

# **TIBCO® Messaging - Apache Kafka Distribution User's Guide**

*Software Release 1.0  
May 2018*

## Important Information

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

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

ANY SOFTWARE ITEM IDENTIFIED AS THIRD PARTY LIBRARY IS AVAILABLE UNDER SEPARATE SOFTWARE LICENSE TERMS AND IS NOT PART OF A TIBCO PRODUCT. AS SUCH, THESE SOFTWARE ITEMS ARE NOT COVERED BY THE TERMS OF YOUR AGREEMENT WITH TIBCO, INCLUDING ANY TERMS CONCERNING SUPPORT, MAINTENANCE, WARRANTIES, AND INDEMNITIES. DOWNLOAD AND USE THESE ITEMS IS SOLELY AT YOUR OWN DISCRETION AND SUBJECT TO THE LICENSE TERMS APPLICABLE TO THEM. BY PROCEEDING TO DOWNLOAD, INSTALL OR USE ANY OF THESE ITEMS, YOU ACKNOWLEDGE THE FOREGOING DISTINCTIONS BETWEEN THESE ITEMS AND TIBCO PRODUCTS.

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

TIBCO, the TIBCO logo, Two-Second Advantage, TIB, Information Bus, TIBCO Messaging, FTL, and eFTL are either registered trademarks or trademarks of TIBCO Software Inc. in the United States and/or other countries.

Enterprise Java Beans (EJB), Java Platform Enterprise Edition (Java EE), Java 2 Platform Enterprise Edition (J2EE), and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle Corporation in the U.S. and other countries.

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

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

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

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

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

Copyright © 2018 TIBCO Software Inc. ALL RIGHTS RESERVED.

TIBCO Software Inc. Confidential Information

# Contents

---

- About this Product ..... 5
- TIBCO Documentation and Support Services .....6
- Architecture ..... 7
- Use Cases ..... 9
- FTL Bridge ..... 10
  - Configuring Kafka Connect ..... 10
  - Configuring the Connectors .....11
  - Configuring Realm Definitions for the Connectors ..... 12
  - Running the Connectors ..... 12
    - Bridge Connector Troubleshooting .....14
- Schema Repository ..... 15
  - Schema Repository Command Line Reference (tibscemad) ..... 15
  - Password Security ..... 17
  - Schema Repository Security .....18
- Java Programmer's Checklist ..... 20
- FTL-Avro Translation Reference .....21
- FTL-JSON Translation Reference .....22

# About this Product

---

TIBCO® is proud to announce TIBCO® Messaging - Apache Kafka Distribution.

This release is the latest in a long history of TIBCO products that leverage the power of Information Bus® technology to enable truly event-driven IT environments. To find out more about how TIBCO® Messaging software and other TIBCO products are powered by TIB® technology, please visit us at [www.tibco.com](http://www.tibco.com).

TIBCO Messaging - Apache Kafka Distribution is part of TIBCO® Messaging software.

## Product Editions

TIBCO Messaging software is available in a community edition and an enterprise edition.

TIBCO Messaging - Community Edition is ideal for getting started with TIBCO Messaging, for implementing application projects (including proof of concept efforts), for testing, and for deploying applications in a production environment. Although the community license limits the number of production processes, you can easily upgrade to the enterprise edition as your use of TIBCO Messaging expands.

The community edition is available free of charge, with the following limitations and exclusions:

- Users may run up to 100 application instances or 1000 web/mobile instances in a production environment.
- Users do not have access to TIBCO Support, but you can use TIBCO Community as a resource (<https://community.tibco.com>).

TIBCO Messaging - Enterprise Edition is ideal for all application development projects, and for deploying and managing applications in an enterprise production environment. It includes all features presented in this documentation set, as well as access to TIBCO Support.

The enterprise edition of TIBCO ActiveSpaces® use the enterprise edition of TIBCO Messaging and include a license for it. The community editions of those related products are compatible with both the enterprise and community editions of TIBCO Messaging.

# TIBCO Documentation and Support Services

---

## How to Access TIBCO Documentation

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

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

## Product-Specific Documentation

Documentation for TIBCO products is not bundled with the software. Instead, it is available on the TIBCO Documentation site. To access the documentation web page for this product from a local software installation, open the following file:

`TIBCO_HOME/akd/bridge/1.0/product_info/TIB_msg-akd_1.0.0_docinfo.html`

*TIBCO\_HOME* is the top-level directory in which TIBCO products are installed.

- On Windows platforms, the default *TIBCO\_HOME* is `C:\tibco`.
- On UNIX platforms, the default *TIBCO\_HOME* is `/opt/tibco`.

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

## How to Contact TIBCO Support

You can contact TIBCO Support in the following ways:

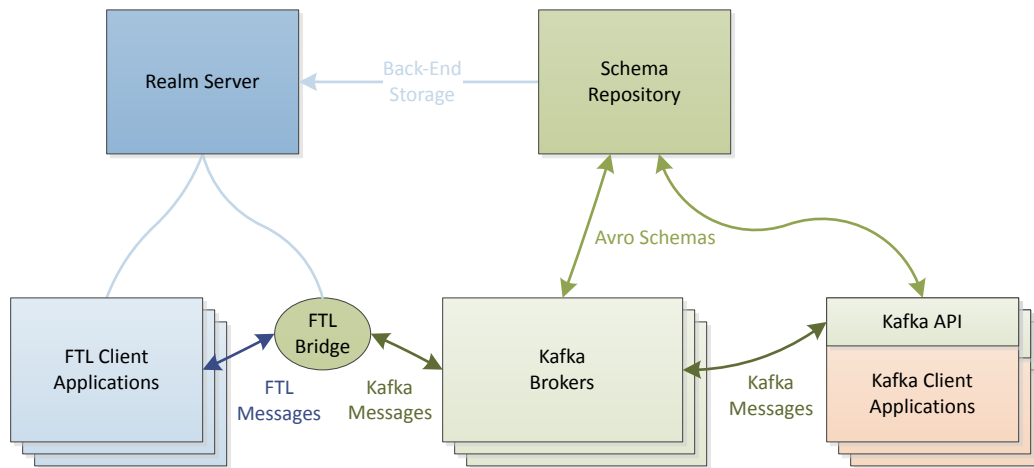
- For an overview of TIBCO Support, visit <http://www.tibco.com/services/support>.
- For accessing the Support Knowledge Base and getting personalized content about products you are interested in, visit the TIBCO Support portal at <https://support.tibco.com>.
- For creating a Support case, you must have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

## How to Join TIBCO Community

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

# Architecture

This diagram illustrates the interactions among the parts of TIBCO® Messaging as messages flow between TIBCO FTL® clients and Kafka clients.



## Message Flow

Messages flow among the parts in the lower level of the diagram. (Components in the upper level of the diagram play supporting roles outside the message path.)

Publishers in FTL applications (lower left, light blue) send FTL messages. The source connector in the FTL bridge can subscribe to those messages. When the bridge (green ellipse) receives FTL messages, it translates them to equivalent Kafka messages, and delivers them to the Kafka broker (light green). Subscribing Kafka applications (orange) can consume the messages from topics in the Kafka broker.

In the opposite direction, publishers in Kafka applications (orange) send Kafka messages to the Kafka broker (light green), which stores them in topics. The sink connector in the FTL bridge (green ellipse) consumes those messages, translates them to equivalent FTL messages, and publishes them on its endpoint. Subscribers in FTL applications (lower left, light blue) receive and process the messages.

## Parts

The diagram shows the parts (green) of TIBCO Messaging - Apache Kafka Distribution:

- **Core** Apache Kafka (light green), including the Kafka client API and the Kafka broker.
- **FTL Bridge** A pair of connectors that run within the Kafka Connect framework (dark green ellipse).
- **Schema Repository** A daemon process (dark green rectangle) to store and retrieve Apache Avro message schemas.

The FTL realm server (dark blue) is also a required part of this system, serving two roles:

- FTL clients rely on the realm server for realm configuration definitions (light blue lines). Clients also send their operating metrics to the realm server.

Notice that FTL bridge connectors are FTL application clients of the realm server.

- The schema repository relies on the realm server as back-end storage for schemas.

Notice that the schema repository is required *only* when the Kafka client applications use Avro messages.

## Message Representations

The bridge connectors support two representations for Kafka messages: Apache Avro messages and JSON string representation.

- **Avro Messages** Apache Avro is a widely used, compact representation. Parsing is fast. Avro supports schema evolution and field name aliases.
- **JSON String** JSON strings impose a minimal structure.
- **JSON Messages with Schemas** Schemas can extend the scope of message interoperability to third-party Kafka sink connectors.

# Use Cases

---

FTL and Kafka applications can exchange messages in several ways. This topic outlines some of those use cases.

## Avro Messages

Messages flow in two directions between FTL applications and Kafka topics.

Kafka applications read and write Avro messages, according to the FTL-Avro translation scheme. For details about message translation, see [FTL-Avro Translation Reference](#). For detailed examples, see the sample programs in `TIBCO_HOME/akd/repo/1.0/samples`.

The TIBCO schema repository assists with schema storage and management.

## JSON Messages without Schemas

Messages flow in two directions between FTL applications and Kafka topics.

Kafka applications read and write JSON messages, *without schemas*, according to the FTL-JSON translation. (For detailed examples, see [FTL-JSON Translation Reference](#).)

## JSON Messages with Schemas

Messages flow in two directions between FTL applications and Kafka topics.

Kafka applications read and write JSON messages *with attached schemas*.

Third-party sink connectors can also read JSON messages *with attached schemas*.

# FTL Bridge

TIBCO FTL Bridge forwards and translates messages between FTL clients and the Kafka broker. The bridge is a subcomponent of TIBCO Messaging - Apache Kafka Distribution.

The bridge operates as a Kafka Connect connector. The Java archive file includes a source connector and a sink connector.

With the source connector you can subscribe to TIBCO FTL message streams. The source connector converts FTL messages to Kafka messages and stores them in an Apache Kafka broker.

With the sink connector you can translate messages stored in Kafka, and publish them on TIBCO FTL endpoints.

## Configuring Kafka Connect

The bridge relies on the Kafka Connect framework. You must configure the Kafka Connect properties file.

Supply the properties file as the first argument in the Kafka Connect command line. (For examples, see [Running the Connectors](#), where the properties file appears as `config/connect-standalone.properties`.)

### Procedure

1. Configure the locations of the Kafka brokers.

Set the `bootstrap.servers` parameter to a list of `host:port` locations of Kafka brokers.

For example:

```
bootstrap.servers=localhost:9092
```

2. Configure the key and value converters.

For example, to convert between FTL messages and Avro messages:

```
value.converter=com.tibco.messaging.kafka.avro.AvroConverter
key.converter=com.tibco.messaging.kafka.avro.AvroConverter
```

The following table describes the compatible converters.

Converter	Description
<code>com.tibco.messaging.kafka.avro.AvroConverter</code>	<p>The Avro converter stores and retrieves Avro messages on disk at the Kafka broker.</p> <p>Use this converter to bridge between FTL applications and Kafka applications that use Avro messages. You can also use this converter to convert between Avro messages and Kafka Connect in-memory representation.</p> <p>The Avro converter requires the realm server (see step 3).</p>

Converter	Description
<code>org.apache.kafka.connect.storage.StringConverter</code>	<p>The string converter stores and retrieves messages in JSON string representation on disk at the Kafka broker.</p> <p>Use this converter to bridge between FTL applications and Kafka applications that use JSON strings without attached schemas.</p>
<code>org.apache.kafka.connect.json.JsonConverter</code>	<p>The JSON converter stores and retrieves JSON messages, optionally with schemas attached.</p> <p>Use this converter to bridge between FTL applications and Kafka applications that use JSON messages with schemas attached.</p> <p>To attach a schema to each message, see step 4.</p>

3. If you use the Avro converter, configure the locations of FTL realm servers and the schema repository.

(These parameters are not relevant to other converters.)

For example,

```
value.converter.ftl.realmserver = http://localhost:8080
key.converter.ftl.realmserver = http://localhost:8080

key.converter.schema.registry.url = http://localhost:8081/schema/v1
value.converter.schema.registry.url = http://localhost:8081/schema/v1
```

4. If you use the JSON converter, configure Kafka Connect to store the schema with each converted message.

(These parameters are not relevant to other converters.)

Storing schemas with JSON messages enables interoperability with some 3rd-party sink connectors.

For example,

```
value.converter.schemas.enable=true
key.converter.schemas.enable=true
```

5. Configure the plugin path.

Set the `plugin.path` parameter to a directory containing the bridge connector archive file, `tibftl-kafka-connect-1.0.0.jar`.

For example,

```
plugin.path=TIBCO_HOME/akd/bridge/1.0/lib
```

## Configuring the Connectors

You can use TIBCO FTL Bridge as a source connector (FTL into Kafka), a sink connector (Kafka into FTL), or both. Configure each direction separately by modifying a copy of the appropriate sample configuration file.

### Configuration Files

Comments in the sample configuration files document all available configuration parameters and their default values.

To transfer messages from FTL into Kafka, copy and modify the source connector configuration file `TIBCO_HOME/akd/bridge/1.0/config/tibftl-kafka-connect-source.properties`.

To transfer messages from Kafka into FTL, copy and modify the sink connector configuration file `TIBCO_HOME/akd/bridge/1.0/config/tibftl-kafka-connect-sink.properties`.

Supply the connector properties files as arguments in the Kafka Connect command line (see [Running the Connectors](#)).

### Procedure

1. Required. Ensure `ftl.realmServers` contains correct URLs for the FTL realm servers.
2. If the realm server requires client authentication, ensure `ftl.username` and `ftl.password` are the correct credentials to identify the bridge connectors as clients to the realm server.
3. If the realm server uses secure communications, ensure that the parameters `ftl.trust.*` are set correctly so that the bridge connectors trust the realm server.
4. Configure schema generation and storage.  
If you use the Avro converter or the JSON converter, configure the *source* connector to store the schema with each converted message.  
For example,  

```
schemas.enable=true
```

  
If you use the string converter, configure the source connector to *not* store schemas (which is the default behavior).  
For example,  

```
schemas.enable=false
```
5. Modify other parameters as needed.

## Configuring Realm Definitions for the Connectors

Each connector is an FTL client application, so you must configure an application definition for each connector.

### Background

Each connector instance is an FTL client application, which either subscribes to an FTL endpoint (source) or publishes to an FTL endpoint (sink). For each connector, complete the following steps.

For further details, see *TIBCO FTL Administration*.

### Prerequisites

You have already configured the Java properties file for each connector.

### Procedure

1. Configure an application definition.  
Use the same application name that you configured in the connector properties file.
2. Configure one endpoint in the application.  
Use the same endpoint name that you configured in the connector properties file.

## Running the Connectors

To run the bridge connectors in standalone mode, complete this task.

Run and manage the bridge connectors as you would any other Kafka connector. It can run in either standalone mode or distributed mode. For details see [Kafka Connect](#).

To quickly begin development and testing, run the connector in standalone mode.

## Prerequisites

You have already configured the bridge connector properties.

You have already configured application definitions in the realm server for the connectors.

## Procedure

1. Ensure access to the native TIBCO FTL client libraries.

If you use Kafka's standard start scripts, you can ensure that `java.library.path` is set correctly by including it in the environment variable `KAFKA_OPTS` before starting the connector.

- On **Linux** and **macOS** platforms, ensure that `java.library.path` contains `TIBCO_HOME/ftl/version/lib`. For example:

```
export KAFKA_OPTS="-Djava.library.path=TIBCO_HOME/ftl/version/lib"
```

- On **Windows** platforms, ensure that the `PATH` variable contains `TIBCO_HOME\ftl\version\bin`. For example:

```
set KAFKA_OPTS="-Djava.library.path=TIBCO_HOME\ftl\version\bin"
```

2. Ensure that the realm server is running and reachable.

The connectors are FTL clients, and depend on the realm server for configuration definitions. For more details about the realm server, see *TIBCO FTL Administration*.

3. Ensure that the bridge connector archive file `tibftl-kafka-connect-1.0.0.jar` is in the plug-ins directory.

Kafka's configuration file `config/connect-standalone.properties` specifies the location of the plug-ins directory as the value of the `plugin.path` property.

You can either set this property to the location of the archive file, or copy the archive file to the plug-ins directory.

4. Navigate to the Kafka installation directory.

For example:

```
cd TIBCO_HOME/akd/core/1.1
```

5. Run the bridge connectors.

Select from the following command line examples and modify as appropriate. Notice that each example command line supplies a configuration file for the Kafka Connect worker, and a configuration file for each bridge connector instance.

- **Source Connector, Standalone**

```
bin/connect-standalone.sh
  config/connect-standalone.properties
  config/tibftl-kafka-connect-source.properties
```

- **Sink Connector, Standalone**

```
bin/connect-standalone.sh
  config/connect-standalone.properties
  config/tibftl-kafka-connect-sink.properties
```

- **Both Connectors, Standalone** For bidirectional communication, supply both configuration properties files.

```
bin/connect-standalone.sh
  config/connect-standalone.properties
  config/tibftl-kafka-connect-source.properties
  config/tibftl-kafka-connect-sink.properties
```

- **Multiple Connectors, Standalone** To publish or subscribe to multiple FTL endpoints, run multiple instances of the connector. Provide a separate configuration file for each instance.

```
bin/connect-standalone.sh
  config/connect-standalone.properties
```

```
config/tibftl-kafka-connect-source-A.properties  
config/tibftl-kafka-connect-source-B.properties  
config/tibftl-kafka-connect-sink-C.properties  
config/tibftl-kafka-connect-sink-D.properties
```

## Bridge Connector Troubleshooting

When running the bridge connectors, this information can help you resolve errors.

### No tibftljni in java.library.path

**Problem:** The connector fails to start, and the Kafka Connect log file contains the following error:

```
java.lang.UnsatisfiedLinkError: no tibftljni in java.library.path
```

**Solution:** Ensure that you have installed a supported version of TIBCO FTL software. Set the `java.library.path` correctly (see [Running the Connectors](#)).

### Could not connect to FTL realm server

**Problem:** The connector does not produce or consume any messages, and the Kafka Connect log file contains the following error:

```
Could not connect to FTL realm server; make sure it is running and reachable;  
retrying...
```

**Solution:** Ensure the TIBCO FTL realm server is running and reachable from the connector host computer.

# Schema Repository

The schema repository manages Avro message schemas.

The schema repository registers Avro schemas, associating each subject name with its schema. As you update a subject's schema, the subject automatically retains its schema history in versions.

Kafka client applications and Kafka brokers can use the schema repository to manage Avro schemas.

Within client processes, TIBCO's Avro serialization library accesses the schema repository by directing REST requests to the schema repository at `schema_repository_host:port`.

For REST API documentation, run the repository, then use a browser to view `http://schema_repository_host:port`.

The schema repository uses the FTL realm server as back-end storage.

## Schema Repository Command Line Reference (tibschmad)

Use the **tibschmad** command line executable to start schema repository process.



The schema repository executable installs in `TIBCO_HOME/akd/repo/1.0/bin/tibschmad`.

### Client Communication

Parameter	Arguments	Description
<b>-l</b> <b>-listen</b>	<i>interface:port</i>	Optional.  The repository listens for requests at this interface and port. Nonetheless, it is good practice to direct requests through the realm server, which redirects the requests to an appropriate schema repository.  When absent, the default location is <code>localhost:8081</code> .

### Realm Server Back-End Storage

Parameter	Arguments	Description
<b>-ftl</b>	<i>URL</i>	Optional.  A list of the realm server URLs, separated by the pipe character ( <code> </code> ).  The schema repository uses the realm server at this location for back-end schema storage.  When absent, the default URL is <code>http://localhost:8080</code>
<b>-u</b> <b>-user</b>	<i>user</i>	Optional. (Required for communication with a secure realm server.)  The schema repository authenticates itself to the realm server with this user name.

Parameter	Arguments	Description
<b>-p</b> <b>-password</b>	<i>password</i>	<p>Optional. (Required for communication with a secure realm server.)</p> <p>The repository authenticates itself to the realm server with this password. Supply one of the following forms:</p> <ul style="list-style-type: none"> <li>• <code>stdin</code></li> <li>• <code>env:environment_var_name</code></li> <li>• <code>file:password_file_path</code></li> <li>• <code>pass:password</code></li> </ul> <p>For further details, see <a href="#">Password Security</a>.</p>
<b>-trust-file</b>	<i>file_path</i>	<p>Optional. (Required for TLS communication with a secure realm server.)</p> <p>When present, the repository process reads the realm server's trust file from this path, and uses that trust data in communications with the secure realm server.</p>
<b>-trust-everyone</b>		<p>Optional.</p> <p>The repository trusts any realm server without verifying trust in the realm server's certificate.</p> <div>  <p>Do not use this parameter except for convenience in development and testing. It is not secure.</p> </div>
<b>-mem</b>		<p>Optional.</p> <p>When present, the schema repository stores schemas only in process memory (which is not persistent) instead of using the realm server for backend storage.</p> <div>  <p>Do not use this mode in production environments.</p> </div>

## Configuration

Parameter	Arguments	Description
<b>-c</b> <b>-config</b>	<i>file_path</i>	<p>Optional.</p> <p>Path to a JSON configuration file.</p> <p>Command line arguments override environment variables, which override configuration file arguments.</p> <p>When absent, the schema repository first looks for <code>./tibschemad</code>, then <code>~/tibschemad</code>.</p>

Parameter	Arguments	Description
<b>-env</b>		Optional. The repository prints the environment variables that would produce its current configuration and exits.
<b>-show-config</b>		Optional. The repository prints the contents of a configuration file that would produce its current configuration and exits.

### Tracing and Logging


Parameter	Arguments	Description
<b>-debug</b>		Optional. When present, print debugging information.
<b>-q</b> <b>-quiet</b>		Optional. When present, the repository prints minimal output.
<b>-v</b> <b>-verbose</b>		Optional. When present, print verbose output.
<b>-version</b>		Optional. When present, the repository outputs version information and exits.

## Password Security

### Password Argument

When you supply a password as a command line argument, that argument is visible to casual observers. For example, command line arguments appear in the output of the UNIX `ps` command.

You can supply password arguments in any of the following four 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>stdin</code>	<p>This form can provide maximum security: after entering the password, it is no longer visible to anyone.</p> <p>You can pipe the password to the realm server executable through <code>stdin</code>. For example, in UNIX environments, you could use this command line:</p> <pre>echo my_password   tibschemad ... --password stdin</pre> <p>You could use an encrypted password management application to supply the password to <code>stdin</code>. In this scenario, the password is not visible during any task step.</p>
<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 realm server's host computer, and ensure the security of that file.</p>
<code>env:environment_var</code>	<p>This form can provide excellent security.</p> <p>You must set an environment variable in the shell where you run the realm server. The value of that variable is the password string. You must ensure that only authorized personnel have access to that shell.</p>
<code>pass:password</code>	<div>  <p>With this form the password remains in the process command line, which is visible to casual observers. Do <i>not</i> use this form except during development and early testing phases.</p> </div>

## Schema Repository Security

The schema repository is secure if and only if the realm server is secure.

When the repository connects and authenticates to a secure realm server, the realm server automatically generates and signs a temporary certificate that identifies the repository to its clients. The repository stores this certificate in process memory only.

Clients need only the realm server trust file to verify the identity of the repository because its certificate is signed by the realm server.

### Repository Certificate Parameters

Some situations that require the repository's temporary certificate to name a specific host name or IP address. The repository embeds the values of two optional parameters in the subject alternate name (SAN) portion of its certificate. When these parameters are absent, empty, or null, the repository attempts to supply reasonable values.

Configuration File Parameter	Environment Variable	Description
<b>reachable_dns</b> Value is a JSON array of strings.	TIBSCHEMAD_REACHABLE_DNS Value is a comma-separated list.	Host name where clients can reach the repository.

Configuration File Parameter	Environment Variable	Description
<b>reachable_ip</b> Value is a JSON array of strings.	TIBSCHEMAD_REACHABLE_IP Value is a comma-separated list.	IP address where clients can reach the repository.

# Java Programmer's Checklist

---

Use this checklist when developing Java programs that use the APIs in TIBCO Messaging - Apache Kafka Distribution.

## Environment

If your Java application serializes or deserializes Avro messages, the CLASSPATH must include an archive file that contains the TIBCO Avro client library classes (*TIBCO\_HOME*/adk/rep0/1.0/lib/tibftl-kafka-avro-1.0.0.jar).

## Compile

If your Java application serializes or deserializes Avro messages, the CLASSPATH must include an archive file that contains the TIBCO Avro client library classes (*TIBCO\_HOME*/adk/rep0/1.0/lib/tibftl-kafka-avro-1.0.0.jar).

TIBCO Avro client library classes require Java 1.8 (or later) 64-bit.

## Run

Ensure that the Kafka broker is running, and reachable from each client process.

If your Java programs use the schema repository, ensure that the repository is running, and reachable from each client process. Ensure that an FTL realm server is running, and reachable from the repository.

TIBCO Avro client library classes requires Java 1.8 (or later) 64-bit.

## FTL-Avro Translation Reference

---

This topic details the translation between FTL messages and Avro messages.

The FTL bridge connectors can use this translation scheme in both directions.

### Numbers

FTL messages treat all integer values as long (64 bits), and all floating point values as double. The translation from Avro to FTL casts numeric values accordingly. The translation from FTL to Avro does not attempt to use smaller types.

### Inbox

FTL inbox fields are not supported outside of FTL. The translation from FTL to Avro discards inbox fields.

### Message Arrays

FTL supports arrays of messages, but Avro does not. The translation from FTL to Avro discards message array fields.

(The JSON string converter can support arrays of messages when schemas are not enabled.)

# FTL-JSON Translation Reference

This topic details the translation between fields in FTL messages and equivalent fields in JSON messages (without schemas).


The FTL bridge connectors can use this translation scheme in both directions. However, they use this translation only when the configuration specifies the string converter without schemas.


## Inbox

FTL inbox fields are not supported outside of FTL. The translation from FTL to JSON discards inbox fields.

## Field Translations

The table juxtaposes the Java code fragment that would create an FTL message against the equivalent JSON representation.

FTL (Java)	JSON
<code>message.setLong("my-long", 1);</code>	<code>"my-long": 1</code>
<code>long[] longArray = {1, 2, 3}; message.setArray("my-long-array", longArray);</code>	<code>"my-long-array": [ 1, 2, 3 ]</code>
<code>message.setString("my-string", "hello");</code>	<code>"my-string": "hello"</code>
<code>String[] stringArray = {"eeny", "meeny", "miny"}; message.setArray("my-string-array", stringArray);</code>	<code>"my-string-array": [ "eeny", "meeny", "miny" ]</code>
<code>message.setDouble("my-double", 9.9);</code>	<div> <code>"my-double": { "_d_": 9.9 }</code> </div> <div>  <p>An FTL double field translates as a JSON object with one field named <code>"_d_"</code>. Its value must be either a JSON number, or one of the following special string values:</p> <ul style="list-style-type: none"> <li><code>"Infinity"</code> (positive infinity)</li> <li><code>"-Infinity"</code> (negative infinity)</li> <li><code>"NaN"</code> (not a number)</li> </ul> </div>
<code>double[] doubleArray = {1.1, Double.POSITIVE_INFINITY, Double.NEGATIVE_INFINITY, Double.NaN}; message.setArray("my-double-array", doubleArray);</code>	<code>"my-double-array": [ { "_d_": 1.1 }, { "_d_": "Infinity" }, { "_d_": "-Infinity" }, { "_d_": "NaN" } ]</code>

FTL (Java)	JSON
<pre>TibDateTime dateTime = new TibDateTime(); dateTime.set(443815200, 0); message.setDateTime("my-dateTime", dateTime);</pre>	<pre>"my-dateTime": {   "_m_": 443815200000 }</pre> <p> An FTL TibDateTime field translates as a JSON object with one field named "_m_". Its value must be a JSON number, which represents the number of milliseconds since the UNIX Epoch.</p> <p>The JSON representation excludes nanoseconds, truncating to the nearest millisecond.</p>
<pre>TibDateTime dt1 = new TibDateTime(); dt1.set(1168365600, 0); TibDateTime dt2 = new TibDateTime(); dt2.set(1003860000, 0); TibDateTime dt3 = new TibDateTime(); dt3.set(1003860000, 0); TibDateTime[] dateTimeArray = {dt1, dt2, dt3}; message.setArray("my-dateTime-array", dateTimeArray);</pre>	<pre>"my-dateTime-array": [   { "_m_": 1168365600000 },   { "_m_": 1003860000000 },   { "_m_": 1003860000000 } ]</pre>
<pre>byte[] opaque = {'H', 'i'}; message.setOpaque("my-opaque", opaque);</pre>	<pre>"my-opaque": {   "_o_": "SGk=" } }</pre> <p> An FTL opaque field translates as a JSON object with one field named "_o_". Its value is a JSON string containing base64-encoded binary data.</p>
<pre>Message nestedMessage = realm.createMessage(null); nestedMessage.setLong("my-nested- message-long", 2); message.setMessage("my-message", nestedMessage);</pre>	<pre>"my-message": {   "my-nested-message-long": 2 } }</pre> <p> A nested FTL message field is represented as a nested JSON object using the same JSON representation as any other FTL message.</p>
<pre>Message m1 = realm.createMessage(null); m1.setString("m1-string", "ftl-is- great"); Message m2 = rlm.createMessage(null); m2.setDouble("m2-double", 4.5); Message m3 = rlm.createMessage(null); m3.setLong("m3-long", 3); Message[] messageArray = {m1, m2, m3}; message.setArray("my-message-array", messageArray);</pre>	<pre>"my-message-array": [   { "m1-string": "ftl-is-great" },   { "m2-double":     { "_d_": 4.5 } },   { "m3-long": 3 } ]</pre>