TIBCO Silver® Fabric Enabler for Generic Adapter

User's Guide

Software Release 1.0
November 2015

Two-Second Advantage®

Important Information

Contents

Figures

Preface

TIBCO Silver ® Fabric Enabler for Generic Adapter ™ is a complementary software component. It allows TIBCO Adapter projects to be deployed in cloud environments based on TIBCO Silver Fabric and leverage Silver Fabric capabilities. This accelerates deployments of Adapter projects, enforces its industry best practices, and provides elastic optimization of computing resources.

Topics

Related Documentation

This section lists documentation resources you may find useful.

TIBCO Silver Fabric Enabler for Generic Adapter Documentation

The following documents form the TIBCO Silver® Fabric Enabler for Generic Adapter™ documentation set:

- TIBCO Silver® Fabric Enabler for Generic Adapter™ *Installation*
  Read this manual for instructions on site preparation and installation.
- TIBCO Silver® Fabric Enabler for Generic Adapter™ *User's Guide*
  Read this manual for instructions on using the product.
- TIBCO Silver® Fabric Enabler for Generic Adapter™ *Release Notes*
  Read the release notes for a list of new and changed features. This document also contains lists of known issues and closed issues for this release.

Other TIBCO Product Documentation

TIBCO Silver Fabric Enabler for Generic Adapter provides the TIBCO Silver Fabric private cloud infrastructure that can run TIBCO Adapter.

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

- TIBCO Silver® Fabric
- TIBCO ActiveMatrix BusinessWorks™
- TIBCO Designer™
- TIBCO Administrator™
- TIBCO Rendezvous®
- TIBCO Hawk®
- TIBCO Runtime Agent™
- TIBCO Enterprise Message Service™

Typographical Conventions

The following typographical conventions are used in this manual.

Table 1 General Typographical Conventions

Convention	Use
TIBCO_HOME *ENV_HOME* *SFGA_HOME* *SILVERFABRIC_HOME*	Many TIBCO products must be installed within the same home directory. This directory is referenced in documentation as *TIBCO_HOME*. The default value of *TIBCO_HOME* depends on the operating system. For example, on Windows systems, the default value is `C:\tibco`. Other TIBCO products are installed into an *installation environment*. Incompatible products and multiple instances of the same product are installed into different installation environments. An environment home directory is referenced in documentation as *ENV_HOME*. The default value of *ENV_HOME* depends on the operating system. For example, on Windows systems, the default value is `C:\tibco`. TIBCO Silver® Fabric Enabler for Generic Adapter™ is installed into a directory that is referenced in documentation as SFGA_HOME. The value of SFGA_HOME depends on the operating system. For example, on Windows systems, the default value is C:\tibco\sfga. TIBCO Silver Fabric is installed into a directory that is referenced in documentation as *SILVERFABRIC_HOME*. The value of *SILVERFABRIC_HOME* depends on the operating system. For example, on Windows systems, the default value can be `C:\fabric`.
`code font`	Code font identifies commands, code examples, filenames, pathnames, and output displayed in a command window. For example: Use `MyCommand` to start the foo process.
`bold code font`	Bold code font is used in the following ways: • In procedures, to indicate what a user types. For example: Type **`admin`**. • In large code samples, to indicate the parts of the sample that are of particular interest. • In command syntax, to indicate the default parameter for a command. For example, if no parameter is specified, `MyCommand` is enabled: `MyCommand [`**`enable`**` | disable]`

Table 1 General Typographical Conventions (Cont'd)

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

Connecting with TIBCO Resources

How to Join TIBCOmmunity

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

How to Access All TIBCO Documentation

All product documentation is available from https://docs.tibco.com

TIBCO Silver Fabric Enabler for Generic Adapter documentation is here:

https://docs.tibco.com/products/tibco-silver-fabric-generic-adapter

How to Contact TIBCO Support

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

- For an overview of TIBCO Support, and information about getting started with TIBCO Support, visit this site:

  http://www.tibco.com/services/support

- If you already have a valid maintenance or support contract, visit this site:

  https://support.tibco.com

  Entry to this site requires a username and password. If you do not have a username, you can request one.

Chapter 1 Introduction

This chapter briefly introduces the TIBCO Silver® Fabric Enabler for TIBCO Generic Adapter .

Topics

Product Overview

TIBCO Silver Fabric Enabler for Generic Adapter is a complementary software component that enables TIBCO Adapter projects to be published in cloud environments based on TIBCO Silver Fabric to leverage Silver Fabric capabilities. This accelerates publishing of TIBCO Adapter projects, enforces its industry best practices, and provides elastic optimization of computing resources.

A TIBCO Silver Fabric Enabler for Generic Adapter is a classic container and component type. It provides scripts to build new Silver Fabric custom distributions from scratch. You can choose and configure the adapter of your choice from the existing TIBCO Adapters and can build distribution for it. For some Adapters, even if the distributions are provided by TIBCO, you can choose to build your own distributions. However, you can choose only the Adapters that use TIBCO Runtime Agent and TIBCO Domain. It is possible to build distributions for SDK custom Adapters also.

Using TIBCO Silver Fabric Enabler for Generic Adapter, you can generate the following common features for any Adapter:

- Create and update a machine
- Start Hawk
- Define Hawk running condition
- Detect deployment
- Logging

If you want any other custom behavior in your distribution, you need to develop it using the script and upload it to the component. Also, to use the TIBCO Silver Fabric Enabler for TIBCO Generic Adapter with an non existing enabler for Adapter that has a specific behavior, it must be first developed and tested by you.

By default TIBCO Silver Fabric Enabler for Generic Adapter supports all the generic features of TIBCO Adapter once deployed in a TIBCO Silver® Fabric environment:

- Adapter Configuration
- Publication Service
- Subscription Service
- Read Schema
- Write Schema (Classic and Manual)
- Delimited File Record

- Layout File Record
- Positional File Record

For more information on the features supported by the Adapter, refer to the documentation of that product.

Main Functionalities

TIBCO Silver Fabric Enabler for Generic Adapter provides the following main functionalities:

- It allows you to quickly set up a TIBCO Domain environment on multiple machines. Then you can use TIBCO Administrator to publish the Adapter projects onto this environment using traditional tools, such as TIBCO Administrator user interface or its command-line tools.
- It allows you to quickly define, set up, and publish a complete stack based on Adapter projects onto a set of virtual or physical machines. These actions include installation of this software, creation of TIBCO Domain, starting up TIBCO Administrator Server, and publish the Adapter projects on one or multiple machines.
- It guarantees that installations follow a set of recommended and supported TIBCO practices to implement fault tolerance, load balancing, software updates, and updates.
- It scales up and down the number of Adapter engines required to process the workload. Therefore, it provides elasticity and optimization of computing resources.

Components

TIBCO Silver Fabric Enabler for Generic Adapter consists of the following components:

- TIBCO Adapter Enabler

  It is used to configure, start, and manage Adapter engines and its published Stack.
- A set of distributions that includes all the software pieces required to run a Adapter environment: TIBCO Runtime Agent, TIBCO Rendezvous, TIBCO Hawk, and TIBCO ActiveMatrix Adapter.

Chapter 2

Creating an Enabler for Generic Adapter Component

This chapter explains how to use the TIBCO Silver Fabric Enabler for Generic Adapter to create, configure and publish Components.

Topics

Overview

TIBCO Silver Fabric Enabler for Generic Adapter is complementary software used with TIBCO Silver Fabric to configure and publish components that are necessary to run projects based on TIBCO ActiveMatrix Adapter.

A TIBCO Silver Fabric Enabler for Generic Adapter Stack assists in the creation of one or more Adapter Components.

To build and run a Stack using TIBCO Silver Fabric Enabler for Generic Adapter Stack, you need to do the following steps:

Table 2 Installation, Configuration, and Publication

Tasks	Notes
1. Install the product	Refer to the *Installation* guide for details.
2. Build custom distributions	Refer to the readme available with `SilverFabric_sfga_distributionBuilder_1.0.0.zip` for details.
3. Deploy the custom distributions in the Silver Broker.	Refer to the 'Grid Libraries' section in the *Installation* guide for details.
4. Create and publish one TIBCO Administrator component.	Refer to *TIBCO Silver Fabric Enabler for TIBCO Administrator - Enterprise Edition User's Guide*
5. Create and publish one or more Generic Adapter Components.	Refer to Creating Generic Adapter Components on page 8.
6. Set a dependency to the TIBCO Administrator Component for each Adapter Components	Refer to Dependency Requirements on page 24.

After completing these tasks, you can run a TIBCO Silver Fabric Enabler for Generic Adapter Stack and update it as necessary.

Creating Custom Distributions

Before creating the TIBCO Generic Adapter component, you must create custom distributions for the TIBCO Adapter of your choice.

You can create distributions for only the Adapters that use TIBCO Runtime Agent and TIBCO Domain. It is possible to build distributions for SDK custom Adapters also.

Use the readme available in the `SilverFabric_sfga_distributionBuilder_1.0.0.zip`, in order to use custom distribution builder.

The minimum versions required to use the ant script are as follows:

- Java (TM) SE Runtime Environment (build 1.6, 1.7)
- Apache ANT (TM) Version 1.8.2 compiled on December 20, 2010.

After the custom distribution are correctly created and deployed to Silver Fabric broker, you can run and deploy the archives using Silver Fabric Generic Adapter as described in Creating Generic Adapter Components on page 8.

Creating Generic Adapter Components

This Component can perform the following tasks:

- Add a virtual machine to TIBCO Domain.
- Start TIBCO Hawk Agent.
- Publish one or more Adapter projects. This task is optional.

If you do not upload the Adapter projects, you still can publish the Adapter projects on the machine where Adapter is registered to the TIBCO Domain using the TIBCO Administrator GUI.

Prerequisite

Before creating the TIBCO Generic Adapter component, you must first create a TIBCO Administrator component. It creates a TIBCO Domain, and starts TIBCO Hawk® services and a TIBCO Administrator server. Refer to *TIBCO Silver Fabric Enabler for TIBCO Administrator - Enterprise Edition User's Guide* for creating and configuring a TIBCO Administrator component.

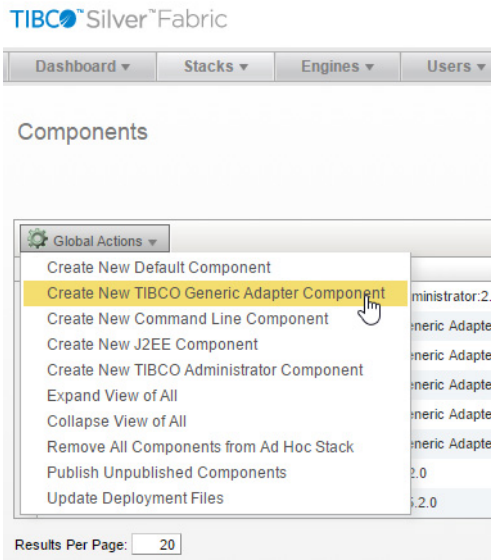
To configure a Generic Adapter Component, perform these tasks:

- Task A, Specify the Component Type and Name, page 9
- Task B, Select the Software Product Versions, page 10
- Task C, Set the Basic Configuration, page 11
- Task D, Load External JAR Files, page 15
- Task E, Upload a Adapter Project, page 15
- Task F, Hawk Application Management Interface (AMI) Configuration (optional), page 17
- Task G, Configure TIBCO Hawk Agent Running Condition, page 18
- Task H, Uploading Hawk MicroAgent Plugin (optional), page 19
- Task I, Add or Edit Enabler-Specific Runtime Context Variables, page 19
- Task J, Add or Edit Runtime Context Variables (optional), page 20
- Task K, Add Enabler and/or Component-Specific Content Files, page 22
- Task L, Add Allocation Rule Settings, page 22
- Task M, Generic Settings, page 23
- Task N, Finish Configuring the Component, page 23

Task A Specify the Component Type and Name

1. In TIBCO Silver Fabric Administrator GUI, select **Stacks > Components**.
2. In the Components page, select **Create New TIBCO Generic Adapter Component** in the **Global Actions** drop-down list.

Figure 1 Creating a New TIBCO Generic Adapter Component



3. Enter a **Distribution Name** for your distribution.
4. In **Distribution Minimum**, enter the minimum version of the distribution for the Silver Fabric to resolve all of the available distributions.
5. In **Distribution Maximum**, enter the maximum version of the distribution for the Silver Fabric to resolve all of the available distributions.
6. Check the **Enable product registration** option for the custom software to register itself as custom software on the TIBCO Administrator.

Figure 2 Configure General Properties

Task B Select the Software Product Versions

You can select the versions of the distributions for TIBCO Generic Adapter to run, as shown in Figure 3.

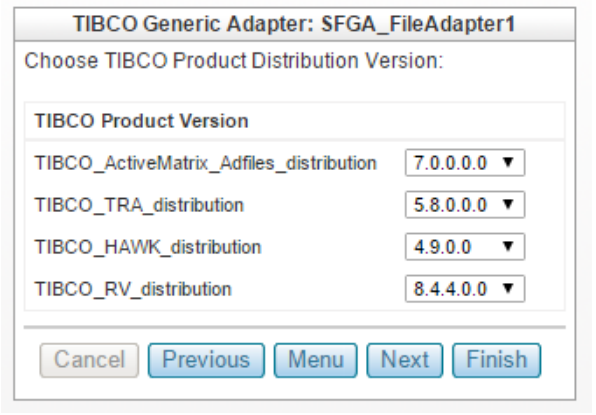
1. Select appropriate versions of the installed distributions from the dropdown and click **Next**.

   All distributions may not be compatible with your specific environment. All versions of the distributions are compatible with the Silver Fabric Enabler, but certain Distributions may support only specific operating systems as deployment destinations.

Refer to the respective product readme files for specific information on which operating systems the product distribution are supported.

It is almost always recommendable to use the latest release version number of a product distribution to take advantage of the most recent feature developments and improvements.

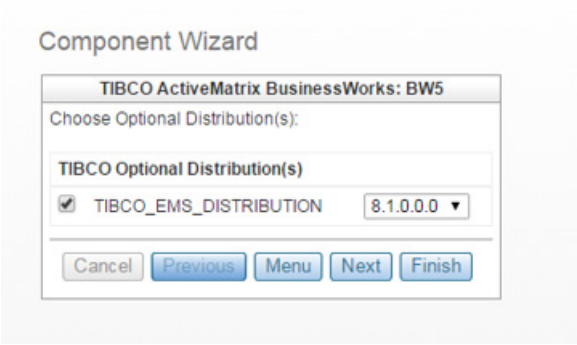
Figure 3 Software Version



2. Choose **Optional Distribution(s)** - If your TIBCO Administrator implementation makes use of TIBCO Enterprise Messaging Service, you must upload the latest TIBCO EMS Distribution to the TIBCO Silver Fabric Broker.

It is valid only for TRA and Administrator 5.9.0 and above. For TRA and Administrator version 5.8.x and below, no need to select this option even if it configured to use EMS transport.

Figure 4 Choose Optional Distributions



Task C Set the Basic Configuration

In the TIBCO Generic Adapter Basic Configuration page, you can upload a JDBC driver, set a specific TIBCO Domain machine name (the TIBCO Logical Machine name - TLM), provide for specification of Fast TLM Restart, and allow uploading of external jar files and prepending/appending the path in the ClassPath. It also provides for the specification of either full or short archive names.

Each of the settings on this page deserves an explanation so they are described in the order of their appearance on the page.

1. **Upload a JDBC Driver for this Database**

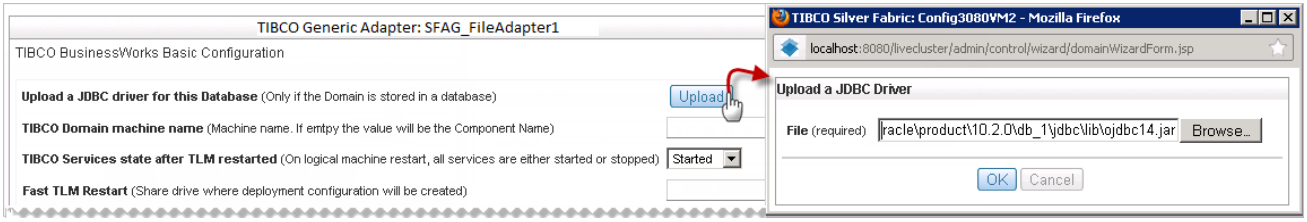
   Specify a JDBC driver to publish with the TIBCO® Generic Adapter Distribution for TIBCO Silver® Fabric so that it can communicate with the TIBCO Administrator Domain database. When the TIBCO Administrator uses a database as the domain storage, you must upload a JDBC driver so the component can interact with it.

   The JDBC driver must match the database type used by the TIBCO Administrator. This procedure is the same as uploading a JDBC driver for the TIBCO Administrator component.

The Adapter component is dependent on TIBCO Administrator. Refer to Dependency Requirements on page 24 for setting that dependency.

If the Domain is not stored in a database, then do not upload a JDBC driver.

Figure 5 Upload a JDBC Driver for TIBCO Domain Storage



2. **Set the Domain Logical Machine Name**

   The TIBCO Domain Logical Machine Name (also known as the TIBCO Logical Machine name - TLM) provides for publishing of a unique component.

   TLM is the machine name you see in the TIBCO domain where Adapter is installed. The TIBCO Domain Logical Machine virtualizes the machine publishing so that component publishing can maintain state when the targeted engine is changed for whatever reason.

   If for example, the target machine is restarted because of an OS update or a hardware change, TLM allows for the virtualized machine to be restarted on different hardware.

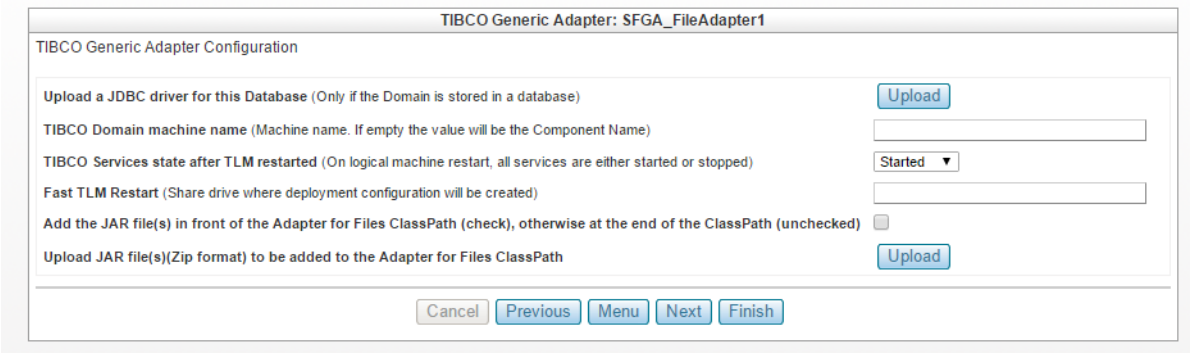
   TIBCO Domain Machine Name - To activate use of the TIBCO Domain Logical Machine Name, enter a value that is unique in the TIBCO Domain machine name field. The machine name is a virtualize name and must be unique. It can use alphanumeric characters, hyphen (-), or underscore (_) characters. Other

special characters, including period or comma, are not allowed. The name length must be less than 64 characters.

When the component is instantiated multiple times, the TLM machine name gets the name: <*Your_TLM_Name*>_<*ComponentInstanceNumber*> , where the <*ComponentInstanceNumber*> is just a sequential incrementing integer.

If <*ComponentInstanceNumber*>=0, the TLM machine name is <*Your_TLM_Name*>. when <*ComponentInstanceNumber*>>0, the TLM machine name is <*Your_TLM_Name*>_<*ComponentInstanceNumber*>.

Figure 6 Configure the TIBCO Logical Machine Name



If the TIBCO Domain machine name field is left blank the component name is used for setting the domain machine name.

3. Specify Drive For Fast TLM Restart Configurations.

Fast TIBCO Logical Machine Restart provides for accelerated restart of the Engine and redeploying of many archives within a reduced amount of time. Enhanced restart times are achieved by using a saved state stored on a shared Network File System drive instead of synchronizing with the domain repository. It updates the deployment configuration file with the new binary location. To set your expectations properly, "fast" does not mean instantaneous or even amazingly fast, but it is faster than if the archives were loaded from the domain repository.

The shared NFS directory drive for Fast TLM Restart requires the following:

— The shared drive must be `READ/WRITE` accessible to all engine daemons running as TIBCO Logical Machines in a TIBCO Silver Cloud. The degree

to which you can guarantee reliability and availability of that shared drive will help ensure runtime continuity.

— All TLM in a stack must run the same OS.

Domain configuration changes made via TIBCO Administrator or the AppManage CLI in the period between the TLM stop and the TLM restart will not be captured.

If Domain configuration changes made during the period after TLM stop and before TLM restart must be captured then the user will need to redeploy the modified application.

Recommendation: For those implementations that use fewer than ten Adapter deployments per component, avoid using Fast TLM Restart to avoid the constraints mentioned above.

If you want to force redeployment (synchronization with the domain) once, add a file call "`ForceRedeploy.txt`" in the `domainDataHome` (*<domainDataDir>/<DomainName>/<TLMNAME>/<DomainName>*. It redeploys all applications (no FAST TLM) at startup and then deletes the file (it is one time action).

To enable Fast TIBCO Logical Machine Restart simply enter the directory path of the shared NFS drive (on Windows servers - a mapped drive) and make sure that the host grants permissions allowing the user who launches the engines to read and write in that location.

4. Status of TIBCO Services after TLM Restart.

Sets the desired services state when the TIBCO Logical Machine is restarted. When the TLM is restarted, the Adapter service instances is republished and appears as either **started** or **stopped**. The stopped services state setting may be convenient for developers who are testing machines with many services that do not need to be started for every logical machine change.

5. Add JAR Files before Adapter Classpath

Add external JAR file(s) to the Adapter Component - As an option, JAR files may be uploaded for publishing with the TIBCO Generic Adapter component.

The checkbox "*Add the JAR file(s) in front of the Generic Adapter ClassPath (check), otherwise at the end of the ClassPath (unchecked)*" means just what it says. The JAR file name may be prepended in front of the ClassPath by checking the box and if it is left unchecked the JAR file name is appended at the end of the ClassPath.

Upload JAR file(s) in ZIP Format to Adapter ClassPath -

Upload individual JAR file(s) to be either appended or prepended to the ClassPath. All uploaded JARs are added in the same way according to how the checkbox is set.

To remove unwanted JAR files use the Menu button to display the list of Wizard configuration steps and select **Add/Override/Customize Enabler and Component-specific content files**. Relative paths to external jar files added may be removed with that window.

Task D Load External JAR Files

Upload and add JARs to the Generic Adapter Classpath with the JAR names as either a prefix or appended to the ClassPath.

The checkbox "*Add the JAR file(s) in front of the Generic Adapter ClassPath (check), otherwise at the end of the ClassPath (unchecked)*" means just what it says. The JAR file name may be prepended in front of the ClassPath by checking the box and if it is left unchecked the JAR file name is appended at the end of the ClassPath.

Upload individual JAR file(s) to be either appended or prepended to the ClassPath. All uploaded JARs will be added in the same way according to how the checkbox is set.

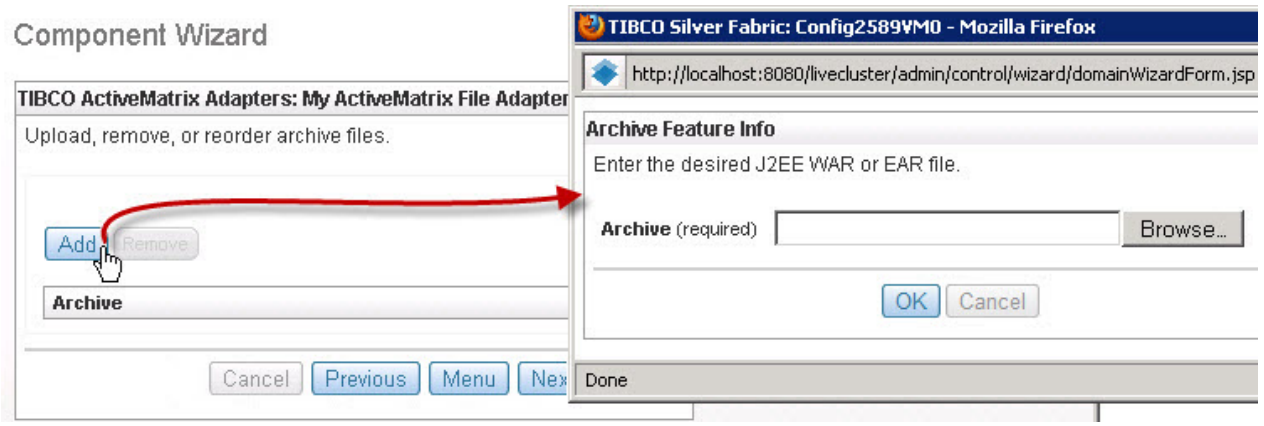
To remove unwanted JAR files use the Menu button to display the list of Wizard configuration steps and select **Add/Override/Customize Enabler and Component-specific content files**. Relative paths to external jar files added may be removed with that window.

Task E Upload a Adapter Project

If you want TIBCO Silver Fabric Enabler for Generic Adapter to publish and run one or more Adapter projects, you need to upload one or more EAR files (or ZIP files). EAR and ZIP files are briefly described in the note following Step 2: TIBCO Designer EAR and ZIP files on page 16. Enable the Generic Adapter project as follows:

1. Click the **Add** button in the Upload, Remove, or Recorder Archive Files panel.

Figure 7 Uploading an EAR or ZIP Archive File



2. Click the **Browse** button and navigate to the EAR, ZIP, or J2EE WAR file.

TIBCO Designer EAR and ZIP files

- EAR File

  When you upload a TIBCO Designer EAR file, it uses the default values of the global variables that were set using TIBCO Designer.

- Zip File

  The Zip file must contain both an EAR file and an XML configuration file. Optionally the Zip file may also contain a *.properties file if you want to define a customized folder structure.

  The XML file contains all the publishing configuration, for example, global variables, Java heap size, and so on.

To create an appropriate XML file, follow these steps:

To create a deployment configuration properties XML file for either an EAR deployment or for an archive continuous deployment, follow these steps:

a. In `TIBCO_HOME/tra/tra_version/bin`, run the following command:
   `AppManage -export -ear EarFile.ear -out DeploymentConfig.xml`

b. Edit the file and set the value for publishing.

c. Create a ZIP file with the file `EarFile.ear` and `DeploymentConfig.xml`.

For details about the creation of the `DeploymentConfig.xml` file, refer to the TIBCO Runtime Agent documentation, *Scripting Deployment User's Guide*..

A `CustomFolder.properties` file put in the zip archive can specify the deployment path. For example create the CustomFolder.properties file with content "**`ApplicationFullPath=aaa/bbb/ccc`**".

When the deployment path is not specified then the deployed EAR/application files can be found in the root folder level of Application Management in TIBCO Administrator.

When the EAR application is deployed to the runtime the project files are placed in the folder structure according to the file folder structure defined in the properties file, xml file, or EAR in the zip file.

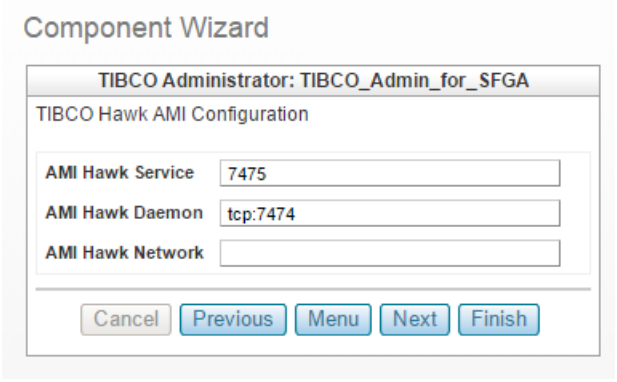
In previous releases of this Enabler, when the deployment path was not specified then the deployed project EAR and application files could be found in the TIBCO Administrator directory shown below:
`Silver Fabric/<ComponentName>/<TLM_Name>/<AppName>`

These files can also be uploaded separately and then deployed, undeployed, started, and stopped by HTTP REST request. Refer to the section on Deploying Archives Directly to Endpoints - Continuous Deployment on page 27 for more information.

Task F Hawk Application Management Interface (AMI) Configuration (optional)

TIBCO Hawk AMI may be used to monitor Generic Adapter for deployments.

Figure 8 TIBCO Hawk AMI Configuration



AMI Hawk Service - specifies the TIBCO Hawk port number, for example, TIBCO Rendezvous will connect with TIBCO Hawk on the default port 7474. AMI Hawk Service and AMI Hawk Daemon must be set with valid values together. Setting only one of them will result in an error.

AMI Hawk Daemon - specifies the location of the TIBCO Hawk Daemon. A value of "tcp:*yyyy*" would correspond to a local Hawk Daemon where "*yyyy*" would be the port number. A Hawk Daemon located elsewhere would be specified by a value of the protocol, IP address, and port number: "tcp:*xxx.xxx.xxx.xxx:yyyy*".

The AMI Hawk Service and the AMI Hawk Daemon ports may be the same or they may be different. By default, different TLMs on the same engine daemon (physical machine) are using the same RV transport (default AMI Hawk Service=7475, and default AMI Hawk Daemon=tcp:7474). This setting enables visibility of all of the deployed applications on each TLM even if some applications are NOT deployed on this particular TLM.

The actual AMI Hawk Service port for the runtime component instance is incremented by an integer according to the engine instance ID. For example, if the user sets the AMI Hawk Service to 6464 and the component is instantiated to run on engine instance 1, then the service is reported in the `hawkagent.cfg` file as 6465. Then when the component is scaled up to other engine instances the AMI Hawk Service value will be incremented higher by the Enabler automatically.

Generally, AMI Hawk Network is an empty string.

Task G Configure TIBCO Hawk Agent Running Condition

Use the following options to configure running conditions for TIBCO Hawk agent:

- **Polling Period (in seconds) for detection of TIBCO Hawk Agent running verification (required)**

  Enter an integer to specify the number of seconds between periodic verification checks that the TIBCO Hawk Agent is still running.
  If the TIBCO Hawk Agent becomes unresponsive to this verification then the process is automatically restarted.

Figure 9 TIBCO Administrator and Hawk Agent Running Conditions

- **Automatically Restart Silver Fabric Engine if TIBCO Hawk Agent fails to restart N successive times (required)**

Enter an integer to specify the number of restart retries for the TIBCO Hawk Agent before the TIBCO Silver Fabric Engine is to be restarted. A successful restart resets the count.

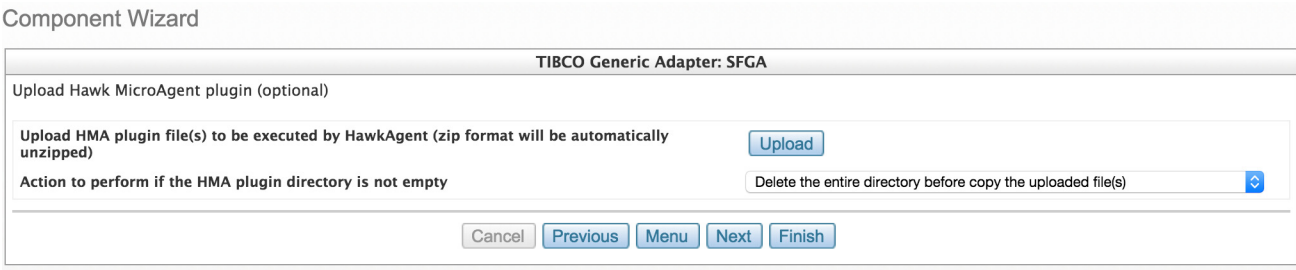
Task H Uploading Hawk MicroAgent Plugin (optional)

You can upload Hawk MicroAgent plug-ins to HawkAgent as a .zip file. These Hawk microagents collect information and operate using that information. They execute specific tasks known as methods.

The zip file gets extracted into the HMA plugin directory. If the directory is not empty, select one of the following options:

- **Delete the entire directory before copying the uploaded files**
- **Keep all existing files and replace the existing files with the uploaded ones**

Figure 10 Upload Hawk MicroAgent Plugin

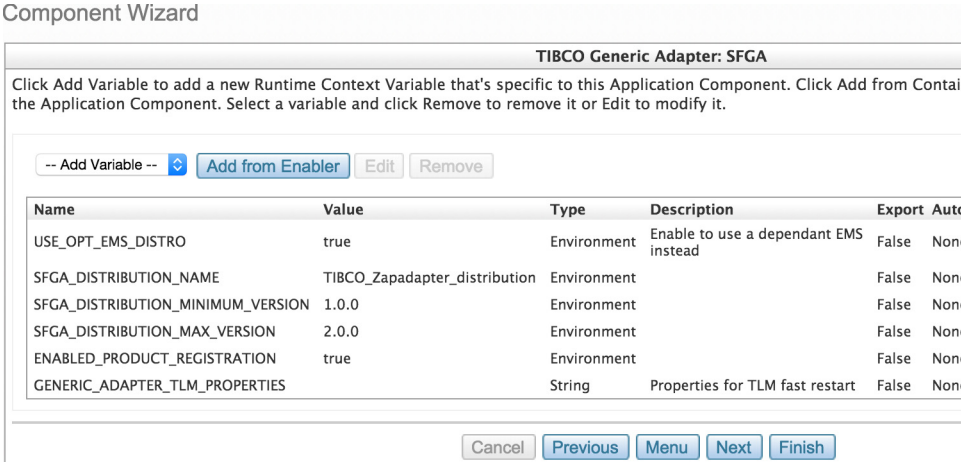


Task I Add or Edit Enabler-Specific Runtime Context Variables

String, **Environment**, **System**, or **Encrypted** variables may be added to the component to define and set runtime specific context variables.

Select a variable type from the **Add Variable** pull-down list or **Add from Enabler** to use a variable from a selected Enabler.

Figure 11 Adding a Runtime Context Variable



Once you have added any runtime context variable you may select the variable (selected row is highlighted) and **Edit** to change its attributes. Selected rows may also be removed.

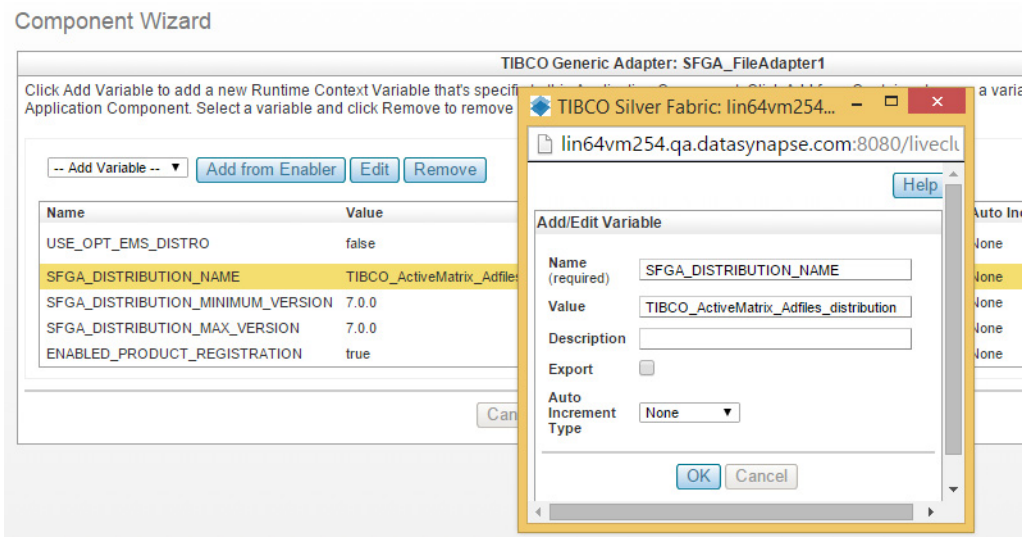
Changes are of course optional, because all variables have default values that are appropriate for the most common use cases.

Variable values from the Enabler may be added to the runtime as well. Use the **Add from Enabler** button to add Enabler-specific context variables.

Task J Add or Edit Runtime Context Variables (optional)

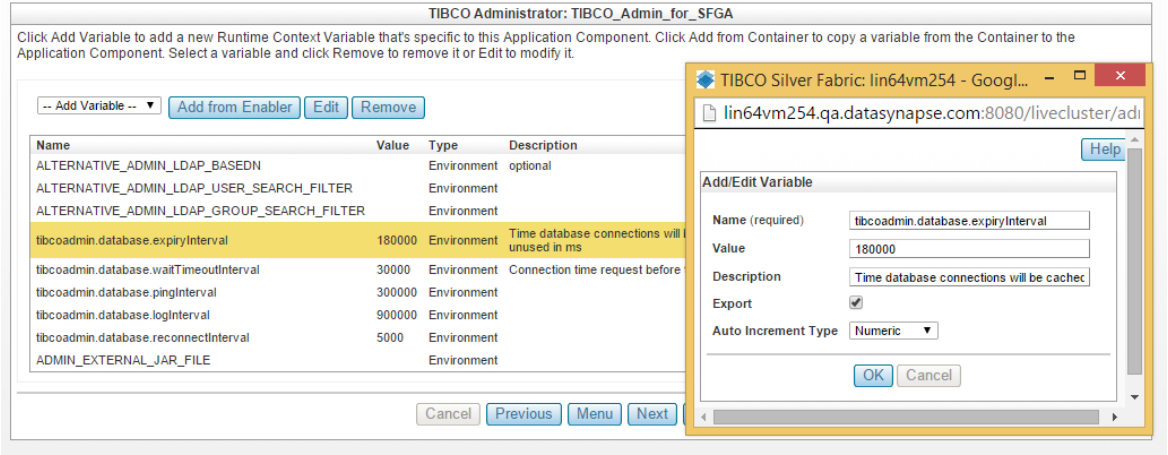
Variables may be added to define runtime specific context variables. **String**, **Environment**, **System**, or **Encrypted** variables may be added. Select a variable type from the **Add Variable** selector if you wish to add or edit a variable.

Figure 12 Editing a Variable



You can add a string variable to change pre-existing context variables like: ARCHIVE_DETECTION_FREQUENCY.

Figure 13 Add or Edit a String Variable



ARCHIVE_DETECTION_FREQUENCY is the periodic interval (default value is 30 seconds) for detection of deployed archives and report to the broker. The broker uses this data to synchronize the archive with scaling rules.

These variables are not exposed in the interface elsewhere, but you can change their values. In this case it would be recommendable to set these variable values higher than the time needed to deploy and start an archive.

Changes are of course optional, because all variables have default values that are usually appropriate for the most common use cases.

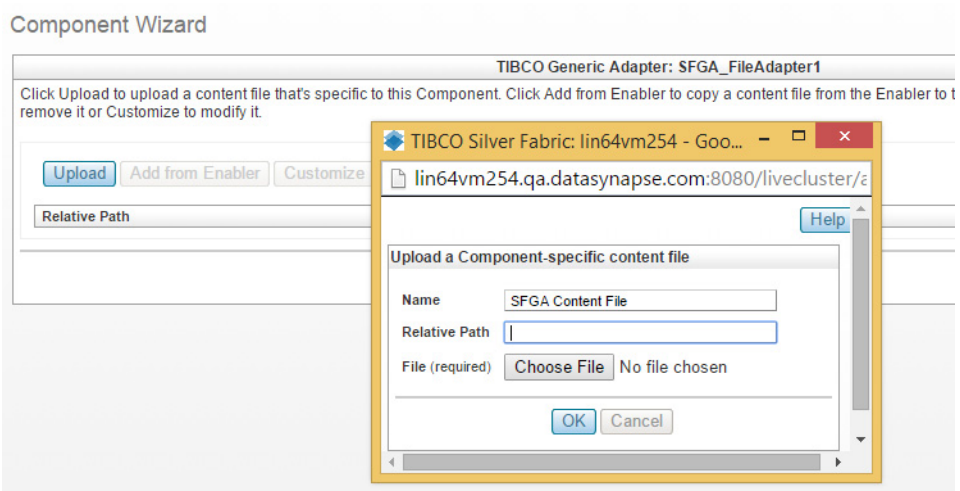
Variable values from the Enabler may be changed as well. Use the **Add from Enabler** button to change values of Enabler-specific context variables.

Task K Add Enabler and/or Component-Specific Content Files

If needed, you can upload a content file (for example, the JDBC Jar file for mysql) specific to this component from this dialog. You can perform the following operations:

- **Upload**: Upload new file.
- **Add from Enabler**: Copy a content file from the Enabler to the component.
- **Customize**: Modify the content file.
- **Remove**: Delete the file.

You can add files to a relative path associated with the Adapter component that can be required for the component to run according to design. Uploading Content Files



Task L Add Allocation Rule Settings

Add rules to specify and set component behavior. Add rules to do the following:

- Specify Resource Preferences,

- Set Thresholds for Activation,
- Set Enablement Conditions,
- Specify component dependency, or
- Set Engine Group Minimums or Maximums

Each rule selection will bring up a slightly different dialog window that allows property selection of a tracked Engine to be evaluated according to a logical operator and a value you specify to define an action.

Task M Generic Settings

The following tasks are generic for all Silver Fabric Enablers. The configuration is optional for the Generic Adapter components.

- Add/edit Application Component scripts
- Edit the configuration file

Task N Finish Configuring the Component

All the other screens are generic for all Silver Fabric enablers. The configuration is optional for Adapter components.

Refer to *TIBCO Silver Fabric User's Guide* for additional information on these configurations.

After you click the **Finish** button, make sure that the Component is published to make it available to create an Application.

To do this, select **Publish Component** in the Actions drop-down list located at the left-hand side of the component you just created.

Dependency Requirements

For each Adapter Component, you must set dependency on the TIBCO Administrator Component.

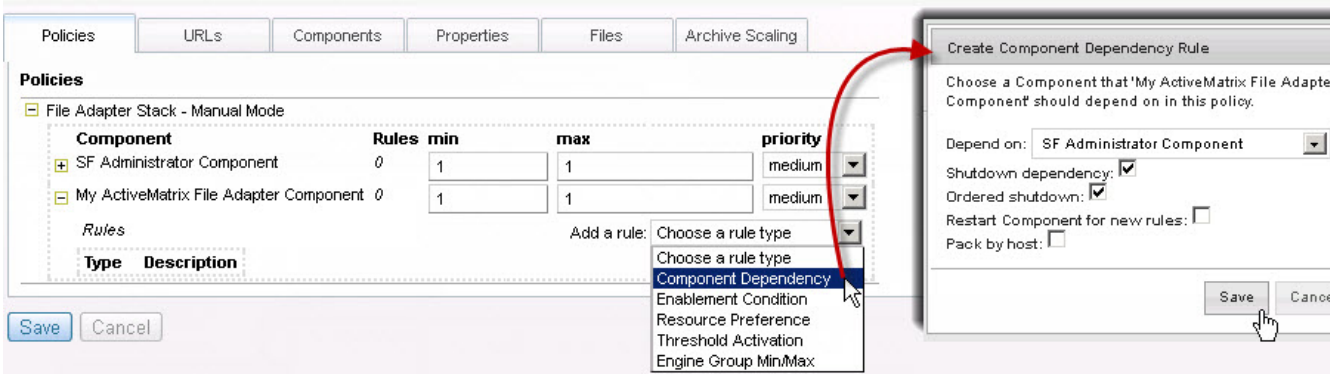
This step is required. If you do not set the dependency, TIBCO Silver Fabric Enabler for Generic Adapter will not work.

The Generic Adapter Component must have the TIBCO Administrator configuration information so that it may publish, unpublish, and communicate with other components (if required) successfully. After setting the dependency, TIBCO Adapter will start after TIBCO Administrator is up and running.

To set dependency, follow these steps:

1. During creation or during edit of the Stack select "**Add/edit default rule settings**" from the component wizard menu.
2. Use the **Add a Rule** pull down to select the **Component Dependency** option.

Figure 14 Setting an Administrator Component Dependency



In this way, the component knows all TIBCO Administrator configuration information. After setting the dependency, TIBCO Adapter starts after TIBCO Administrator is up and running.

3. In the **Depend On** `Reference Component` field, select the name of the TIBCO Administrator Component that runs inside your Stack.

If you run TIBCO Administrator in the Fault Tolerant mode, uncheck the **Shutdown Dependency** checkbox. Otherwise, all Adapter Components will stop if TIBCO Administrator stops working.

If the Administrator Component was configured to **Use dependent EMS server,** then you must set that dependency here also.

When using a dependent TIBCO Enterprise Message Service server, set the dependency in the TIBCO Administrator Component, which must also have a dependency on TIBCO EMS Server Component.

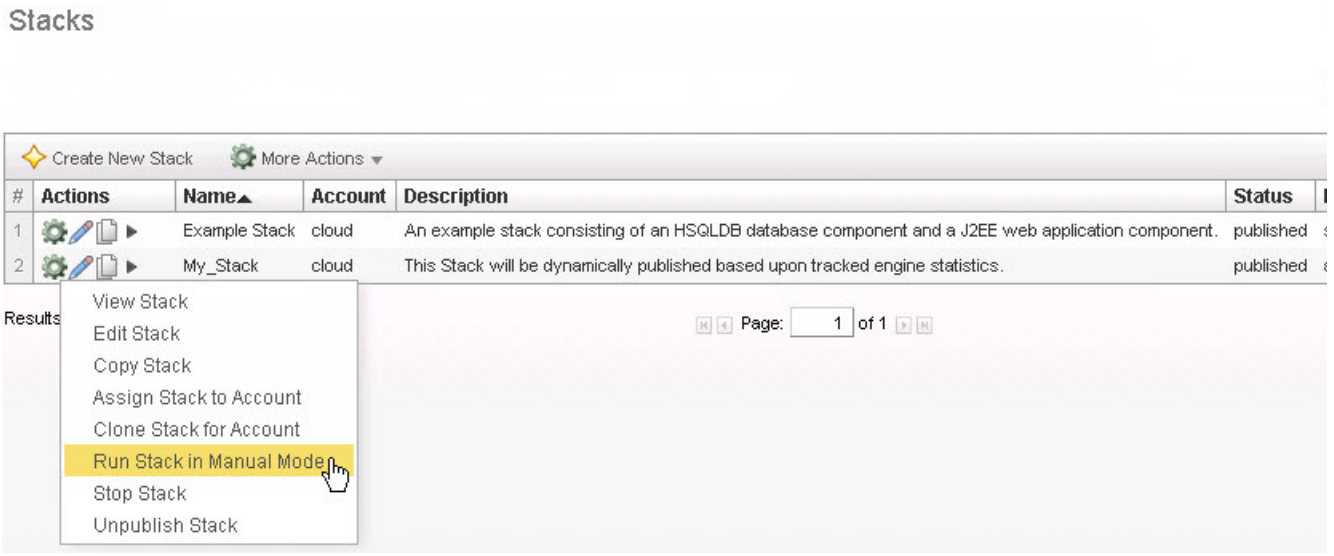
It is recommended that you check Ordered Shutdown to close dependent components when the TIBCO Administrator is shut down.

Running Stacks

After you have created your stack, use **Publish Stack**, available from the **Actions** drop-down list, to make it available to run.

After publishing, select **Run Stack In Manual Mode** in the **Actions** drop-down list as shown in Figure 15 to run the stack immediately on available resources.

Figure 15 Running a Stack



Alternatively, if the stack is defined with a Policy (schedule), you can **Run** (the) **Stack in Auto Mode**. The stack runs according to the schedule selected for the stack on available resources.

Updating the Stack

When a stack is published and running, you can still make changes to the stack such as adding other components, changing allocation rules, changing threshold activation rules, or deploying and starting archives on the runtime Adapter Application instantiated on the engine.

Making changes to the stack is as easy as editing, saving, and publishing those changes to any instances that are running. Some changes might require restart of the changed resource, so consult the TIBCO Silver Fabric documentation for best practices prior to making changes to a production system.

After making any changes to a stack, **Save** the changes and then from the Actions list in the main stack page, select **Publish Changes**. The specified engines are affected by the changes immediately

If you want to change a Adapter component, you do not need to stop and restart your entire stack. If you want to deploy, start, stop, and undeploy the project archives here are a couple of ways to accomplish that:

Micro-scaling - Start and stop Adapter archives based on your defined rules when they are already in your component..

Continuous Deployment (deploy archives directly to endpoints) - publish (deploy), unpublish (undeploy), start, or stop archives without having to change any stacks, components, or engines. Deploying application archives via REST and cURL commands is described in the next section.

Deploying Archives Directly to Endpoints - Continuous Deployment

There are situations where deploying archives directly to a running TIBCO Generic Adapter component can be useful, such as when an archive needs to be deployed and run on a system that is already running. This is known as continuous deployment. Archives can be directly deployed to Adapter instances already running on Silver Fabric Engines using the command line interface (CLI), Silver Fabric API, Ant scripts, or an HTTP REST command sent using cURL or a Java client. Refer to the *TIBCO Silver Fabric Cloud Administration Guide* for more information on the CLI or the Silver Fabric API. Archive deployment, undeployment, starting, and stopping application archives via REST are described in further detail here. For details on Ant scripts, refer the chapter "Silver Fabric Ant Tasks" in the *TIBCO Silver Fabric Developers Guide*.

TIBCO Silver Fabric supports many HTTP REST commands to `GET`, `PUT`, `POST`, and `DELETE` objects and managed resources for use with archive scaling, brokers, components, daemons, enablers, gridlibs, schedules, stacks, and Skyway.

TIBCO Silver Fabric REST Services are documented in the *TIBCO Silver Fabric Administration Tool* at **Admin > REST Services**. They are grouped by resource. Click any method name to see possible parameters if any and example responses.

Continuous deployment is discussed in good detail in the section on *Deploying Archives Directly to Components* in the *TIBCO Silver Fabric Administration Guide*.

Prior to using REST CLI with TIBCO Silver Fabric 5.6 you must change the setting, ***Strict Validation***, at **Config** > **Broker** > **General** to "false".

The TIBCO Silver Fabric Enabler for Generic Adapter adds more REST methods to enable control of Adapter archives.

Continuous Deployment Life Cycle

The continuous deployment life cycle has four REST operations that can be executed using cURL methods:

Deploy Send an archive to a Silver Fabric Engine running an Generic Adapter component that meets the criteria specified. The Deploy REST method enables specification of a properties files with criteria dictating where and how the archive should be deployed.

Start Start an archive that was deployed to an engine.

Stop Stop an archive that is running on an engine.

Undeploy Remove an archive from an engine, stopping any running instances of that archive on that engine.

Deploy

The Adapter application archives may be deployed directly to an appropriate Silver Fabric Engine (TIBCO Logical Machine) by making REST calls. In this document cURL syntax is used to show the REST inputs in a generic form:

```
curl -u UserName:Password \
     -X POST \
     -H "Accept:application/json" \
     -H "Content-Type: multipart/form-data" \
     -F "archiveFile=@YourArchiveName.zip" \
     -F "deploymentFile=@YourDeploymentFileName.properties" \
    [-F "LogicalAnd=false"]
```

```
     -v "http://YourSFBroker.com:<port>/livecluster/rest/v1/sf/eng
ines/archives"
    [-F "AppName=YourDirABC/YourAppName"]
    [-F "AppSettings.element1.element2=SomeValue"]
    [-F "ArchiveSettings.element1.element2=SomeValue"]
    [-F "Archives=Archive_A,Archive_B,Archive_X"]
    [-F "configurationFile=YourConfigurationfile.xml"]
    [-F "ForceDeploy=true"]
    [-F "GV=globalVariableA=123,globalVarB=SomeString"]
    [-F "InstanceSettings.element1.element2=SomeValue"]
    [-F "NoDeploy=true"]
    [-F "NoStart=true"]
    [-F "VariableProvider=Some_PROVIDER(S)_Name"]
```

Expressions in the generic form cURL expression as mentioned earlier were separated by line breaks to help with readability, but normally a string is submitted in the execution as in the example that follows:

Example 1 An example cURL archives deploy statement:

```
curl -u admin:admin -X POST -H "Accept:application/json" -H "Conte
nt-Type: multipart/form-data" -v http://MySFBroker.com:8080/livecl
uster/rest/v1/sf/engines/archives -F "archiveFile=@MyProcessOrder.
ear" -F "AppName=MyOrders/MyProcOrder" -F "deploymentFile=@MyDeplo
yCriteria.properties"
```

Where the user specified by `-u` must have Silver Fabric administrator level permissions, and the form-data fields (`-F "property=value"`) may be specified in any order. The form-data fields (`-F`) are described:

Mandatory form-data contain the files necessary for the deployment:

`archiveFile:` specifies the archive file (.zip, .par, or .ear file) to upload to the Silver Fabric Broker which will then publishes the archive to the appropriate Silver Fabric engine. Multiple application archives may be deployed in a single archive .zip or .ear file. You can deploy and run them all (default behavior) or you can selectively run a list of archives by specifying that list with the **`Archives`** form-data field. Archives may also be uploaded using the component wizard.

`deploymentFile:` specifies the properties file that defines the endpoint selection criteria described as follows in more detail.

```
     -F "deploymentFile=@YourDeploymentFileName.properties"
```

`LogicalAnd` (optional)**`:`** by default **all** criteria specified in the deployment properties file must be satisfied for deployment to an application endpoint, but that can be toggled to mean **any** (meaning logical OR) of the deployment properties criteria by setting: `-F "LogicalAnd=false"`

`-v:` specifies the target of the cURL POST execution and asks for a verbose response. The cURL `-v` expression should specify the appropriate Silver Fabric directory. For the default installation, that expression would look like the following:

```
 -v "http://YourSilverFabricBrokerName.com:<port>/livecluster/rest/
v1/sf/engines/archives"
```

Where the default http <port> is 8080 and optional form-data fields can specify other continuous deployment behavior.

`AppName:` specifies the directory location where the application archives are deployed and what the application is named.

Where your archive application deploys and what it is called depends on what you specify with `AppName`. For example, when you use TIBCO Designer to create an `.ear` file with a name such as *MyAppArchive*, varying the `AppName` specification gives following behavior:

- If the `AppName` form-data field is not specified, *MyAppArchive* is deployed at the top level of the directory.
- If `-F "AppName=`***`A`***`"` is submitted in the `curl` request, *MyAppArchive* is renamed *A* and deployed at the top level.
- If `-F "AppName=`***`A/`***`"` is sent, the directory folder *A* is used or it is created and *MyAppArchive* is deployed within that subdirectory.
- If `-F "AppName=`***`A/B`***`"` is sent, the subdirectory *A* is used or created, and *MyAppArchive* is deployed there and renamed *B*.
- If `-F "AppName=`***`A/B/`***`"` is sent, the folder *A* with a subfolder *B* is used or created and *MyAppArchive* is published within the subfolder *B*.

The full application name is derived from the `AppName` directory location and the application archive name as it is deployed.

Optional form-data fields do not need to be specified unless you want to change the default behavior. By default, all archives in the archive file are deployed and started unless an archive of the same name is already deployed and started, in which case that archive is allowed to run without interruption or replacement.

`AppSettings` (optional): specifies settings that your application uses when deployed.

All deployment configuration cURL expressions takes the form:

```
 -F "AppSettings.element1.element2=SomeValue"
Some examples:
 -F "AppSettings.localRepoInstance.encoding=UTF-8"
 -F "AppSettings.description=This%20application%20deployment%20is%20fo
r%20validation%20testing."
```

Where the element is one of the following (plus any subordinate *element2* where applicable):

a. **description :**a string describing the application.

b. **contact** :a string to name the person responsible for the deployment.

c. **maxDeploymentRevision** :specifies the default number of application revisions to keep in the revision history for each deployed application. Leave the value at -1 to keep all revisions by default.

d. **localRepoInstance** : for Enabler installed components and application archives installed with continuous deployment, a local file (or directory of files) is used as the deployment repository instance.

When deploying applications your domain is automatically configured to establish a local application repository managed by TIBCO Runtime Agent. This helps to ensure proper functionality of deployed applications when using Fast TLM restart and HTTP discovery.

e. **`encoding`**: specifies encoding for the repository instance. If this element is not specified, the encoding for the admin server is used. If the admin server is not available, the default for this element is ISO8859-1.

All TIBCO components working in the same domain must always use the same encoding for intercommunication.

`Archives` (optional): form-data parameter that specifies a comma delimited list of archives that are to be deployed within the .zip file that are to be deployed. If an **`archives`** list is omitted, all archives in the application archive package are deployed. Example:

```
-F "Archives=Archive_A,Archive_B,Archive_X"
```

`ArchiveSettings` (optional): form-data parameter specifies settings for the archive.

a. **`enabled`**: `true` or `false`. Only enabled services are deployed. Disabling a service effectively undeploys just that service while letting all other services in the application run as normal. This can be useful when you wish to deploy an application that includes a service, for which you do not have the required software. A deployment configuration cURL expressions takes the form:

```
-F "ArchiveSettings.enabled=true"
```

b. **`av`**: Specify values for archive runtime variables with a comma-separated string with each key value pair joined by an equal (=) sign. For example:

```
-F "ArchiveSettings.av=Deployment=T2.HTTP_GET-Tomcat,Domain=Mine"
```

Refer to the appendix of the *TIBCO Runtime Agent Scripting Deployment User's Guide* for a full list of Archive Settings properties, parameters, descriptions, and usage. Not all elements and properties are supported for use by the TIBCO Silver Fabric Enabler for Generic Adapter . The following is a first attempt at listing them all.

c. **`hbInterval`**: heartbeat interval. The heartbeat interval determines the time (in milliseconds) between heartbeat messages.

d. **`activationInterval`**: activation interval. This field specifies the amount of time to expire since the last heartbeat from the master before the secondary restarts the process starters and process engines.

e. **`preparationDelay`**: preparation interval. This field is used to specify a delay before the master engine restarts.

f. **`ruleBases`**: rule bases for the archive

   `ruleBase.uri`: location of the rulebase file. Example syntax:

   ```
   -F "ArchiveSettings.ruleBases.ruleBase.uri=someValue"
   ```

   `ruleBase.data`: Rulebase content. Do not change this setting.

g. **`failureEvents.failureEvent`**:

   `failureType`: *ANY, FIRST, SECOND, Subsequent*

   If an event is used, type must be specified.

   Example syntax:

   ```
   -F "ArchiveSettings.failureEvents.failureEvent.failureType=ANY"
   ```

   `restart`: `true` or `false`. If `true`, the service instance is restarted upon failure.

   `description`: information that describes this operation.

   `alertAction.performPolicy`: the policy to perform. `Once`: generates an alert only for the first occurrence. `Always`: generates an alert for each occurrence.

   `alertAction.enabled`: `true` or `false`. If `true`, the action occurs when conditions for the action are `true`. If `false`, the action is not called.

   `alertAction.level`: *High, Medium, Low*

   `alertAction.message`: The message that displays when this alert is triggered.

h. **failureEvents.emailAction**:

performPolicy: the policy to perform. Once: generates an alert only for the first occurrence. Always: generates an alert for each occurrence.

enabled: true or false. If true, the action occurs when conditions for the action are true. If false, the action is not called.

message: the message to send.

to: a comma-separated list of email addresses to which the messages are sent.

cc: a comma-separated list of email addresses to which copies of the messages are sent.

subject: the subject of the email message.

sMTPServer: The mail server (SMTP server) to use to send the message. Specify the host name or the host IP address.

i. **failureEvent.customAction**:

performPolicy: the policy to perform. Once: generates an alert only for the first occurrence. Always: generates an alert for each occurrence.

enabled: true or false. If true, the action occurs when conditions for the action are true. If false, the action is not called.

command: specify the script to execute. Script files are highly recommended.

arguments: the list of arguments for the command

j. **suspendProcessEvents.suspendProcessEvent**:

restart: true or false. If true, the service instance is restarted upon failure.

description: information that describes this operation.

alertAction.performPolicy: the policy to perform. Once: generates an alert only for the first occurrence. Always: generates an alert for each occurrence.

alertAction.enabled: true or false. If true, the action occurs when conditions for the action are true. If false, the action is not called.

alertAction.level: High, Medium, Low

alertAction.message: the message that displays when this alert is triggered.

emailAction.performPolicy: the policy to perform. Once: generates an alert only for the first occurrence. Always: generates an alert for each occurrence.

`emailAction.enabled`: `true` or `false`. If `true`, the action occurs when conditions for the action are `true`. If `false`, the action is not called.

`emailAction.message`: the message to send.

`emailAction.to`: a comma-separated list of email addresses, to which the messages are sent.

`emailAction.cc`: a comma-separated list of email addresses, to which copies of the messages are sent.

`emailAction.subject`: the subject of the email message.

`emailAction.sMTPServer`: the mail server (SMTP server) to use to send the message. Specify the host name or the host IP address.

`customAction.performPolicy`: the policy to perform. `Once:` generates an alert only for the first occurrence. `Always:` generates an alert for each occurrence.

`customAction.enabled`: `true` or `false`. If `true`, the action occurs when conditions for the action are true. If `false`, the action is not called.

`customAction.command`: specify the script to execute. Script files are highly recommended.

`customAction.arguments`: the list of arguments for the command

k. **`logEvents.logEvent`**:

`restart`: `true` or `false`. If `true`, the service instance is restarted upon failure.

Example syntax:

```
-F "ArchiveSettings.logEvents.logEvent.restart=true"
```

`match`: The string in the log file to match.

`description`: information that describes this operation.

`alertAction.performPolicy`: the policy to perform. `Once:` generates an alert only for the first occurrence. `Always:` generates an alert for each occurrence.

`alertAction.enabled`: `true` or `false`. If `true`, the action occurs when conditions for the action are true. If `false`, the action is not called.

`alertAction.level`: *High, Medium, Low*

`alertAction.message`: The message that displays when this alert is triggered.

`emailAction.performPolicy`: the policy to perform. `Once:` generates an alert only for the first occurrence. `Always:` generates an alert for each

occurrence.

`emailAction.enabled`: `true` or `false`. If `true`, the action occurs when conditions for the action are `true`. If `false`, the action is not called.

`emailAction.message`: the message to send.

`emailAction.to`: a comma-separated list of email addresses, to which the messages are sent.

`emailAction.cc`: a comma-separated list of email addresses, to which copies of the messages are sent.

`emailAction.subject`: the subject of the email message

`emailAction.sMTPServer`: The mail server (SMTP server) to use to send the message. Specify the host name or the host IP address.

`customAction.performPolicy`: the policy to perform. `Once`: generates an alert only for the first occurrence. `Always`: generates an alert for each occurrence.

`customAction.enabled`: `true` or `false`. If `true`, the action occurs when conditions for the action are `true`. If `false`, the action is not called.

`customAction.command`: specify the script to execute. Script files are highly recommended.

`customAction.arguments`: the list of arguments for the command

l. **`failureCount`**: The value in this field defines how many restarts should be attempted before resetting the error counter to 0.

m. **`failureInterval`**: The value in this field defines how much time should expire before resetting the error counter to 0.

n. **`SFGAprocess`**:

`name`: the name of the process

`enabled`: `true` or `false`. Only enabled processes are deployed.

`maxjob`: specifies the maximum number of process instances that can concurrently be loaded into memory.

Note: `maxjob` may not apply to all adapters. Please refer to the TIBCO Adapter product documentation for details.

`activation`: the use activation limit

`flowlimit`: specifies the maximum number of currently running process instances to start before suspending the process starter.

Note: `flowlimit` may not apply to all adapters. Please refer to the TIBCO Adapter product documentation for details.

`InstanceSettings` - (optional) Some syntax examples:

```
-F "InstanceSettings.initHeapSize=64"
-F "InstanceSettings.maxHeapSize=512"
-F "InstanceSettings.threadStackSize=512"
```

a. **`description`**: specify any pertinent information about the binding.

b. **`contact`**: name the person responsible for this application instance.

c. **`startOnBoot`**: when `true`, the service instance starts when the computer is restarted. The default value is `false`.

d. **`enableVerbose`**: when `true`, sets the enabler for verbose tracking for service instances. The default value is `false`.

e. **`maxLogFileSize`**: sets the maximum size (in kilobytes) that a log file can reach before the engine switches to the next log file.

f. **`maxLogFileCount:`** specifies the maximum number of log files to use. When the maximum number of log files have been written, the engine begins writing to the first log file again.

g. **`threadCount`**: specifies the number of threads to use to execute process instances. The number of threads determines how many process instances can execute concurrently.

h. **`prepandClassPath`**: the values supplied here are prepended to your CLASSPATH environment variable.

i. **`appendClassPath`**: the items you supply here are appended to your CLASSPATH environment variable.

j. **`initHeapSize`**: specifies the initial size (in MB) for the JVM used for the process engine. The default is 32 MB.

k. **`maxHeapSize`**: specifies the maximum size (in MB) for the JVM used for the process engine. The default is 256 MB.

l. **`threadStackSize`**: specifies the size of the thread stack. The default is 256 KB.

m. **`runAsNT`**: when set to `true` the service is run as a Microsoft Windows Service. You can then manage the engine as you would any other service, and you can specify that it starts automatically when the machine reboots.

n. **`startUpType`**: specifies the instance service startup type as either: `Automatic`, `Manual`, or `Disabled`.

o. **`login`**: specifies the login account for the service, if any. The domain name must be specified as well.

p. **`password`**: sets the password for that service, if any.

q. **checkpoint**: when set to `true`, the process engine waits for all jobs to finish (up to the maximum timeout) before shutting down the engine, rather than remove jobs at their next checkpoint.

r. **timeout**: the maximum timeout in seconds the process engine will wait for jobs to finish before shutting down the engine. A zero `(0)` value means 0 seconds, which effectively turns the graceful shutdown into an immediate shutdown.

s. **iv** : this element uses a comma-separated string with name-value pairs with each key value pair joined by an equal (=) sign. For example:

configurationFile (optional): form-data parameter used to include an XML configuration file created to modify archive properties if needed. Example syntax:

```
-F "configurationFile=YourConfigurationfile.xml"
```

Where your XML configuration file should use the same format as an enabler or component level `configure.xml` file with the outermost XML element as follows:

```
<archiveConfig name="YourArchiveName">
   ...
</archiveConfig>
```

For more information on writing an archive configuration file, see the "Using the Silver Fabric SDK" chapter of *Silver Fabric Developer's Guide*.

ForceDeploy (optional): redeploy, forces a stop and overwrite of a pre-existing archive or set of archives with the same name. By default `ForceDeploy` is set to false and so a second deployment does not overwrite a pre-existing deployment of the same name. If there is a change of the archive file then `ForceDeploy` should be set to *true* so that the new application archive is redeployed. If `ForceDeploy` is used with `-F Archives` specifying a comma delimited list, then only those archives are stopped, undeployed, and redeployed.

```
-F "ForceDeploy=true"
```

FTWeight (optional): sets the relative FT (fault tolerance) weight for the enabler binding of the deployed archives. Adapter archives can be deployed on multiple machines to support fault tolerance and load balancing. Setting different `FTWeight` values for different application endpoints will require invoking the REST service at least twice with different deployment properties criteria, once for each endpoint.

VariableProvider (optional): specifies a variable provider to set global variables for applications deployed with REST. The variable provider is a Java Class extension compiled into a JAR and loaded into a Silver Fabric directory with an appropriate XML so that it may be called by REST during application deployment. It supports multiple VariableProviders at the same time. You can add multiple variable provider names with a comma separated list. If you use the global variable name in both variable providers, the latest in the list is used.

```
 -F "VariableProvider=Some_PROVIDER_Name"
```

Creating a variable provider for use by a REST call

1. Create a class that extends `com.datasynapse.fabric.broker.userartifact.variable.AbstractVariableProvider` and override the methods as needed. For example:

```
package com.tibco.sf.providers;
import java.util.Properties;
import
com.datasynapse.fabric.broker.userartifact.variable.AbstractVariab
leProvider;
public class My_Var_Provider extends AbstractVariableProvider
{
  private static final long serialVersionUID = 1L;
  @Override
  public Properties getVariables()
  {
    Properties p = new Properties();
    p.setProperty("MyApp/vars/name", "SomeString");
    p.setProperty("MyApp/vars/episodic", "true");
    p.setProperty("MyApp/vars/Xduration", "60");
    return p;
  }
  @Override
  public void destroy()
  {}
  @Override
  public void init() throws Exception
  {}
}
```

2. Export a JAR from it and save it to the TIBCO Silver Fabric Broker directory:

```
SILVERFABRIC_HOME/webapps/livecluster/deploy/config/variableProvid
ers
```

3. Create an XML file with the properties shown in this sample to associate the new class for TIBCO Silver Fabric.

```
<variableProvider class="com.tibco.sf.providers.My_Var_Provider">
   <property name="name" value="Some_PROVIDER_Name"/>
   <property name="description" value="GV values are in the JAR"/>
   <property name="enabled" value="true"/>
</variableProvider>
```

The variable provider name used by the REST statement is specified as "name" in the XML file.

Save the XML file in the same directory as the JAR:

```
SILVERFABRIC_HOME/webapps/livecluster/deploy/config/variableProvid
ers
```

You can verify what variable providers are ready for use by REST invocation on the TIBCO Silver Fabric Administrator > Admin > Variables page.

`NoStart -` (optional) default value is false, meaning that the archives are both deployed and started by default. If `NoStart` is set to true, the application is deployed but not started.

```
-F "NoStart=true"
```

Deployment File

When deploying an archive by REST you must include a deployment file, which specifies at least one selection criteria for determining which engine and component receives the deployed archive. The deployment file is a simple properties text file specified by a form-data field like the following for REST upload with the archive:

```
-F "deploymentFile=@YourDeploymentFileName.properties"
```

The deployment file contains one or more logical statements with a criteria, a comparator, and a value, delimited by spaces:

`criteria comparator value`

Some criteria require an argument to be specified in parentheses:

`criteria(argument) comparator value`

For example, what follows is a simple statement to deploy an archive to an engine running a component with a component type that contains *Adapter* as part of the name **and** which has a domain name of *YourDomain*:

```
ComponentType contains Adapter
ImportedVariable(TIBCO_DOMAIN_NAME) = YourDomain
```

By default **all** deployment file criteria statements must be satisfied for the deployment to occur, but you can change how the properties file criteria are evaluated to make them logical **`OR`** statements by using the optional cURL form-data switch: `-F "LogicalAnd=false"`

Table 3 Supported Criteria

Name	Definition
`ComponentName`	The name of the component.
`ComponentType`	The type of the component.
`EnablerName`	The name of the Enabler running the component.
`EnablerVersion`	The version of the Enabler running the component.
`Account`	The name of the account running the component.

`EngineProperty(`*name*`)`	The following named engine properties can be used:
	`Engine Id`
	`Engine GUID`
	`Engine Instance`
	`IP Address`
	`Host Name`
	`Number of CPUs`
	`Total CPU Processing Power`
	`Total Memory (KB)`
	`Free Memory (KB)`
	`Free Disk Space (MB)`
	`OS Platform`
	`OS Version`
	`OS Username`
	`Location`
	`vimTemplate`
	`Group`
	`Description`

`ActivationInfoProperty(`*name*`)`	A named property, such as `ClusterName`, or `HTTP_STATIC_ROUTE_PREFIX` within the component's `ActivationInfo` object. Archives that can be scaled elastically keep a list of `ActivationInfo` properties and their respective values for failover Broker.
`ImportedVariable(`*name*`)`	A variable imported into a component.
`ExportedVariable(`*name*`)`	A variable exported from a component.
`DependencyComponent`	A named dependency on a component.
`DependencyEngine(`*componentName*`)`	A named dependency on an engine running the named component.

Comparator Valid comparators include `=`, `!=`, `>`, `<`, `<=`, `>=`, `matches`, `contains`, `!matches`, and `!contains`.

Example 2 A sample deployment file

```
# Sample deployment file

ComponentType = "TIBCO Generic Adapter"

ActivationInfoProperty(ClusterName) matches dev_cluster.*
```

Successful Deployment

REST returns a Status of 200 (OK) when the archive is successfully deployed. A successful REST execution returns the `Engine-Instance` and `Engine-Id` where the archive deployed. Example response (application/JSON):

```
{
    "result": {
        "name": "Archive Deployment",
        "value": {
            "message": "ENGINE '[1885509966828815621-5 :
my_archive.ear]' DEPLOYED",
            "Engine-Instance": "5",
            "Engine-Id": "1885509966828815621"
        }
    },
    "status": 200
}
```

Knowing the `Engine-Id, Engine-Instance`, and the full `ArchiveName` allows for invocation of START, STOP, and UNDEPLOY methods to enable full control of the Archive life cycle.

An unsuccessful deployment returns an error of Status 500 or some other appropriate error status depending on the cause.

Continuous Deployment Timeout

Continuous deployment transfers can timeout due to one or more of the following factors:

- Large archive size
- A slow network or high latency
- Long start-up time for archives

If you are encountering timeout issues, you can set a higher socket timeout between the broker and engines. This can be set in the Silver Fabric Administration Tool at **Config > Broker > Communications** under the section **HTTP Connections > Engines**. The Socket Timeout parameter configures the HTTP connections established from brokers to clients and engines. Set the timeout value to the longest of the following three:

- The longest scaling archive download time from the broker to engine

- The longest starting or stopping time for an archive
- The longest undeploy time

Files larger than 1GB should not be deployed using continuous deployment.

Start

Using REST you can also start application archives that are deployed but not started. You must know the `Engine-Id`, `Engine-Instance`, and the full application `ArchiveName`.

Example 3 Here is a generic cURL example that starts an Archive.

```
curl -u UserName:Password \
     -X POST \
     -H "Accept:application/json" \
     -H "Content-type: multipart/form-data" \
     -v "http://<YourSFBroker.com>:<port>/livecluster/rest/v1/sf/e
ngines/{engine-id}/{engine-instance}/archives/{appname}/start"
[-F "Version={VersionNumber}"] #supported only for 6.x component
```

Expressions in this generic form cURL expression example were separated by line breaks to help with readability, but normally an unbroken string is submitted when executed.

The value of {***engine-instance}*** can be obtained from the response to the successful cURL deployment REST execution or the TIBCO Silver Fabric Administrator > **Engines** page > expanding the row to see Engine details.

The Start and Stop both have an optional option to specify the version number. If the version number is not specified, an attempt to complete the action is made on the deployed application with the same name.

There are also Silver Fabric REST methods to GET the engine-id and engine-instance for all instances running on a daemon. Refer to TIBCO Silver Fabric REST Services documented in the *TIBCO Silver Fabric Administration Tool* at **Admin > REST Services**.

`Version -` (optional) The version form-data field (`-F`) lets you specify the version to be started or stopped. If not specified, the version number is obtained from the deployed application that has the same application name.

Example 4 Another cURL example that starts an archive

```
curl -u admin:admin
     -X POST
     -H "Accept:application/json"
     -H "Content-Type:multipart/form-data" http://lin64vm205.qa.da
tasynapse.com:8080/livecluster/rest/v1/sf/engines/5103224222683646
```

```
426/3/archives/tibco.sfga.sample.binding.soap.http.BookStore.appli
cation/start
     -F "Version=1.0"
```

The value of **{*appname*}** is usually the application name or the full application archive name.

When the **{*appname*}** is used, it is URL-encoded in a cURL statement so that spaces are converted to "**%20**" and forward slashes (**/**) are represented by "**%2F**". Likewise, other special characters must be encoded appropriately.

Example 5 A cURL example (URL encoded) that starts an archive.

The full archive name shown had to be URL encoded:

```
/Silver Fabric/SFGA 22E5/DomainMachineName/processOrder/Orders/MyP
rocOrder/Process_Archive.par
```

...to be passed in the cURL statement shown here:

```
curl -u admin:admin -X POST -H "Accept:application/json" -H
"Content-Type:multipart/form-data" -v
"http://192.168.72.189:8080/livecluster/rest/v1/sf/engines/8735490
097077982598/1/archives/tibco.sfga.sample.palette.http.PersistentC
onnection.application/start" -F "Version=1.0"

curl -u admin:admin -X POST -H "Accept:application/json" -H
"Content-Type: multipart/form-data" -v
"http://lin64vm121.qa.datasynapse.com:8080/livecluster/rest/v1/sf/
engines/4649861604167227205/1/archives/%2FSilver%20Fabric%2FSFGA%2
022E5%2FDomainMachineName%2FprocessOrder%2FOrders%2FMyProcOrder/Pr
ocess_Archive.par/start"
```

Stop

You can stop application archives that are running on an engine by REST method by submitting a cURL expression to the Silver Fabric Broker. You must know the `Engine-Id`, `Engine-Instance`, and the full application `ArchiveName`.

```
curl -u UserName:Password
     -X POST
     -H "Accept:application/json"
     -H "Content-type: multipart/form-data" \
     -v "http://<YourSFBroker.com>:<port>/livecluster/rest/v1/sf/e
ngines/{engine-id}/{engine-instance}/archives/{appname}/{archive-id}/stop"
```

`Version -` (optional) The version form-data field (`-F`) lets you specify the version to be started or stopped. If not specified, the version number is obtained from the deployed application that has the same application name.

Example 6 A cURL example that stops an Archive:

```
curl -u admin:admin
```

```
     -X POST -H "Accept:application/json"
     -H "Content-Type:multipart/form-data" http://lin64vm205.qa.da
tasynapse.com:8080/livecluster/rest/v1/sf/engines/5103224222683646
426/3/archives/tibco.sfga.sample.binding.soap.http.BookStore.appli
cation/tibco.sfga.sample.binding.soap.http.BookStore.application/s
top
```

Refer to the `Start` method for a description on how to obtain the values of ***{engine-id}***, ***{engine-instance}***, and **{*appname*}**.

Undeploy

You can undeploy application archives that are running on an engine by REST method by submitting a cURL expression to the Silver Fabric Broker. You must know the `Engine-Id`, `Engine-Instance`, and the full application `ArchiveName`.

```
curl -u UserName:Password
     -X POST
     -H "Accept:application/json"
     -H "Content-type: multipart/form-data" \
     -v "http://<YourSFBroker.com>:<port>/livecluster/rest/v1/sf/e
ngines/{engine-id}/{engine-instance}/archives/{appname}/undeploy"
```

Refer to the `Start` method for a description on how to obtain the values of ***{engine-id}***, ***{engine-instance}***, and **{*appname*}**.

`DeleteApp` - (optional - use with undeploy only) The default value is false and it can be omitted. When the `DeleteApp` parameter is false, an undeploy archive action leaves the application configurations of global variables and bindings so that they can be used again. The archive and the application are only undeployed and not deleted.

> Setting `DeleteApp` to true deletes the application and the associated variable settings from the runtime engine after the archive is undeployed.

Example 7 A cURL example of an undeploy call

```
curl -u admin:admin
     -X POST
     -H "Accept:application/json"
     -H "Content-Type:multipart/form-data"
     -v "http://lin64vm205.qa.da
tasynapse.com:8080/livecluster/rest/v1/sf/engines/5103224222683646
426/3/archives/tibco.sfga.sample.binding.soap.http.BookStore.appli
ca
tion/undeploy"
```

Here the application name is used and there is no need for archiveid.

For more information on REST methods, including parameters and return responses, see the online REST help in the Silver Fabric Administration Tool. For information on using REST services, see "Using REST Services" in the *Silver Fabric Developer's Guide*.

The Archive Management Support Feature

When components are activated, by default all archives are deployed in order, and then started in order. This behavior can be changed by editing the component in the component wizard, and editing the Archive Management Support feature, which has an option that can be cleared to prevent archives from starting when the component is activated.

Ant Scripts

In addition to the Administration Tool, Command Line Interface tool, and REST interface, you can also configure Components and Stacks using a set of Silver Fabric Ant tasks, which are provided for Apache Ant. This alternative provides a method more granular than the CLI which can be easier to use in some configuration scenarios, and also provides an easy way to configure Silver Fabric from within an IDE with a built-in Ant support.

For more details on installation and tasks, refer to chapter 10 - Silver Fabric Ant Scripts in *TIBCO Silver Fabric Developer's Guide.*

Samples

The Silver Fabric Command Line Interface installation contains a samples directory, which contains example build files using Ant tasks for several Enablers. See the enclosed readme for more information on the examples.

You can also create a sample build.xml and build.properties based on any of your published Stacks. See "Packaging Stacks For Ant Task Deployment" in the *TIBCO Silver Fabric Cloud Administrator's Guide* for more information.

You can use the `fabric-cli.properties` file from the Silver Fabric installation along with the following files:

- build.xml
- build.properties

build.xml

Here is the sample build.xml file.

```
<project name="sfgabuild" default="release" basedir="." xmlns:sf="antlib:com.datasynapse.fabric.ant">

    <property file="build.properties"/>
    <property file="../../fabric-cli.properties" />

    <sf:connection-props brokerurl="http://lin64cdc201.qa.datasynapse.com:8000" username="admin" password="admin"
clientssltrustfile="${DSSSLTrustFile}" />

    <target name="release" depends="release-component, release-stack"/>
    <target name="clean" depends="clean-stack, clean-component"/>
    <target name="release-component">
        <echo message="Building ${component.name}" />
        <sf:component action="create" name="${component.name}" description="${component.description}"
```

```
type="${component.type}" enablername="${enabler.name}" enablerversion="${enabler.version}" utility="${utility}" />

    <sf:option name="${component.name}" action="replace">
      <sf:property name="Department" value="${department}" />
      <sf:property name="Location" value="${location}" />
      <sf:property name="Partition" value="${partition}" />
      <sf:property name="Engine Blacklisting" value="false" />
      <sf:property name="Failures Per Day Before Blacklist" value="0" />
      <sf:property name="Archive Scale Up Timeout" value="${archive.scale.up.timeout}" />
      <sf:property name="Archive Scale Down Timeout" value="${archive.scale.down.timeout}" />
      <sf:property name="Maximum Deactivation Time" value="${deactivation.timeout}" />
      <sf:property name="Maximum Activation Time" value="${activation.timeout}" />
      <sf:property name="Maximum Capture Time" value="${maximum.capture.time}" />
      <sf:property name="Maximum Instances Per Host" value="${max.instances.per.host}" />
      <sf:property name="Separator Tags" value="${separator.tags}" />

      <sf:property name="Activation Delay" value="0" />
      <sf:property name="Engine Reservation Expiration" value="${engine.reservation.expiration}" />
    </sf:option>

    <sf:default-settings name="${component.name}" action="update">
      <sf:property name="Default Min Engines" value="${min}" />
      <sf:property name="Default Max Engines" value="${max}" />
      <sf:property name="Default Priority" value="${priority}" />
    </sf:default-settings>

    <sf:feature name="${component.name}" action="add" feature="Application Logging Support" />
    <sf:feature name="${component.name}" action="update" feature="Application Logging Support">
      <sf:property name="Archive Application Logs" value="${archive.logs}" />
      <sf:property name="Checkpoint Frequency In Seconds" value="${checkpoint.frequency}" />
      <sf:property name="Log File Pattern" value="${log.file.pattern}" />
    </sf:feature>

    <sf:feature name="${component.name}" action="add" feature="HTTP Support" />
    <sf:feature name="${component.name}" action="update" feature="HTTP Support">
      <sf:property name="HTTP Enabled" value="${http.enabled}" />
      <sf:property name="HTTPS Enabled" value="${https.enabled}" />
      <sf:property name="Routing Prefix" value="${routing.prefix}" />
      <sf:property name="Route Directly To Endpoints" value="${routing.direct}" />
    </sf:feature>

    <sf:feature name="${component.name}" action="add" feature="Archive Management Support" />
    <sf:feature name="${component.name}" action="update" feature="Archive Management Support">
      <sf:property name="Start Archives On Activation" value="true" />
    </sf:feature>
```

```
    <sf:publish type="component" name="${component.name}" />
  </target>

  <target name="clean-component">
    <echo message="Cleaning ${component.name}" />
    <sf:unpublish type="component" name="${component.name}" onerror="ignore" />
    <sf:remove type="component" name="${component.name}" onerror="ignore" />
  </target>

  <target name="release-stack">
    <echo message="Building ${stack.name}" />
    <sf:stack action="create" name="${stack.name}" description="${stack.description}"/>
    <sf:stack-component action="add" name="${stack.name}" components="${component.name}" />
    <sf:stack-policy action="update" name="${stack.name}">
      <sf:policy manualpolicy="true">
        <sf:compalloc component="${component.name}" min="${min}" max="${max}" priority="${priority}">
          <sf:allocationrule type="Resource Preference">
            <sf:property name="propertyName" value="os" />
            <sf:property name="propertyValue" value="linux" />
            <sf:property name="operator" value="contains" />
            <sf:property name="affinityPos" value="2" />
          </sf:allocationrule>
        </sf:compalloc>
      </sf:policy>
    </sf:stack-policy>

    <sf:publish type="stack" name="${stack.name}" />
  </target>

  <target name="clean-stack">
    <echo message="Cleaning ${stack.name}" />
    <sf:unpublish type="stack" name="${stack.name}" onerror="ignore" />
    <sf:remove type="stack" name="${stack.name}" onerror="ignore" />
  </target>
</project>
```

build.properties

Here is the sample of build.properties file.

```
# stack properties
stack.name=Example Stack for ${component.name}
stack.description=Example Stack for ${component.name} built by Ant
```

```
# component properties
component.name=SFGA Example Component
component.description=SFGA Example Component built by Ant
component.type=TIBCO Generic Adapter:1.0.0
enabler.name=TIBCO Generic Adapter container
enabler.version=3.1.0.0
utility=false

# load balancing
routing.direct=false
routing.prefix=${cluster.name}

# logging
archive.logs=true
log.file.pattern=\
/../domaindata/logs/domainutility.log,\
/../domaindata/tra/${TIBCO_DOMAIN_NAME}/logs/Hawk.log,\
/../domaindata/tra/${TIBCO_DOMAIN_NAME}/logs/tsm.log,\
/../domaindata/tra/${TIBCO_DOMAIN_NAME}/logs/msghma.log,\
/../domaindata/tra/${TIBCO_DOMAIN_NAME}/logs/ApplicationManagement.log,\
/../domaindata/tra/${TIBCO_DOMAIN_NAME}/application/logs/.*\.log
checkpoint.frequency=300

# instances
max.instances.per.host=1
min=1
max=4
priority=3

# http
http.enabled=true
https.enabled=true

# build directories
content.dir=content
configure.dir=configure
script.path=

# timeouts
archive.scale.up.timeout=120
archive.scale.down.timeout=120
activation.timeout=300
deactivation.timeout=120
maximum.capture.time=300
stats.collection.frequency=120
```

engine.reservation.expiration=300

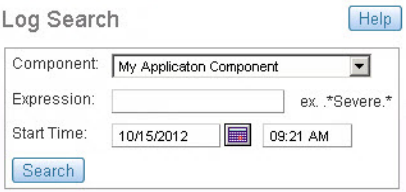
 # options
department=
location=
partition=
separator.tags=

Log Files

You can retrieve some Adapter log files from the TIBCO Silver® Fabric Administration Tool. To do so, follow these steps:

1. In TIBCO Silver® Fabric Administration Tool, select **Engines > Log Search**.
2. Select the component, from which you want to see the log files, as shown in Figure 16.
3. Optionally you can search for a regular expression using the **Expression** field.
4. Select the Start Time to see the logs since that time.

Figure 16 Log Files



Retained Log Files

The log files generated for components that use TIBCO Administrator contains the following log files:

- TIBCO Hawk Agent Log Files, page 52
- TIBCO Administrator Log Files, page 53
- Adapter Log Files, page 54

TIBCO Hawk Agent Log Files

When TIBCO Administrator runs in the Fault Tolerant mode, all files are not located under the TIBCO Silver® Fabric *$ENGINE_WORK_DIR* directory. The Hawk® log files do not appear in the TIBCO Silver® Fabric Administrator GUI.

For both TIBCO Administrator components and Adapter components, TIBCO Silver Fabric Enabler for Generic Adapter uses TIBCO Hawk® and TIBCO Hawk® Agent. Table 4 lists the retained TIBCO Hawk® log files.

Table 4 Retained TIBCO Hawk® Agent Log Files

Name	Location	Purpose of the Log
`Hawk.log`	*`$ENGINE_WORK_DIR`*`/domaindata/tra/`*`TIBCO_DOMAIN`*`/logs`	Log file of the Hawk call in the Hawk® Agent.
`tsm.log`	*`$ENGINE_WORK_DIR`*`/domaindata/tra/`*`TIBCO_DOMAIN`*`/logs`	Log file of the Hawk® Agent.
`msghma.log`	*`$ENGINE_WORK_DIR`*`/domaindata/tra/`*`TIBCO_DOMAIN`*`/logs`	Log file for `tibhawkhma`.

TIBCO Administrator Log Files

When TIBCO Administrator runs in the Fault Tolerant mode, all files are not located under the TIBCO Silver® Fabric *`$ENGINE_WORK_DIR`* directory. The TIBCO Administrator log files do not appear in the TIBCO Silver® Fabric Administration Tool.

Table 5 lists the retained TIBCO Administrator log files.

Table 5 Retained TIBCO Administrator Log Files

Name	Location	Purpose of the Log
`audit.log`	*`$ENGINE_WORK_DIR`*`/domaindata/admin/`*`TIBCO_DOMAIN`*`/logs`	All information about TIBCO Administrator activities.
`tomcat.log`	*`$ENGINE_WORK_DIR`*`/domaindata/admin/`*`TIBCO_DOMAIN`*`/tomcat/logs`	Technical log of TIBCO Administrator that runs on Tomcat.
`domainutility.log`	`$ENGINE_WORK_DIR/tibco/tra/tra_version_2digits/logs`	Log file of the `domainUtilitty` command used to create the domain or add the machine. This file is common for all Engines.

Adapter Log Files

Table 6 lists the retained Adapter log files.

Table 6 Retained TIBCO Adapter Log Files

Name	Location	Purpose of the Log
`ApplicationManagement.log`	*`$ENGINE_WORK_DIR`*`/domaindata/tra/`*`TIBCO_DOMAIN`*`/logs`	Generated by `Appmanage`, which publishes Adapter Applications.
`domainutility.log`	*`$ENGINE_WORK_DIR`*`/tibcoga/tra/`*`tra_version_2digits`*`/logs`	Log file of the `domainUtilitty` command used to create a domain or add a machine. This file is common for all Engines.
Adapter Application Name, for example, `OrderConsolidation-Order_Consolidation.log`	*`$ENGINE_WORK_DIR`*`/domaindata/tra/`*`TIBCO_DOMAIN`*`/application/logs`	These are the most important log files, as they are the log files from Adapter and trace all activities that have happened.

HTTP Virtual Router and Load Balancer

In a private cloud, when you run a Adapter or TIBCO Administrator instance for 5.x component, you do not know in advance the address of the machine where it will run unless you set resource preferences in the rules.

For better interaction between Adapter activities, an HTTP URL (a web service using SOAP over HTTP transport or an HTTP Receiver activity) is required. The HTTP Load Balancer, an HTTP redirector, fulfills this requirement.

HTTP Load Balancer for Adapter Component

The URL to access or invoke the Adapter archive endpoint (a web service using SOAP over HTTP transport or an HTTP Receiver activity) uses the form:

http://{BrokerMachineName}:{BrokerPort}/{TIBCO_ADMIN_DOMAIN}/{GA-APPLICATION-PATH.NAME}/{GA_HttpPortName}

Where the URL is composed of the following parts:

- ***{BrokerMachineName}*** - The Machine name or IP Address where you installed the virtual router. The default is the broker machine.
- ***{BrokerPort}*** – Typically this is the port of the Silver® Fabric Administrator GUI. The default value is **8080**.
- ***{TIBCO_ADMIN_DOMAIN}*** – The value of the Domain name you entered when you configured TIBCO Administrator component.
- ***{GA-APPLICATION-PATH.NAME}*** – The deployment directory and name of the Adapter application with folders and name delimited by periods. For example if the Adapter application were named *AppName* and was deployed in the TIBCO Admin directory: `aaa/bbb/ccc/`
  then the ***{GA-APPLICATION-PATH.NAME}*** would be: *aaa.bbb.ccc.AppName*

  The ***{GA-APPLICATION-PATH.NAME}*** may be copied from the **TIBCO Silver Fabric Administrator > Dashboard > Scaled Archives** page.
- ***{GA_HttpPortName}*** – This global variable, endpoint URI, is defined at archive design-time as the HTTP port created to accept requests. Using TIBCO Designer create the variable (the name is arbitrary) in the global variable group and it should be set as modifiable at the service level to allow the value to be changed at runtime.
  - — If deployed from TIBCO Administrator UI or AppManage, it is the default value set in TIBCO Designer,
  - — If deployed from GA Enabler Component (REST, .ear file uploaded) , it is the HTTP port calculated to avoid port conflict.

The base port value for HTTP request activity and SOAP/HTTP Web service activity may be changed at the component level as was shown previously.

Figure 17 HTTP Port Management

For example:

`http://MyBrokerMachine(IP)Name:8080/MyDomain/{SFGA-APPLICATION-PATH}/MyArchiveName.par/`HTTPVariables/HTTP_Port

Index

Symbols

A

C

D

E

F

H

I

J

L

M

N

O

R

S

T

U

V

Z