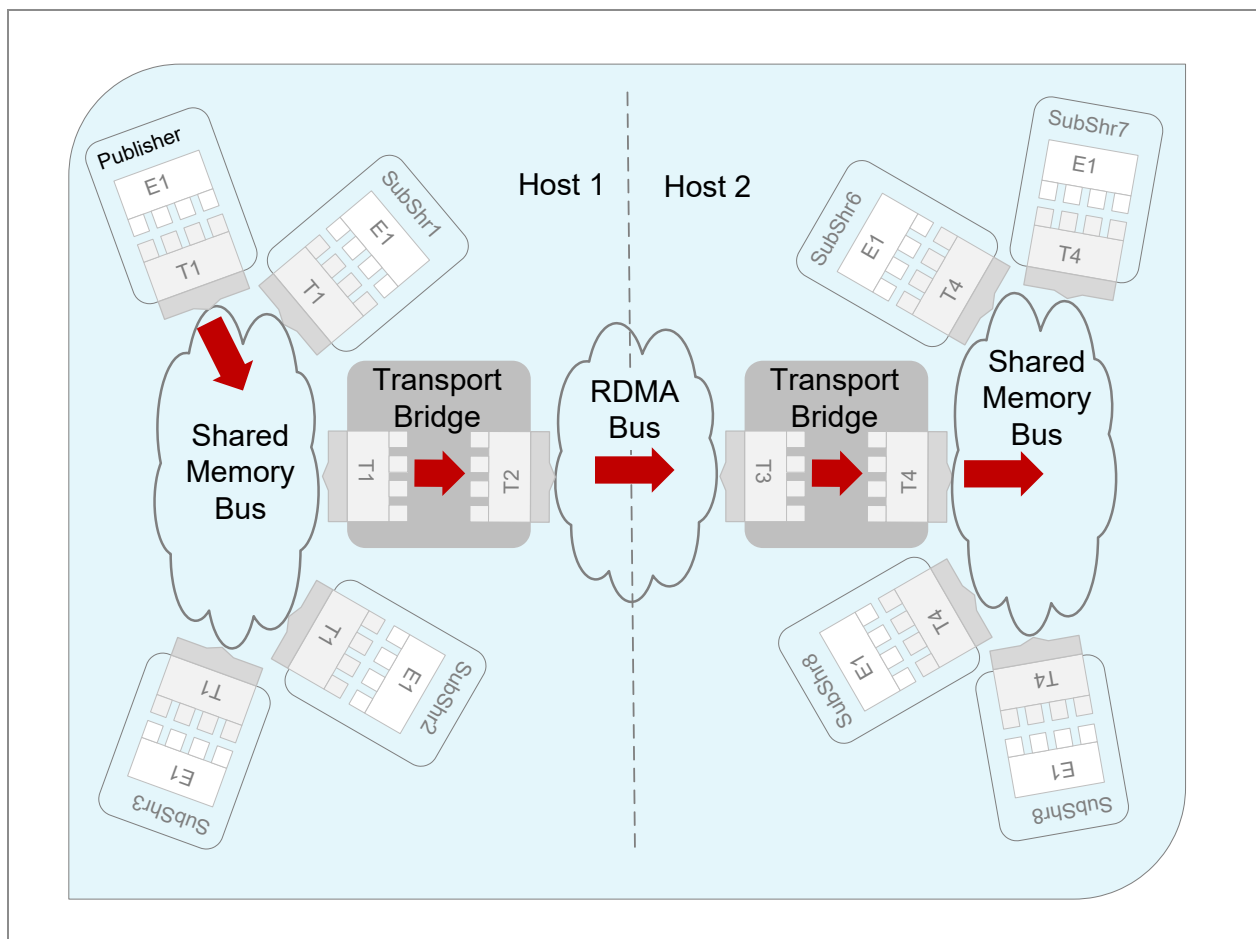
Transport Bridge to Cross the Boundary of Shared Memory

Access to a shared memory transport is limited to processes running on the computer that hosts the transport's shared memory segment. To escape this natural boundary, you can arrange two transport bridges in series.

The following diagram depicts such an arrangement, connecting shared memory transports on separate host computers.

One FTL server must provide a transport bridge on the computer that hosts the shared memory segment of T1, while another FTL server must provide a transport bridge on the computer that hosts the shared memory segment of T4.

Figure 28: Transport Bridge: Connecting Shared Memory on Two Computers



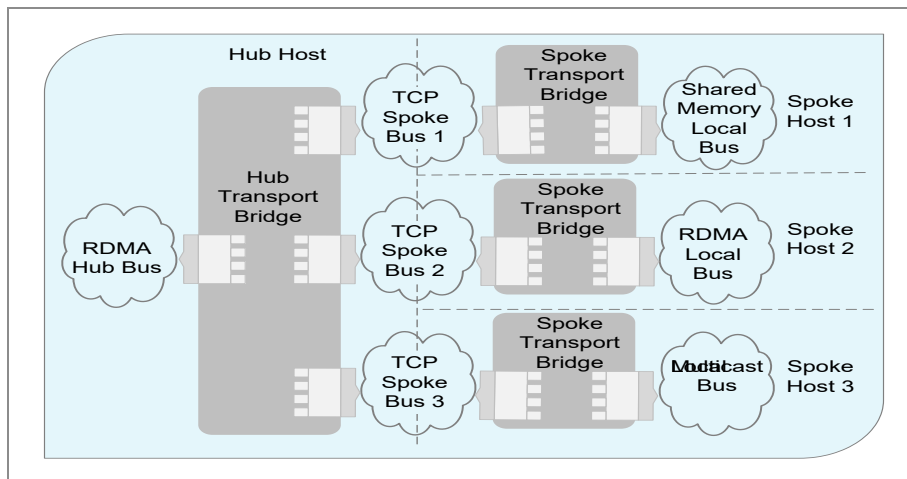
See Also: [Latency](#)

Transport Bridge to Isolate Spokes

Transport bridges can connect partner enterprises that must cooperate, and yet remain separate.

The following diagram shows a central control hub for the main enterprise, with spokes to partner enterprises. Although each partner enterprise must share data with the main hub, the partners must *not* share data with one another.

Figure 29: Transport Bridge: Common Hub with Mutually-Isolated Spokes



This use case is interesting in two respects:

- It is an example of a complex bridge topology (see [Hub-and-Spoke Bridges](#)).

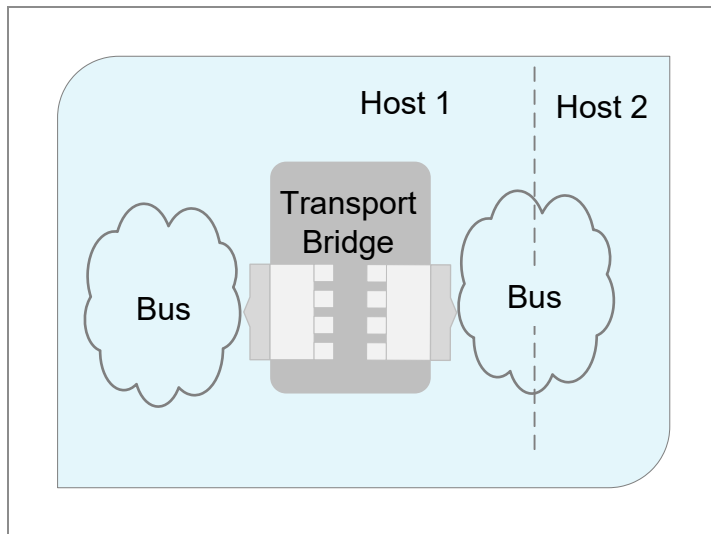
Transport Bridge Topologies

Transport bridges support only the three topologies presented in the topics that follow.

Any topology that includes any kind of loop is prohibited.

One Bridge

The basic topology is a single bridge between two transports.

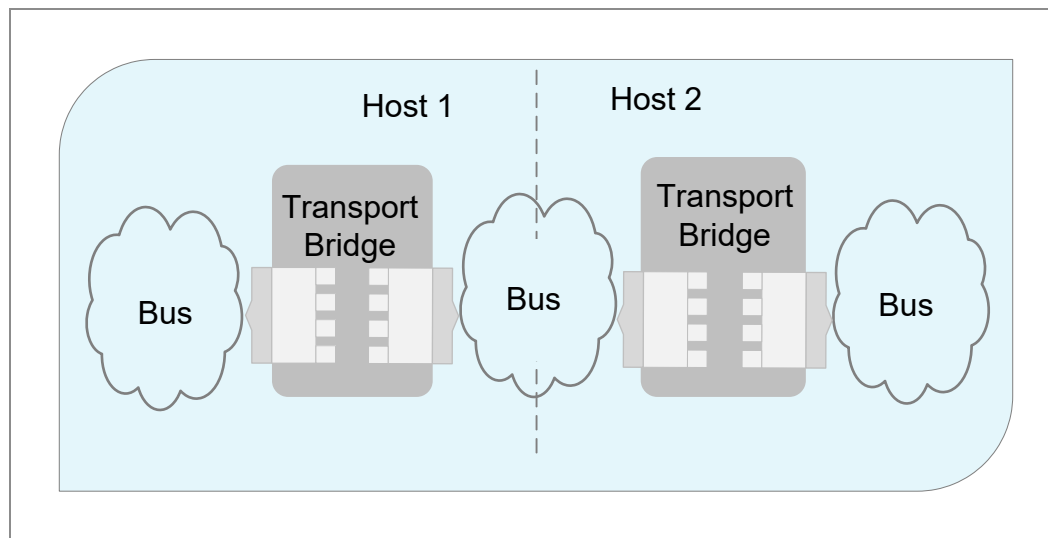
Figure 30: Transport Bridge Topology: One Bridge

You may extend this topology by adding transport sets, or by adding transports to the transport sets.

Two Serial Bridges

Topologies with two serial bridges are useful for bridging shared memory transports on different hosts.

For more information about this use case, see [Transport Bridge to Cross the Boundary of Shared Memory](#).

Figure 31: Transport Bridge Topology: Two Bridges

You may extend this topology by adding transport sets, or by adding transports to the transport sets.

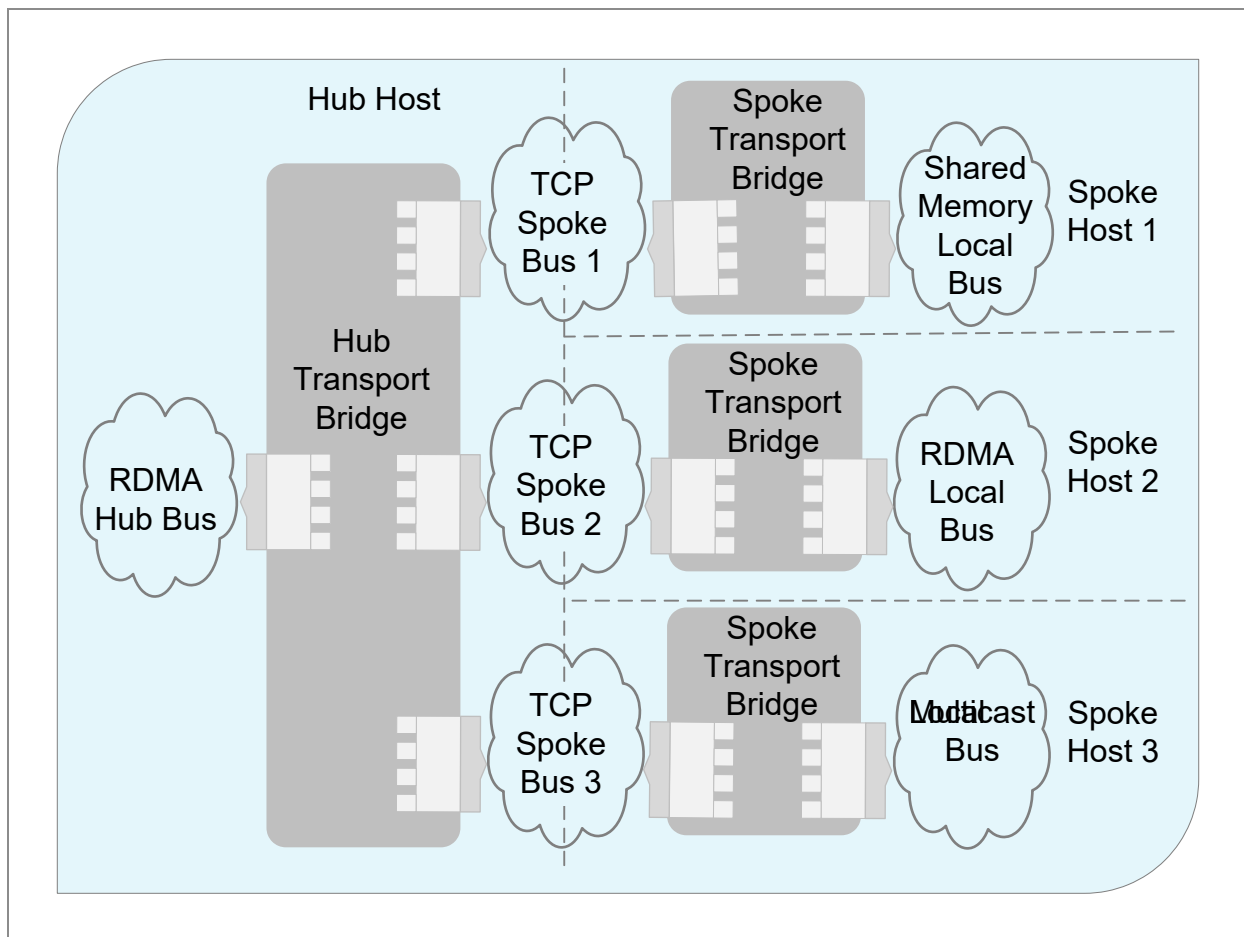
You may *not* extend this topology by adding a third serial bridge.

Hub-and-Spoke Bridges

Topologies featuring hub-and-spoke bridges are useful for connecting partner enterprises, especially at remote sites.

For more information, see the use case [Transport Bridge to Isolate Spokes](#).

Figure 32: Transport Bridge Topology: Hub-and-Spoke Bridges



You may extend this topology by adding transport sets, or by adding transports to the transport sets. You may extend this topology by adding more spokes, each consisting of a spoke bus, a spoke bridge, and a local bus.

You may *not* extend this topology by adding a third serial bridge, neither to the hub bus, nor to any local bus. You may *not* extend this topology by adding a second branching spoke bridge to any spoke bus.

Transport Bridge Restrictions

Although transport bridges are flexible, these restrictions apply.

- Transport bridges support only bidirectional transports. For example, a bridge cannot support a multicast transport configured with a send group but without a listen

group, nor vice versa.

- Transport bridges support only non-blocking transports. A transport that blocks could impede data flow through the bridge.

Transport Bridge Configuration

You can arrange several bridges within a realm. Each bridge requires a separate bridge definition with a distinct bridge name, at least two transport sets, and at least one transport in each transport set.

Bridge Definitions

To configure bridge definitions, administrators can use the FTL server's bridge configuration GUI or the FTL server's web API. For details, see [Bridges Grid](#) or [Bridge Definition Objects](#).


Configuring a bridge definition in either way automatically embodies the bridge in a bridge application definition within the realm. Bridge application definitions supply bridge services with instructions for binding their transports and endpoints.

Use Transports Uniquely

A *transport* can be in at most one transport set within a bridge definition.

You may use the same *transport definition* in more than one bridge. However, it is your responsibility to avoid creating a bridge topology that contains loops.

Bridges Grid

The Bridges grid presents bridge definitions in the realm. In edit mode, you can create new bridge definitions and modify existing bridges. In the FTL GUI, click the **Bridges** icon .

Levels

- Bridge

- Transport Set
- Transport

Bridge Level

| GUI Parameter | Description |
|---------------|--|
| Bridge | Required. Name of the bridge. Bridge names must be globally unique within the realm. All names are limited to a maximum length of 256 characters. |
| Last Modified | This timestamp indicates the date and time of the most recent change to this bridge definition. |
| Description | Optional. This column presents a text field. You can annotate a bridge with a comment describing its purpose. |

Transport Set Level

| GUI Parameter | Description |
|---------------|---|
| Transport Set | Required. You must define <i>at least two</i> transport sets in each bridge. The transport set names are generated automatically. You cannot change them. |

Transport Level

| GUI Parameter | Description |
|---------------|---|
| Transport | <p>Required.</p> <p>Select a transport definition from the drop-down menu. The menu lists all the transports you have defined in the realm.</p> <p>In each transport set, you must configure <i>at least one</i> transport.</p> |
| Protocol | These columns display details of the transport definition. You cannot modify them in this grid. |
| Address | |
| Port | |
| Mode | |

Bridge Service

Bridge services implement the transport bridges that you define in the realm definition. Any FTL server can provide and manage a bridge service.

FTL servers provide bridge services according to the FTL server configuration file.

Each bridge service can implement one or more bridge objects as defined in the realm definition. These bridge objects are independent: even though one bridge service could implement several bridge objects, messages do not cross from one bridge to another bridge.

If you modify the bridge definitions, or the transport definitions that they use, the FTL server automatically restarts the affected bridge services that implement those bridge objects.

The bridges status table displays bridge services. You cannot disable nor purge them using the GUI.

Arranging Fault-Tolerant Bridge Services

You can configure two or more bridge services for fault-tolerant operation. Although this feature ensures automatic and quick recovery, failover might leave some messages unforwarded.

Weights can be specified in the FTL bridge service to coordinate its fault-tolerant operations to set a precedence order among all the bridge server instances. If a group of fault-tolerant bridges is configured, the bridge service with the highest weight becomes active. For example, if `ft.weight` is set on server 1 at 100, server 2 at 200, and server 3 at 300, server 3 will become active. This applies to all logical bridges managed by the bridge service. To see how to set this up, see the sample configuration file in [Bridge Service Configuration Parameters](#).

Before you begin

Fault-tolerant bridge services rely on the group facility to coordinate fault tolerance.

Procedure

1. Configure bridge objects in the realm definition (see [Transport Bridge Configuration](#)).
2. Configure two or more bridge services in separate FTL servers. Specify the bridges in the common section, or specify identical configuration parameters for each FTL server that provides the bridge service (see [Bridge Service Configuration Parameters](#)).

Result

The group facility automatically ensures that only one of the fault-tolerant bridge services is *active* at a time. The bridge service with the highest weight becomes active. This applies to all logical bridges managed by the bridge service. See [Bridge Service Configuration Parameters](#) to assign weight.

Bridges among Dynamic TCP Meshes

You can use transport bridges to link the disjoint mesh topologies that a dynamic TCP mesh transport establishes in satellite locations.

A dynamic TCP mesh definition establishes a bus with full mesh topology among all its endpoints, as long as their applications connect to a localized cluster of FTL servers. When

your enterprise uses satellite FTL servers, a single dynamic TCP transport definition establishes a *separate* mesh in each localized satellite cluster. Each mesh includes all the applications that are clients of any FTL server in the local cluster (and that use that dynamic TCP transport). You can use transport bridges to link those otherwise disjoint meshes.

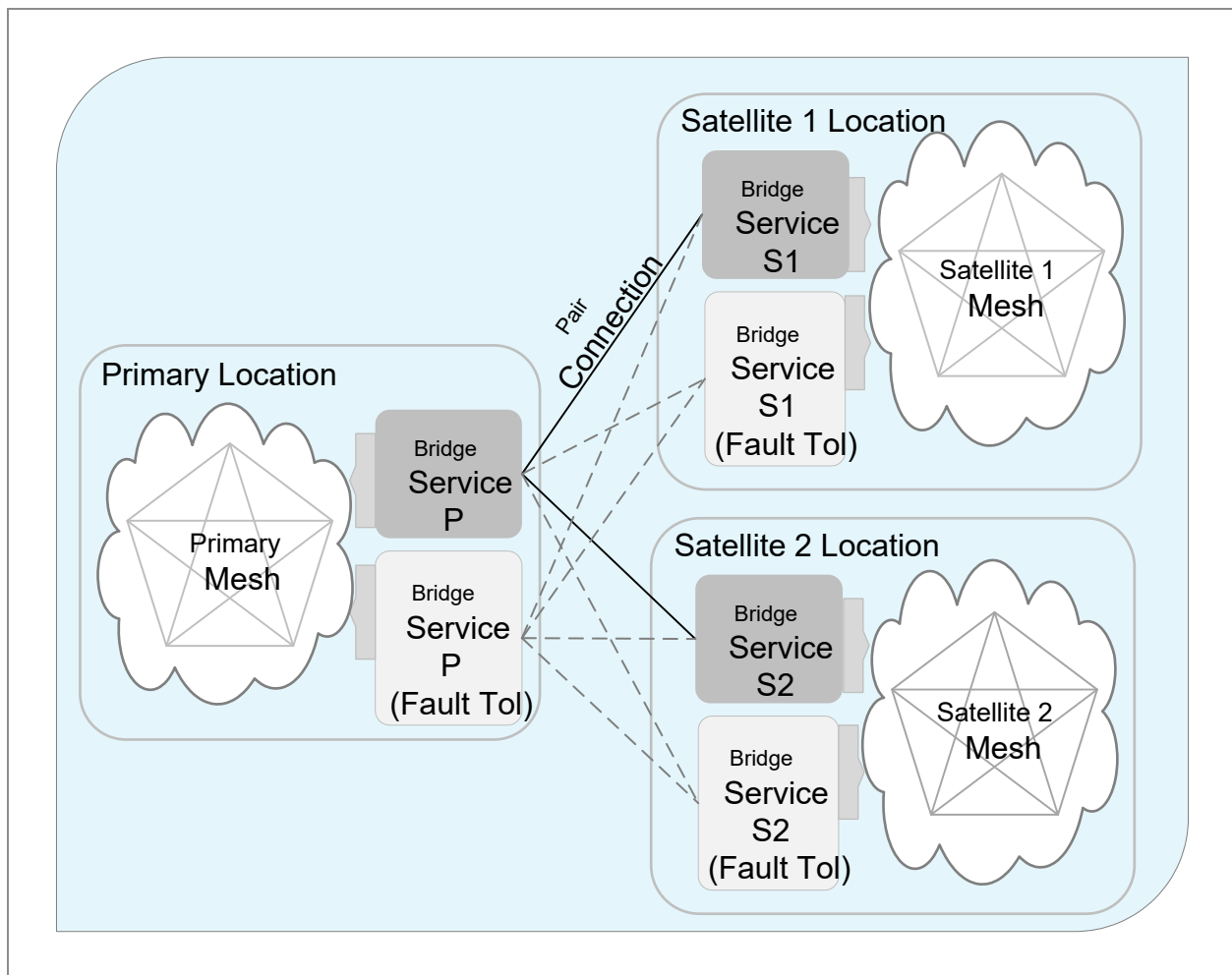
You can define a transport bridge to interconnect dynamic TCP meshes similarly to other bridges.

For more background information, see [Dynamic TCP Transport](#).

Principles of Operation

The following diagram depicts the general use case of a dynamic TCP bridge.

Figure 33: Bridging for Dynamic TCP Meshes

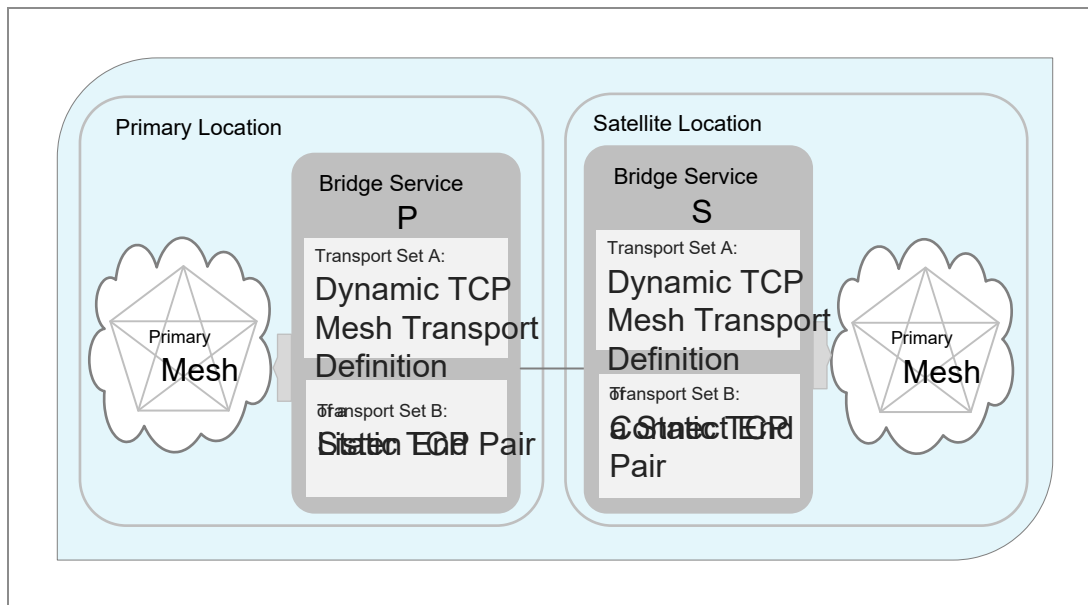


This enterprise includes three locations: the primary location (left) and two satellite locations (right). Each location has an affiliated FTL server cluster. At least one FTL server in each cluster provides a bridge service. The realm defines a single dynamic TCP transport definition, which establishes a mesh at each location. The meshes are disjoint from one another.

To bridge the meshes requires a bridge service at each location (orange), and a bridge definition that links the locations using a pair of *static* TCP listen and connect definitions. Include the listen end at the primary location, and the connect end at the satellite locations. The resulting pair connections link each of the satellite bridge services to the primary bridge service (solid lines). (Ensure that the static TCP port is open on the bridge service host computers.)

To extend to a fault-tolerant scenario, add redundant bridge services to FTL servers at each location (light orange). Configure these additional bridge services in the FTL server configuration files. Specify the identical bridge names. You can configure any number of redundant bridge services at each location (one per FTL server) even though the diagram shows only one redundant service per location.

Figure 34: Example



To implement the preceding example (that is, the simple case, without fault tolerance), define two bridge definitions in the realm, Primary Location and Satellite Location:

- The Primary Location, Bridge Service P, has two transport sets.
 - Set A contains only the Dynamic TCP Mesh Transport Definition.
 - Set B contains only the *listen end* of a static TCP pair.
- The Satellite Location, Bridge Service S, has two transport sets.
 - Transport Set A contains only the Dynamic TCP Mesh Transport Definition.
 - Transport Set B contains only the *connect end* of the static TCP pair.

Bridges Status Table

The Bridges status table reports the state of bridge objects. Bridge objects are clients of the FTL server.

| Column | Description |
|--------|---|
| Status | For descriptions of these values, see Client Status . |
| Name | The bridge name string. |
| ID | The FTL server assigns each bridge client a unique identifier. |
| Host | The host name string of the bridge client's host computer (if available). |

Persistence: Stores and Durables

Stores and durables provide for persisted messages and persistent interest.

For a general introduction, see “Persistence: Stores and Durables” in TIBCO FTL [Concepts](#).

Purposes of Persistence

The persistence infrastructure of stores and durables can serve four purposes:

- Delivery assurance
- Apportioning message streams
- Last-value availability
- Key/value maps

These purposes correspond to four types of durables: standard durables, shared durables, last-value durables, and map durables.

Delivery Assurance

An ordinary subscriber receives messages from a publisher only when a transport connects the subscriber with that publisher. That is, an ordinary subscriber cannot receive a message through a direct path transport unless *both* of these conditions hold at the time that the publisher sends the message:

- The subscriber object exists.
- The transport is connected.

With persistence, subscribers can receive every message with high assurance, despite temporary lapses in these conditions. For example, a persistence store can retain data for subscriber recovery after application exit and restart, temporary network failure, or application host computer failure.

Stores can retain data indefinitely, until subscribers acknowledge delivery. (In contrast, even reliable transports retain data only for a finite interval.)

For delivery assurance, use standard durables. Standard durables serve one subscriber at a time, and assure that it can receive the whole message stream.

Apportioning Message Streams

In some situations it is convenient to use a shared durable to apportion a message stream among a set of cooperating subscribers so that only one subscriber receives and processes each message, but acting together the subscribers process every message in the stream. For details, see [Basic Definitions for Persistence](#), Shared Durable.

Last-Value Availability

When new subscribers need to receive the current state immediately, but do not need to receive a complete historical message stream, a persistence store can supply the most recent (i.e., *last-value*) message.

The store continues to forward subsequent messages to subscribers, discarding the previous message as each new message arrives.

For last-value availability, use last-value durables. Last-value durables can serve many subscribers.

A last-value durable divides its input message stream into distinct output streams based on the value of a distinguished key field. In this way, one last-value durable holds more than one last value: one for each key.

Key/Value Map

A key/value map is like a last-value durable without a message stream. It behaves like a simple database table with two columns.

Instead of an inbound message stream to supply values, programs use the map API calls to store and access values. See "Key/Value Maps" in TIBCO FTL [Development](#).

Persistence Architecture

Persistence is flexible. Administrators can tailor various aspects of persistence to meet the needs of applications.

Storage, Replication, and Fault Tolerance

Persistence services manage stores and durable subscriptions (durables). A store holds messages until they are consumed. A *persistent store* is a store that is replicated to further ensure against loss of messages.

Replication of stores across a cluster of persistence services protects against hardware or network failures on a small scale. (However, this replication scheme cannot guarantee delivery after catastrophic failures.)

Note that for clusters with only one member it is possible to configure a replicated store.

Message Swapping

Stores hold message data typically in process memory to avoid the latency associated with disk I/O. However, with optional message swapping, if storage requirements exceed configured memory limits, excess messages are temporarily written to disk as needed. The use of message swapping can hedge against bursts. Memory threshold limits can be set on both a per-store and per-durable basis.

When messages are swapped to disk, the message body is freed from process memory. However, message metadata is retained in process memory. Therefore, for each message stored, some memory will be consumed. The amount of memory can depend on the type of durable and the number of subscriptions matched by the message. The best practice is to empirically determine the amount of memory needed for your desired message backlog. Then, size the environment accordingly.

You can set a memory limit (Swap Byte Limit) on a per-store and a per-durable basis. If a message exceeds either limit, it is swapped out.

As a good practice, for last-value durables, either a swap memory limit of zero (swap everything to disk), or a limit high enough to contain everything in that durable. Otherwise performance for the durable may be variable.

For standard durables without prefetch, configuring a swap memory limit greater than zero is not expected to increase throughput, because messages are typically delivered on the direct path.

For shared durables and standard durables with prefetch, configuring a swap memory limit greater than a typical backlog size for those durables may improve throughput. However, throughput can vary if either the durable or the store swap memory limits are exceeded.

You can enable message swapping from the administrative GUI, via REST API, or via YAML configuration file. For example, in the YAML file, to enable message swapping on the default cluster:

```
servers:
  <ftlserver name>:
    - realm:
        default.cluster.disk.swap: true
```

Disk-Based Persistence

You can store FTL messages and metadata to multiple disks when disk access is more readily available and cost effective than using memory. These messages and metadata can also then be automatically recovered on a full restart of the persistence cluster.

Disk persistence is enabled on a by-cluster basis, in one of two modes:

- **sync** - The client returns from a send-message call after the message has been written to a majority of disks. This mode generally provides consistent data and robustness, but at the cost of increased latency and lower throughput. If the cluster restarts, no data is lost; performance is subject to disk performance.
- **async** - The client may return from a send-message call before the message has been written to disk by majority of the FTL servers. This mode generally provides less latency and more throughput, but messages could be lost if a majority of servers restart shortly after the API call.

You can enable disk-based persistence from the administrative GUI, via REST API, or via YAML configuration file. For example, in the YAML file, to enable disk persistence on the default cluster:

```
servers:
  <ftlserver name>:
    - realm:
        default.cluster.disk.persistence: sync
```

Non-replicated stores are never persisted to disk, though they may be swapped to disk if disk swapping is enabled.

Setting Automatic Disk Persistence File Compaction

A persistence service can compact its disk persistence files while running online. There is no interruption to active publishers or subscribers. See [Compact Disk Persistence Files with Persistence Service Online](#).

You can enable automatic file compaction from the administrative GUI, via REST API, or via YAML configuration file. For example, in the following YAML file, auto compaction is disabled by default in the initial realm configuration for the default cluster.

```
servers:
  <ftlserver name>:
    - realm:
        default.cluster.disk.nocompact: true
```

Auto compaction can only be disabled by YAML file for the default cluster. For other clusters the GUI or REST API has to be used.

Latency

Using persistence for delivery assurance is consistent with high-speed message delivery with low latency. Delivery assurance operates *alongside* regular direct-path delivery. Transports carry messages directly from publishers to subscribers without an intermediary hop through a persistence service, which would add message latency. Separate transports carry messages from publishers to standard durables in a store in the persistence services, which retain them for as long as subscribers might need to recover them.

However, using persistence to apportion message streams or for last-value availability emphasizes throughput rather than the lowest latency. Delivery through durables *replaces* direct-path delivery. The persistence service is an intermediary hop, which adds message latency.

Meanwhile, a message broker emphasizes the convenience of a well-known pattern and minimal configuration, at the cost of added latency.

Wide-area stores involve the inherent latency of a WAN.

Publisher Quality of Service

For each store, administrators can balance appropriately between performance requirements and the need to confirm message replication.

Subscriber Acknowledgment

Within application programs, subscribers can acknowledge message delivery automatically or explicitly.

Administrators can configure durables to receive individual acknowledgments synchronously, or in asynchronous batches.

Durable Creation

Administrators can arrange for **dynamic durables**, which applications create as needed. Dynamic durables require minimal administrative configuration. Programmers take responsibility for the number of durables and their names.

Administrators can define **static durables** in the realm. Static durables require more administrative configuration and greater coordination between programmers and administrators. Administrators control the number of durables and their names.

Durables can be configured for Total Time to Live (TTL). Durables with a low TTL value are considered *ephemeral durables*.

Persistence Effectiveness

The flexibility of FTL persistence allows for various levels of persistence effectiveness, depending on factors such as the number of replicated stores, data limits store and persistence service hosting, and durable TTL. Persistence is generally considered adequately effective when stores are replicated and durables are non-ephemeral.

Logs

The persistence service reports an estimate of its own disk usage, and other statistics, via the monitoring stream and, periodically, in the log.

The persistence service periodically logs statistics about message rates. If disk persistence is configured, statistics about disk usage and disk write are also periodically logged.

See [GET persistence/clusters/<clus_name>/servers](#) and [Catalog of Persistence Metrics](#).

Persistence Service Disk Capacity

The FTL Server configuration parameter `max.disk.fraction` monitors disk capacity and prevents a disk full state to keep the persistence cluster running in the event that disk space is not available. For example, assume a disk size is 10GB and the default `max.disk.fraction` is set at 0.95. The persistence service would stop accepting messages once the disk usage reaches 9.5GB.

Considerations

- Persistence services must have disk persistence enabled to enforce `max.disk.fraction`.
- Only replicated stores are persisted to disk.
- Even with `max.disk.fraction` enabled, you still need to configure reasonable byte limits and message limits at the cluster level, store level, or both. See [max.disk.fraction](#) in [Persistence Service Configuration Parameters](#).
- The persistence service measures total disk usage, not just its own, and compares that to `max.disk.fraction`. For example, if several persistence services use the same disk, `max.disk.fraction` is compared to all disk usage across all persistence services and any other processes.
- The disk volume with the persistence data directory should have the same disk capacity for each persistence service in a cluster. Each persistence service in a cluster should also use the same value of `max.disk.fraction`.

Values and Behavior

The `max.disk.fraction` default value is 0.95. Publish calls fail once the total disk usage approaches the `max.disk.fraction` setting multiplied by the capacity of the disk that contains the persistence data directory. The persistence service may go over or under the limit by a small amount. A best practice is to allow for some overage so the persistence service continues to process subscriber acknowledgments while the disk is nearly full. The default value of 0.95 should allow for sufficient overage in common scenarios. In high fan-out cases, where many subscribers must acknowledge the same message before it can be deleted, consider reducing `max.disk.fraction`.

The impact of publish failures due to the disk usage limit depends on the publisher mode. (Also see [Publisher Mode](#).)

- If the publisher mode is `store_confirm_send`, the publish will be retried automatically by the FTL client library for the publisher's retry duration. This allows some time for subscribers to consume messages and free disk space before the publish call returns an exception.
- If the publisher mode is `store_send_noconfirm`, there is no retry, and the call returns immediately with no exception.

Once enough disk space has been freed, the publish call will no longer fail.

Values for `max.disk.fraction` of less than 0 and more than 1 are not allowed.

A value for `max.disk.fraction` of 0 disables the feature which means there is no limit on disk usage. When set to 0 and persistence services have disk persistence enabled, publish calls will not stop, and an overfull backlog may cause a system failure.

Disk Space for Backup or Compaction

There must be enough disk space available to hold the current message backlog when backing up or compacting disk persistence files. When the persistence service is online, backup and compaction cannot run at the same time.

- The persistence service will consume additional disk space when compacting or backing up.
- If disk usage approaches the disk usage limit (obtained by multiplying disk capacity by `max.disk.fraction`), or the message backlog is large, the persistence service may abort a backup or compaction and log a warning.
- For manual compaction, the best practice is to perform compaction when the message backlog is small.

If storage limits are not configured, the persistence service may consume disk space up to `max.disk.fraction` for the message backlog. At this point it may be difficult start a backup or compaction even after the message backlog is consumed. In situations like this, the offline compaction tool can be used. See [Compact Disk Persistence Files with Persistence Service Offline](#).

The persistence service prioritizes avoiding disk full errors over storage for pending messages. The persistence service prioritizes storage for pending messages over backups and compactions.

Compact Disk Persistence Files with Persistence Service Online

With disk persistence enabled, a persistence service can compact its disk persistence files while running. There is no interruption to active publishers or subscribers.

Once started, compaction proceeds in the background until complete. In general, all persistence services in a given cluster will compact their disks at about the same time. A restart of the persistence service cancels any ongoing compaction.

Compaction Methods

You can compact disk persistence files with the following methods. Backup and online compaction cannot run at the same time.

- The persistence service can automatically start online compaction as set in the realm configuration. See [Automatically Start Online Compaction](#).
- You can manually start online compaction via the REST API. See [Compact Disk Persistence Files with Persistence Service Online](#).
- If online compaction is impractical because the disk is nearly full or other reason, see [Compact Disk Persistence Files with Persistence Service Offline](#), section [Alternative to Online Compaction Procedure](#).

Automatically Start Online Compaction

When automatic disk persistence compaction is enabled, the persistence services in the cluster automatically start a compaction when certain conditions are met. Compaction only runs when there is space to be reclaimed. If there is not enough disk space, the persistence service will not start compaction and will log a warning.

Automatic disk persistence compaction is set at the persistence cluster level in the realm configuration. Automatic compaction is enabled by default for newly created persistence clusters. The defaults follow. See [Cluster Details Panel](#) for more information.

- `disk_compact=true` (Disk compaction is enabled.)
- `min_disk_inuse_ratio=0.05`. (If the ratio of disk inuse size to disk allocated size falls below this value, the persistence service can start a compaction.)

Manually Start Online Compaction via the REST API

When the REST API compaction is manually invoked, all members of the persistence cluster start compaction. See [POST persistence/clusters/<clus_name>](#), cmd: compactdisk.

You can invoke compaction for a cluster from the GUI, [Persistence Clusters Status Table](#).

Compaction runs even if there is no space to be reclaimed. If disk persistence is not enabled, an error results. If there is not enough disk space, the persistence service will not start compaction and will log an error message.

The log reflects when compaction is complete. Compaction can also be monitored via the field compaction_in_progress from the REST API, [GET persistence/clusters/<clus_name>/servers](#).

Other Considerations

The amount of disk that can be reclaimed can be estimated by comparing disk space in use and allocated disk space. The larger the difference between inuse size and allocated size, the more disk can be reclaimed. See [Catalog of Persistence Metrics](#), disk size (3013) and disk in use size (3019).

For details on disk space during compaction, see [Persistence Service Disk Capacity](#), section [Disk Space for Backup or Compaction](#).

Compaction requires additional disk bandwidth, so some performance degradation is possible while compaction is running. A higher rate of live data, or a larger amount of disk space inuse, will increase the disk activity associated with compaction.

If the performance degradation due to compaction is not tolerable, you may want to disable automatic compaction in the realm configuration. If disabled, you can do one of the following:

- With the persistence service running, start a manual compaction during non-peak hours. See [Manually Start Online Compaction via the REST API](#).
- Use the offline compaction tool after shutting down the persistence service. See [Compact Disk Persistence Files with Persistence Service Offline](#).

Compact Disk Persistence Files with Persistence Service Offline

Use the FTL administration utility to compact persistence files for a given persistence service when the persistence service is offline and not running. An error results if you invoke the utility while the persistence service is running. For details on compacting disk persistence files while the persistence service is running, see [Compact Disk Persistence Files with Persistence Service Online](#).

The utility must be invoked once for each persistence service.

A sample command line follows:

```
tibftladmin --compact_offline --name <name> --datadir <path>
```

When complete, the file will be smaller, depending on how much disk space was reclaimable.

Running the compaction tool can be advantageous if there is large difference between allocated disk space and inuse disk space.

See [Catalog of Persistence Metrics](#), 3013 Disk Size and 3019 Disk In Use Size. Also see the [GET persistence/clusters/<clus_name>/servers](#) log for disk_size and disk_inuse_size.

Compaction During Downtime Procedure

Use this procedure for compaction during downtime:

1. Shutdown all persistence services.
2. Use the tibftladmin utility to compact all files for all persistence services.
3. Restart all persistence services.

Compaction During Rolling Upgrade Procedure

Use this procedure for compaction during a rolling upgrade:

1. Shutdown one persistence service.
2. Use the tibftladmin utility to compact only the files for the one persistence service.

3. Restart the persistence service and wait for the persistence cluster to form a quorum and sync.
4. Repeat for the remaining persistence services.

Alternative to Online Compaction Procedure

If online compaction is impractical because the disk is nearly full or some other reason, use this alternative compaction procedure.

Shutdown one persistence service.

1. Copy the data directory for this persistence service to a different location.
2. Use the `tibftladmin` utility to compact the files at the new location.
3. Copy the data directory back to the original disk, replacing the old data directory.
4. Restart the persistence service.

Coordination for Persistence

Administrators and application developers must coordinate the details of durables.

See [TIBCO FTL Durable Coordination Form](#).

Stores for Delivery Assurance

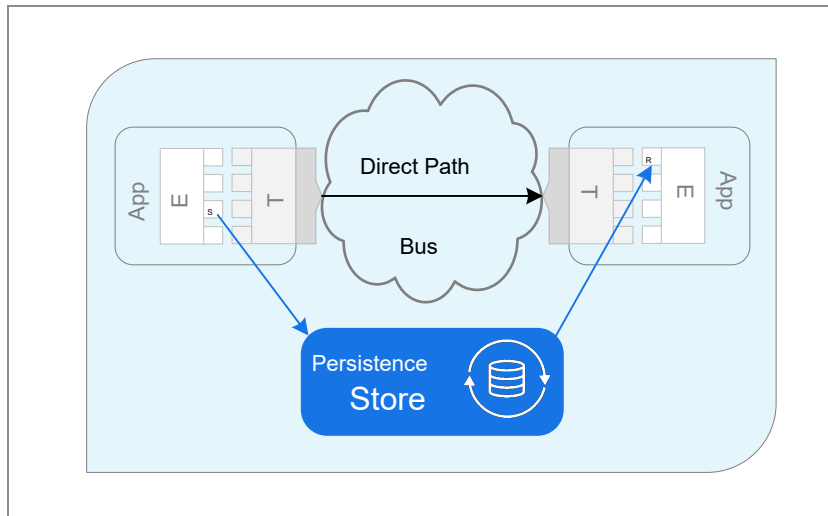
The following topics present the details of delivery assurance for architects and administrators.

Delivery Assurance: Topology

When using a persistence store for delivery assurance, the store is similar to a second transport. Like the direct path, it mediates message communication among a network of endpoints. It operates in parallel with direct-path transports, as an additional path from publishers to subscribers.

Because the store offers a redundant path, one can say that the store *backs* the direct paths. Depending on the focus, one can also say that the store *backs* the network of endpoints, or that it *backs* the message stream that travels among those endpoints.

Figure 35: Persistence Store as Redundant Message Path



The preceding diagram depicts a direct path through the transport bus, and another path from endpoint to endpoint through a store. (Conceptually, communications to and from the store travel through the endpoint instances. TIBCO FTL software arranges a communication mechanism for this path.)

i Note: To configure direct-path delivery assurance, you must disable the durable prefetch feature. For more information, see [Durable Prefetch Count](#).

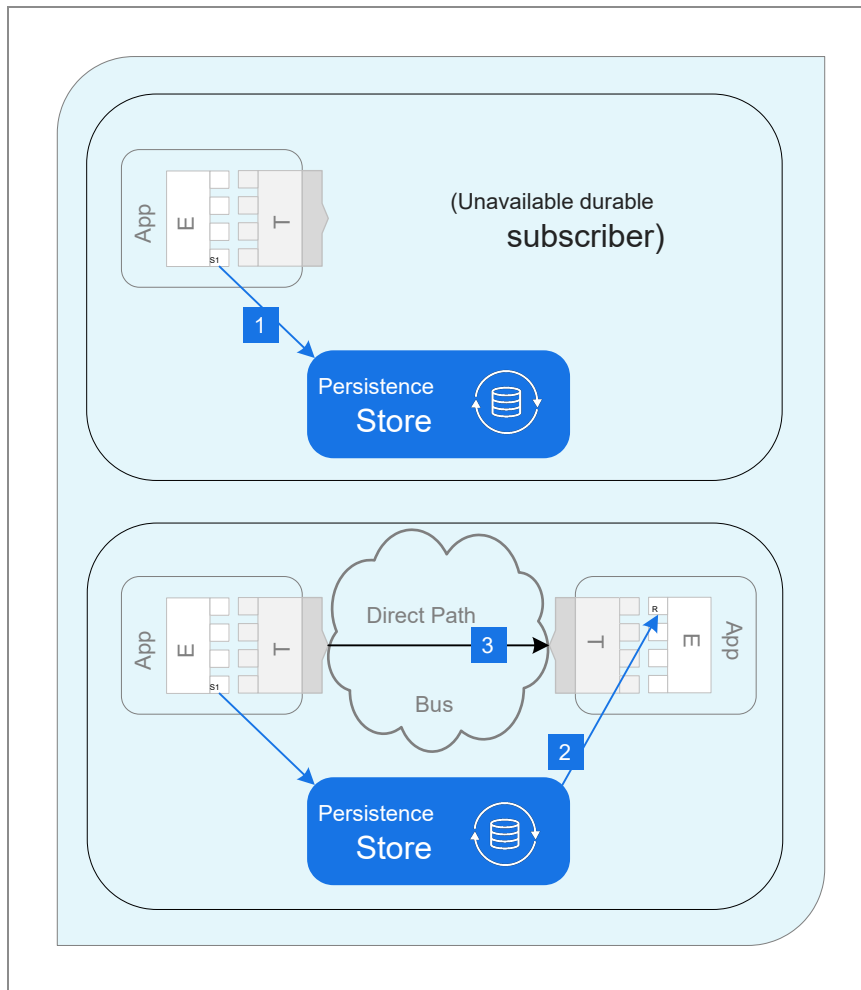
Direct-Path and Latency

For delivery assurance with *low latency*, you must configure a direct path. If a persistence store is the *only* path between a publishing endpoint and a subscribing endpoint, it is an intermediate hop that introduces latency.

Delivery Assurance: Retention and Delivery

A persistence store adds value by strengthening message delivery assurances. You could imagine a persistence store as a bus that can retain its message stream for subscribers that are temporarily unavailable.

Figure 36: Store for Message Recovery



In the first frame of the preceding diagram, the durable subscriber is unavailable, so it misses messages from the publisher. The persistence store collects those messages (1). In the second frame, the durable subscriber resumes, recovers the missed messages from the store (2), then continues receiving messages directly from the publisher (3).

Delivery Assurance: Larger Networks of Endpoints

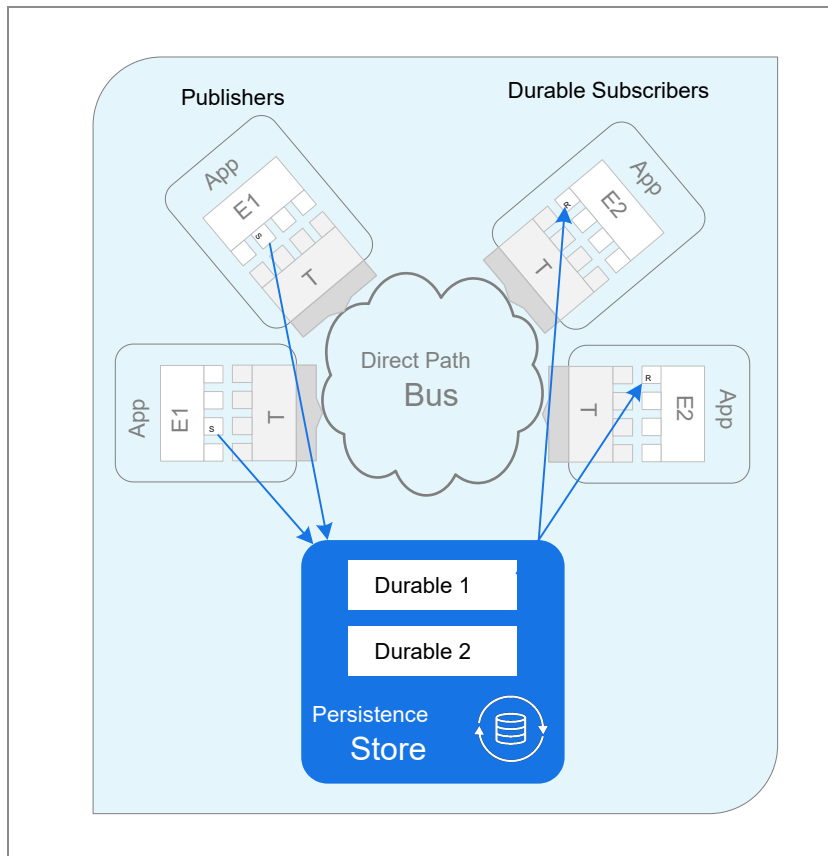
Persistence stores connect endpoints to endpoints.

Earlier diagrams showed the persistence store connecting one publisher endpoint to one subscriber endpoint. More generally, a store can connect many publisher endpoints, each with many publishers, to many subscriber endpoints, each with many subscribers.

The store in the following diagram collects messages from two publishers on endpoint E1 (left). Two durable subscribers on endpoint E2 (right) can recover those messages from the store.

A direct path connects each of the two E1 publishers to each of the two E2 subscribers.

Figure 37: Store Serves Several Publishers and Subscribers



Notice that each of the durable subscribers in the diagram depends on a unique standard durable within the store.

Delivery Assurance: Combined Message Stream

A persistence store collects messages from all its publishing endpoints, and merges them into a combined message stream. The store then ensures that all of its subscribers receive every message of that combined message stream. (If a subscriber has a content matcher, it receives every *matching* message.)

In the preceding diagram, the store merges the message streams from the two publishers, and feeds the combined stream to both subscribers. Each subscriber then receives the entire combined message stream.

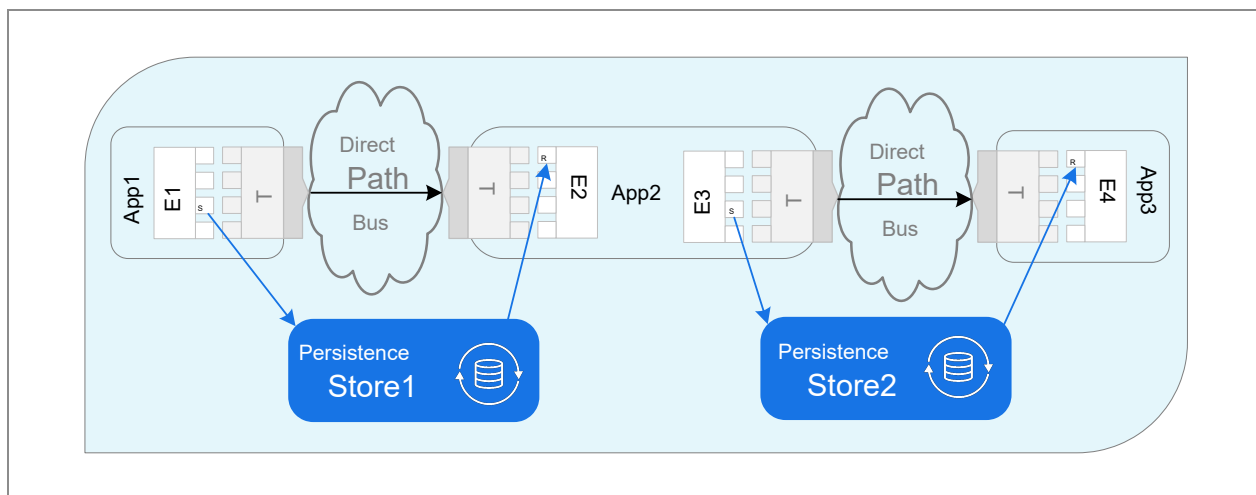
Delivery Assurance: Multiple Stores

A persistence store can serve as a redundant message path for *exactly one* combined message stream. When a suite of application programs communicate using two or more disjoint message streams, use a separate store to back each of the message streams.

Note: While persistence stores can potentially contain large quantities of message data, each store is itself a lightweight object. When the situation requires multiple stores, use as many as is appropriate.

Using one store inappropriately to back two or more direct message streams is a configuration error.

Figure 38: Two Stores

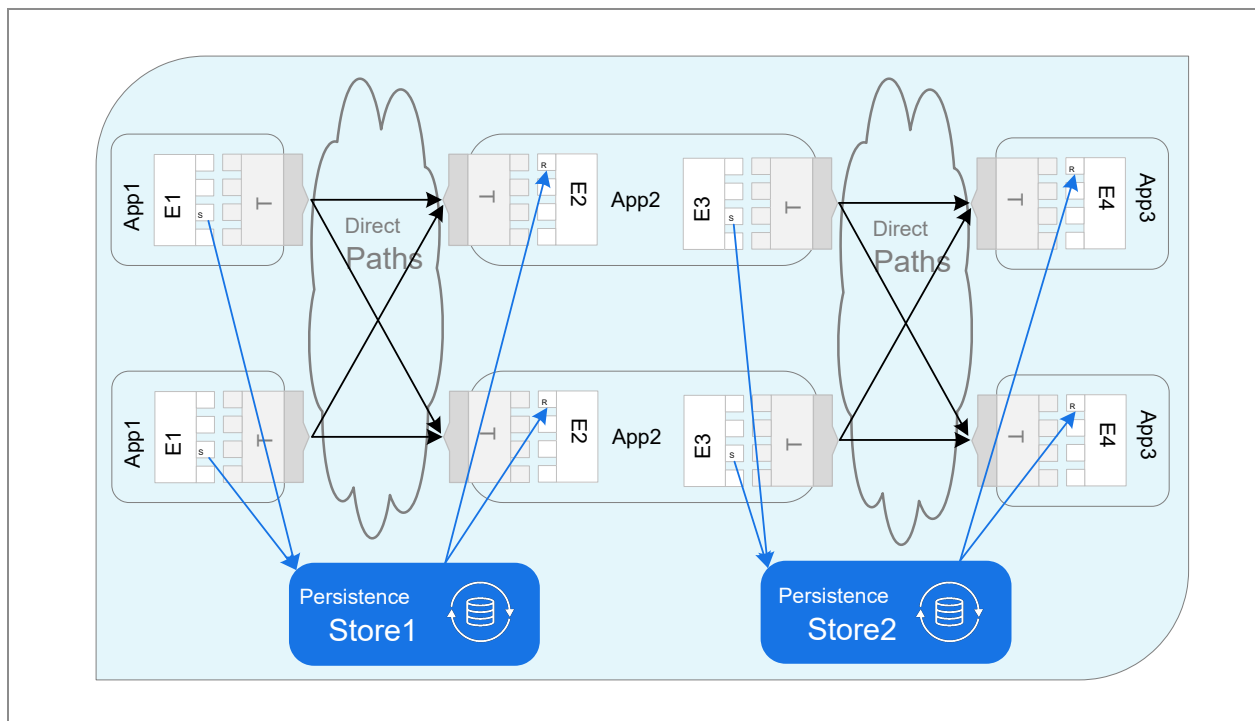


Two message streams are *disjoint* when they travel among disjoint networks of endpoints. For example, the first diagram (preceding) depicts a suite of three application programs: App1 sends a stream of messages from endpoint E1 to endpoint E2 in App2. App2 in turn sends a different stream of messages from endpoint E3 to endpoint E4 in App3. The two message streams are disjoint, even though E2 and E3 are both endpoints in App2. Because the message streams are disjoint, each requires its own store.

Larger Disjoint Networks of Endpoints

The preceding diagram depicted only one publisher feeding into each persistence store, and only one subscriber depending on each store. The following diagram depicts the same application suite in a more complex configuration, with two instances of each program.

Figure 39: Two Stores with Many Publishers and Subscribers



The second diagram shows that each of the two stores serves a separate network of endpoints. Store1 serves E1 and E2, while Store2 serves E3 and E4.

Notice that in the second diagram each E1 publisher has a direct path to every E2 subscriber. Similarly, every E3 publisher has a direct path to every E4 subscriber. The stores provide parallel indirect paths.

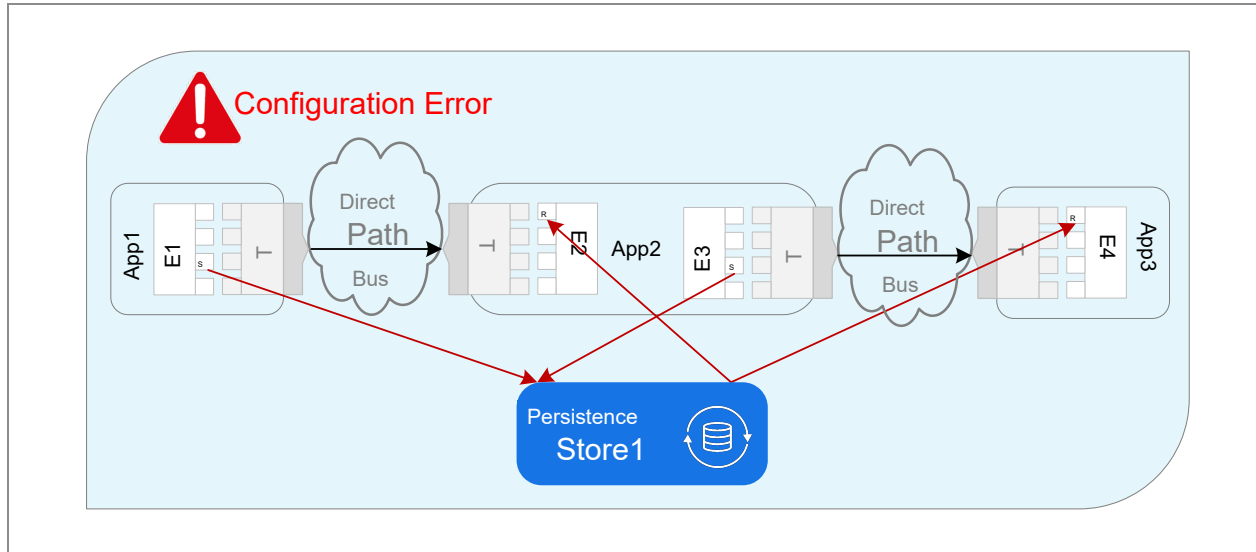
Store1 merges the message streams from all the E1 publishers, and guarantees that all the E2 subscribers receive all the messages in that combined message stream. Similarly, Store2 merges the message streams from all the E3 publishers, and guarantees that all the E4 subscribers receive all the messages in that combined message stream.

Violation of the Non-Merging Requirement

Violating the non-merging requirement is a configuration error, and causes incorrect results.

The third diagram (following) illustrates this error by modifying the example of the first diagram to erroneously use one persistence store instead of two.

Figure 40: Erroneously Using One Store Instead of Two



In this erroneous configuration Store1 collects messages from publishers at E1 in App1 and E3 in App2, and merges them into a combined message stream. Then Store1 ensures that all of its subscribers, at E2 in App2 and E4 in App3, receive every message of the combined message stream. That is, messages that App1 sends to App2 erroneously arrive at App3. Messages that App2 sends to App3 also erroneously arrive at App2.

These results are clearly incorrect. They do not match the intent. Compare this third diagram with the first diagram. Furthermore, this outcome results directly from violating the non-merging requirement:

- Store1 connects E1 to E4, even though no direct path connects them.
- Store1 connects E3 to E2, even though no direct path connects them.

Delivery Assurance: Durable Collision

Each standard durable can serve at most one durable subscriber object at a time. The persistence store resolves collisions in favor of the most recent durable subscriber.

For example, if a durable subscriber in process A is using durable D, and a new durable subscriber in process B subsequently claims D, then B wins the durable from A. The persistence store forces subscriber A to close.

The assumption behind this behavior is that the older durable subscriber is malfunctioning, and the newer one has started in order to replace it.

Collision does not apply with shared durables or last-value durables, both of which can support more than one subscriber at a time.

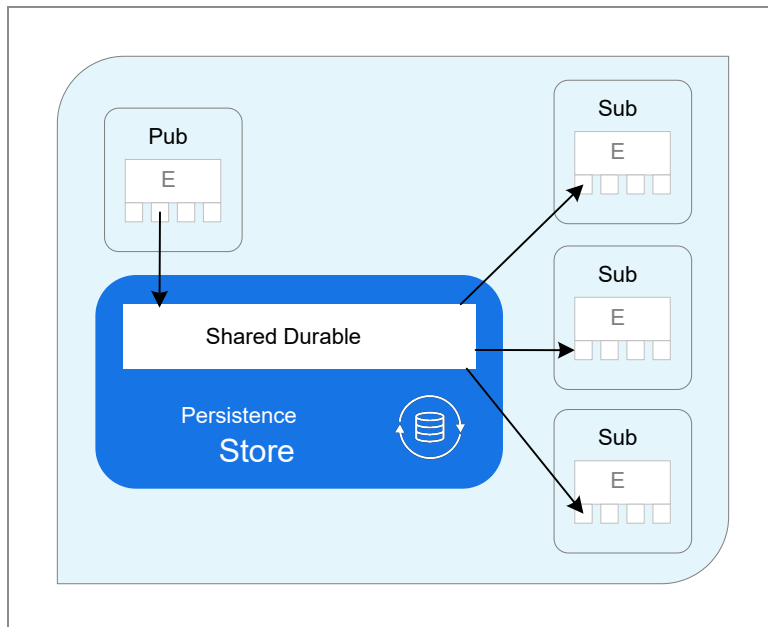
Stores for Apportioning Message Streams

This topic presents the details of apportioning message streams for architects and administrators.

Topology

When using a persistence store to apportion a message stream, the store is an intermediary hop between publishing and subscribing applications. In the following diagram, notice the absence of any direct path. The store is the *only* path from the publishing endpoint to the subscribing endpoints.

Figure 41: Persistence Store as Apportioning Intermediary



Apportioning

The shared durable in the preceding diagram apportiones the message stream among its subscribers. That is, each subscriber receives a portion of the message stream, rather than every message. Exactly one subscriber consumes each message in the message stream.

Acknowledgment and Redelivery

A subscriber acknowledges each message it receives. The shared durable tracks those acknowledgements. If a subscriber disconnects leaving one or more messages unacknowledged, the shared durable redelivers those messages to other subscribers.

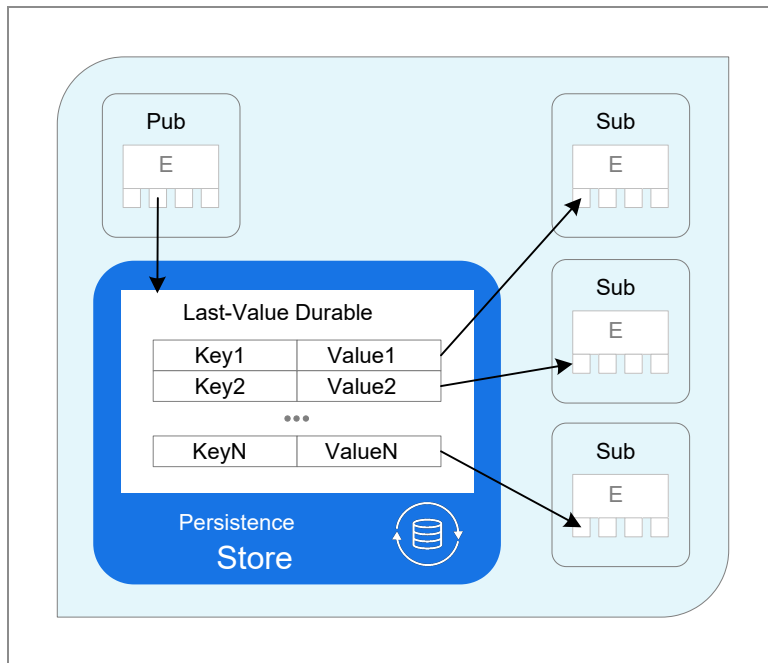
Stores for Last-Value Availability

This topic presents the details of last-value availability for architects and administrators.

Topology

When using a persistence store to deliver the most recent values, the store is an intermediary hop between publishing and subscribing applications. In the following diagram, notice the absence of any direct path. The store is the *only* path from the publishing endpoint to the subscribing endpoints.

Figure 42: Persistence Store as Last-Value Intermediary



Dividing a Message Stream by Keys

The last-value durable in the preceding diagram divides its input message stream according to the value of the key field. That is, the durable stores exactly one message for each unique string value of the key field: that is, the most recent message containing that key value.

Each subscriber specifies a key value in its content matcher, and receives a corresponding sub-stream: that is, messages with that key value in the key field.

When a new subscriber links to the durable, it receives the last matching message stored in the durable. Thereafter, it continues to receive subsequent matching messages.

Acknowledgment

A subscriber does not acknowledge the messages it receives. The last-value durable does not track acknowledgments.

Key/Value Map

The essence of a key/value map is a two-column database, like the green rectangle in the preceding diagram. Programs can store key/value pairs and access the table using API

calls.

FTL implements a map as a last-value durable. These two perspectives on the same type of durable are functionally equivalent.

Stores for Message Broker

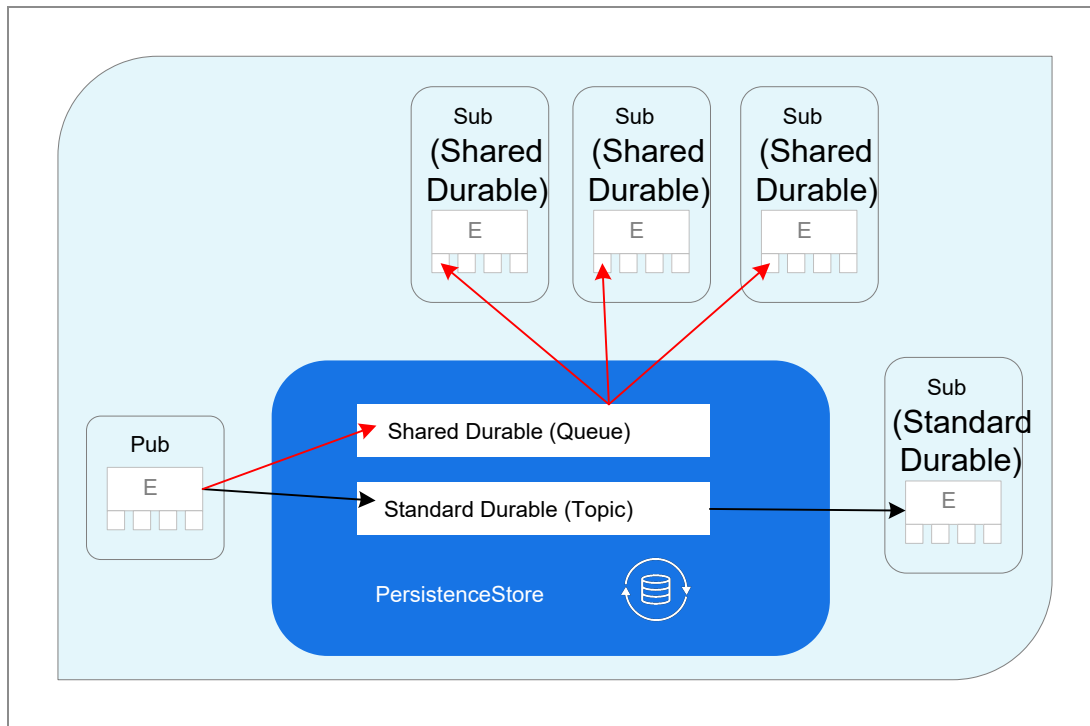
This topic presents information for architects and administrators about using stores to implement a message broker.

The default message broker uses built-in configuration values, which administrators do not need to configure explicitly (see [FTL Server: Message Broker](#)). Alternatively, administrators may explicitly configure a message broker to meet application requirements.

Topology

When using a store for message broker functionality, the store is an intermediary hop between publishing and subscribing applications. In the following diagram, notice the absence of any direct path. The store is the *only* path from the publishing endpoint to the subscribing endpoints.

Figure 43: Store as Intermediary Broker



Note: To configure a message broker, you must enable the durable prefetch feature. For more information, see [Durable Prefetch Count](#).

Topics and Queues

Message brokers can organize messages in topics or queues.

A shared durable is equivalent to a *queue*. Red arrows in the diagram indicate delivery of each message to exactly one of the three subscribers.

A standard durable is equivalent to a *topic*. Black arrows in the diagram indicate delivery of each message to both subscribers.

Replication

Message broker stores can be replicated or non-replicated.

Replicated stores provide fault-tolerance through redundancy. (However, replication requires a cluster of three or more message brokers.)

In contrast, when a *non-replicated store* receives messages from publishers, the message broker does not replicate them to other message brokers in the cluster. In using non-replicated storage, message brokers implicitly risk lost messages (for example, if the broker exits with undelivered messages). In exchange, non-replication reduces consumption of I/O bandwidth, process memory, and CPU cycles.

Default Durable

You can use the *default durable* to add delivery assurance to any application, even applications that do not explicitly supply a durable name or a subscriber name in their create subscriber calls.

To use the default durable, administrators may map the default subscriber name (`_default`) to a durable in one or more application instance definitions.

Restrictions

- Each application instance definition supports at most one default durable per endpoint. That is, for each application endpoint, each application instance definition can map the default subscriber name to at most one durable name. (However, separate application instance definitions can map their respective default subscribers to different durable names.)
- At any moment, at most one running process can claim a specific standard durable.

This restriction applies to all standard durables, but pay special attention to prevent inadvertent violations by processes that use default durables.

(This restriction does not apply to shared durables nor last-value durables.)

See Also: [Configuring a Default Durable](#)

Durable Behavior

Administrators configure the behavior of a durable and its interaction with durable subscribers.

For the configuration interface, see [Durable Details Panel](#).

Types of Durables: Standard, Shared, Last-Value, Map

Each type of durable supports a different set of use cases.

Standard Durable

A durable that represents the interest of one subscriber object, strengthening delivery assurance for that subscriber.

Use a standard durable when a subscriber must receive every message in a message stream, even if the subscriber is intermittently disconnected from the message bus. The type of persistence for a standard durable depends on whether the *prefetch* feature is enabled or disabled:

- Prefetch Enabled — Messages are received and sent by the server to subscribers. The durable functions as a message broker, with no messages traveling peer-to-peer from publisher to subscriber.
- Prefetch Disabled — Under normal conditions, messages flow directly from publisher to subscriber, with the store serving as a backup.

Shared Durable

A durable that represents the identical interest of *two or more* cooperating subscriber objects, apportioning a message stream among those subscribers.

Use a shared durable when a set of subscribers collectively process a message stream in a distributed or parallel fashion, processing every message exactly once.

Last-Value Durable

A durable that represents the interest of potentially many subscriber objects. It stores only the most recent message from one or more message streams.

Use a last-value durable to hold recent context for initializing new subscribers.

Map Durable

A durable that stores a key/value mapping. When a message arrives, a map durable uses the value in the message's key field as the key, and stores the full message as its value. For each distinct key, the durable stores only the most recent value. (Notice that one map durable can hold many key/value pairs.)

Applications can use the map API to get the current value of a key, or to store a key/value pair.

Key values must be strings.

Standard Durables

A *standard durable* strengthens delivery assurance for a subscriber.

A standard durable with prefetch disabled is intended to be used alongside a direct path to strengthen delivery assurance for a subscriber. However, the direct path remains the primary path.

A standard durable with prefetch enabled uses no direct path. Messages are always brokered by the persistence service.

Browsing is not supported for standard durables and is only supported for shared durables.

Quality of Service

A standard durable ensures that every message is received and acknowledged by its subscriber.

If the subscriber does not acknowledge a message, the persistence store retains the message, and the subscriber automatically recovers the message.

Delivery Count and Maximum

A standard durable with a nonzero prefetch setting counts the delivery attempts for each message. If the administrator sets an optional maximum limit for a durable, the durable can discard a message after the number of failed delivery attempts exceeds that limit.

While this mechanism never discards a message before reaching the limit, it does *not* guarantee a discard at exactly the limit.

Message Retention

A standard durable retains each message until its subscriber acknowledges it.

If a prefetch is nonzero, and a delivery limit is set, a message may be discarded once the maximum number of delivery attempts has been reached for that message. This is true regardless of whether a retention time is set.

If the message TTL is enabled, a message may be discarded once the message's age exceeds the TTL. This is true regardless of whether a retention time is set.

If the prefetch is nonzero, and a retention time is set, the message may not be deleted immediately after being acknowledged. Instead, the message is retained until the message's age (as measured from publish time) exceeds the retention time. However, an acknowledged message is not delivered to a subscriber unless the durable is rewound. For more information see, [Rewinding a Durable](#).

Subscribers

A standard durable can serve at most one subscriber at a time. That subscriber receives the entire message stream.

If a standard durable already serves a subscriber object in one client process, and another client process creates a subscriber that connects to the same standard durable, then the persistence store resolves collisions in favor of the more recent subscriber, deactivating the older durable subscriber. See "SUBSCRIBER_FORCE_CLOSE" in TIBCO FTL [Development](#).

If a subscriber to a standard durable already exists within a client process, and the same client process subsequently attempts to create another subscriber on the same standard durable, that subscriber create call fails.

Direct Path

Administrators may configure a direct path from publisher to subscriber. In this arrangement, a standard durable provides a parallel path though the persistence service, which adds an intermediary hop. This durable path is only for recovering missed messages. See "Stores for Delivery Assurance" in TIBCO FTL® - Enterprise Edition [Administration](#).

However, in some use cases, a direct path is not required for standard durables:

- When using a message broker
- When using a wide-area store

Content Matcher

A standard durable accepts, but does not require, a content matcher.

If it has a content matcher, then the subscriber must specify either an identical content matcher, or no content matcher.

Acknowledgments

A standard durable tracks acknowledgments from subscribers.

Dynamic Durable

Before a program can create a dynamic standard durable, the administrator must have already enabled standard dynamic durables on the endpoint. In TIBCO FTL® - Enterprise Edition Administration, see “Enabling Dynamic Durables” and “Inbox Durable Templates”.

When a program creates a dynamic durable, it must supply the following information in the subscriber create call:

- Endpoint Name
- Durable Name
- Key Field Name

When creating a new dynamic durable, the key field name becomes part of its content matcher.

- Content Matcher

When creating a new dynamic durable, the content matcher becomes part of the new durable. The content matchers of subsequent subscribers to the durable must be identical.

If a durable with the specified durable name and key field name already exists, that durable forwards messages to the new subscriber.

If a durable with the specified durable name does *not* yet exist, the persistence store creates a dynamic durable using the durable name, key field name, and content matcher supplied in the subscriber create call.

Static Durable

Before a program can subscribe to a static durable, the administrator must have already defined that durable in the persistence store associated with the endpoint. See “Defining a Static Durable” in TIBCO FTL® - Enterprise Edition [Administration](#)

When a program subscribes to a static standard durable, it must supply the following information in the subscriber create call:

- Endpoint Name

- Durable Name or Subscriber Name

To understand this distinction, see “Durable Subscribers” in TIBCO FTL® - Enterprise Edition [Development](#).

- Optional: Content Matcher

The content matcher must either be null, or be identical to the content matcher of the durable as configured by the administrator. The best practice is to supply null.

Shared Durables

A *shared durable* apportions a message stream among its subscribers.

Quality of Service

A shared durable ensures that every message is received and acknowledged by a subscriber. Each subscriber receives a portion of the input message stream. Together, the subscribers receive the entire message stream.

If a subscriber does not acknowledge a message, the durable can redeliver it to another subscriber.

Delivery Count and Maximum

A shared durable counts the delivery attempts for each message. If the administrator sets an optional maximum limit for a durable, the durable can discard a message after the number of failed delivery attempts exceeds that limit.

While this mechanism never discards a message before reaching the limit, it does *not* guarantee a discard at exactly the limit.

Message Retention

A shared durable retains each message until a subscriber acknowledges it.

If a delivery limit is set, a message may be discarded once the maximum number of delivery attempts has been reached for that message. This is true regardless of whether a retention time is set.

If message TTL is enabled, a message may be discarded once the message's age exceeds the TTL. This is true regardless of whether a retention time is set.

If a retention time is set, the message may not be deleted immediately after being acknowledged. Instead, the message is retained until the message's age (as measured from publish time) exceeds the retention time. However, an acknowledged message cannot be delivered to a subscriber unless, the durable is rewound. For more information, see [Rewinding a Durable](#)

Subscribers

A shared durable can support multiple subscribers simultaneously.

Each subscriber receives a portion of the message stream.

Direct Path

Subscribers receive messages *only* through the persistence store. The shared durable semantic precludes direct-path delivery, even if a direct path is configured.

Content Matcher

A shared durable accepts, but does not require, a content matcher.

If it has a content matcher, then every subscriber must specify either an identical content matcher, or no content matcher.

Acknowledgments

A shared durable tracks acknowledgments from subscribers.

Shared Durable Browsing

Use a client API to browse messages stored in a shared durable, for example, to run analytics on the data or delete a message. Messages are browsed from oldest to newest. A matcher can be used to browse a subset of the messages. Browsing does not affect delivery of messages to subscribing clients. Messages can be delivered to other subscribing clients while simultaneously being browsed. The effective matcher is a logical AND of the browser matcher and the durable matcher.

Messages returned to a durable while a browser is running are not browsed unless the browser is restarted or re-created. This may happen, for example, because a consumer failed while processing some messages.

In the event of leader failover or connection loss, browsers need to be restarted or closed.



Warning: Creating too many browsers is not recommended, as it can put a strain on the persistence service resources, including both CPU and memory usage.

Monitoring

The REST API and UI show the number of browsers on a shared durable. See [GET persistence/clusters/<clus_name>/stores/<stor_name>/durables](#).

Logging

A log message warns if there are too many browsers on a shared durable.

Dynamic Durable

Before a program can create a dynamic shared durable, the administrator must have already enabled shared dynamic durables on the endpoint, In TIBCO FTL® - Enterprise Edition [Administration](#), see “Enabling Dynamic Durables” and “Inbox Durable Templates”.

When a program creates a dynamic durable, it must supply the following information in the subscriber create call:

- Endpoint Name
- Durable Name
- Key Field Name

When creating a new dynamic durable, the key field name becomes part of its content matcher.

- Content Matcher

When creating a new dynamic durable, the content matcher becomes part of the new durable. The content matchers of subsequent subscribers to the durable must be identical.

If a durable with the specified durable name and key field name already exists, that durable forwards messages to the new subscriber.

If a durable with the specified durable name does *not* yet exist, the persistence store creates a dynamic durable using the durable name, key field name, and content matcher supplied in the subscriber create call.

Static Durable

Before a program can subscribe to a static durable, the administrator must have already defined that durable in the persistence store associated with the endpoint. See “Defining a Static Durable” in TIBCO FTL [Administration](#).

When a program subscribes to a static shared durable, it must supply the following information in the subscriber create call:

- Endpoint Name
- Durable Name or Subscriber Name
To understand this distinction, see “Durable Subscribers” in TIBCO FTL [Development](#).
- Optional: Content Matcher
The content matcher must either be null, or be identical to the content matcher of the durable. The best practice is to supply null.

Last-Value Durables

A *last-value durable* preserves only the most recent message for subscribers. It does not track message acknowledgments from subscribers.

Browsing is not supported for last-value durables and is only supported for shared durables.

Quality of Service

A last-value durable divides its input message stream into output sub-streams based on the string value of a key field in each message. For each sub-stream, the durable stores the most recent message.

A last-value durable ensures that every new subscriber immediately receives the most recent message, if one is stored. The subscriber continues to receive the sub-stream of subsequent messages until the subscriber closes, but without any assurance of delivery.

Message Retention

A last-value durable retains only one message for each output sub-stream. Delivery limits are not permitted for last-value durables. If message TTL is enabled, a message may be discarded once the message's age exceeds the TTL. Retention time is not permitted for last-value durables.

Subscribers

A last-value durable can support multiple subscribers simultaneously.

Each subscriber can receive exactly one output sub-stream.

Direct Path

Subscribers receive messages *only* through the persistence store. The last-value durable semantic precludes direct-path delivery, even if a direct path is configured.

Content Matcher

Subscribing to a last-value durable requires a content matcher.

For static last-value durables, the administrator configures the key field name, and the program must match a specific value of that field in the create subscriber call.

For dynamic last-value durables, the program must supply the key field name as a property in the create subscriber call, and match a specific value of that field.

The key field must contain data of type string. Content matchers must match a string value against the key field.

Notice that each subscriber must test the key field, but each subscriber can match a different value of that field. The value determines the output sub-stream that the subscriber requests from the last-value durable.



Caution: A logical OR match is not permitted on the key field in the matcher used by a last-value durable subscriber. A subscriber must specify exactly one string value for the key field.

Acknowledgments

A last-value durable does not track acknowledgments from subscribers.

Unlike standard and shared durables, a subscriber to a last-value durable subscriber does not consume messages from the durable. Instead, new subscribers to a sub-stream continue to receive the stored value until a publisher sends a new value, which replaces it.

Dynamic Durable

Before a program can create a dynamic last-value durable, the administrator must have already enabled dynamic last-value dynamic durables on the endpoint. In TIBCO FTL® - Enterprise Edition Administration, see “Enabling Dynamic Durables” and “Inbox Durable Templates”.

When a program creates a dynamic durable, it must supply the following information in the subscriber create call:

- Endpoint Name
- Durable Name
- Key Field Name

When creating a new dynamic durable, the key field name becomes part of its content matcher.

- Content Matcher

When creating a new dynamic durable, the content matcher becomes part of the new durable. The content matchers of subsequent subscribers to the durable must be identical.

If a durable with the specified durable name and key field name already exists, that durable forwards messages to the new subscriber.

If a durable with the specified durable name does *not* yet exist, the persistence store creates a dynamic durable using the durable name, key field name, and content matcher supplied in the subscriber create call.

For example, suppose the store does not yet contain a durable named `ticker`. A program then creates a subscriber to a last-value durable named `ticker`, with key field name `Symbol` and content matcher `{"Symbol": "IBM", "Exchange": "NYSE"}`. In response, the store creates a new dynamic durable named `ticker`, with content matcher `{"Symbol": true, "Exchange": "NYSE"}`. That is, the new durable generalizes from the key field name to a content matcher clause that tests for the *presence* of the key field rather

than a specific value, and integrates the rest of the clauses from the subscriber's content matcher.

If the create subscriber call does not supply a content matcher, the new durable uses only the key field clause as its content matcher.

Subsequent subscribers to the durable must specify compatible content matchers: that is, they must be identical except for the key field value, which can be different. A subscriber create call with an incompatible content matcher throws an exception.

Static Durable

Before a program can subscribe to a static durable, the administrator must have already defined that durable in the persistence store associated with the endpoint. See “Defining a Static Durable” in TIBCO FTL® - Enterprise Edition [Administration](#).

When a program subscribes to a static last-value durable, it must supply the following information in the subscriber create call:

- Endpoint Name
- Durable Name or Subscriber Name

To understand this distinction, see “Durable Subscribers” in TIBCO FTL [Development](#).

- Content Matcher

The content matcher must include a test for a specific value of the key field: that is, it must express interest in a sub-stream of the last-value durable, as determined by a specific key string.

Message Interest

A persistence store collects all the messages from its publishers, but a subscriber might require only a subset of those messages, and different subscribers might require different subsets. (Developers and administrators coordinate to configure this behavior.)

Store All Messages

The durable collects the full message stream. (This behavior is available only for standard durables and shared durables.)

Store Matching Messages

The durable collects a sub-stream determined by a content matcher.

Store Messages with Key

The durable collects the sub-stream of messages that contain a key field. You must specify the `Key Field Name` parameter. This behavior is available only for last-value durables.



Tip: From the moment that a subscriber object first interacts with a durable, you may no longer change a durable's message interest, nor its matcher fields. If these configuration values must subsequently change, you can instead map the subscriber to a new durable with the correct message interest and matcher fields.

See Also:

- [TIBCO FTL Durable Coordination Form](#)
- “Content Matchers” in TIBCO FTL [Development](#)

Acknowledgment Mode

Durable subscribers acknowledge each message as they process it. However, the client library can transmit acknowledgments to the cluster in either of two modes: synchronous or asynchronous.

Synchronous

Each time the application program acknowledges a message, the client library immediately communicates the acknowledgment to the cluster, and waits for the cluster to confirm that it has safely recorded that acknowledgment.

Asynchronous

The client library gathers batches of acknowledgments, and communicates the batches to the cluster. (The application does *not* receive confirmation that the cluster has safely recorded the batch of acknowledgments.)

In some situations, batching can improve throughput and latency performance. However, after application failover, durable subscribers could receive duplicate delivery of messages that they have already acknowledged (because the cluster never received the most recent batch of acknowledgments).

i Note: Acknowledgment mode as described here is orthogonal to the developer's choice between automatic acknowledgment versus an explicit acknowledge call.

Acknowledgment mode does not apply to last-value durables.

Retention Time

You may configure a durable or durable template with a retention time. Only standard durables with prefetch and shared durables may have a retention time.

For more information, see [Durable Details Panel](#)

The retention time feature allows users to keep messages for a certain period. This would not affect the ability of consumers to receive and acknowledge messages normally. The messages may be replayed later by rewinding the durable, for example if some analysis is required, or lost data needs to be recovered. For more information, see [Rewinding a Durable](#).

Administrators should be mindful of how much data is retained. A long retention time can greatly increase storage requirements.

When retention time is set, and a subscriber acknowledges a message, the persistence service does not always delete the message immediately. Instead, if the message's age, as measured from send time, is less than the retention time, the message is retained. Later, once the message's age exceeds the retention time, it is deleted permanently.

When retention time is set, and a message's age exceeds the retention time, but the message has not been acknowledged, the message is not deleted. The message is deleted immediately if it is acknowledged.

Acknowledged and retained messages cannot be browsed or delivered to subscribers. The administrator or the client application must rewind the durable to make those messages available again.

Administrators can monitor the number of acknowledged or unacknowledged messages. FTL server reports the total message count for each durable (which includes acknowledged and unacknowledged messages), as well as the unacked message count for each durable. For more information, see [Durables List](#).

Retention time does not override byte limits, message limits, delivery limits, or message TTL. Once any of those limits is reached, the message may be discarded, regardless of

whether it has been acknowledged and regardless of whether the retention time has been set.

For example, if message TTL and retention time are both set, and the message's age is greater than the TTL, the message will be discarded, regardless of retention time. Similarly, if the message has been acknowledged, and its age is greater than the retention time, the message will be discarded, regardless of message TTL. Message TTL, byte limit, message limit, and delivery limit should be thought of as storage limits, orthogonal to retention time.

Rewinding a Durable

When retention time is enabled, see, [Retention Time](#) an administrator or client application may rewind the durable. Rewinding a durable means that all messages present in the durable, including messages previously acknowledged, are now delivered to newly created subscribers (or browsers). This allows you to replay messages from a durable.

Administrators can rewind any durable through the REST API [POST persistence/clusters/<clus_name>/stores/<stor_name>/durables/<dur_name>](#)Or, administrators can rewind a durable through the user interface, see [Durables List](#).

Client applications can rewind any durable via the `tibRealm_RewindSubscription()` API, or equivalent language binding. If permissions are enabled for persistence, the client must have the subscribe permission to do so. For more information, see [:Required Permissions for API Calls](#)

Before rewinding a durable, all subscribers on the durable must be closed. Once rewind completes, subscribers can be created normally, possibly using the exact code as before to process and acknowledge messages.

Rewinding a durable clears the acknowledged status of all messages. However, rewinding a durable does not reset the message's age. For example, rewinding a durable does not cause an acknowledged message to be retained for any longer than normal, and it does not affect the behavior of message TTL.

Endpoint Store Inboxes

Starting from FTL 6.8.0 and later, inbox subscriber applications can create an inbox durable in the endpoint's configured store, as long as there is no direct path transport. If the inbox

subscriber's endpoint has a direct path transport, the transport is used and no inbox durable is created, even if the endpoint has a store.

This behavior is controlled by the realm property `use_endpoint_store_for_inbox`, which is set to `true` by default in new realm configurations. If migrating from a release prior to 6.8.0, the endpoint store inbox store behavior is not enabled by default. The pre-FTL 6.8.0 inbox behavior of routing to one of two inbox stores (`ftl.system.inbox.store` or `ftl.routing.inbox.store`) is used if `use_endpoint_store_for_inbox` is set to `false`.

When sending to an endpoint store inbox, the message matches only the corresponding inbox durable in the endpoint's store. All non-inbox durables are excluded, even if those durables use null matchers. In this way, it is possible to use the same store for one-to-many and one-to-one sends. Releases prior to FTL 6.8.0 relied on dedicated stores for inbox traffic.

Endpoint store inboxes are functional in all store configurations (including disk persistence and/or swapping) and in any zone type (hub-spoke, full-mesh, hub-spoke-1hop). In releases prior to FTL 6.8.0, the dedicated inbox stores were configured to be non-replicated and to use a publisher mode of `store_send_noconfirm`. If that behavior is desirable, consider using a similarly configured store for inbox messages.

The FTL request-reply APIs use inboxes implicitly, and therefore will use an endpoint store inbox if that behavior is enabled. If the requestor wishes to use separate stores for sending requests and receiving inbox replies, the application may specify an explicit reply endpoint.

Changing the value of the `use_endpoint_store_for_inbox` property will require a restart of all store-based inbox publishers and subscribers.

Inbox Durable Templates

Inbox durables are always dynamic durables. By default, FTL automatically generates a durable template for inbox subscribers. The default inbox template has a substantial prefetch and is configured for async acknowledgments.

Administrators may override the default template by defining any standard durable template with prefetch in the appropriate store and setting the `inbox_durable` field in the endpoint definition. An inbox durable template must be standard with prefetch, and a small durable TTL is automatically applied even if none is configured.

The inbox durable template is used only for inbox subscribers. Other subscribers continue to use the generic `dynamic_durable` template, if configured.

It is not possible to explicitly name an inbox durable. It is not necessary to unsubscribe from an inbox durable. The inbox durable is automatically destroyed when the subscriber is closed.

Wide-Area Forwarding

Client applications can publish and subscribe across network boundaries. To enable this behavior, administrators define forwarding zones and wide-area stores.

Background

Ordinarily, a network limits the geographic scope of persistence. That is, a persistence cluster implements a store, and the durables of the store are accessible only to publishers and subscribers running within the network that connects the persistence services in the cluster.

Yet in some enterprises, it is important to access messages remotely, across network boundaries. That is, the publishers and subscribers are in separate networks, connected by a wide-area network (WAN). *Wide-area stores* support this situation.

Motivation

Consider a chain of retail outlets that generate data about sales, inventory, and expenses. In each outlet, some application programs publish messages with this data, and other application programs subscribe to those messages.

With wide-area stores, application programs at corporate headquarters can also subscribe to those messages, automatically receiving the messages from the retail outlets.

Infrastructure

The durables of a *wide-area store* are accessible to publishers and subscribers across a WAN link.

To designate a wide-area store, set its scope to *zone*, and assign the store to a zone.

Forwarding zones delineate the scope of wide-area stores. A forwarding zone contains a set of persistence clusters, which cooperate to forward messages among all the persistence clusters in that zone. (A zone can govern more than one wide-area store. A cluster can participate in more than one zone.)

i Note: Forwarding applies to only the *dynamic* durables in a wide-area store. (Static durables do not support wide-area behavior.)

i Note: Standard durables, shared durables, and last-value durables support wide-area forwarding. Maps do *not* support forwarding.

Persistence: Local and Wide-Area

Within each persistence cluster, a wide-area store replicates messages across the persistence services of the cluster. The persistence cluster assures message replication, retention, and delivery - subject to factors such as persistence limit parameters, store replication, the number of persistence services in the cluster, and local network connectivity. *Local connectivity* is sufficient to assure that messages are never lost, even if *WAN connectivity* is intermittent among the clusters of a zone.

Wide-Area Forwarding Zone Types

A wide-area store with a scope set to *zone* is assigned to a defined zone type:

- Peer to Peer (Full Mesh)
- Hub-and-Spoke (2 Hops)
- Limited Hub-and-Spoke (1 Hop)

Peer to Peer (Full Mesh)

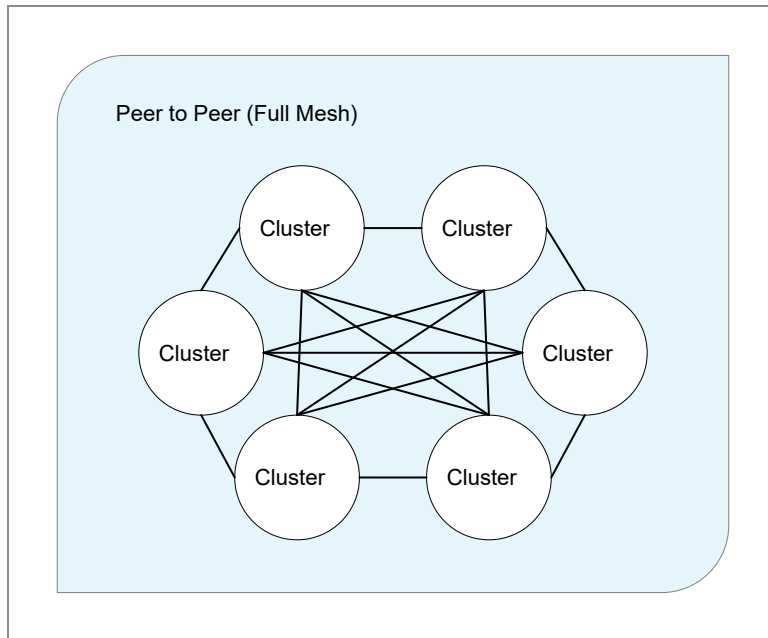
Messages flow directly between every pair of clusters in the zone in a single hop.

To scale:

- Add more clusters.
- Make multiple clusters appear as one. For example, a pool of clusters may all provide the same service with requests handled in round-robin, randomly, or another method.

This option features low latency (at the expense of network bandwidth and processing power in all persistence clusters). This zone type could result in best performance when adding persistence services to a cluster to expand client connection capacity.

Figure 44: Peer to Peer (Full Mesh)



Hub-and-Spoke Zone Type Commonalities

The following is true for the Hub-and-Spoke and Limited Hub-and-Spoke (One Hop) zone types:

- All communications must go through a hub. Spokes do not communicate directly to each other. Hubs can communicate to each other.
- Spoke clusters forward to the hub clusters, and the hub cluster forwards to all the spokes.
- One specific spoke only communicates to one specific hub.
- To scale, you can add hubs or spokes.
- Persistence services in the hub cluster use more resources than those in the spoke clusters.
- To scale, add hubs and/or spokes.
- Multiple hub machines can be made to look like a single hub through load balancing

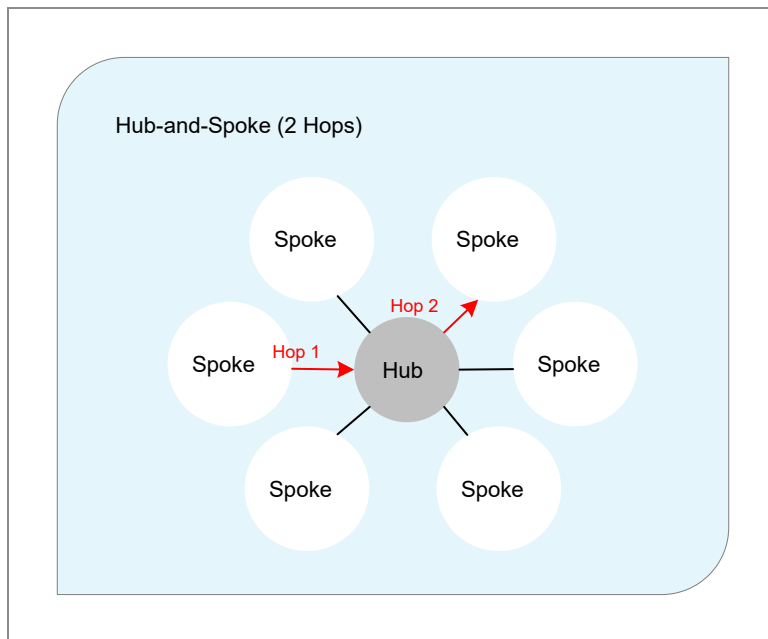
tools. This can make managing the spokes easier.

- Spokes can be used to partition the workload.

Hub-and-Spoke (2 Hops)

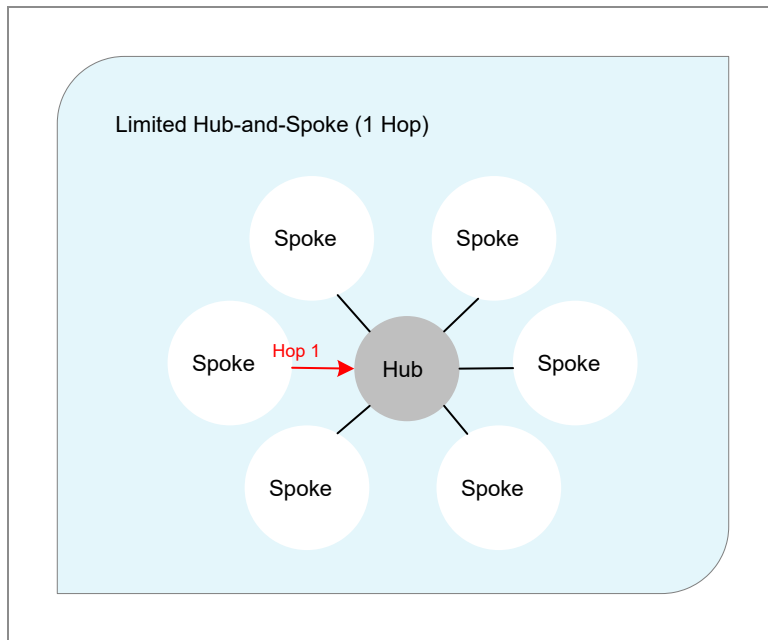
When using the Hub-and-Spoke zone type, a message from one spoke to another flows by way of the hub in two hops.

Figure 45: Hub-and-Spoke (2 Hops)



Limited Hub-and-Spoke (1 Hop)

When using the Hub-and-Spoke zone type, messages travel no farther than one hop, so a message cannot flow indirectly from spoke to spoke. Select this zone type to restrict the flow of information.

Figure 46: Limited Hub-and-Spoke (1 Hop)

Example: Wide-Area Behavior

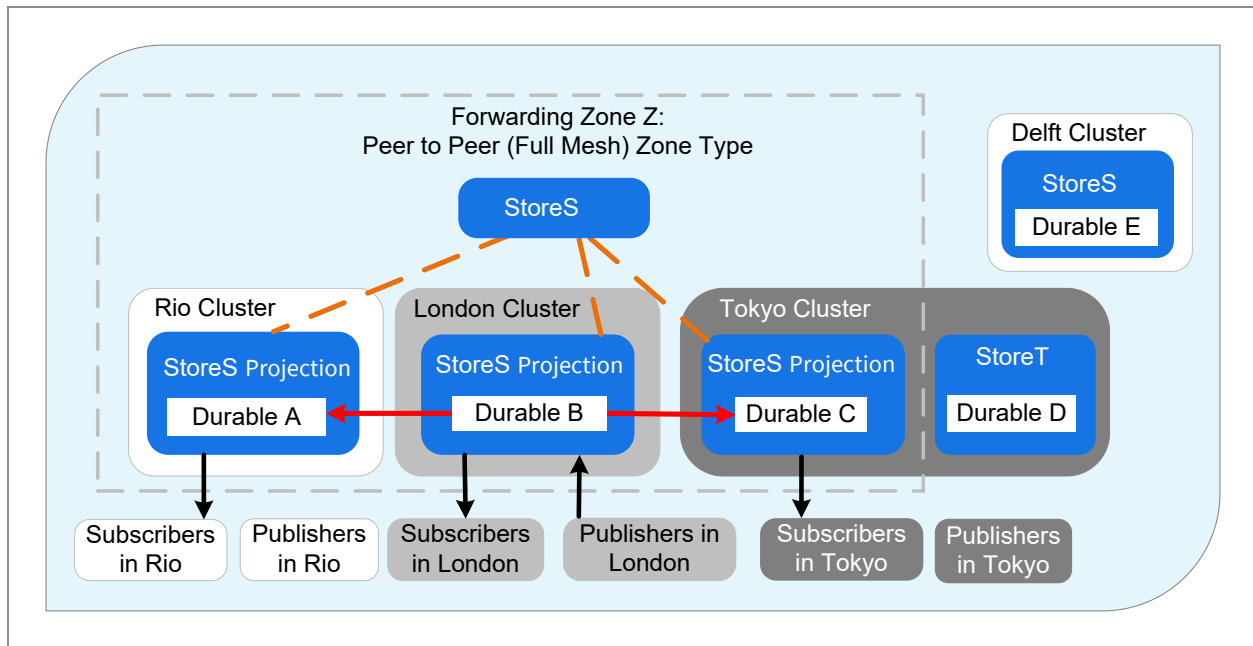
For example, in the diagram, StoreS is designated as a wide-area store, and assigned for forwarding in Zone Z. All the clusters in Zone Z implement projections of StoreS (orange dashed lines). The zone type of Z is a Peer to Peer (Full Mesh).

Application programs in Rio, London, and Tokyo can publish and subscribe to the durables of StoreS, and the clusters automatically ensure that durable subscriptions and messages are available as needed in all the projections.

In this example, when a publisher in London publishes a message, the message is potentially available to all matching durables and their subscribers throughout Zone Z (red arrows).

If the Rio cluster did not have any subscribers, then the London cluster would not forward it to Rio. This behavior conserves WAN bandwidth and persistence service memory.

Figure 47: Forwarding the Messages of Wide-Area Stores



Notice that two conditions are necessary to enable wide-area forwarding - a forwarding zone and a wide-area store assigned to it. Consider the backward-compatible results when either condition is false.

- The Delft cluster is outside of forwarding Zone Z. Even though the Delft cluster implements StoreS, which is designated as a wide-area store of Zone Z, the cluster does *not* forward StoreS, neither out from nor in to the Delft cluster.
- StoreT is not a wide-area store, so it remains functionally outside Zone Z. Instead it is local to the Tokyo cluster. Subscriptions to durables of StoreT do not cause messages to flow in from other clusters. Publishers to StoreT do not cause messages to flow out to other clusters.

A durable subscription is required to forward messages *into* a cluster, but is *not* required to forward messages *out* of a cluster. For example, even if durable B were not present in London, messages from London publishers on StoreS would still be available to durable subscribers in Rio and Tokyo.

Arranging a Wide-Area Store

To define a wide-area store, use the stores grid of the FTL server GUI.

Procedure

1. Configure the zone transport.

- a. For each persistence service in each cluster of the zone, configure the zone transport parameter.

Configure this parameter in the persistence service details panel. For flexibility, select one of the automatic transport types. Otherwise select one of the TCP types. Configure all the services analogously.

- b. For each cluster, configure the externally reachable addresses parameter.

If (and only if) you chose one of the automatic transport types in the previous step, then you must also configure externally reachable addresses for each cluster in the zone. Configure this parameter in the cluster details panel.

For each of the cluster's FTL core servers, configure its host and port address. Configure the same addresses that the services of the cluster use to contact the core servers.

2. Define a zone.

In the zones grid of the FTL server GUI, create a new zone.

Select the zone type. For details, see [Wide-Area Forwarding Zone Types](#).

- **Peer to Peer (Full Mesh)**
- **Hub-and-Spoke**
- **Limited Hub-and-Spoke (1 Hop)**

3. Add persistence clusters to the zone.

If you selected one of the two hub-and-spoke types, designate the role of one cluster as the hub and the others as spokes.

You may add additional clusters to a zone at any time.

You may change the hub or spoke role of a cluster only when the cluster is empty and not running.

4. Define a wide-area store.

In the stores grid of the FTL server GUI, create a new store. Set its scope parameter to *Zone*.

5. Assign the store to a forwarding zone.

In zone column of the stores grid, select a zone from the drop-down menu.

Assigning the store to a *zone* makes the store available in all clusters in the zone.

6. Bind the wide-area store to application endpoints.

In the applications grid, associate the appropriate endpoints with the wide-area store.

What to do next

Ensure runtime topology conditions; for an explanation, see [Wide-Area Forwarding: Run Time Conditions](#).

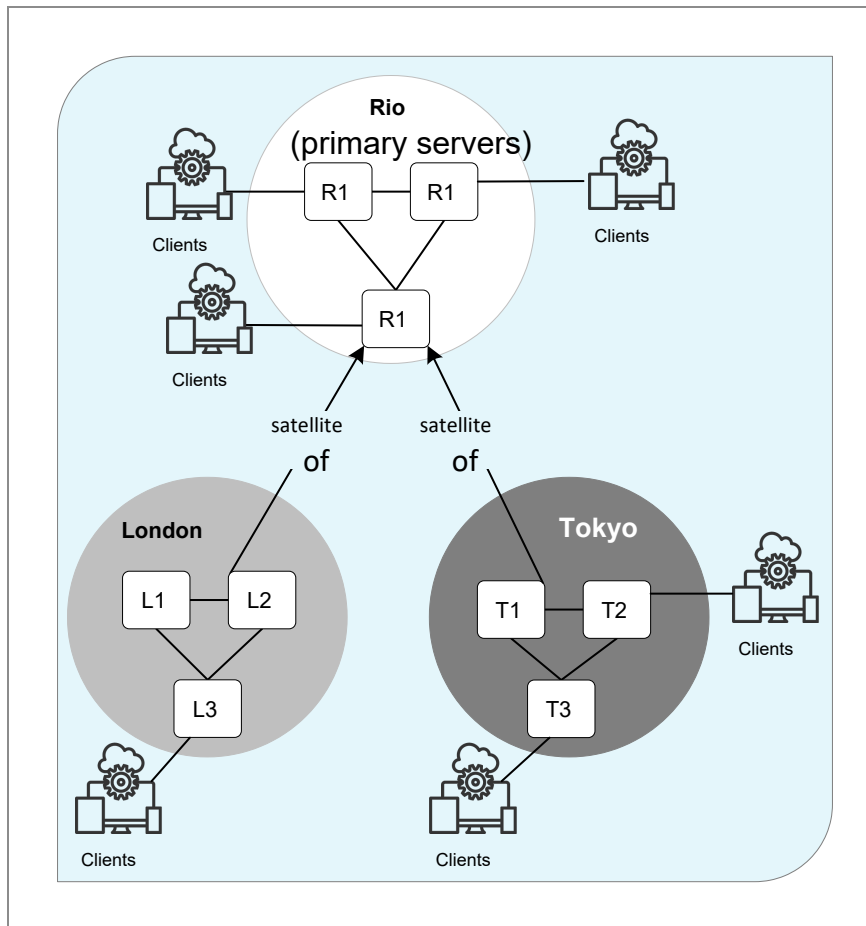
Wide-Area Forwarding: Run Time Conditions

Wide-area forwarding places configuration prerequisites on the runtime topology of FTL servers, and clusters of persistence services.

Intra-Realm Scope

Wide-area forwarding requires that all servers and services operate within a single realm, even though they are located at different geographic locations. The example in the diagram satisfies this condition by configuring the core servers in Rio as primary servers, and the core servers in London and Tokyo as satellites of the Rio primaries. Configure these relationships in the FTL server configuration file.

Figure 48: Example Intra-Realm Scope

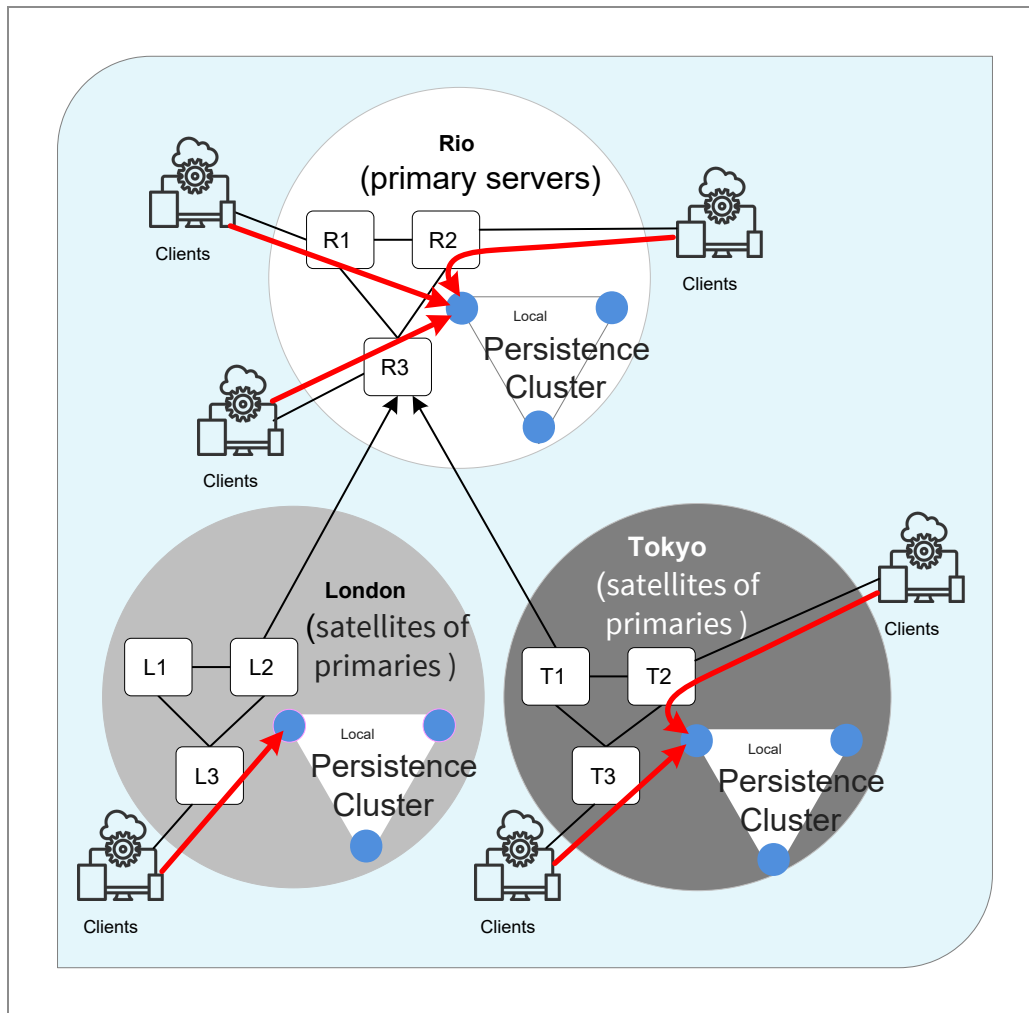


Core Server Clusters

The local realm services direct the client to the *local* persistence cluster (red arrows).

This mechanism places two conditions on the topology of the FTL servers and persistence services.

Figure 49: Example Core Server Clusters



If two persistence services belong to the same persistence cluster, then they must connect to realm services within the same cluster of core servers. Conversely, if two persistence services belong to two separate persistence clusters, then they must connect to realm services in different clusters of core servers.

The example in the diagram satisfies these conditions. For example, the persistence services (triplet of three blue services) in London all connect to realm services in FTL servers L1, L2, or L3. The diagram illustrates this connection as the arrow into L2, implying similar arrows from all three London persistence services to all three London FTL servers.

In contrast, a persistence service in London must *not* connect to FTL servers in Rio or Tokyo. The dashed arrow into T1 would *contradict* this condition.

Configure these relationships in the FTL server configuration file.

Durable Identity and Core Servers

As mentioned previously, the local realm services direct the client to the *local* persistence cluster (red arrows). This mechanism yields two corollaries and a condition. The corollaries are:

- If two clients connect to different core servers within the same cluster, and each subscribes to durable D, then they access the same durable.
- In contrast, if two clients connect to *distinct* clusters, and each subscribes to durable D, then they access different durables (which could have different message content).

The condition follows. To maintain consistent access to a durable, a client must always connect (and reconnect) to the same cluster of core servers.

Persistence Services and Clusters

If a cluster of FTL servers provide persistence services, those persistence services cooperate in a *persistence cluster*. A persistence cluster maintains a set of *persistence stores* which store messages in *durables*.

Administrators define persistence clusters and services within the realm definition. You can define any number of persistence clusters in the realm. A cluster consists of a set of named persistence services.

Each FTL server can provide any number of named persistence services. You assign each named persistence service to a specific FTL server in the FTL server configuration file. See [Persistence Service Configuration Parameters](#).

The persistence services within a cluster communicate among themselves to *replicate* message data and acknowledgment data. Replication ensures that the services can continue to deliver messages to subscribers, even if some of the individual services become unavailable.

A persistence cluster as a whole maintains a set of persistence stores, replicating their data among the persistence services for fault tolerance.

To ensure the benefits of a persistence cluster, namely, in-memory replication and fault-tolerant service, run each FTL server on a dedicated host computer.

Quorum and Leader

A persistence cluster can interact with clients *only* when a *quorum* of persistence services exists. This requirement ensures correct replication and fault tolerance.

When a quorum exists, one of the services takes the role of *leader*, that is, the service that interacts with clients. All other services in the quorum replicate the leader's data, but do not interact with clients.

Administrators can set the *weight* of each persistence service to determine an order of relative preference in which services become the leader.

See Also

- [Quorum Conditions: General Rule](#)
- [Persistence Service Fault Tolerance](#)

Quorum Conditions: General Rule

A valid quorum ensures correct replication and fault tolerance. As a general rule, a group of candidate services must satisfy *all* three of the following conditions to form a valid quorum.

(For special exceptions, see [Quorum Behaviors](#).)

- **Majority:** The number of candidate services that satisfy these conditions must be strictly greater than half of the number of services in the cluster definition (that is, a simple majority).
- **Reachable:** A majority of the defined services must be running and able to communicate with one another.
- **Non-Empty:** A majority of the defined services must be non-empty. (A service is *empty* when it starts or restarts without loading saved state. A service that has successfully joined a quorum is no longer empty.)

Cluster Size

For delivery assurance with fault tolerance, use a minimum cluster size of 3 services, so that a quorum can exist even when one service is unavailable.

For replication, use a minimum cluster size of 3 services.

A cluster of 2 services is invalid, because when one service is unavailable, the remaining service cannot form a quorum on its own. It is an error to configure only 2 services.

For apportioning message streams, a cluster of only 1 service is a valid configuration.

Similarly, a message broker cluster of only 1 service is a valid configuration.

Quorum Behaviors

The persistence services in a cluster exhibit these behaviors as they form and maintain a quorum.

Cluster Start

When the cluster starts for the first time, all the services are empty. The services wait until *all* the services in the cluster have started, and are running simultaneously. Then they constitute a quorum for the first time, and can begin interacting with clients to store messages.

Steady State

After forming that initial quorum, client interaction continues while the quorum exists (that is, a majority of services continue to satisfy all three conditions of [Quorum Conditions: General Rule](#)).

Tail End Loss

When a persistence service has a stored state, but might have suffered the loss of a tail end of the state, the persistence service is considered empty for the purpose of quorum formation.

The persistence service assumes tail end loss in the following circumstances:

- The persistence service was running at a disaster recovery site, and the disaster recovery site was activated.
- The persistence service was restored from a saved state. This applies only to situations where the administrator manually saved the state of the persistence

cluster. This does not apply when the disk persistence feature is enabled and the persistence service recovers from its non-backup disk persistence file. See [Saving and Loading Persistence State](#).

- The persistence service was restored from a backup disk persistence file. See [Persistence Clusters Status Table](#), [Persistence Clusters Status Table](#).

The tail-end loss assumption is cleared once the persistence service successfully joins a quorum.

Rejoining the Quorum

A service can rejoin an existing quorum, for example, after that service restarts, or becomes reachable (after recovery from network segmentation). The rejoining service copies the data state of the quorum leader.

The current leader remains in place, even when the rejoining service has higher weight. That is, weights are relevant only for leader election. Election occurs only when the quorum does not already have a leader.

Reforming a Quorum

If the services no longer satisfy the quorum conditions, client interaction stops. The quorum must reconstitute through any of the scenarios in the following table.

Reforming a Quorum

| Scenario | Description |
|--------------|---|
| Complete | The cluster can form a quorum of non-empty services (that is, they satisfy all three conditions of Quorum Conditions: General Rule). For example, this scenario can occur after recovery from network segmentation. Alternatively, it can occur after suspending the cluster, saving the state of its services, and, finally, restarting the services. |
| Force Quorum | <p>A majority of the defined services are reachable, but the candidates cannot satisfy the non-empty condition. After verifying preconditions, the administrator may force a quorum (using the FTL server monitoring interface). For more details, see Before Forcing a Quorum. Also see Cluster Details Panel, Force Quorum Setting: Force Quorum Delay.</p> <p>The services in the forced quorum copy the most recent message data state.</p> |

Persistence Service Fault Tolerance

Replication of data among a quorum of persistence services ensures that the cluster as whole remains complete and correct, despite temporary outage of some of its services.

For best results, insulate each persistence service host from the *other* persistence services and the factors that might make *them* unavailable. For example, ensure that service hosts rely on separate power circuits and separate network infrastructure.

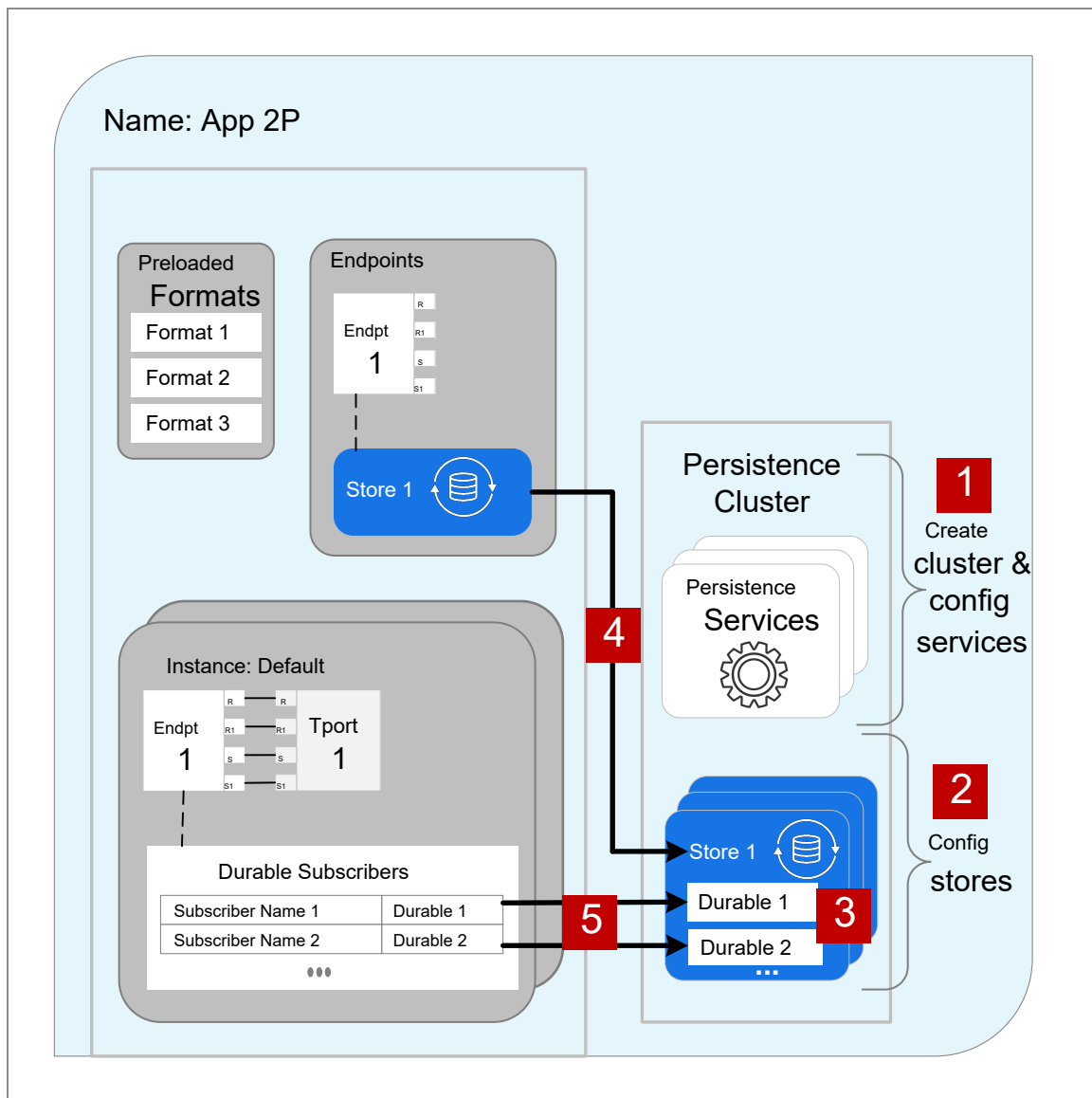
Do *not* locate persistence service hosts at geographically distant sites, as that would increase communication latency.

i Note: This level of fault tolerance alone is *not* sufficient to support disaster recovery. For information about the disaster recovery feature, see [Disaster Recovery](#).

Configuring Persistence

The realm definition schematic in the following diagram indicates five configuration tasks for persistence stores and durables. (Numbers in the diagram correspond to the task steps that follow.)

Figure 50: Configuring Persistence and Durables



Before you begin

Before starting this task, developers and administrators coordinate to determine the need for durables, and the parameters for those durables. See [TIBCO FTL Durable Coordination Form](#).

Procedure

1. Create a cluster and configure the persistence services that make up the cluster.

For concepts, see [Persistence Services and Clusters](#).

For the configuration interface and API considerations, see [Clusters Grid](#), and [Persistence Service Details Panel](#).

2. Configure the stores within the cluster.

For concepts, see [Publisher Mode](#).

For the configuration interface, see [Clusters Grid](#), and [Stores Grid](#).

3. Configure the dynamic durable templates and the static durables within each store.
See [Durable Behavior](#).

For the configuration interface, see [Stores Grid](#) and [Durable Details Panel](#).

4. Associate endpoints with stores, as appropriate.

For the configuration interface, see [Applications Grid](#).

5. Optional. Map durable subscribers to static durables.

If application programs use only dynamic durables, or use only durable names when subscribing to static durables, then you may skip this step.

For concepts and the configuration interface, see [Configuration of Durable Subscribers in an Application or Instance](#).

Stores Grid

The Stores grid presents persistence store definitions and their details. In edit mode, you can add and delete store definitions, and modify existing store definitions.

For consequences of modifying a store definition, see [Persistence Modifications: Store and Durable](#).

Levels

- Store
- Durable
- Content Matcher

Store Level

From the left menu, select **Stores**.

| Column | JSON Attribute | Description |
|------------------|--------------------|--|
| Store | name | <p>Required.</p> <p>Name of the store.</p> <p>Store names must be unique within a cluster.</p> <p>All names are limited to maximum length of 256 characters.</p> |
| Scope | | <p>Required.</p> <p>Select either cluster or zone from the drop-down menu.</p> |
| Cluster/Zone | | <p>Required.</p> <p>Assign the store to a persistence cluster or forwarding zone. The drop-down menu offers the available clusters or zones, depending on the value in the scope column.</p> |
| Publisher Mode | publisher_settings | <p>This parameter governs communication among publisher send calls, the store, and the endpoint's regular transports to produce different qualities of service. For complete details, see Publisher Mode.</p> |
| Replicated | replicated | <p>When enabled, the cluster replicates the store across its persistence services.</p> <p>When disabled, the cluster does not replicate the store. Only one persistence service holds the store's data.</p> |
| Store Size Limit | byteLimit | <p>The store may grow until it exhausts available memory or disk of any one of its persistence services. This parameter limits the volume of message data in the store. If publishing another message would exceed this limit, the</p> |

| Column | JSON Attribute | Description |
|---------------|----------------|--|
| | | store rejects that message. Zero is a special value, indicating no limit. The store may grow until it exhausts available memory of any one of its services. See Size Units Reference . |
| Last Modified | | This timestamp indicates the date and time of the most recent change to this store definition. |

Durable Level

The durable level presents the definitions of static durables and dynamic durable templates in the store.

From the left menu, select **Stores**. Click the expand button by **Durable Name** to display the following information for the durable.

| Column | JSON Attribute | Description |
|--------------|----------------|--|
| Durable Name | name | Required. Name of the static durable or dynamic durable template. (The name of an <i>individual</i> dynamic durable comes from subscriber create call, and not from administrative configuration.) Names must be unique within each persistence store. All names are limited to maximum length of 256 characters. |
| Durable Type | type | Select the type of durable definition from the drop-down menu. You may define either a <i>static</i> durable or a <i>dynamic</i> |

| Column | JSON Attribute | Description |
|----------------|----------------|---|
| | | <p>durable template of the following types:</p> <p>Standard</p> <p>The durable retains a message stream for exactly one durable subscriber.</p> <p>Shared</p> <p>Many subscribers can share this durable, which distributes each message to only one of those subscribers.</p> <p>Last-Value</p> <p>The durable retains only the most recent message for each sub-stream.</p> <p>For background information, see Durable Behavior.</p> |
| Acknowledgment | ack_settings | <p>Select an acknowledgment mode from the drop-down menu.</p> <p>This parameter applies only to <i>standard</i> and <i>shared</i> durables.</p> <p>When a subscriber finishes processing a message, the client library sends an acknowledgment to the durable. This interaction can be either Synchronous or Asynchronous. For complete details, see Acknowledgment Mode.</p> |
| Interest | interest | <p>Message interest determines a stream of messages:</p> <p>Store All Messages</p> <p>The durable collects the full message stream. (This behavior is available only for standard durables and shared durables.)</p> <p>Store Matching Messages</p> |

| Column | JSON Attribute | Description |
|-----------|-----------------------------|---|
| | | <p>The durable collects a sub-stream determined by a content matcher.</p> <p>Store Messages with Key</p> <p>The durable collects the sub-stream of messages that contain a key field. You must specify the <code>Key Field Name</code> parameter. This behavior is available only for last-value durables.</p> <p>Select a type of message interest from the drop-down menu.</p> |
| Key Field | <code>key_field_name</code> | <p>Required. Applies only to static last-value durables.</p> <p>Supply the name of the key field.</p> <p>The durable stores only the most recent message for <i>each</i> distinct value of that key field.</p> <p>Developers and administrators coordinate to determine the appropriate behavior (see TIBCO FTL Durable Coordination Form).</p> <p>When you set the durable type to static last-value, the GUI automatically sets this parameter value to the placeholder value <code>default-key</code>. You must explicitly change this placeholder to the actual key field name.</p> |

Content Matcher Level

This level applies only to static durables, and only when the [Interest](#) column is `Store Matching Messages`.

Developers and administrators coordinate to determine the appropriate behavior (see [TIBCO FTL Durable Coordination Form](#)).

Administrators configure a content matcher at this level to implement that behavior. A content matcher can match one or more fields.

For background information, see [Message Interest](#).

| Column | Description |
|-------------|---|
| Match Field | Enter a field name. For static last-value durables, one matcher clause must test the existence of the key field. |
| Match Type | Select a data type or a field-existence predicate from the drop-down menu. |
| Match Value | Enter a value to match. |

Store Detail Panel

The Store details panel presents the details of a persistence store definition. In edit mode, you can modify the definition.

From the left menu, select **Stores**. Select the three dots above a store, then select **View Details**.

| GUI Parameter | JSON Attribute | Description |
|------------------------|-----------------------|---|
| Dynamic Durables Limit | dynamic_durable_limit | Optional. This parameter limits persistence service memory growth by restricting the number of dynamic durables that the store can create. When a program attempts to create a dynamic durable in excess of this maximum, the create subscriber call throws an exception. Zero is a special value, indicating no limit. When absent, the default value is zero. For further details, see Persistence Limits . |
| Maximum Message | message_limit | This parameter limits the number of messages that can be held by the store. When this limit is reached, the client |

| GUI Parameter | JSON Attribute | Description |
|----------------------|-------------------------------|---|
| Limit | | <p>receives an exception on <code>tibPublisher_Send</code> or <code>tibPublisher_SendMessages</code> (when <code>store_confirm_send</code> mode is set on the store). However, the client library retries the sends. To have the exception presented to the user, the client application must specify a finite persistence retry duration.</p> <p>Note that even when messages are stored on disk, message metadata is still stored in memory. Hence, use this setting to constrain memory use.</p> <p>Zero is a special value indicating no limit. The store may grow until it exhausts available memory or disk of any one of its persistence services.</p> |
| Maximum Message Size | <code>max_message_size</code> | <p>Optional.</p> <p>Set the maximum allowed message size. If the application's message exceeds the configured limit, <code>tibPublisher_Send/tibPublisher_SendMessages</code> calls get an exception (if the store publisher mode is <code>confirm_send</code>). See Maximum Message Size.</p> |
| Swap Byte Limit | <code>swap_bytelimit</code> | <p>Optional.</p> <p>If message swapping is enabled, set a byte quantity threshold for the store above which messages are swapped to disk to keep store memory use limited. Valid values are -1 (unlimited), 0, or any positive byte count. Positive byte counts can be expressed with a suffix. For example, 100, 100m, 100mb are all valid. Default is 0.</p> <p>If <code>disk_swap</code> is set to <code>false</code>, this parameter is ignored.</p> |

Durable Details Panel

The Durable details panel presents the parameters of a static durable or a dynamic durable template. In edit mode, you can modify the parameter values.

From the left menu, select **Stores**. Select the left arrow above a store to display the durables. Select the three dots above a durable then select **View Details**.

The title of the panel indicates the location and name of the durable or template: Store / Durable.

For consequences of modifying this configuration, see [Persistence Modifications: Store and Durable](#).

For background information, see [Durable Behavior](#).

TTL Settings

| GUI Parameter | Description |
|----------------------|--|
| Message Time to Live | <p>Optional.</p> <p>This parameter limits persistence service memory growth by limiting the time (in seconds) that messages remain in the durable. When a message arrives in the durable, a timer counts this interval. The persistence service can delete that message at any time after the interval elapses.</p> <p>Zero is a special value, indicating no time limit on messages in this durable.</p> <p>When absent, the default value is zero.</p> <p>For further details, see Persistence Limits</p> |
| Durable Time to Live | <p>Optional.</p> <p>This parameter limits persistence service memory growth by limiting the time (in seconds) that a dynamic durable without any subscribers can remain in the store.</p> <p>When a dynamic durable no longer has any connected subscribers, a timer counts this interval. The persistence service can delete the durable at any time after the interval elapses. During the interval, the presence of any subscriber stops the timer and prevents deletion.</p> <div><p>Note: Using this feature weakens the quality of service of the dynamic durable. The durable no longer assures delivery to clients that disconnect without closing subscribers.</p></div> |

| GUI Parameter | Description |
|------------------------|--|
| | <p>Zero is a special value, indicating no time limit on dynamic disables without subscribers.</p> <p>When absent, the default value is zero.</p> <p>For further details, see Persistence Limits.</p> |
| Message Retention Time | <p>Acknowledged messages may be retained for this amount of time (in seconds), as measured from the time when the message was published. Zero is a special value indicating that message retention is disabled. When absent, the default value is zero.</p> <p>For further details, see Retention Time</p> |

Limits and Discard Policy Settings

| GUI Parameter | Description |
|----------------|--|
| Message Limits | <p>Optional.</p> <p>This parameter limits persistence service memory growth by limiting the number of messages in a durable. When a message arrives, the persistence service checks the number of messages in the durable. If the new message would exceed the limit, the service discards a message (see the discard parameter).</p> <p>A value of 0 indicates no limit on the number of messages in this durable.</p> <p>(When a durable discards the newest message, the publisher does not throw an exception.)</p> <p>You can set this parameter for static durables and for dynamic durable templates. If you change its value on a dynamic durable template, all of that template's individual dynamic durables reflect the new behavior.</p> <p>For further details, see Persistence Limits.</p> |
| Byte Limit | Optional. |

| GUI Parameter | Description |
|---------------------|---|
| Discard Policy | <p>This parameter limits persistence service memory growth by limiting the number of bytes in a durable. When a message arrives, the persistence service checks the bytes in the durable to ensure the total message size in a durable does not increase continuously past the durable byte limit. If the new message would exceed the limit, the service discards a message (see the discard parameter) and logs a warning.</p> <p>A value of 0 indicates no limit on the number of bytes in this durable.</p> <p>For further details, see Persistence Limits.</p> |
| Maximum Delivery | <p>With the toggle button, you can set this parameter to govern the message discard behavior of the message limit or byte limit parameter. The durable can discard either its oldest message or the new message.</p> <p>You can set this parameter for static durables and for dynamic durable templates. If you change its value on a dynamic durable template, all of that template's individual dynamic durables reflect the new behavior.</p> <p>Applies only to shared durables and standard durables with prefetch.</p> <p>This parameter limits the number of times that a persistence service will attempt to deliver each message in the shared durable.</p> <p>Zero is a special value, indicating no limit on delivery attempts.</p> <p>Administrators can this set parameter to zero or any positive integer strictly greater than one. (The value 1 is invalid.)</p> <p>When absent, the default value is zero.</p> <p>Note: Robust message callback methods verify field data types before operating on field data content.</p> <p>Nonetheless, sometimes a message with unexpected data content can cause a subscribing application to abruptly exit. When the persistence service repeatedly delivers such a message to the subscribers of a shared durable, it could potentially cause all the subscribing applications to exit. To limit the damage that such messages can cause, set this parameter to a non-zero value that is less than the expected number of subscribed applications</p> |

Prefetch Settings

| GUI Parameter | Description |
|----------------|---|
| Prefetch Count | <p>With prefetch enabled, when a subscriber is ready, the store delivers a portion of the message stream. This value, a positive integer, limits the number of messages in a portion. To disable prefetch, supply a value of 0.</p> <p>Prefetch can only be disabled for standard durables. When prefetch is disabled, a direct path transport between publisher and subscriber is the primary means of message delivery.</p> <p>For more information, see Durable Prefetch Count.</p> <p>You cannot enable or disable prefetch <i>after</i> a durable configuration has been deployed to the FTL server. However, you can still change the prefetch value.</p> |

Swap Settings

| GUI Parameter | Description |
|-----------------|--|
| Swap Byte Limit | <p>Optional.</p> <p>If message swapping is enabled, set a threshold above which messages are swapped to disk to keep store memory use limited.</p> <p>Choose one of the following options.</p> <ul style="list-style-type: none">• Swap Everything: All messages are swapped.• Do Not Swap: The persistence service may keep all messages in memory.• Swap After: The persistence service may keep messages in memory, up to the specified limit in bytes. |

| GUI Parameter | Description |
|---------------|---|
| | If disk_swap is set to false for the persistence cluster, this parameter is ignored. Note: For shared durables or standard durables with prefetch, setting this value to a higher limit can improve throughput in some cases. |

Acknowledge Batch Settings

| GUI Parameter | Description |
|---------------|---|
| Batch Size | <p>In asynchronous acknowledgment mode, each client immediately sends accumulated acknowledgments when the number of waiting acknowledgments exceeds this number, even if the batch time interval has not elapsed.</p> <p>This setting only applies to durables or durable templates with asynchronous acknowledgment mode.</p> |
| Batch Timeout | <p>In asynchronous acknowledgment mode, each client immediately sends accumulated acknowledgments when the oldest acknowledgment has waited for this interval, in seconds, even before exceeding the batch count.</p> <p>This setting only applies to durables or durable templates with asynchronous acknowledgment mode.</p> |

Durable Prefetch Count

The prefetch count parameter of a durable limits the number of messages per batch that the durable can deliver to a subscriber. The backlog buffer size of the persistence client transport also limits the number of messages that can be delivered at once. (To set the **Backlog Buffer Size**, see [Persistence Service Details Panel](#).) These values affect delivery performance.

Positive integer values enable prefetch, and determine the maximum batch size. Zero disables prefetch, however, zero is a legal value only for standard durables.

Appropriate values for the prefetch count depend on the use case.

For *standard* durables that will *back a direct path*, set this parameter to zero for best performance. (Standard durables are already optimized for direct path delivery without prefetch.)

For *standard* durables that will deliver messages in the *absence of a direct path*, set this parameter to a non-zero starting value of 1024. You can tune this parameter to optimize delivery speed. If the durable's message backlog grows, try increasing this value. If the subscriber is overloaded and discards messages, try decreasing this value.

For *shared* durables, you can tune this parameter empirically to improve message throughput:

- If some subscribers are idle while others are overloaded, try decreasing this value.
- If the durable has a large message backlog, yet subscribers could process more messages, try increasing this value.

For *last-value* durables, you can tune this parameter to control the flow of messages from a fast publisher.

With *wide-area stores*, the best practice is non-zero prefetch with either shared durables or standard durables.

i Note: It is illegal to enable or disable prefetch *after* a durable configuration has been deployed to the FTL server. (Nonetheless, you may still change the prefetch value from one positive integer to another positive integer.)

Clusters Grid

The Clusters grid presents definitions of persistence clusters and persistence services, along with selected details. In edit mode, you can add and delete persistence cluster and service definitions, and modify existing definitions.

For background information, see [Persistence Services and Clusters](#).

For consequences of modifying this configuration, see [Persistence Modifications: Store and Durable](#), and [Quorum Behaviors](#).

Levels

- Cluster
- Service Set
- Service

Cluster Level

From the left menu, select **Clusters**.

| Column | Description |
|-----------------------|--|
| Cluster | <p>Required.</p> <p>Name of the persistence cluster.</p> <p>Cluster names must be unique within the realm.</p> <p>All names are limited to maximum length of 256 characters.</p> |
| Last Modified | <p>This timestamp indicates the date and time of the most recent change to this persistence cluster definition.</p> |
| Disk Persistence Mode | <p>Select the disk persistence mode. This option applies to replicated stores only. If you turn on disk persistence but have not turned on swapping, the messages are still held in memory. Once you turn on swapping, the messages are swapped out to disk and freed from memory.</p> <p>None</p> <p>Disable disk persistence.</p> <p>sync</p> <p>The client returns from a send-message call after the message has been written to a majority of disks. This mode generally provides consistent data and robustness, but at the cost of increased latency and lower throughput. If the cluster restarts, no data is lost; performance is subject to disk performance.</p> <p>async</p> <p>The client may return from a send-message call before the message has</p> |

| Column | Description |
|-------------|---|
| | been written to disk by majority of the FTL servers. This mode generally provides less latency and more throughput, but messages could be lost if a majority of servers restart shortly after the API call. |
| DR Enabled | <p>Disaster Recovery Replication</p> <p>In the GUI, select this checkbox to enable replication to the disaster recovery service set.</p> <p>In the web API, set this attribute to <code>true</code> to enable replication to the disaster recovery set.</p> |
| DR Protocol | <p>If you enable disaster recovery replication, the services in the cluster use this transport protocol to establish a communication bus for that purpose.</p> <p>You can set other parameters of the transport for each service in the cluster.</p> |
| Primary Set | <p>In the GUI, select one service set to be primary.</p> <p>In the web API, set this attribute to the name of the primary service set.</p> <p>The opposite service set is the standby set.</p> |

Service Set Level

Each persistence cluster can contain either one or two sets of services. If the cluster *enables* disaster recovery replication, then configure two service sets. If the cluster does *not* enable disaster recovery replication, then only one service set is relevant. (See the DR Enabled column.) For background information, see [Disaster Recovery](#).

| Column | Description |
|----------|---|
| Set Name | <p>Name of the service set.</p> <p>You may use the default set names or supply other names.</p> <p>Set names must be unique within the cluster.</p> <p>All names are limited to maximum length of 256 characters.</p> |

| Column | Description |
|----------------------|---|
| Cluster Set Protocol | <p>Persistence services with a cluster's service set communicate with one another using this transport.</p> <p>Select the transport protocol from the drop-down menu in this column. As needed, set the host, subnet, and port parameters in service-level columns. Set other parameters in the service details panel.</p> <p>Notice that all the services in a service set use the same cluster protocol, though individual services can supply different transport parameters.</p> <p>For connection-oriented protocols, such as static TCP, the realm service automatically arranges the listening ends and connecting ends.</p> |
| Client Protocol | <p>Client applications communicate with persistence services in the cluster using the client transport.</p> <p>Select the client transport protocol from the drop-down menu in this column. As needed, set the host, subnet, and port parameters in service-level columns. Set other parameters in the service details panel.</p> <p>Notice that all the services in a cluster use the same client protocol, though individual services can supply different transport parameters.</p> <p>For connection-oriented protocols, such as static TCP, the persistence service always is the listening end, and clients the connecting end.</p> |

Service Level

If the cluster *enables* the disaster recovery feature, then each set must contain equivalent services. For background information, see [Disaster Recovery](#).

| Column | JSON Attribute | Description |
|--------------|----------------|---|
| Service Name | name | <p>Required.</p> <p>Name of the persistence service.</p> <p>Persistence service names must be globally unique within the realm.</p> |

| Column | JSON Attribute | Description |
|---------------------|----------------|--|
| | | All names are limited to a maximum length of 256 characters. |
| Weight | weight | <p>Optional.</p> <p>This value must be an integer in the range [1,10]. Larger values indicate greater preference that the service becomes the leader.</p> <p>When absent, the default weight is 10.</p> |
| Cluster Host/Subnet | | Persistence services inherit the cluster transport protocol from the service set. |
| Cluster Port | | <ul style="list-style-type: none"> For connection-oriented transports, set the host name and port for establishing a connection (both are required). For dynamic TCP transports, you may set a subnet mask and port (both are optional). |
| Client Host/Subnet | | Persistence services inherit the client transport protocol from the service set. |
| Client Port | | <ul style="list-style-type: none"> For connection-oriented transports, set the host name and port for establishing a connection (both are required). For dynamic TCP transports, you may set a subnet mask and port (both are optional). |
| DR Host | | Persistence services inherit the disaster recovery transport protocol from the cluster. |
| DR Port | | Set the host name and port for establishing a connection (both are required). |

Cluster Details Panel

The Cluster details panel presents the details of a persistence cluster definition. In edit mode, you can modify the parameter values.

From the left menu, select **Clusters**, select the three dots above a cluster, then select **View Details**.

Heartbeats and Timeouts

You can adjust the persistence heartbeat and timeout parameters, which detect persistence service availability. For best results, use the default values, except to resolve specific issues or as directed by TIBCO personnel.

| GUI Parameter | JSON Attribute | Description |
|---|-----------------------------------|---|
| Persistence Heartbeat (Server to the Client) | client_ pserver_ heartbeat | The leader persistence service sends heartbeats to its clients at this interval, in seconds. The default is 2 seconds. |
| Timeout Interval (Server to Client) | client_ timeout_ pserver | When the leader's heartbeat is silent for this interval, in seconds, its clients seek to connect to a new leader from among the other services in the cluster. The default is 5 seconds. When a client's heartbeat is silent for this interval, the persistence service clears all state associated with the client. |
| Persistence Heartbeat (Server to Server) | pserver_ pserver_ heartbeat | Persistence services in the cluster exchange heartbeats at this interval, in seconds. The default is 0.5 seconds. |
| Timeout Interval (Server to Server) | pserver_ timeout_ pserver | When the heartbeat of any service in the cluster is silent for this interval, in seconds, the remaining services attempt to form a new quorum. The default is 3 seconds. |
| Persistence Heartbeat | inter_ cluster_ | The leader persistence services of different persistence clusters in a forwarding zone exchange heartbeats at this |

| GUI Parameter | JSON Attribute | Description |
|--|-----------------------|--|
| (Cluster to Cluster) | heartbeat | interval, in seconds. The default is 2 seconds. |
| Timeout Interval (Cluster to Cluster) | inter_cluster_timeout | When the heartbeat from a leader persistence service in a second cluster is silent for this interval, in seconds, the leader of the first cluster will attempt to reconnect. The default is 5 seconds. |

Disk Settings

You can enable or disable the ability for this cluster's store to use disk-based persistence, or disk swap space when needed.

You can enable automatic disk persistence compaction so the persistence services in the cluster automatically starts a compaction when certain conditions are met. For details, see [Compact Disk Persistence Files with Persistence Service Online](#).

| GUI Parameter | JSON Attribute | Description |
|------------------|----------------|---|
| Message Swapping | disk_swap | <p>Toggle switch to enable or disable message swapping for this cluster. The default setting is Disabled.</p> <p>When message swapping is enabled, the server can swap messages from process memory to disk. Swapping allows the server to free process memory for more incoming messages, and to process message volume in excess of memory limit. When the server swaps a message to disk, a small record of the swapped message remains in memory.</p> <p>This can be enabled with or without disk persistence. If enabled without disk persistence, data is written to a temporary swap file. This option can apply to replicated or non-replicated stores.</p> |

| GUI Parameter | JSON Attribute | Description |
|---------------------------------|--|--|
| | | <p>Note: When using message swapping, if all FTL servers in the cluster are restarted simultaneously, messages in the store prior to the restart are not retrievable after the restart unless disk persistence is also enabled.</p> |
| Disk Persistence Mode | disk_persistence | <p>Select the disk persistence mode.</p> <p>None</p> <p>Disable disk persistence.</p> <p>sync</p> <p>The client returns from a send-message call after the message has been written to a majority of disks. This mode generally provides consistent data and robustness, but at the cost of increased latency and lower throughput. If the cluster restarts, no data is lost; performance is subject to disk performance.</p> <p>async</p> <p>The client may return from a send-message call before the message has been written to disk by majority of the FTL servers. This mode generally provides less latency and more throughput, but messages could be lost if a majority of servers restart shortly after the API call.</p> |
| Automatic compaction | disk_compact | <p>Set to enable or disable automatic disk compaction. See the "Compact when in-use ration below" setting which follows in this table.</p> <p>Changing the setting does not require a restart.</p> |
| Compact when in-use ratio below | disk_compact_settings.min_disk_inuse_ratio | <p>Set a value between 0 and 1, inclusive. The persistence service attempts to keep the ratio of disk inuse size to disk allocated size at or above the specified value. If the ratio of in-use to allocated size falls below this value, the persistence service can start a compaction.</p> |

| GUI Parameter | JSON Attribute | Description |
|---------------|----------------|---|
| | | Automatic compaction does not occur if the disk inuse size is extremely small. |
| | | To use <code>disk_compact_settings.min_disk_inuse_ratio</code> , automatic compaction must be enabled via <code>disk_compact</code> . |

Cluster Limits and Force Quorum Settings

| GUI Parameter | JSON Attribute | Description |
|--------------------|---------------------------------|---|
| Force Quorum Delay | <code>force_quorum_delay</code> | <p>Automatically force a quorum after this delay, in seconds. Zero is a special value indicating that automatic force formation is disabled.</p> <p>This setting can be used to ensure that the quorum always forms without administrator intervention. A majority of non-empty servers is usually enough to form a quorum, however in some situations (such as DR activation or recovery from a backup file), the quorum can't form unless all members are present. If some servers are still missing, this option forces the quorum to form automatically after the specified delay.</p> <p>Administrators may still manually force a quorum if this setting is configured. See Quorum Behaviors and Before Forcing a Quorum.</p> |
| Byte Limit | <code>bytelimit</code> | <p>This parameter limits the volume of message data in the cluster. If publishing another message would exceed this limit, the cluster rejects that message.</p> <p>The store may grow until it exhausts available memory or disk of any one of its persistence services.</p> <p>Zero is a special value, indicating no limit. The store may</p> |

| GUI Parameter | JSON Attribute | Description |
|---------------|----------------|--|
| | | <p>grow until it exhausts available memory of any one of its services.</p> <p>When messages are stored on disk, message metadata is still stored in memory. Use this setting to constrain memory for message metadata.</p> <p>See Size Units Reference.</p> |
| Message Limit | message_limit | <p>This parameter limits the number of messages that can be held by the cluster. When this limit is reached, the client receives an exception on <code>tibPublisher_Send</code> or <code>tibPublisher_SendMessages</code> (when <code>cluster_confirm_send</code> mode is set on the cluster). However, the client library retries the sends. To have the exception presented to the user, the client application must specify a finite persistence retry duration.</p> <p>Note that even when messages are stored on disk, message metadata is still stored in memory. Hence, use this setting to constrain memory use.</p> <p>Zero is the default value indicating no limit. The cluster may grow until it exhausts available memory or disk of any one of its persistence services.</p> |

Externally Reachable Addresses

When using persistence with zones of clusters that use an auto inter-cluster transport to address the cluster (or to address a load balancer's interface), use this field to identify one or more ports for the core servers of this cluster (or the port for the load balancer).

| GUI Parameter | Description |
|---------------|--|
| Host | Enter either an IP address or a resolvable hostname. |
| Port | Enter the port number to use to access this host. |

Persistence Service Details Panel

The Persistence Service details panel presents the details of a persistence service definition. In edit mode, you can modify the definition.

From the left menu, select **Clusters**, select the three dots above a cluster's **Server Name**, then select **View Details**.

The title of the panel indicates both the cluster and service names: Cluster / Service.

For background information, see [Persistence Services and Clusters](#).

For consequences of modifying this configuration, see [Persistence Modifications: Store and Durable](#).

Service Parameters

| GUI Parameter | Description |
|---------------|--|
| Server Weight | Optional. This value must be an integer in the range [1,10]. Larger values indicate greater preference that the service becomes the leader. When absent, the default weight is 10. |

Transports

Each persistence service definition must configure a cluster transport and a client transport.

You can also configure an alternate client transport, and a disaster recovery transport, each of which enables optional functionality.

Supply the transport parameters as needed. The available parameters depend on the transport protocol. In general, only a subset of parameters for that transport protocol are available, as the FTL server automatically supplies values for the remainder.

For transport parameter details, see [Transport Concepts](#), and [Transport Details Panel](#).

For background information, see [Persistence Service Transports](#).

| GUI Parameters Grouping | Description |
|----------------------------------|--|
| Client Transport | <p>Client applications communicate with persistence services in the cluster using this transport.</p> <p>Persistence services inherit the client transport protocol from the service set. Select the transport Protocol in the Client Protocol column in the clusters grid.</p> <p>Set other parameters in this section:</p> <ul style="list-style-type: none"> • The Backlog Buffer Size of the persistence client transport limits the number of messages that can be delivered at once. (Messages may also be limited by the prefetch count parameter. See Durable Prefetch Count.) • Receive Spin Limit is an advanced function. See Receive Spin Limit. <p>See also Client Transport and Non-Blocking Send.</p> |
| Cluster Transport | <p>Persistence services with the cluster communicate with one another using this transport.</p> <p>Persistence services inherit the cluster transport protocol from the service set. Select the transport protocol in the Cluster Protocol column of the clusters grid.</p> <p>Set other parameters in this section.</p> |
| Alternate Client Transport | <p>Optional.</p> <p>When defined, clients on the associated hosts communicate with this persistence service using the alternate transport. All other clients use the regular client transport defined for the cluster.</p> <p>To define an alternate client transport for clients on specific host computers, supply a transport protocol and other transport parameters. Add hosts to the list of Associated Client Host Names.</p> <p>For background information, see Alternate Client Transport.</p> <p>See also Client Transport and Non-Blocking Send.</p> |

| GUI Parameters Grouping | Description |
|-----------------------------------|---|
| Disaster Recovery Transport | <p>Optional.</p> <p>When two persistence services need to communicate across the WAN for disaster recovery replication, they establish a transport using this definition.</p> <p>Persistence services inherit the disaster recovery transport protocol from the cluster. Select the transport protocol in the DR Protocol column of the clusters grid.</p> <p>Set other parameters in this section.</p> <p>For background information, see Disaster Recovery.</p> |
| Zone Transport | <p>Optional.</p> <p>When two persistence services need to communicate across the WAN to forward wide-area stores, they establish a transport bus using parameter values in this definition.</p> <p>You may modify the default parameter values in this section.</p> |

Zones Grid

The Zones grid presents definitions of forwarding zones and clusters for wide-area durables, along with selected details. In edit mode, you can add and delete zone definitions, and modify existing definitions.

For background information, see "Wide-Area Durables."

Levels

- Zone
- Cluster

Zone Level

| Column | Description |
|--------|--|
| Zone | Required. Name of the forwarding zone. Zone names must be unique within the realm. All names are limited to a maximum length of 256 characters. |
| Active | You can activate or deactivate forwarding of wide-area durables with a zone. |
| Type | A zone can operate as a full mesh, as a hub-and-spoke network, or as a limited hub-and-spoke network. For details, see Arranging a Wide-Area Store . |

Cluster Level

| Column | Description |
|---------|--|
| Cluster | Required. Name of a persistence cluster. All names are limited to maximum length of 256 characters. |
| Role | When the zone type is either hub-and-spoke or limited hub-and-spoke, one cluster in the zone has the <i>hub</i> role, while all the others have the <i>spoke</i> role. |

Zone Settings Panel

The Zone settings panel presents the details of a forwarding zone definition. In edit mode, you can modify the parameter values. The panel is split into Hub Settings and Spoke Settings, containing the same parameters.

From the left menu, select **Zones**, select the three dots above a zone, then select **View Details**.

Durable Settings that Depend on Cluster Role

You can set parameter values that apply to message delivery and acknowledgment between clusters of the zone. Settings depend on the role of each cluster within the zone.

Hub Settings and **Spokes Settings** follow:

- **TTL Settings:** Message Time To Live
- **Message Limit and Discard Policy Settings:** Message Limit, Discard Policy, Maximum Delivery Attempts, Byte Limit
- **Prefetch Settings:** Prefetch Count
- **Acknowledgement Batch Settings:** Batch Size
- **Swap Byte Limit**

For descriptions of these parameters, see [Durable Details Panel](#).

Note that the **Acknowledgement Batch Settings > Batch Size** parameter refers to how often one persistence cluster acknowledges messages forwarded from another persistence cluster. However, this does not set a maximum time limit before an acknowledgement is sent. If the count of pending messages never reaches the batch size, those messages are acknowledged at the discretion of the persistence service.

Defining a Static Durable

Administrators must define each and every static durable in the realm definition.

(Dynamic durables are simpler to use because they do not require pre-configuration of each individual durable.)

Procedure

1. Coordinate with application developers to determine the requirements.
2. Define the endpoint in the application.
See [Stores Grid](#)
3. Associate the endpoint with a persistence store.
See [Endpoint Details Panel](#)
4. Define static durables.

The parameter values depend on the type of durable.

5. Optional. Define a subscriber name mapping.

i Note: Subscriber name mapping is an advanced topic.

If application programs supply only durable names (and not subscriber names) when subscribing to static durables, then you may omit this step.

6. Ensure that the persistence services are running.

Enabling Dynamic Durables

By defining a dynamic durable template, administrators enable application programs to create many similar dynamic durables without further administrative action.

Procedure

1. Coordinate with application developers to determine the requirements.
2. Define the endpoint in the application.
See [Stores Grid](#)
3. Associate the endpoint with a persistence store.
See [Endpoint Details Panel](#)
4. Select or define a dynamic durable template.
Select one of the built-in templates with its default parameter values.
Alternatively, define your own template and set its parameter values.
5. Associate the endpoint with that template.
See [Endpoint Details Panel](#).
6. Ensure that the persistence services are running.

Result

Programs can use the template to dynamically create any number of dynamic durables, each with a different durable name.

Enabling Key/Value Maps

By defining a dynamic durable template, administrators enable application programs to use the map API without further administrative action. Programs can use the map API to access a persistence store as if it were a simple database, mapping from keys to values.

Procedure

1. Coordinate with application developers to determine the requirements.
2. Define or select a store to contain one or more maps.
3. Define the endpoint in the application.

See [Stores Grid](#)

4. Associate the endpoint with a persistence store.

See [Endpoint Details Panel](#)

5. Select or define a dynamic durable template.

Select the built-in template named `ftl.map.template`, with its default parameter values.

Alternatively, define your own template with durable type `Dynamic/Last-Value` and set its parameter values.

6. Associate the endpoint with that template.

See [Endpoint Details Panel](#).

7. Ensure that the persistence services are running.

Result

Programs can use the template to dynamically create any number of maps, each with a different map name.

Built-In Dynamic Durable Templates

TIBCO FTL defines the following built-in dynamic durable templates with default parameter values.



Tip: The simplest way to use durables is with these built-in templates. (Nonetheless, if you need to customize the parameter values you can define your own dynamic durable templates.)

| Template Name | Description |
|-------------------------------------|---|
| <code>ftl.pubsub.template</code> | Template for dynamic durables with message broker semantics. |
| <code>ftl.standard.template</code> | Template for dynamic standard durables with default parameter values. |
| <code>ftl.shared.template</code> | Template for dynamic shared durables with default parameter values. |
| <code>ftl.lastvalue.template</code> | Template for dynamic last-value durables with default parameter values. |
| <code>ftl.map.template</code> | Template for dynamic map durables with default parameter values. |

Configuration of Durable Subscribers in an Application or Instance

When developing new applications that use static durables, application developers and administrators coordinate to determine either durable names or subscriber names. Administrators must configure the realm accordingly.



Tip: This topic does not apply when an application uses only dynamic durables. Dynamic durables are always simpler to use. Static durables continue to support existing projects.

Background

Each application process supplies either a unique *durable name* or a unique *subscriber name* in each create subscriber call. To understand this distinction, see “Durable Subscribers” in TIBCO FTL [Development](#).

Durable Considerations

If the application supplies only durable names, the administrator need not define a mapping, because the mapping applies only to subscriber names.

Similarly, if the application uses only dynamic durables, the administrator need not define a mapping, because the mapping applies only to static durables. See instead, [Enabling Dynamic Durables](#).

If you change the name of a currently deployed durable, you will receive a message like: Changing the name of a currently deployed durable will cause its stored message data to be deleted when the new realm configuration is deployed.

Subscriber Name Mapping

i Note: Subscriber name mapping is an advanced topic.

If the application program supplies subscriber names, the administrator must define a mapping from each subscriber name to a durable name: that is, to a static durable defined within the store. Administrators configure this mapping in the endpoint details panel.

The subscriber name and the durable name may be either identical strings or different strings. For administrative simplicity, use identical strings if possible.

For consequences of modifying this mapping, see [Persistence Modifications: Store and Durable](#).

See Also:

- [Endpoint Details Panel](#)
- [TIBCO FTL Durable Coordination Form](#)

Instance Matching for Subscriber Name Mapping

The FTL server's instance matching algorithm determines the mapping from subscriber names to durables.



Note: Subscriber name mapping is an advanced topic.

[Instance Matching](#) describes the matching algorithm that the FTL server uses to select an application instance, and so determine the implementation of endpoint publisher and subscriber objects using transport connectors.

After a first pass determines those transport connections, a second pass uses the identical matching algorithm to determine a mapping from subscriber names to durables.

Because the two passes are independent, you can configure the subscriber name mapping independently from instance connections in separate application instance definitions. For example, you could key transport connections to the host, and key the subscriber name mapping to the identifier.

You can use this independence to reduce the number of application instance definitions.

Configuring a Default Durable

To configure the default durable on an endpoint, open the endpoint's detail panel, add a subscriber name mapping, and supply `_default` as the subscriber name.

The durable name can be any durable you have defined in the store.



Note: Subscriber name mapping is an advanced topic.

For background information, see [Default Durable](#).

Persistence Service Transports

Each persistence service uses special-purpose transports of two kinds: client transports and cluster transports.

- *Client transports* communicate between a persistence cluster and its client processes.

- *Cluster transports* communicate within a persistence cluster.

**Note:**

Administrators usually use the default values for these transports.

However, when needed, you can configure the details of these transports as part of the persistence service definition (rather than as separate transport definitions). The remainder of this topic presents use cases for non-default values.

To set non-default values, see [Persistence Service Details Panel](#).

Mixed Environment

Environments that combine components from Release 6.x *and* Release 5.x, such as during migration to Release 6.x, could require non-default values carried over from Release 5.x.

Client Transport and Non-Blocking Send

Administrators can choose whether the client transport uses blocking or non-blocking send. This setting affects only the client end of the transport, as the persistence service end always uses non-blocking send.

This choice is relevant *only* when the publisher mode is [Store - Send](#), because otherwise the application publishing rate is limited by confirmations from the persistence service. (See [Publisher Mode](#).)

Furthermore, this choice has an effect on behavior only during peak message bursts, that is, if application publishing overwhelms the persistence service leader:

- To favor maximum application publishing speed while accepting potential message loss in the store, choose non-blocking send.
- To favor minimum message loss in the store while throttling application publishing speed, choose blocking send.

Disaster Recovery Transport

For background information, see [Disaster Recovery](#).

For configuration, see [Preparing FTL Servers for a Disaster Recovery Site](#), and [Persistence Service Details Panel](#).

Alternate Client Transport

Consider a heterogeneous environment in which the client host computers have different transport capabilities. For best performance, each client would use the fastest available transport protocol.

In such situations you can define a second client transport, called the *alternate client transport*, and assign clients on specific hosts to use that alternate transport. All other clients use the regular client transport.

To configure this transport, see [Persistence Service Details Panel](#).

Publisher Mode

When an endpoint uses a persistence store, its publisher send calls communicate outbound messages to the store, as well as to the endpoint's regular transports. Depending upon application requirements, the publisher send call can also wait for the store to confirm that it has received and stored each message. Different sequences of these subtask elements produce different qualities of service.

Administrators must select the publisher mode that best suits the application's needs.

Persistence Store Publisher Modes

| Publisher Mode | JSON Attribute | Description |
|-----------------------|-----------------------|---|
| Store - Send | store_send_noconfirm | <p>Highest throughput. Minimal message latency. More robust delivery than regular reliability.</p> <p>Send the message to the store. Do not wait for confirmation.</p> <p>Send the message on the endpoint's transports.</p> <p>There is no back pressure from the persistence service to the producer application. As a result the producer must ensure that it does not send faster than the persistence services can</p> |

| Publisher Mode | JSON Attribute | Description |
|------------------------------|----------------------------|---|
| | | <p>replicated data on the network (or write it to disk, if disk persistence is enabled).</p> <p>Data can be lost if the connection between the producer and the persistence services is broken for any reason while the producer is sending. This includes network failures and restarts or upgrades of the persistence services.</p> |
| Store - Confirm - Send | store_ confirm_ send | <p>Guarantees delivery in exchange for lower throughput and higher message latency.</p> <p>If the send call succeeds, then the store guarantees that the appropriate durable subscribers will eventually receive the message. Conversely, if the store operation fails, the send call also fails, and subscribers do not receive the message through direct transports, nor from the store. The exact behavior depends on the send policy. See .</p> <p>Send the message to the store.</p> <p>Wait for the cluster to confirm storage.</p> <p>Send the message on the endpoint's transports.</p> <p>There is back pressure from the persistence service to the producer application. The producer will automatically be throttled to a sustainable data rate.</p> |

Publisher Mode and Send Policy (Administrators)

Administrators set the publisher's mode when configuring a store. The publisher mode is either `store_confirm_send`, which offers maximum delivery assurance, or `store_send_noconfirm`, which offers minimal delivery assurance.

Applications may set the publisher's send policy when creating a publisher object. In addition, administrators may set a default send policy in the realm configuration. For the publisher send policy realm properties, see the following web API topics:

- [Realm Definition and Properties](#)
- [GET realm/properties](#)

- [POST realm/properties](#)

The send policy may be inline or non-inline. In general inline send offers better latency, while non-inline send offers better throughput.

When the publisher mode is `store_send_noconfirm`, the send call returns immediately, regardless of the send policy. The application is not notified as to whether the message was persisted or lost. In this case, the non-inline send policy may offer somewhat improved throughput in exchange for higher latency.

When the publisher mode is `store_confirm_send`, the FTL library will attempt to persist the message for the publisher's retry duration. The retry duration can be set by the application. Administrators may set a default value in the realm configuration. See the [Realm Properties Details Panel](#), section [Realm Properties Details Panel](#).

If the publisher mode is `store_confirm_send`, and the send policy is inline, then the send call will block until the message is persisted, or the retry duration has elapsed. If the FTL library could not ensure that the message was persisted, an exception is raised by the send call. This is a synchronous send call, where performance is tied directly to the latency to the persistence service. It may be necessary to use the batched send call (`tibPublisher_SendMessages`), or multiple publishers in parallel, to achieve acceptable performance.

If the publisher mode is `store_confirm_send`, and the send policy is non-inline, the send call may return immediately, provided that the maximum batch count has not yet been reached. The application may set the maximum batch count. See the [Development](#) guide, [Publisher Mode and Send Policy](#).

When the send policy is non-inline, the FTL library attempts to persist the message in the background for the publisher's retry duration. If the latency to the persistence service is large, a non-inline send may offer a substantial improvement in throughput compared to inline send. However, failures are signaled to the application asynchronously, and handling failures may require a more sophisticated application.

If using a non-inline send, it is critical that the application flushes or closes the publisher before exiting. If the application merely exits, the send call may not be complete, and the message may be lost.

Note: Calls to set keys in a map (`tibMap_Set` and `tibMap_SetMultiple`) are always synchronous. That is, they behave as if the publisher mode is `store_confirm_send` and the send policy is inline.

Persistence Limits

Administrators can set parameters to limit the memory growth of persistence services.

These parameters are *not* suitable for administering data validity or message security.

The time-to-live (TTL) parameters rely on timers. The timers reside in the persistence service leader. If the quorum elects a new leader, all the timers reset and restart.

This table presents an overview of the persistence limits and the ways you can use them to protect against common forms of memory growth. You can find more complete details for each parameter in the topic listed in the GUI Location column of this table.

| GUI Parameter | GUI Location | JSON Parameter | Usage |
|----------------------|-----------------------|----------------|--|
| Store Size Limit | Stores Grid | bytelimit | <p>This coarse-grained limit can protect the persistence service from exceeding available memory by limiting the maximum number of bytes in the store.</p> <p>The persistence server discards new messages that would cause the store to exceed this limit.</p> |
| Message Time-to-Live | Durable Details Panel | message_ttl | <p>This fine-grained limit protects the store from retaining messages that contain obsolete information.</p> <p>A message in a durable expires after this timeout (in seconds) elapses since it arrived in a durable. The persistence service deletes it from the durable.</p> |
| Durable Time-to-Live | Durable Details Panel | durable_ttl | <p>This medium-grained limit protects the store from retaining obsolete durables.</p> <p>A dynamic durable expires after this timeout (in seconds) elapses without any connected subscribers. The persistence service deletes the durable from the store, along with any messages it contains.</p> |

| GUI Parameter | GUI Location | JSON Parameter | Usage |
|------------------------|-----------------------|-----------------------|---|
| Message Limit | Durable Details Panel | message_limit | <p>This fine-grained limit protects the store from durable growth. The byte limit grows when subscribers are either absent or too slow to keep pace with publishers.</p> <p>Limits the maximum number of messages in a durable. When this limit is exceeded, the persistence service discards messages according to the durable's discard policy.</p> |
| Byte Limit | Durable Details Panel | bytelimit | <p>This fine-grained limit protects from durable growth. The byte limit can be configured for a static durable, a durable template, or a routing durable (for example, zone settings).</p> <p>The limit ensures that the total message size in a durable does not increase continuously past the durable byte limit. When this limit is exceeded, the persistence service discards messages according to the durable's discard policy and logs a warning.</p> |
| Dynamic Durables Limit | Store Details Panel | dynamic_durable_limit | <p>This medium-grained limit protects the store from subscribing clients that create too many dynamic durables (even when those durables do not receive any messages from publishers).</p> <p>Limits the number of durables in the store.</p> |

Memory Reserve for Persistence Services

Memory reserve is pre-allocated memory with which a persistence service could continue limited operations after exhausting available memory.

Caveat

For best results, avoid exhausting available memory. Techniques include pre-production resource allocation testing and appropriate configuration of the persistence store's byte limit parameter.

Memory reserve does not guarantee continued operation, merely a best effort attempt. For example, other processes could interfere with memory reserve operation.

Background Information

Stored messages occupy process memory within a persistence service. As a persistence service accumulates message data from publishers, it allocates memory. As clients consume stored messages, the persistence service reclaims their memory.

If the operating system denies a request to allocate memory, the persistence service must discontinue normal operation. However, the persistence service can continue partial service using its memory reserve.

Behavior

The following sequence describes the behavior of memory reserve:

1. Message data growth exhausts available memory on the persistence service host computer.
Such growth could indicate that subscribing clients are operating slowly, or not operating at all. Alternatively, the issue could result from high demand for memory by *another* process on the persistence service host computer.
2. The persistence service rejects all new inbound data messages from publishing clients.
3. Meanwhile, the persistence service attempts to use memory from the reserve for limited operation, for example, to continue replication, and to service data requests and acknowledgements from subscribing clients.
4. The demand for memory subsides.

Any of these factors can help reduce demand:

- Subscribing clients consume and acknowledge stored messages.
- Administrators selectively purge durables.

- Other processes on the persistence service host computer deallocate memory.
5. When the persistence service can reallocate its full memory reserve, it resumes accepting and storing new messages from publishing clients.

Size

Administrators can specify the memory reserve using the configuration file parameter `mem-reserve`.

As a rule of thumb, reserve an additional 10% of the maximum expected volume of stored message data. For best results, test persistence service behavior in a pre-production environment, and adjust the size of the memory reserve accordingly.

Starting a Persistence Service

Administrators have full responsibility for persistence service configuration and operation.

Procedure

1. Configure persistence clusters and services using the FTL server interface.
2. Configure persistence in the FTL server configuration file.
Assign each FTL server to manage a distinct named persistence service.
Configure other parameters as appropriate.
3. If a state file exists, check that it is current. If it is stale, remove it.
For more information, see [Saving and Loading Persistence State](#), and [Restarting a Persistence Cluster with Saved State](#).
4. Start the FTL servers on their host computers.
The FTL servers start and manage their persistence services.

Stopping or Restarting a Persistence Service

To permanently stop an individual persistence service, terminate the FTL server process that manages it. To stop and restart an individual persistence service, use the shutdown command.

Procedure

1. Send this web API request to the FTL server that provides the persistence service.

```
curl -X POST http://<ftl_svr_host>:<ftl_svr_port>/api/v1/server -d  
'{"cmd":"shutdown"}'
```

The FTL server automatically restarts the persistence service.

What to do next

To monitor or confirm the service's status, use the FTL server interface (see [Servers List](#)).

See Also: [Suspending a Persistence Cluster](#)

Persistence Monitoring and Management

The following topics present tools in the FTL server GUI to help monitor and manage persistence.

Persistence Clusters Status Table

The persistence clusters status table presents the state of persistence clusters and services. In the top bar of the FTL GUI, click **Clients** then **Clusters**.

Information and actions are listed in the following table. You can click a cluster row to expand a [Servers List](#), detailing the persistence services in the cluster.

Columns

| GUI Item | Description |
|----------------|---|
| Cluster Status | The health of the cluster based on the aggregated state values of its services. For service state values, see Persistence Service States . |
| Name | Name of the persistence cluster. |

| GUI Item | Description |
|----------|--|
| Servers | Status summary of the persistence services in the cluster. |
| Stores | List of persistence stores that the cluster implements. |

Note: The **Attention** tally in the GUI top bar does not include persistence services or eFTL services unless *at least one* service from the cluster is present. Nonetheless, even in a temporary situation where a cluster is configured but none of its services are present, the status table indicates that the cluster and its absent services are offline.

Commands

Icons at the right of each cluster row trigger the following commands:

- Suspend all services in the cluster, in preparation for saving the contents of its persistence services. For background information, see [Saving and Loading Persistence State](#), and its subtopics. You need to manually restart all persistence services to bring the cluster back online. *This command is used only when disk persistence is disabled.*
- Perform a manual disk compaction so all members of the persistence cluster start compaction. The compaction button only works if the persistence cluster is running. Compaction runs even if there is no space to be reclaimed. If disk persistence is not enabled, an error results. If there is not enough disk space, the persistence service will not start compaction, and will log an error message. For details, see [Compact Disk Persistence Files with Persistence Service Online](#).
- Save the data state of all the persistence services in the cluster. This command is visible only when it is available. For background information, see [Saving and Loading Persistence State](#), and its subtopics. *This command is used only when disk persistence is disabled.*
- Back up the disk(s). When disk persistence is enabled, this command backs up the files associated with disk persistence. See [Disk Persistence Backup and Restore](#).
- Force this cluster to form a quorum. This command icon is visible only when it is relevant. Before using this command, read [Before Forcing a Quorum](#).

Servers List

To see more details about a persistence service, go to the top bar of the FTL GUI, click **Clients** then **Clusters** and, finally, click the row in the **Persistence Clusters** status table to see the **Servers List**.

Columns

| GUI Item | Description |
|---------------|---|
| Server Status | Status of the persistence service process (as a string). |
| State | If the service is running, this string indicates its state or role with respect to the quorum and disaster recovery. For explanations of these state values, see Persistence Service States later in this topic. |
| Name | The name of the persistence service. This name originates in the realm definition, and is required in the FTL server configuration file. |
| ID | Client ID of the persistence service (as a client of the realm service). Administrators could use this client ID to retrieve client monitoring metrics about the persistence service using the FTL server web API. |
| Set | The name of the disaster recovery set to which this service belongs. |
| Host | The host name string of the persistence service's host computer (if available). |
| History | This field indicates the data state of a persistence service. The first numeric value is the service's quorum number. The second numeric value is the data state of the service. Higher values indicate more recent data. A history value of 0,0 indicates an empty service. |
| Consistency | This field indicates whether a service is up to date. |

| GUI Item | Description |
|------------------|--|
| | <ul style="list-style-type: none"> • Up to date: This service is up to date with respect to the other services in the cluster. • A history value pair indicates the quorum number and data state when this service was most recently up to date with respect to the other services in the cluster. <p>When the cluster does not contain enough up to date services to form a quorum, administrators may force a quorum, using the most up to date service (see Before Forcing a Quorum).</p> |
| Disk | When the service loads a state file from disk, or saves its state to disk, this JSON attribute reflects the state of the load or save operation. |
| Disk Persistence | <p>None</p> <p>Disk persistence disabled.</p> <p>sync</p> <p>The client returns from a send-message call after the message has been written to a majority of disks. This mode generally provides consistent data and robustness, but at the cost of increased latency and lower throughput. If the cluster restarts, no data is lost; performance is subject to disk performance.</p> <p>async</p> <p>The client may return from a send-message call before the message has been written to disk by majority of the FTL servers. This mode generally provides less latency and more throughput, but messages could be lost if a majority of servers restart shortly after the API call.</p> |
| Disk Swap | <p>Message swapping is enabled (<code>true</code>) or disabled (<code>false</code>).</p> <p>When message swapping is enabled (with or without disk persistence), message data is freed from persistence service process memory according to configured limits. However, the system may choose to use idle memory as a cache for the message data on disk. Tools used to report the memory usage of the persistence service process may include this cache memory, which is normal behavior. If the system experiences memory pressure, it reclaims the cache memory for use in other processes.</p> |

Commands

Icons at the right of each cluster row trigger commands:

- Change logging level of a persistence service.

For the meaning of log level values, see "Log Level Reference" in [TIBCO FTL Development](#).

- Save the persistence service's data state to a file.

This command icon is visible only when the command is available. For background information, see [Saving and Loading Persistence State](#) and its subtasks.

Persistence Service States

| Role | Description |
|---------------------|---|
| Leader | Functioning properly. This service is the leader. |
| Replica | Functioning properly. This service is a replica. |
| Offline | <p>This persistence service is configured, but not running.</p> <p>The persistence service or the FTL server that provides it might have exited, or a network segmentation prevents connection.</p> |
| Forming Quorum | <p>This process is running, but the service is not yet functioning as part of a valid quorum.</p> <p>If this state continues, ensure that all persistence services are running, and check network connectivity.</p> |
| Timed Out | The FTL server no longer detects this persistence service. |
| Manual Intervention | <p>Potential danger of data loss.</p> <p>This service is the provisional leader, but it cannot safely form a quorum even though other services are reachable.</p> <p>To proceed, see Before Forcing a Quorum.</p> |

| Role | Description |
|---------------|---|
| Needs Restart | <p>Warning.</p> <p>The administrator is testing a deployment with configuration changes that would require this persistence service to restart. Meanwhile, this service is still functioning properly using the old realm definition.</p> |
| Out of Sync | <p>Error.</p> <p>This persistence service is still functioning, but its realm definition is not the most recent deployment.</p> <p>(This situation can occur when a network partition separates persistence services. When the network reconnects the FTL server GUI could alert you to stop the persistence service so that the FTL server automatically restarts it with the newer deployment.)</p> |
| Exception | <p>Recoverable error.</p> <p>A non-fatal error occurred. After briefly registering this status, the persistence service will attempt to form a quorum.</p> <p>Check log files to determine the reason for the error.</p> |
| DR Leader | <p>Functioning properly. This service is the disaster recovery leader in the standby set.</p> <p>See Persistence Service Sets: Primary and Standby</p> |
| DR Replica | <p>Functioning properly. This service is a disaster recovery replica in the standby set.</p> <p>See Persistence Service Sets: Primary and Standby</p> |

Also see [Catalog of Persistence Metrics](#)

Persistence Stores Status Table

The Persistence Stores status table presents the state of the persistence stores. In the top bar of the FTL GUI, click **Clients** then **Stores**.

| | | | | | |
|---------------------|---------------------------------------|--------------|---------------|----------------|---------------|
| TIBCO® FTL 6.9.0 V1 | 08/04/2022 @ 11:34am Last Deployed | 0 Clients | 2 Services | - Attention | |
| Realm | | | Clients 0 | Stores 7 | Clusters 1 |

Clicking a store row expands a [Durables List](#), summarizing the durables in the store.



Persistence Stores columns follow.

Columns

| Column | Description |
|-------------------------|--|
| Status | State of the persistence store: Running or Offline. |
| Store Name | Name of the persistence store. |
| Durables | Number of durables in the store. |
| Cluster | The persistence cluster that implements the store. |
| Used/Max Bytes | The amount of storage consumed by messages in the store, and the maximum amount configured. |
| Consumed Byte Limit | Progress bar showing storage consumed, by bytes. |
| Used/Max Messages Count | The number of messages in the store, and the maximum number configured. |
| Consumed Message Limit | Progress bar showing storage consumed, by message count. Or, instead of a progress bar, No limit defined by User displays. |
| Total Messages / Bytes | The number of messages in the store, and the amount of storage they consume |


Commands

Icons at the right of each store row trigger commands:

- Purge durables ( broom icon). This action purges all messages from all the durables in the store. You cannot undo this action.
- Delete durables ( trash can icon). This action deletes all durables in the store. You cannot undo this action.

Durables List

To see more details about the durables in a persistence store, click the **Durable** value in the [Persistence Stores status table](#).

| Persistence Stores | | | |
|---|------------|----------|-------------------------|
| Status | Store Name | Durables | Cluster |
|  Running | st1 | 1 | RealmPersistenceCluster |


Columns

| GUI Item | Description |
|--------------|--|
| Durable Name | Name of the durable. |
| Durable Type | The type of durable. |
| Subscribers | <p>Number of subscribers with client processes on the durable.</p> <p>To see more details about subscribers, click this number. See Subscribers List.</p> |
| Browsers | <p>Number of client processes with browsers on the durable.</p> <p>Browsers only work with shared durables. This is not supported for standard durables.</p> <p>To see more details, about browsers, see Browsers List</p> |

| GUI Item | Description |
|--|--|
| Used/Max Bytes | Number of bytes used by messages in the durable, followed by the durable bytelimit ("unlimited" if there is no limit). |
| Used/Max Messages | Number of messages in the durable, followed by the durable message limit ("unlimited" if there is no limit). |
| Matcher | The content matcher for the durable. For example, Match ALL. |
| Key | The key field to use with the matcher. |
| Template Name (Appears only when relevant.) | The name of the template used to create a dynamic durable. |
| Unacked | The number of unacked messages in the durable. This can be different from the total number of messages when the retention time is set. For More information, see Retention Time |

Purge and Delete

Icons at the right of each durable row trigger commands:

- Purge the durable** (TIBCO FTL® - Enterprise Edition Administration



Rewind (clock icon): This action rewinds the durable. For more information, see [Rewinding a Durable](#).

This action is only available for durables with retention time set. For more information, see [Retention Time](#).

Subscribers List

To see more details about the subscribers in a persistence store, click the **Subscribers** value in the [Persistence Stores status table](#).

| Persistence Stores | | | | | | | | | |
|---|----------------------------------|-------------|---------------------|----------------|----------------------|--------------------------|--------------------------|---------------------|--|
| Status | Store Name | Durables | Cluster | Used/Max Bytes | Consumed Byte Limit | Used/Max Messages Count | Consumed Message Limit | | |
| Running | ftl.nonpersistent.store | 1 | ftl.default.cluster | 0 / 1gb | 0% | 0 / Unlimited | No limit defined by User | | |
| Durables List | | | | | | | | | |
| Durable Name | Durable Type | Subscribers | Browsers | Consumed Bytes | Current/Max Messages | Consumed | Matcher | Template Name | |
| _ephmFB76194E-0F55-4634-A780-C9CA6313B68D | Dynamic/Standard (With Prefetch) | 1 | 0 | 0 | 0 / Unlimited | No limit defined by User | Match All | ftl.pubsub.template | |

For shared durables and last-value durables the GUI shows a Subscribers List followed by the subscriber's ID.

Columns






Columns appear only as relevant to the durable's type.

| GUI Item | Description |
|--------------------------|---|
| Client ID | The client ID of the application that created the subscriber (assigned by the realm service). |
| Subscriber ID | Subscriber ID (assigned by the persistence service). |
| Last ACKed (in sec) | Approximate elapsed time in seconds since the subscriber acknowledged a message. |
| Last Dispatched (in sec) | Approximate elapsed time in seconds since the client dispatched a message for the subscriber (from an event queue). |

| GUI Item | Description |
|--------------------|---|
| Last Send (in sec) | Approximate elapsed time in seconds since the persistence service sent a batch of messages to the subscriber. |
| Number unACKed | Number of messages sent to this client subscriber that remain unacknowledged. |

Browsers List

To see more details about the browsers in a persistence store, click the **Browsers** value in the [Persistence Stores status table](#).

| Persistence Stores | | | | | | | | | |
|---|----------------------------------|-------------|---|----------------|----------------------|--------------------------|--------------------------|---|---|
| Status | Store Name | Durables | Cluster | Used/Max Bytes | Consumed Byte Limit | Used/Max Messages Count | Consumed Message Limit | | |
|  Running | ftl.nonpersistent.store | 1 | ftl.default.cluster | 0 / 1gb | 0% | 0 / Unlimited | No limit defined by User |  |  |
| Durables List | | | | | | | | | |
| Durable Name | Durable Type | Subscribers | Browsers | Consumed Bytes | Current/Max Messages | Consumed | Matcher | Template Name | |
| _ephmFB76194E-0F55-4634-A780-C9CA6313B68D | Dynamic/Standard (With Prefetch) | 0 | 1 | 0 | 0 / Unlimited | No limit defined by User | Match All | ftl.pubsub.template |   |

Browsers are only supported with shared durables. the GUI shows a Browsers List followed by the browser's ID.

Columns

Columns appear only as relevant to the durable's type.

| GUI Item | Description |
|---------------------|--|
| Client ID | The client ID of the application that created the browser (assigned by the realm service). |
| Subscriber ID | Subscriber ID (assigned by the persistence service). |
| Matcher | The matcher supplied by the application when the browser was created. |
| Last ACKed (in sec) | Approximate elapsed time in seconds since the subscriber acknowledged a message. |

| GUI Item | Description |
|--------------------|---|
| Last Send (in sec) | Approximate elapsed time in seconds since the persistence service sent a batch of messages to the subscriber. |

Before Forcing a Quorum

When a persistence cluster contains enough reachable persistence services, but not enough *non-empty* services to form a quorum, an administrator may force a quorum.

Check Safety

Before forcing a quorum, first verify that it is safe to do so.

Determine the process status of each unreachable service.

- If all of the reachable services are not running, and have no disk persistence state, it is safe to force a quorum.
- If any services are actually running but unreachable, or are not running but have disk persistence state, then it is *not* safe to force a quorum. Those services could contain more recent message data than the quorum candidates, and forcing a quorum would forfeit those messages.

Instead, restore reachability by repairing the network problem. Then the quorum can reform itself without administrator intervention.

i Note: Do not rely on service history state to determine whether it is safe to force a quorum. History state information that appears in the monitoring interface and in persistence service log output is not accurate enough for this purpose.

Example: Persistence Services, Forcing a Quorum

Consider a persistence cluster that defines three services. Only two services are reachable. The third service is unreachable.

Persistence Services: Forcing a Quorum

Persistence Clusters

| Cluster Status | Name | Servers | Stores | | | | | |
|--|------------------------------|----------------|-----------------------------------|-------|------|---------|-------------|----------------------|
| <div><div>Needs Intervention</div></div> | Cluster1 | 2 of 3 Running | <div><div></div><div></div></div> | | | | | |
| Servers List | | | | | | | | |
| Server Status | State | Name | ID | Set | Host | History | Consistency | Disk |
| <div><div></div><div>Timed Out</div></div> | Quorum Offline | s1 | 6053 | Set 1 | ben | | | |
| <div><div></div><div>Running</div></div> | Quorum Forming | s2 | 6063 | Set 1 | ben | 0, 0 | Up to date | Inactive <div></div> |
| <div><div></div><div>Running</div></div> | Manual Intervention Required | s3 | 6057 | Set 1 | ben | 8, 0 | Up to date | Inactive <div></div> |
| <div><div>Offline</div></div> | Realm Persistence Cluster | 0 of 1 Running | mapstore, pstore, sstore, | | | | | |

The two reachable services cannot form a quorum because s2 is empty (see [Quorum Conditions: General Rule](#)). If the administrator determines that s1 has actually exited, then it is safe to force a quorum. The administrator can explicitly force a quorum using the force quorum command icon in the cluster row of the persistence clusters status table.

If, on the other hand, s1 is still running, it could hold a more recent history state than s3 (the candidate with the most recent history state). Instead of forcing a quorum, the administrator should endeavor to repair the network problem that partitions s1 from the other candidates.

See Also: [Force Quorum](#)

Disk Persistence Backup and Restore

Disk persistence backup can be performed without interruption to clients while servers are running.

Disk persistence backup saves a copy of message and acknowledgment data for a persistence service at a single point in time.

By default, backup copies are written to the data directory of each persistence service. A custom backup directory can be configured by specifying the `savedir` parameter in the FTL server YAML configuration file. See [Persistence Service Configuration Parameters](#).

If the original disk persistence files are lost, the persistence state can be restored from a backup copy. This effectively sets the persistence service back in time to the point when the backup was taken.

See [FTL Administration Utility](#) for the commands to backup or restore persistence data.

Some example commands to backup/restore follow:

```
tibftladmin --ftlserver <url> --backup_persist --cluster <cluster_name>  
tibftladmin --restore_persist --name <persistence_service_name> --datadir  
<path> --backupdir <path>
```

Note the following differences between backups and saving the state of the persistence cluster.

- You use backups only when disk persistence is enabled.
- Unlike saving the state of the persistence cluster (which involves suspending the cluster), backups do not prevent clients from sending or receiving messages while the backup is in progress.

For details saving and loading the state of any persistence cluster, including in-memory clusters, see [Saving and Loading Persistence State](#).

Saving and Loading Persistence State

The persistence GUI includes a facility to save the content of a persistence service's stores to a file. When a new persistence service starts, it loads its initial state from that file. You can save and load a persistence state for in-memory or disk-based clusters.

You can use this facility for nightly backups, during upgrade migration, or to migrate a persistence cluster to a new physical location. Clients are interrupted for this operation.

See the following steps:

Procedure

1. [Suspending a Persistence Cluster](#)
2. [Saving the State of a Persistence Service](#)
3. [Restarting a Persistence Cluster with Saved State](#)
4. [Starting a New Persistence Service with Saved State](#)

Backup and online compaction cannot run at the same time. For details about using the persistence service for compaction, see [Compact Disk Persistence Files with Persistence Service Online](#).

For information on disk persistence backup, which does not interrupt clients, see [Disk Persistence Backup and Restore](#).


Suspending a Persistence Cluster


You can suspend *all* the services in a persistence cluster, however, you cannot suspend a service individually.

Suspending a cluster is a prerequisite subtask to saving the state of its services (the subtask that follows).

In addition, the suspend command can be used with the disaster recovery feature for planned failback. Once the suspend command is issued at the site that is currently active, the persistence services at that site will stop accepting message and acknowledgement data from clients and/or routes. Instead, they will finish replicating all stored data to the current standby site.

If authentication and authorization are enabled, you must be in the `ftl-admin` authorization group to do this task.

 **Note:** Services in a suspended cluster transition to a suspended state. Suspended services do not communicate with clients, nor with other services. Their only available operation is to save their state to a file. Disaster recovery persistence services can also be activated after the active persistence services were suspended.

 **Note:** You cannot suspend the internal cluster `ftl.default.cluster`.

Procedure

1. Open the persistence clusters status table.
2. Locate the cluster to suspend, and click its **Suspend** icon.
3. Confirm.
4. Wait until all the services in the cluster are suspended.

To verify suspension, check for the *pause icon* in the Service Status column of the services list, or check the persistence service log messages.

What to do next

[Saving the State of a Persistence Service](#)

Saving the State of a Persistence Service

Administrators can save the state of suspended persistence services.

Before you begin

The cluster and all the services in it must be suspended.

Procedure

1. Open the services list.

You may save all the services, or only a subset. The choice depends on available disk space, network speed and bandwidth. If you save only a subset, ensure that you save the quorum leader or the service with the latest history numbers.


2. Locate the service to save, and click the **Save** icon in the corresponding row. Confirm.


3. Wait for the service to finish saving, and verify success.

When the save operation is complete, the Disk column will indicate the value Saved. Alternatively, you can verify the save operation by checking the service log messages and the timestamp on the saved state file.

A save operation could fail if the disk is full, for example.

If a save operation fails, take corrective action and save again.

 **Note:** The save operation includes only replicated state. If replication is not enabled for a store, then the save operation excludes the content of that store from the state file.

 **Note:** The state file name is <service_name>.state.

Note: If the state file already exists, the save operation renames it to `<service_name>.state.backup` before writing a new state file.

4. Optional. Repeat steps 2 and 3 for other services.

Alternatively, you can save the state of *all* the services with one command. Click the **Save** icon in the cluster row.

What to do next

[Restarting a Persistence Cluster with Saved State](#)

Restarting a Persistence Cluster with Saved State

Administrators can restart a persistence cluster using previously saved state.

Before you begin

The cluster and all the services in it must be suspended.

You have saved the state of the suspended services

If you saved the state of all the persistence services in the cluster, the recovery is quicker because the services do not need as much time to synchronize.

Nonetheless, if you saved the state of only one persistence service, then the entire cluster can still recover by synchronizing.

Procedure

1. Determine the order in which you will restart the services.

If you saved the state of *only one* service, you must begin with that service.

If you saved the state of *all* the services, you may begin with any service.

After selecting the first server, you may order the others according to your convenience.

2. Ensure that each relevant state file is in the correct location.

The default location is:

```
<data_dir>/<service_name>.state
```

If you specified a non-default location using the persistence load parameter in the FTL configuration file, then ensure that the state file is in that location.

3. Restart the suspended services, one at a time, in the order you determined.

- a. Stop the service.

Send a stop request to the FTL server that provides the persistence service.

```
curl -X POST http://<ftl_svr_host>:<ftl_svr_port>/api/v1/server -d '{"cmd":"shutdown"}'
```

The FTL server shuts down.

- b. Restart the FTL server.
 - c. Wait for the service to report that it has status Running.
 - d. Repeat this step for the next suspended service.
4. Verify cluster status in the persistence clusters status table, and service state in the services list.

Persistence functionality resumes when the services can form a quorum. For details, see [Quorum Conditions: General Rule](#).

To prevent reloading out-of-date state files at a subsequent restart, the persistence service moves the state file automatically after successfully loading the state file.

Starting a New Persistence Service with Saved State

Similar to restarting a persistence cluster with a saved state, administrators can restart a persistence service in a new cluster using previously saved state from a different cluster.

Before you begin

The old cluster and all the services in it must be suspended.

Perform the following steps:

Procedure

1. Save the state of the persistence service running in the old cluster.
2. In Realm Configuration, reproduce all store definitions from the old cluster into the new cluster. The store definitions must match.
3. Start a persistence service running in the new cluster and configure it to load the state saved in step 1.

Clock Synchronization, Affiliated FTL Servers, and Persistence

Persistence services depend on clock synchronization among their host computers.

Ensure clock synchronization among affiliated persistence service hosts to within a few milliseconds.

When persistence services in a cluster communicate with one another, they rely on timestamps that they receive from their FTL servers. If a persistence service moves to a different host computer, and its clock is not synchronized with the previous host, then the persistence services could have difficulty forming a quorum.

For example, this difficulty could occur after switching operations to a disaster recovery site.

If you encounter this difficulty, consider forcing a quorum; see [Before Forcing a Quorum](#).

Message Tracing

For debugging and development purposes, the persistence service can trace individual messages as they are sent by a publisher or delivered to a subscriber. Since this impacts the performance of the persistence service, message tracing is not generally recommended for production environments.

To enable message tracing, set the log level of all persistence services in the cluster to “msg:debug”. Then, for each client application that needs to trace messages, set the client’s loglevel to “msg:debug”. Both the persistence service leader and the client trace the messages.

i Note: The persistence service can only trace messages for clients whose log level is “msg:debug”.

The persistence service log level is usually set in the FTL server yaml configuration file. For details, see [Persistence Service Configuration Parameters](#). The persistence service log level may also be set through the REST API while the persistence service is running.

For details, see [POST persistence/clusters/<clus_name>](#)

The client log level is usually set through the client API call, `tib_SetLogLevel` (or language equivalent). The client’s log level may also be set via the REST API while the client is running. For details, see [POST clients/<ID>](#) and [POST clients](#)

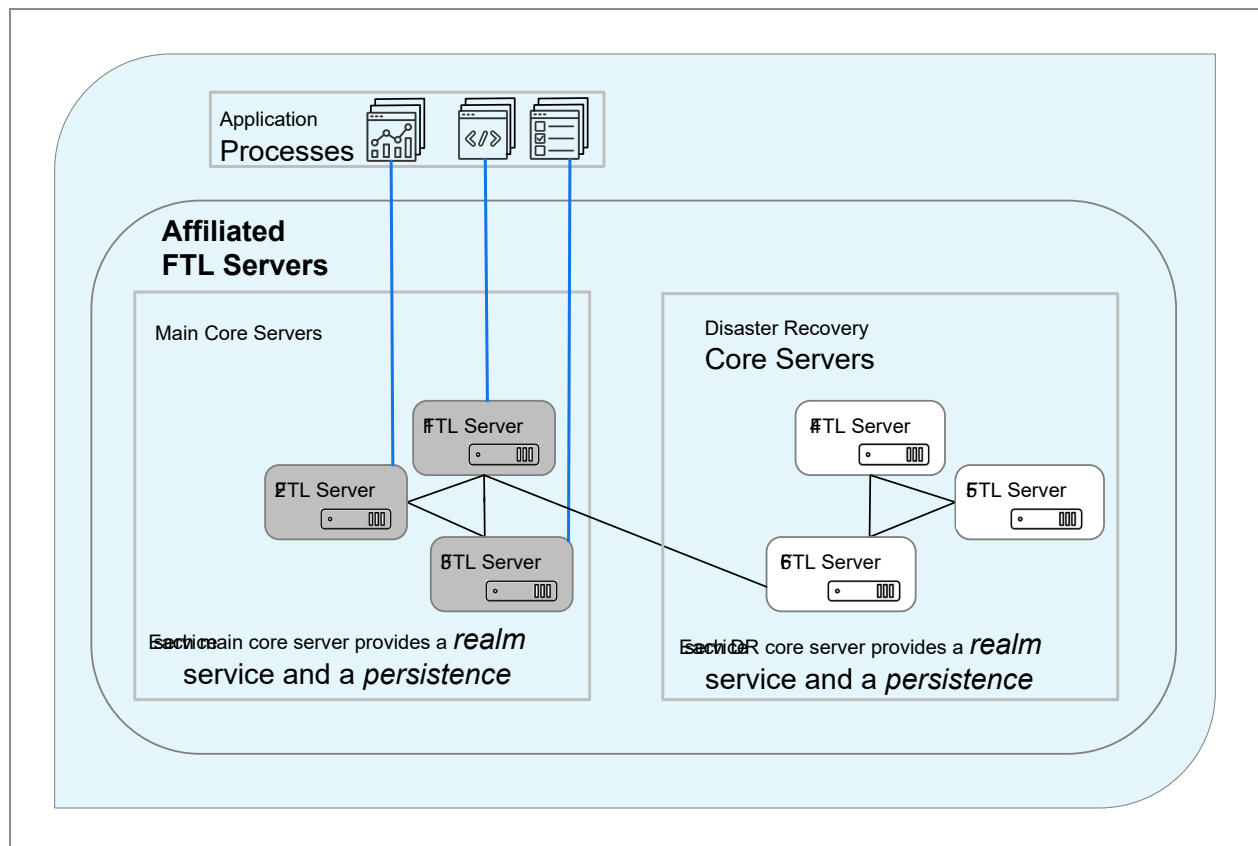
Disaster Recovery

You can use the disaster recovery feature to resume FTL communications after the main operations site becomes disabled. Application systems can continue after the interruption using replicated persistence data at a remote disaster recovery site.

The `TIBCO_HOME/ftl/<n.n>/samples/yaml` directory contains `dr` and `dr-secure` samples.

Prepared for Disaster

Figure 51: Servers Providing Realm and Persistence Services Configured for Recovery



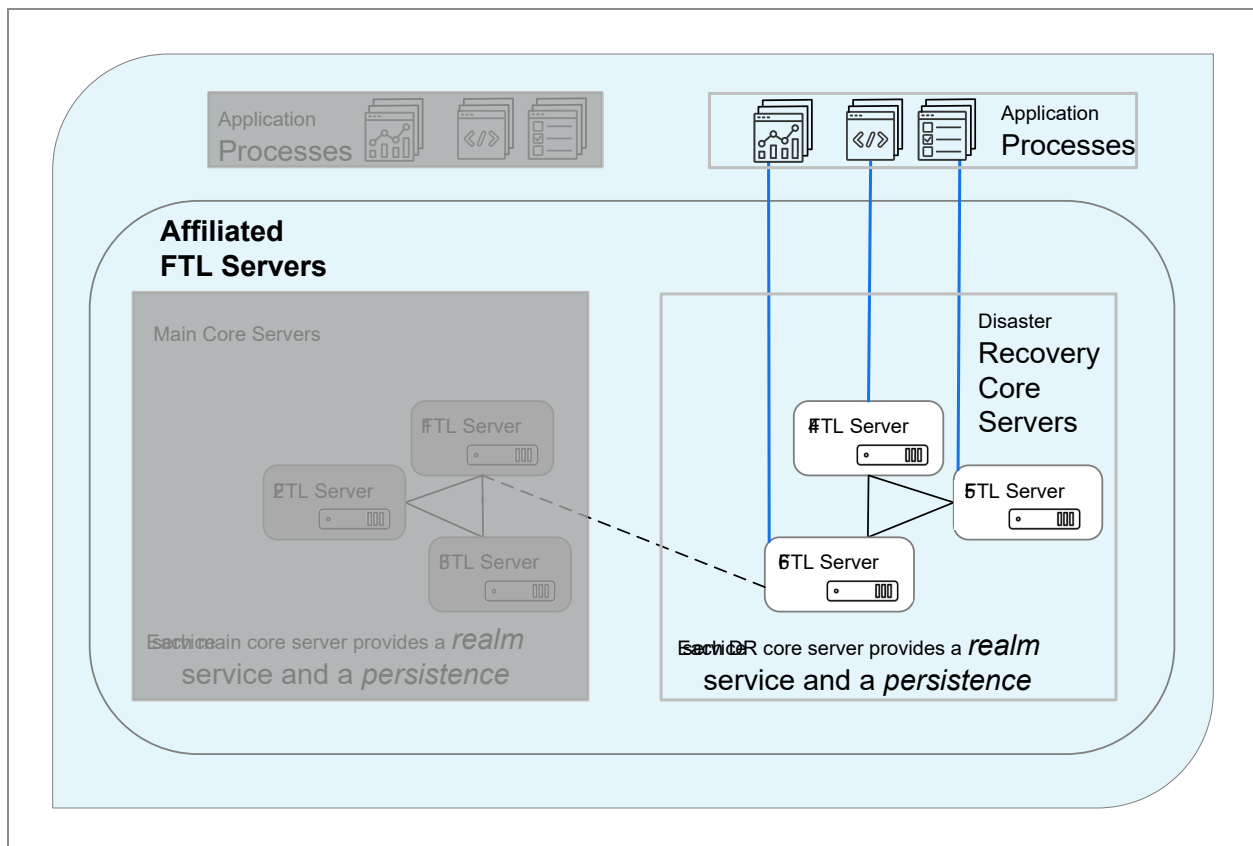
The diagram above illustrates a set of FTL servers configured to prepare for recovery from a potential disaster, along with the services they provide and the application processes that they serve. Servers at the main site are on the left (gray). Components at the disaster recovery site are on the right (white).

At each site, each of three core main core servers explicitly provides a *realm* service and a *persistence* service. An application can connect to any local core server (blue lines).

Realm services and persistence services synchronize to the state of the local cluster leader. The leader communicates via WAN link, to synchronize the disaster recovery site with the latest realm definition and persistence data.

After Recovery

Figure 52: After Recovery



In the diagram above, a disaster has disabled the main site.

Administrators have manually cut over to the disaster recovery site:

1. Administrators promote the disaster recovery FTL servers to primary FTL servers via the REST API.
2. In the realm configuration, Administrators have reconfigured the standby set of persistence services to be the primary set. Disaster recovery replication has been

disabled for the persistence cluster.

Applications connect to the primary servers. The disaster recovery site now operates as the new main site.

Administrators can begin to set up another disaster recovery site.

Scope of the Disaster Recovery Feature

The disaster recovery feature ensures realm availability and protects persistence data.

A comprehensive disaster recovery plan should protect employees and data resources, and enable an enterprise to resume business operations soon after a large-scale disaster.

The disaster recovery feature of TIBCO FTL enables disaster preparation and recovery for two kinds of resources:

- **Realm Availability** FTL servers at the disaster recovery site track the deployment state of the realm definition at the main site. If the main site is disabled, administrators can promote the disaster recovery servers to primary servers. When application clients restart at the disaster recovery site, they can connect to those FTL servers and resume messaging operations.
- **Persistence Data** Persistence services at the disaster recovery site replicate the data state of persistence services at the main site. After administrators reconfigure the persistence services at the disaster recovery site as the primary set, application clients can connect to them, and continue to send and receive persistent message streams.

Data replication to the disaster recovery site is asynchronous, so it does not add latency to message operations at the main site. However, asynchronous replication always lags slightly behind the most recent data, so some small amount of data could be lost in a disaster. For more information, see [Data Gaps and WAN Capacity](#).

Disaster Recovery Prerequisites

The disaster recovery feature depends on appropriate physical infrastructure, realm configuration, and running processes.

To use the disaster recovery feature of TIBCO FTL, you must satisfy these requirements:

- **Equivalent Hardware** Outfit the disaster recovery site with hardware equivalent to equipment at the main site.
- **WAN Capacity** Link the two sites with sufficient bandwidth to replicate persistence data in real time, even during periods of peak volume. It is good practice to link the sites with redundant WAN lines for fault tolerance.
- **Parallel Families of FTL Servers** Run an equivalent set of FTL server processes at each site.
- **Parallel Sets of Persistence Services** Configure an equivalent set of persistence services at each site. Clients connect only to services in the primary set at the main site. Standby services at the disaster recovery site replicate data, but clients do not connect to them during normal operation.

DNS Remapping

For some enterprises, remapping DNS addresses can simplify the task of directing application clients to FTL servers and other components after a disaster.

Alternatively, the scripts that restart application clients can supply the correct FTL server addresses.

Persistence Service Sets: Primary and Standby

To prepare for disaster recovery, each persistence cluster divides its services into two parallel sets: *primary* and *standby*.

Primary Set

Application processes interact with the leader of the primary set. The leader replicates persistence data to the other members of the primary set. The leader also replicates its data to the leader of the standby set.

Standby Set

The leader of the standby set receives data from the leader of the primary set, and replicates it to other members of the standby set. Application processes do not interact with members of the standby set.

Usage

During cut-over to the disaster recovery site, administrators redefine the primary sets, so the standby set becomes primary.



The same redefinition can facilitate migration to a new main site.

Status

To see the status of the persistence services, view each service set from the perspective of any *local* FTL server affiliate. For example, consider the following two screen captures.

Primary FTL Server's View of Persistence Services in the Primary Set

Persistence Clusters

| Cluster Status | Name | Servers | Stores | |
|---|-----------|----------------|--------|---|
|  Running | drCluster | 3 of 6 Running | st1, |  |

Servers List



















| Server Status | State | Name | ID | Set | Host | History | Consistency | Disk | |
|---|----------------|-------|------|-------|------|-----------|-------------|----------|---|
|  Running | Replica | psrv1 | 1027 | _setA | rach | 1, 675578 | Up to date | Inactive |  |
|  Running | Replica | psrv2 | 1029 | _setA | rach | 1, 675529 | Up to date | Inactive |  |
|  Running | Leader | psrv3 | 1031 | _setA | rach | 1, 675529 | Up to date | Inactive |  |
|  Offline | Quorum Offline | psrv4 | 0 | _setB | | | | | |
|  Offline | Quorum Offline | psrv5 | 0 | _setB | | | | | |
|  Offline | Quorum Offline | psrv6 | 0 | _setB | | | | | |

Figure 53: Disaster Recovery FTL Server's View of Persistence Services in the Standby Set

Persistence Clusters

| Cluster Status | Name | Servers | Stores | |
|---|-----------|----------------|--------|---|
|  Running | drCluster | 3 of 6 Running | st1, |  |

Servers List

| Server Status | State | Name | ID | Set | Host | History | Consistency | Disk |
|---|----------------|-------|------|-------|------|---------|-------------|---|
|  Offline | Quorum Offline | psrv1 | 0 | _setA | | | | |
|  Offline | Quorum Offline | psrv2 | 0 | _setA | | | | |
|  Offline | Quorum Offline | psrv3 | 0 | _setA | | | | |
|  Running | DR Follower | psrv4 | 1029 | _setB | rach | | |  |
|  Running | DR Follower | psrv5 | 1031 | _setB | rach | | |  |
|  Running | DR Leader | psrv6 | 1033 | _setB | rach | | |  |

In this example, the realm defines six persistence services, divided into two sets. The three services in `_setA` run at the main site. The three services in `_setB` run at the disaster recovery site.

(In an actual enterprise, each persistence service would run on a separate and distinct host computer for fault tolerance. However, in this simplified example, the servers all share a host computer.)

The two clusters of FTL servers at the primary and disaster recovery sites are separate, as are the two sets of persistence services that they manage. Each FTL server cluster displays the status of *only* those persistence services at its local site.

Conversely, an FTL server does not receive monitoring data from the persistence services at other sites. The FTL server GUI displays the status of the remote persistence services as *offline*, indicating that those remote persistence services are defined in the realm, but are not client services at the local site.

Data Gaps and WAN Capacity

A *data gap* consists of persistent message data that has not yet been replicated to persistence services at the disaster recovery site. This data is at risk, and could be lost in a disaster.

Replication of persistence data to a disaster recovery site requires sufficient WAN capacity to transfer data in a timely manner. You must ensure sufficient WAN capacity for expected peak data volume.

A small data gap will always exist because replication to the disaster recovery site is asynchronous. That is, publishers of a persistent message stream do not wait for replication to the disaster recovery site (even if they wait for confirmation from the main site).

The data gap can grow in two situations:

- Peak message activity exceeds the WAN capacity. The data gap grows until the message volume decreases. When message volume decreases, replication at the disaster recovery site can catch up to the data state at the main site, and the data gap shrinks.

✓ **Tip:** It is good practice for administrators to cross-check the active and standby persistence services during peak periods. If the history values are not identical, raise the capacity of the WAN links as needed.

- The WAN link malfunctions. Replication completely stops, unless you have redundant WAN lines for fault tolerance, and the data gap grows rapidly. Even with redundant lines, a data gap can grow if the backup capacity is less than the message volume. The data gap can also grow if failover introduces a significant delay.

Preparing FTL Servers for a Disaster Recovery Site

The first task in arranging disaster recovery for TIBCO FTL is to configure and run FTL servers that parallel the servers at the main site.

Given the complexity of this procedure, it is strongly recommended that you review the `samples/yaml/dr` or `samples/yaml/dr-secure` subdirectory of your FTL installation, and follow along according to the steps in `samples/yaml/readme.txt`.

Before You Begin

The physical infrastructure at the disaster recovery site must be operational.

The communications infrastructure connecting the main site to the disaster recovery site must be operational.

If authentication is enabled, define administrative users (with the `ftl-internal` role) for use by both sites. Authentication services must be available at both sites.

Procedure

1. Start the FTL servers at the primary site.

You may use the following for reference:

- `samples/yaml/dr/tibftlserver_primary.yaml`
- `samples/yaml/dr-secure/tibftlserver_primary.yaml`

In each YAML configuration file, define the `drto` parameter, which is a list of

addresses that the FTL server will use to connect to FTL servers at the disaster recovery site. For details see [FTL Server Configuration Parameters](#).

If authentication is required, configure the user and password parameters with credentials that the primary servers can use to authenticate themselves to the disaster recovery FTL servers. Ensure that this username is in the `ftl-internal` authorization group (according to the authentication service at the disaster recovery site). See [FTL Server Authorization Groups](#).

If TLS security is required, distribute the keystore file and trust file to all FTL servers. See [Securing FTL Servers](#).

If you have FTL servers and a persistence cluster already running, you may issue the `enable_dr` REST command to any primary FTL server, with the URLs of the disaster recovery site as the argument. This adds DR connectivity without requiring a restart. In the event of a restart, the `drto` URLs are persisted even if they do not appear in the YAML configuration file. The `drto` URLs may be added to the YAML configuration file at any later time. For details on `enable_dr`, see [POST cluster](#).

2. Start the FTL servers at the disaster recovery site.

You may use the following for reference:

- `samples/yaml/dr/tibftlserver_dr.yaml`
- `samples/yaml/dr-secure/tibftlserver_dr.yaml`

In each YAML configuration file, the `drfor` parameter must be defined, which is a list of addresses that the FTL server will use to connect to FTL servers at the primary site. For details see [FTL Server Configuration Parameters](#).

If authentication is required, configure the user and password parameters with credentials that the disaster recovery servers can use to authenticate themselves to the primary FTL servers. Ensure that this username is in the `ftl-internal` authorization group (according to the authentication service at the primary site). See [FTL Server Authorization Groups](#).

If TLS security is required, distribute the keystore file and trust file to all FTL servers. See [Securing FTL Servers](#).

3. Configure a persistence cluster and stores.

You may use the following for reference:

- `samples/yaml/dr/dr-cluster-sample.json`
- `samples/yaml/dr-secure/dr-cluster-sample.json`

At the persistence cluster level, [DR Enabled](#) must be checked.

At the persistence store level, the store should be replicated. Inspect each persistence store definition, and ensure that the [Replicated](#) checkbox is selected.

The set of persistence services should consist of two equivalent and parallel subsets:

- Services at the main site
- Services at the disaster recovery site

Select the main site subset as the primary set.

Configure the disaster recovery transport of each persistence service so that every persistence service at the main site can communicate with every persistence service at the disaster recovery site on this transport. That is, enable full mesh connectivity within each persistence cluster. (Nonetheless, only one pair of services at a time connect across the WAN link.)

In addition to configuring the disaster recovery transports in the realm, it is good practice to verify the configuration of routers, firewalls, and other network infrastructure that connects the two sites.

Deploy the realm definition.

4. Verify replication of data to the disaster recovery site.

The administrative GUI can be used to verify that the same number of messages are stored at the primary and disaster recovery sites.

Recovering after Disaster

Add these steps to your enterprise's comprehensive plan for switching business operations to the disaster recovery site. When disaster disables the main site, administrators complete these steps as part of the comprehensive plan.

Procedure

1. Optional. Remap DNS addresses.

If your disaster recover plan includes remapping the DNS addresses of FTL servers, then complete that remapping first.

2. Issue the `activate_dr` command at the standby site. From this point on, the `drfor` parameter in the standby configuration file will be ignored, even if the FTL servers

are restarted. Optionally remove the `drfor` parameter from the configuration file. For details on `activate_dr`, see [POST cluster](#). For details on `drfor`, see [FTL Server Configuration Parameters](#).

3. Modify the primary set of each persistence cluster.

In the persistence clusters grid of the FTL server GUI, change the primary set of each participating cluster so that the persistence services at the disaster recovery site become the primary set. Disable DR replication by clearing `DR Enabled` at the cluster level. See [Clusters Grid](#).

Deploy the modified realm definition.

You may use the following for reference:

- `samples/yaml/dr/dr-cluster-sample-dr-activate.json`
- `samples/yaml/dr-secure/dr-cluster-sample-dr-activate.json`

4. Ensure that the persistence services at the disaster recovery site form a quorum.

In the persistence clusters status table, verify that the cluster status is `Running`. In the services list sub-tables, verify that all the services in the cluster are synchronized.

If the cluster cannot form a quorum, clients cannot connect to its services. Consider forcing a quorum; see [Before Forcing a Quorum](#).

5. Direct application clients to FTL servers at the disaster recovery site.

Choose *only one* of these two alternatives:

- If you remapped the DNS addresses of FTL servers:
 - Verify that the new DNS information has propagated.
 - Verify that all clients automatically connect or reconnect to FTL servers at the disaster recovery site, and are operating correctly.
- Otherwise, when you restart all application clients, explicitly supply the locations of the new FTL servers where the clients can access realm services and persistence services.

The disaster recovery site is now the new active site of FTL operations.

Fail Back to the Original Site

After recovering from a disaster, the disaster recovery feature can be used to fail back to the primary site once it is available again. If returning to the original primary site, there

must be nothing left at the original site, including any lingering FTL processes or any data directories used by FTL. Clean up all processes and files as needed.

You can also use the disaster recovery feature to migrate FTL operations to a different site, even though no disaster has occurred.

Procedure

1. Start the FTL servers at the original primary site (the site to which FTL operations will be migrated).

You may use the following for reference:

- `samples/yaml/dr/tibftlserver_primary_failback.yaml`
- `samples/yaml/dr-secure/tibftlserver_primary_failback.yaml`

These FTL servers will act as disaster recovery servers for the FTL servers running at the disaster recovery site (which are acting as primary servers).

In each YAML configuration file the `drfor` parameter must be defined, which is a list of addresses that FTL server will use to connect to FTL servers at the primary site. For details see [FTL Server Configuration Parameters](#).

If authentication is required, configure the user and password parameters with credentials that the primary servers can use to authenticate themselves to the disaster recovery FTL servers. Ensure that this username is in the `ftl-internal` authorization group (according to the authentication service at the disaster recovery site). See [FTL Server Authorization Groups](#).

If TLS security is required, distribute the keystore file and trust file to all FTL servers. See [Securing FTL Servers](#).

2. Enable DR connectivity at the disaster recovery site.

This will allow the FTL servers at the disaster recovery site to connect to the FTL servers at the primary site. No restart is required.

Issue the `enable_dr` REST command at the disaster recovery site, using the URLs of the primary FTL servers as the argument. From this point on these URLs are persisted even if FTL servers at the disaster recovery site are restarted. Optionally, at some later time, add the `drto` URLs to the configuration file at the disaster recovery site. For details on `enable_dr`, see [POST cluster](#). For details on `drto`, see [FTL Server Configuration Parameters](#).

3. Update the realm configuration to re-enable DR replication.

You may use the following for reference:

- `samples/yaml/dr/dr-cluster-sample-dr-failback.json`
- `samples/yaml/dr-secure/dr-cluster-sample-dr-failback.json`

At the persistence cluster level, [DR Enabled](#) must be checked.

Deploy the updated realm definition.

4. Stop all client applications at the disaster recovery site.

All publisher and subscriber activity must be stopped so that no data is lost when the primary site is re-activated.

5. Verify replication of data to the primary site.

The administrative GUI can be used to verify that the same number of messages are stored at the primary and disaster recovery sites.

6. Stop all FTL servers at the disaster recovery site.

At this point the disaster recovery site has been disabled.

7. Activate the FTL servers at the primary site.

Issue the `activate_dr` REST command at the primary site. From this point on `drfor` in the primary site configuration file will be ignored, even if the FTL servers at the primary site restart.

Optionally, at some later time, remove `drfor` from the primary site configuration file. For details on `activate_dr`, see [POST cluster](#). For details on `drfor`, see [FTL Server Configuration Parameters](#).

8. Update the realm configuration to activate the persistence services at the primary site.

You may use the following for reference:

- `samples/yaml/dr/dr-cluster-sample-primary-activate.json`
- `samples/yaml/dr-secure/dr-cluster-sample-primary-activate.json`

In the persistence clusters grid of the FTL server GUI, change the primary set of each participating cluster so that the persistence services at the primary site become the primary set. Disable DR replication by clearing [DR Enabled](#) at the cluster level.

Deploy the modified realm definition.

9. Ensure that the persistence services at the primary site form a quorum and then direct application clients to the primary site.

At this point, all operations should be migrated to the primary site. The remainder of this procedure deals with configuring the disaster recovery site to function again as a backup.

10. Clear all data at the disaster recovery site.

When new disaster recovery FTL servers at the disaster recovery site are started, there must be no leftover FTL processes or FTL data directories.

11. Start FTL servers at the disaster recovery site.

You may use the following for reference:

- `samples/yaml/dr/tibftlserver_dr.yaml`
- `samples/yaml/dr-secure/tibftlserver_dr.yaml`

In each YAML configuration file, the `drfor` parameter must be defined, which is a list of addresses that the FTL server will use to connect to FTL servers at the primary site. For details see [FTL Server Configuration Parameters](#).

If authentication is required, configure the user and password parameters with credentials that the disaster recovery servers can use to authenticate themselves to the primary FTL servers. Ensure that this username is in the `ftl-internal` authorization group (according to the authentication service at the primary site). See [FTL Server Authorization Groups](#).

If TLS security is required, distribute the keystore file and trust file to all FTL servers. See [Securing FTL Servers](#).

12. Enable DR connectivity at the primary site.

This will allow the FTL servers at the primary site to connect to the FTL servers at the disaster recovery site. No restart is required.

Issue the `enable_dr` REST command at the primary site, using the URLs of the disaster recovery FTL servers as the argument. From this point on, these URLs are persisted, even if FTL servers at the primary site are restarted. Optionally, at some later time, add the `drto` URLs to the configuration file at the primary site. For details on `enable_dr`, see [POST cluster](#). For details on `drto`, see [FTL Server Configuration Parameters](#).

13. Update the realm configuration to re-enable DR replication.

You may use the following for reference:

- `samples/yaml/dr/dr-cluster-sample.json`

- `samples/yaml/dr-secure/dr-cluster-sample.json`

At the persistence cluster level, [DR Enabled](#) must be checked.

14. Verify replication of data to the disaster recovery site.

The administrative GUI can be used to verify that the same number of messages are stored at the primary and disaster recovery sites.

Disaster Recovery for Routes

This section describes procedures for disaster recovery failover and failback when routed persistence stores are in use. The information in this section assumes that you understand the previous sections concerning disaster recovery.

At a minimum, you must run FTL server processes at four different sites. The following describes how each site functions in the initial configuration:

- **Primary site:** This site controls FTL configuration and offers messaging through one or more persistence clusters.
- **Disaster recovery site:** This site connects to the primary site and serves as a standby for the configuration and messaging functions of the primary site. The persistence services at this site connect to persistence services at the primary site, serving as an asynchronous backup for the primary site persistence services.
- **Active satellite site:** This site connects to the primary site to retrieve the FTL configuration. This site may offer messaging through one or more persistence clusters. The persistence services at this site may connect to the primary site or other active satellite sites for routing purposes.
- **Standby satellite site:** This site connects to the primary site to retrieve the FTL configuration. The persistence services at this site connect to persistence services at the active satellite site, serving as an asynchronous backup for the active satellite of site persistence services.

You can run any number of satellite pairs (active and standby). There must be exactly one primary site and one disaster recovery site. All FTL realm configuration changes are made at the primary site.

For detailed FTL server YAML configuration files and FTL realm configuration files, see the sample files in:

`samples/yaml/satellite-dr.`

i Note: The disaster recovery feature is not supported for the default persistence cluster. The disaster recovery feature requires configuring your own persistence clusters.

Enabling Disaster Recovery for Routed Persistence Clusters

The sample YAML and JSON files at `samples/yaml/satellite-dr` demonstrate how to enable disaster recovery for routed persistence clusters. These samples create a full-mesh forwarding zone with two persistence clusters (c1 and c2) and one store (s1).

The FTL servers are distributed across four sites:

- Site 1: The primary FTL site, consisting of `ftlserver1-3`. This site controls FTL configuration. It also runs persistence services `pserver1-3`, which make up the active set of cluster c1.
- Site 2: The disaster recovery standby for the primary FTL site, consisting of `ftlserver4-6`. This site serves as a backup for the FTL configuration. It also runs persistence services `pserver4-6`, which make up the standby set of cluster c1.
- Site 3: An active satellite site, consisting of `ftlserver7-9`. This site offers messaging only. It runs persistence services `pserver7-9`, which make up the active set of cluster c2.
- Site 4: A standby satellite site for site 3, consisting of `ftlserver10-12`. This site runs persistence services `pserver10-12`, which make up the standby set of cluster c2.

Follow these general guidelines before you begin:

1. Ensure that all FTL servers and clients are at version 7.0 or later.
2. Configure two server sets for each persistence cluster (excluding the default cluster). One server set is active and the other is standby.
3. Each site (that is, each set of FTL core servers) hosts one persistence server set. So, each persistence cluster is split across two sites, with one site active and one site on standby.

If desired, a site can host active or standby server sets from more than one persistence cluster. Do not mix active and standby server sets at a given site.

4. If disaster recovery replication of messages is desired, enable disaster recovery replication for the persistence cluster. Otherwise the standby set is simply emptied until activated.
5. All persistence stores that require disaster recovery replication must also be replicated (ensure that the 'Replicated' checkbox is selected)
6. If using auto transports, ensure that each server set has "externally reachable addresses" configured. Servers at other sites use these addresses when connecting to the server set.

Often, these addresses are the same as the FTL core server addresses where the server set is running. Or, the externally reachable address might be the address of a load balancer.

7. Site 1 and site 2 are the only sites that manage FTL configuration. Site 1 needs "drfor" in its YAML file, and site 2 needs "drto".
8. All other sites are satellites that provide messaging only. There is no need for "drfor"/"drto" in the satellite YAML files, but "satelliteof" must be present.

Use the following sample files as a reference:

- samples/yaml/satellite-dr/tibftlserver_primary.yaml
- samples/yaml/satellite-dr/tibftlserver_dr.yaml
- samples/yaml/satellite-dr/tibftlserver_sat_primary.yaml
- samples/yaml/satellite-dr/tibftlserver_sat_dr.yaml
- samples/yaml/satellite-dr/satellite-dr-sample.json

The procedure that follows demonstrates activating site 2, failing back to site 1, activating site 4, and failing back to site 3.

Failover to the Disaster Recovery Site

If the servers at site 1 are no longer viable, activate site 2. Once site 2 is activated, the FTL configuration can be managed at site 2. The messaging functions of site 1 are been available at site 2.

1. Use the REST API to activate the FTL configuration at site 2 (command "activate_dr"). Optionally remove "dr_for" from the site 2 YAML files.
2. Clients and satellites need to reconnect to site 2. Remap the DNS or restart them.

Reference files:

- samples/yaml/satellite-dr/tibftlserver_sat_primary2.yaml
 - samples/yaml/satellite-dr/tibftlserver_sat_dr2.yaml
3. Update the realm configuration to activate messaging at site 2. This requires two changes to the persistence cluster: disable disaster recovery replication, and make site 2's server set the active server set. (Disaster recovery replication can be re-enabled later for a planned failback.)

Reference files:

- samples/yaml/satellite-dr/satellite-dr-sample-dr-activate.json

Planned Failback to the Primary Site

At a high level, to failback to the primary site you simulate a failover to site 1. To ensure that no messages are lost, you may issue the "suspend" command to site 2 at the appropriate time (see the steps below). The "suspend" command causes the persistence services at site 2 to stop accepting messages from both clients and routes. This allows site 2 to finish replicating all data to site 1 before site 1 is activated. Then, once DNS is remapped, site 1 picks up where site 2 left off, accepting pending messages from clients and routes.

1. Clear all data directories at site 1.
2. Start FTL servers at site 1, this time with "dr_for" in the YAML file.

Reference files:

- samples/yaml/satellite-dr/tibftlserver_primary_failback.yaml
3. Use the REST API to make site 2 recognize site 1 as a disaster recovery standby for FTL configuration (command "enable_dr"). Optionally set "dr_to" in the site 2 YAML files.
 4. Update the realm configuration to re-enable disaster recovery replication for the affected persistence cluster. Verify disaster recovery replication of any pending

messages (in the user interface).

Reference files:

- samples/yaml/satellite-dr/satellite-dr-sample-dr-failback.json
5. When ready for a planned failback, suspend messaging at site 2. Use the REST API (command "suspend"). For details, see “POST cluster”
 6. Wait for the standby persistence services at site 1 to report their status as suspended (in the user interface).
 7. Shut down site 2.
 8. Use the REST API to activate the FTL configuration at site 1 (command "activate_dr"). Optionally remove "drfor" from the site 1 YAML files.
 9. Clients and satellites need to reconnect to site 1. Remap the DNS or restart them.

Reference Files:

- samples/yaml/satellite-dr/tibftlserver_sat_primary.yaml
 - samples/yaml/satellite-dr/tibftlserver_sat_dr.yaml
10. Update the realm configuration to activate messaging at site 1. This requires two changes to the persistence cluster: disable disaster recovery replication, and make site 1's server set the active server set. (Disaster recovery replication is re-enabled later once site 2 is brought back.)

Reference files:

- samples/yaml/satellite-dr/satellite-dr-sample-primary-activate.json
11. Clear all data directories at site 2.
 12. Start FTL servers at site 2. Make sure "drfor", and not "drto", is specified in the YAML file.
 13. Use the REST API to make site 1 recognize site 2 as a disaster recovery standby for FTL configuration (command "enable_dr"). Optionally set "drto" in the site 1 YAML files.
 14. Update the realm configuration to re-enable disaster recovery replication for the affected persistence cluster. Verify disaster recovery replication of any pending messages (in the user interface).

Reference files:

- `samples/yaml/satellite-dr/satellite-dr-sample.json`

15. At this point, the system is ready for another failover.

Failover to the Standby Satellite Site

If the servers at site 3 are no longer viable, activate site 4. Once site 4 is activated, the messaging functions of site 3 are available at site 4. FTL configuration must still be managed at site 1 (or site 2, if site 2 happens to be active at the time).

i Note: This procedure is similar to activating site 2, with the exception that the FTL configuration does not need to fail over, since it is managed by site 1, not site 3.

1. Clients and servers need to reconnect to site 4. Remap the DNS or restart them.
2. Update the realm configuration to activate messaging at site 4. This requires two changes to the persistence cluster: disable Disaster recovery replication, and make site 4's server set the active server set. (Disaster recovery replication can be re-enabled later for a planned failback.)

Reference files:

```
samples/yaml/satellite-dr/satellite-dr-sample-satdr-activate.json
```

Planned Fail-back to the Active Satellite Site

At a high level, to fail-back to the active satellite site you need to simulate a failover to site 3. To ensure that no messages are lost, you may issue the suspend command to site 4 at the appropriate time (see the steps below). The suspend command causes the persistence services at site 4 to stop accepting messages from both clients and routes. This allows site 4 to finish replicating all data to site 3 before site 3 is activated. Then, once DNS is remapped, site 3 picks up where site 4 left off, accepting pending messages from clients and routes.

FTL configuration must still be managed at site 1 (or site 2, if site 2 happens to be active at the time).

i Note: This procedure is similar to fail-back to site 1, with the exception that the FTL configuration does not need to fail over, since it is managed by site 1.

1. Clear all data directories at site 3.
2. Start FTL servers at site 3. No change to the YAML file is needed.

Reference files:

- samples/yaml/satellite-dr/tibftlserver_sat_primary.yaml

3. Update the realm configuration to re-enable DR replication for the affected persistence cluster. Verify DR replication of any pending messages (in the user interface).

Reference files:

- samples/yaml/satellite-dr/satellite-dr-sample-satdr-failback.json

4. When ready for a planned failback, suspend messaging at site 4. Use the REST API (command `suspend`). For details, see `POST cluster`.
5. Wait for the standby persistence services at site 3 to report their status as suspended (in the user interface).
6. Shut down site 4.
7. Clients and servers need to reconnect to site 3. Remap the DNS or restart them.
8. Update the realm configuration to activate messaging at site 3. This requires two changes to the persistence cluster: disable disaster recovery replication, and make site 3's server set the active server set. (Disaster recovery replication is re-enabled later once site 4 is brought back.)

Reference files:

- samples/yaml/satellite-dr/satellite-dr-sample-satpri-activate.json

9. Clear all data directories at site 4.
10. Start FTL servers at site 4. No change to the YAML file is needed.

Reference files:

- samples/yaml/satellite-dr/tibftlserver_sat_dr.yaml

11. Update the realm configuration to re-enable disaster recovery replication for the

affected persistence cluster. Verify disaster recovery replication of any pending messages (example in the user interface).

Reference files:

- `samples/yaml/satellite-dr/satellite-dr-sample.json`

12. At this point, the system is ready for another failover.

IPv4 and IPv6

TIBCO FTL software supports both IPv4 and IPv6.

TIBCO FTL components accept either IPv4 and IPv6 addressing in the following cases:

- FTL server GUI fields configuring these transport parameters:
 - RUDP transport: Host field
 - Static TCP transport: Host field
 - Dynamic TCP transport: Subnet Mask field
- Arguments to client API calls that specify client connections to FTL servers
- Values that specify ports, either in FTL server configuration files, or in FTL server administration utility command line arguments

All other cases accept only IPv4 addressing.

Notation

Specify IPv4 addresses in 4-part dot-decimal notation.

Specify IPv6 addresses in hexadecimal notation, with colon (:) separators, surrounded by square brackets. Square brackets *must* be present.

In situations where it is correct and meaningful to specify *star* (that is, asterisk, *) as a wildcard address, TIBCO FTL software interprets this wildcard as *both* an IPv4 wildcard *and* an IPv6 wildcard. (That is, it listens on all relevant interfaces in *both* protocols.)

You can specify the loopback address in IPv4 as either `localhost` or `127.0.0.1`, and in IPv6 as `[::1]` or an equivalent notation.

Multithreading with Direct Publishers and Subscribers

On hardware with multiple processor cores, multithreading can boost performance of applications that communicate using direct publishers and direct subscribers. You can use multithreading to implement a variety of communicating sub-applications within a single process, and to decrease context-switching latency as they communicate.

For best results, match the size of a multithreaded application program to the host computer hardware. Allot one processor core for each direct publisher and its reserve-fill-send loop, and one processor core for each direct subscriber object and its dispatch loop. Too few cores can increase latency.



Note: A suite of application processes that communicate over a direct shared memory bus may contain *only one* direct publisher object at a time. It is the shared responsibility of the application developers and the administrators to ensure this condition.

Docker Containerization for FTL

You can containerize TIBCO FTL servers and their applications, and run them on hosts that support the Docker environment.

Docker containers provide a convenient way to migrate FTL applications to the cloud.

Arranging an FTL server and its services alongside your application clients in a single image can simplify deployment.

Alternatively, you can use one Docker image for the FTL server and its services, and separate Docker images for your FTL applications.

Build the docker image using docker files shipped with the product and then load the docker images.

Transports and Docker

Application processes running in Docker containers can communicate *only* using dynamic TCP transports.

Specify dynamic TCP as the transport protocol when you configure application definitions in the FTL server.

Building a Docker Image

You can build your own Docker images for the FTL server to include the desired services. You can also build an image that includes all the sample client programs.

A script is available for building Docker images. For information and assistance on building Docker images, see file `samples/docker/README.txt`.

i Note: The included Dockerfile and script, when used in accordance with the instructions here, will download and install third-party components to create Docker images. We recommend that you review the script to identify the website (s) from which the components are downloaded to ensure that you understand which license terms apply to these components and what is required to comply with those terms, and to track their security status and determine if and when to update or replace them for security purposes.

Starting a Default FTL Server in a Docker Container

To start an FTL server in a Docker container, complete this task.

Before you begin

Docker must already be installed on the host computer, and an FTL server Docker image must already be built.

Procedure

1. Run the `build_images` script provided in `samples/docker`.
2. Start the FTL server container.

Publish the FTL server port using Docker's `-p` parameter. This example uses the default port, 8585.

This example command line omits an FTL server configuration file, so the server uses a *default* configuration. Instead of specifying the server's host and port in the configuration file, supply an *extended name* as the argument to the FTL server's `--name` parameter. The extended name string specifies the server name, host, and port.

```
docker run -p 8585:8585 ftl-tibftlserver:7.0.1
           --name <srvr_name>@<host>:8585
```

Three Default FTL Servers

To start three default FTL servers in Docker containers, use the command line variant in this example.

```
docker run -p 8585:8585
            ftl-tibftlserver:7.0.1
            -n <svr_name>@host1:8585
            --core.servers
            svr1@host1:8585|svr2@host2:8585|svr3@host3:8585
```

Run the appropriate command on each of the three hosts you specify in the `--core.servers` argument.

For reference, see [FTL Server Executable](#).

Starting a Cluster of FTL Servers in Docker Containers

To start FTL servers that require explicit configuration, complete this task.

Before you begin

Docker must already be installed on the host computer.

Procedure

1. Arrange an FTL server configuration file.

All the servers in the cluster can use the same configuration file.

```
globals:

  core.servers:
    ftl1: host1:8585
    ftl2: host2:8585
    ftl3: host3:8585
```



```
servers:
  ftl1:
    - realm: {}

  ftl2:
    - realm: {}

  ftl3:
    - realm: {}
```

The configuration file can include core servers, auxiliary servers, or both.

2. Run the `build_images` script provided in `samples/docker`.

| Option | Description |
|-----------------|--------------------------------------|
| No eFTL service | <code>ftl-tibftlserver:7.0.1</code> |
| eFTL service | <code>eftl-tibftlserver:7.0.1</code> |

3. Start the FTL server container.

To make the configuration file available within the container, bind the external file system with Docker's `-v` parameter.

Publish the FTL server port using Docker's `-p` parameter. This example uses the default port, 8585.

Supply the configuration file name as the argument to the FTL server's `-c` parameter.

Supply the server name as the argument to the FTL server's `-n` or `--name` parameter. The name refers to a server defined in the configuration file.

```
docker run -v /var/tmp:/tmp
  -p 8585:8585
  ftl-tibftlserver:7.0.1
    -c /tmp/ftl/ftlconfig.yml
    -n <server_name>
```

4. Repeat the preceding step for each FTL server in the configuration file.

In this example, start three servers:

- Start server ftl1 on host1.
- Start server ftl2 on host2.
- Start server ftl3 on host3.

Coordination Forms

Coordination forms help developers and administrators to agree upon application details and to document those details.

From the FTL [Product Guides](#) list, you can download the PDF forms listed below, rename the copies, and complete them online. Alternatively, you can print the downloaded forms and complete them on paper.

- Application Coordination Form
- Durable Coordination Form
- Endpoint Coordination Form
- Format Coordination Form

Upgrading or Migrating to a New Release

Read these instructions before upgrading from an earlier TIBCO FTL release.

The upgrade or migration tasks you must complete depend upon the release from which you are upgrading.

Upgrading from Release 6.x

If you are upgrading from a release earlier than TIBCO FTL Release 6.7.1, you must first upgrade to Release 6.10.1 and then follow instructions for upgrading from 6.7.1 or later to TIBCO FTL 7.0.0.

You can upgrade to TIBCO FTL Release 7.0.0 directly from TIBCO FTL release 6.7.1 or later.

This procedure uses a rolling upgrade, which is to upgrade one server at a time. This lets you upgrade a network without a total service interruption.

Upgrading from Release 6.7.1 or Later

This procedure uses a rolling upgrade, which is to upgrade one server at a time. This lets you upgrade a network without a total service interruption.

This procedure enables you to upgrade to TIBCO FTL Release 7.0.1 from TIBCO FTL release 6.7.1 or later.

Procedure

1. Determine which core servers are leaders, and plan the order in which you want to upgrade host computers to the new release of TIBCO FTL. A recommended sequence is as follows:
 - a. disaster recovery location, non-leader core servers
 - b. disaster recovery location, leader core server
 - c. disaster recovery location, auxiliary server

- d. primary location, non-leader core servers
 - e. primary location, leader core server
 - f. primary location, auxiliary server
 - g. satellite location, non-leader core servers
 - h. satellite location, leader core server
 - i. satellite location, auxiliary server
2. Check the persistence clusters status table and its services list sub-table to verify that all the persistence services in the cluster are a) running, b) part of the quorum, and c) up to date.
See [Persistence Clusters Status Table](#) , [FTL Servers Status Page](#), and [Servers List](#).
 3. Address the first or next host computer in your plan and stop its TIBCO FTL server process.
This stops all services under that TIBCO FTL server automatically.
 4. Uninstall the old TIBCO FTL installation package from that server's host computer. See instructions in *TIBCO FTL Installation*.
 5. Install the new release of the full TIBCO FTL product on that host computer. See instructions in *TIBCO FTL Installation*.
 6. Ensure that all server computers in the cluster have their clocks synchronized.
 7. Repeat steps 2-6 for the next TIBCO FTL server in your plan. Continue for each server until you have upgraded all the TIBCO FTL servers.
 8. Upgrade all application clients.

Migration With Disk Persistence

Migrating TIBCO FTL Servers to TIBCO FTL Release 7.0.0 from TIBCO FTL Release 6.7.1 or Later 6.x.x versions

TIBCO FTL Server 7.0.1 disk persistence database is different from TIBCO FTL server 6.x.x disk persistence database, hence when you upgrade to TIBCO FTL server 7.0.1, a downgrade to an older version is not possible without the TIBCO FTLServer data directory backup from the earlier version.

i Note: During upgrade to TIBCO FTL release 7.0, TIBCO FTL server must make a copy of all pending data into the new database. Therefore disk space requirements temporarily double during migration. For example, if the data directory for TIBCO FTL 6.x.x was 100 GB, ensure that at least 200 GB is provisioned during upgrade.

Follow the procedure in [Upgrading from Release 6.x](#)

Once the upgrade is complete, the old database can be moved or deleted. The old database files will have the suffix ".imported" appended to the file name.

Migrating TIBCO FTL Servers to TIBCO FTL Release 7.0.1 from TIBCO FTL Release 6.7.1 or Later 6.x.x Versions When Provisioned Disk Space Cannot be Expanded

Determine which TIBCO FTL servers are leaders, and plan the order in which you want to upgrade host computers to the new release of TIBCO FTL.

Suppose ftls1 is the leader of the persistence cluster, and ftls2 and ftls3 are followers. This is the recommended sequence

- a. Shutdown one of the servers which is the follower (ftls3).
- b. Delete the data directory of this server (ftls3) that was just shutdown.
- c. Start a TIBCO FTL 7.0.1 server and let it catch up from the other two servers from 6.x.
- d. Shutdown another one of the 6.x TIBCO FTL servers which is also a follower (ftls2).
- e. Delete the data directory of this server (ftls2) that was just shutdown.
- f. Start a TIBCO FTL 7.0.1 server and let it catch up from the other two servers.
- g. Once the two 7.0.1 TIBCO FTL servers are fully in sync, shutdown the 6.x leader (ftls1) TIBCO FTL server.
- h. Delete the data directory of this TIBCO FTL server (ftls1).
- i. Start a TIBCO FTL 7.0.1 server and let it catch up from the other two servers already from 7.0.1.

Migrating TIBCO FTL servers from one data center to another

When you are migrating from TIBCO FTL release 6.7.1 or later where disk persistence is enabled, for data center migration or upgrades, you can shut down the servers and copy the disk persistence related database files for data center migration use cases. You must shut down all servers before copying these files.

Enabling Disk Persistence for the First Time

When migrating from in-memory persistence to disk-based persistence, two rolling upgrades are required. First, upgrade to 6.7.0 (or later), enable disk persistence or disk swap, then restart the servers again.

Migrating to a Different Host

You can migrate an TIBCO FTL server installation to a replacement host computer. Complete the following procedure.

Procedure

1. Install the full TIBCO FTL product on the new host computer. See instructions in *TIBCO FTL Installation*.
2. On the new host computer, ensure that the TIBCO FTL server is not running.
3. On the old host computer, shut down the TIBCO FTL server.
4. Copy the data directory for the realm service and, if using disk persistence, the persistence service(s) run by the server.
5. Copy any security-related files that were distributed to the server as part of the `--init-security` or `--init-auth-only` procedure (i.e., the `ftl-tpport.p12` and `ftl-trust.pem` files).
6. Remap DNS so that any FTL clients or FTL servers that are currently running are able to reconnect to the new host computer.
7. Start the TIBCO FTL server on the new host computer.

Eliminating the TIBCO FTL Keystore (Authentication Only)

For TIBCO FTL 6.x configurations that used authentication, but did not use TLS, you had to generate an TIBCO FTL keystore by using `tibftlserver --init-auth-only`. Then, you had to distribute the TIBCO FTL keystore and trust files to the data directory of each TIBCO FTL server.

To continue running TIBCO FTL as you did for TIBCO FTL 6.x, do not take any action after upgrading.

When using version TIBCO FTL 7.x, the TIBCO FTL keystore is no longer necessary for authentication. Users that want to enable oauth2 authentication may optionally eliminate the TIBCO FTL keystore after upgrading to TIBCO FTL 7.x. For example, this will allow TIBCO FTL server to enforce oauth2 token expirations.

To eliminate the TIBCO FTL keystore, you must follow this procedure because TIBCO FTL servers that have the TIBCO FTL keystore cannot communicate with TIBCO FTL servers that do not have the TIBCO FTL keystore. This procedure requires a period of time where all TIBCO FTL servers are shut down.

Procedure

1. Upgrade all TIBCO FTL servers to TIBCO FTL 7.x. For more information, see [Upgrading from Release 6.x](#)
2. Upgrade all TIBCO FTL clients to TIBCO FTL 7.x. For more information, see [Upgrading from Release 6.x](#)
3. Save the state of all in-memory persistence clusters to preserve pending messages. For more information, see [Saving and Loading Persistence State](#). If all persistence clusters use disk persistence, no action is needed.
4. Shut down all TIBCO FTL servers, including TIBCO FTL servers at satellite or DR sites.
5. For each TIBCO FTL server, remove `ftl-tport.p12` and `ftl-trust.pem` from the data directory of the server.
6. Restart all TIBCO FTL servers. Clients reconnect automatically.

Eliminating FTL-Generated Certificates (Authentication and TLS)

For TIBCO FTL 6.x configurations that used both authentication and TLS, you had to generate TIBCO FTL certificates by using `tibftlserver --init-security`. Then, you had to distribute the TIBCO FTL keystore and trust files to the data directory of each TIBCO FTL server.

To continue running TIBCO FTL as you did for TIBCO FTL 6.x, do not take any action after upgrading.

When using version TIBCO FTL 7.x, FTL-generated certificates are no longer necessary for TLS. Users that want to control TLS certificates or enable oauth2 authentication may optionally eliminate FTL-generated certificates after upgrading to TIBCO FTL 7.x. For example, this will allow TIBCO FTL server to enforce oauth2 token expirations.

However, to eliminate FTL-generated certificates, and provide their own certificates, you must follow the procedure in this section because TIBCO FTL servers that use FTL-generated certificates cannot communicate with TIBCO FTL servers that do not use the FTL-generated certificates. This procedure requires a period of time where all TIBCO FTL servers are shut down. Also note that when using user-defined certificates, secure peer-to-peer transports are not permitted. Only secure server-based transports are permitted (for example, persistence service or group service transports).

Procedure

1. Ensure that no client applications are using secure peer-to-peer transports.
2. Upgrade all TIBCO FTL servers to TIBCO FTL 7.x. For more information, see [Upgrading from Release 6.x](#)
3. Upgrade all TIBCO FTL clients to TIBCO FTL 7.x. For more information, see [Upgrading from Release 6.x](#). When you restart the TIBCO FTL client at version 7.x, provide the trust certificates that correspond to the user-defined certificates that you plan to use later.
 - a. If the trust certificates are installed in the system trust store, install them before restarting the client.
 - b. If the trust certificates are passed to the client API as a PEM file, concatenate the trust certificates with the FTL-generated trust file (`ftl-trust.pem`). Pass the

resulting combined PEM file to the client API.

4. Save the state of all in-memory persistence clusters to preserve pending messages. For more information, see [Configuring Persistence](#). If all persistence clusters use disk persistence, no action is needed.
5. Shut down all TIBCO FTL servers, including TIBCO FTL servers at satellite or DR sites.
6. For each TIBCO FTL server, remove `ftl-tpport.p12` and `ftl-trust.pem` from the data directory of the server. Make the following changes to the TIBCO FTL server `yaml` configuration file.
 - a. Remove `tls.secure`.
 - b. Add the user-defined certificates (`tls.server.cert`, `tls.server.private.key`, `tls.server.private.key.password`). Ensure that each certificate is appropriate for the specific TIBCO FTL server's hostname.
 - c. Add the trust certificates corresponding to the user-defined certificates (`tls.client.trust.file`). Alternatively, install them in the system trust store. For more information, see the [Enabling TLS for FTL Server](#)
7. Restart all TIBCO FTL servers. Clients reconnect automatically by using the trust information provided earlier.

TIBCO FTL Processes as Windows Services

You can arrange for FTL servers to run as Windows services.

To arrange Windows services, use the prunsv tool, which is part of the Apache Procrun package. The TIBCO FTL installer includes this package on Windows platforms. For documentation, see <http://commons.apache.org/proper/commons-daemon/procrun.html>.

Prerequisite

Before arranging TIBCO FTL processes as Windows services, ensure that both Java and the TIBCO FTL Windows package are installed on a local disk of the host computer (not on a mapped network drive).

Installing the FTL Server as a Windows Service

One command installs the FTL server as Windows service.

Model your command line on this template:

```
prunsv.exe //IS/tibftlserver
--DisplayName="TIBCO FTL Server"
--Install=<TIBCO_HOME>\ftl\<version>\bin\prunsv.exe
--StartMode=exe
--StartImage=<TIBCO_HOME>\ftl\<version>\bin\tibftlserver.exe
--LibraryPath=<TIBCO_HOME>\ftl\<version>\bin
--StartParams=-n;<ftl_svr_name>;-c;<ftl_svr_config_file_path>
--StopMode=exe
--StopImage=<TIBCO_HOME>\ftl\<version>\bin\tibftladmin.exe
--StopParams=--ftlserver;http://<host>:<port>;-x
```

Notice these aspects of this command line template:

- --Install is the file path of the prunsv executable.
- --LibraryPath is the directory containing TIBCO FTL DLL files.
- --StartParams contains the command line parameters for the FTL server executable

(`tibftlserver.exe`). This template illustrates only the minimum required parameter set. Semicolon (;) is the separator character, as `prunsv` does not allow spaces.

- `--StopParams` contains the command line parameters for the FTL server administration utility (`tibftladmin`). This template illustrates the command to shut down the FTL server. The URL is the location corresponding to the server name, as specified in the FTL server configuration file. Semicolon (;) is the separator character, as `prunsv` does not allow spaces.

Uninstalling a Windows Service

One command uninstalls any Windows service that you installed using `prunsv`.

Model your command line on this template:

```
prunsv.exe //DS/<service_name>
```

TIBCO Documentation and Support Services

For information about this product, you can read the documentation, contact TIBCO Support, and join [TIBCO Community](#).

How to Access TIBCO Documentation

Documentation for TIBCO products is available on the [Product Documentation website](#), mainly in HTML and PDF formats.

The [Product Documentation website](#) is updated frequently and is more current than any other documentation included with the product.

Product-Specific Documentation

Documentation for TIBCO FTL® - Enterprise Edition is available on the [TIBCO FTL® - Enterprise Edition Product Documentation](#) page.

TIBCO eFTL™ Documentation Set

TIBCO eFTL software is documented separately. Administrators use the FTL server GUI to configure and monitor the eFTL service. For information about these GUI pages, see the documentation set for TIBCO eFTL software.

How to Contact Support for TIBCO Products

You can contact the Support team in the following ways:

- To access the Support Knowledge Base and getting personalized content about products you are interested in, visit our [product Support website](#).
- To create a Support case, you must have a valid maintenance or support contract with a Cloud Software Group entity. You also need a username and password to log in to the [product Support website](#). If you do not have a username, 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](#). For a free registration, go to [TIBCO Community](#).

Legal and Third-Party Notices

SOME CLOUD SOFTWARE GROUP, INC. (“CLOUD SG”) SOFTWARE AND CLOUD SERVICES EMBED, BUNDLE, OR OTHERWISE INCLUDE OTHER SOFTWARE, INCLUDING OTHER CLOUD SG SOFTWARE (COLLECTIVELY, “INCLUDED SOFTWARE”). USE OF INCLUDED SOFTWARE IS SOLELY TO ENABLE THE FUNCTIONALITY (OR PROVIDE LIMITED ADD-ON FUNCTIONALITY) OF THE LICENSED CLOUD SG SOFTWARE AND/OR CLOUD SERVICES. THE INCLUDED SOFTWARE IS NOT LICENSED TO BE USED OR ACCESSED BY ANY OTHER CLOUD SG SOFTWARE AND/OR CLOUD SERVICES OR FOR ANY OTHER PURPOSE.

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

This document 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 Cloud Software Group, Inc.

TIBCO, the TIBCO logo, the TIBCO O logo, FTL, eFTL, and Rendezvous are either registered trademarks or trademarks of Cloud Software Group, Inc. in the United States and/or 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. You acknowledge that all rights to these third party marks are the exclusive property of their respective owners. Please refer to Cloud SG’s Third Party Trademark Notices (<https://www.cloud.com/legal>) for more information.

This document includes fonts that are licensed under the SIL Open Font License, Version 1.1, which is available at: <https://scripts.sil.org/OFL>

Copyright (c) Paul D. Hunt, with Reserved Font Name Source Sans Pro and Source Code Pro.

Cloud SG 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 a specific version of Cloud SG software 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. CLOUD SG MAY MAKE IMPROVEMENTS AND/OR CHANGES IN THE PRODUCT(S), THE PROGRAM(S), AND/OR THE SERVICES DESCRIBED IN THIS DOCUMENT AT ANY TIME WITHOUT NOTICE.

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 "README" FILES.

This and other products of Cloud SG may be covered by registered patents. For details, please refer to the Virtual Patent Marking document located at <https://www.cloud.com/legal>.

Copyright © 2009-2024. Cloud Software Group, Inc. All Rights Reserved.