TIBCO Silver® Fabric Enabler for Adapter for Files (Unix/Win) User's Guide

Software Release 3.0.0
June 2016

Two-Second Advantage®

TIBCO®

Contents

Figures

Introduction

TIBCO Silver Fabric Enabler for Adapter for Files is a complementary software component that enables TIBCO ActiveMatrix File Adapter projects to be published in cloud environments based on TIBCO Silver Fabric and leveraging Silver Fabric capabilities.

This accelerates publishing of ActiveMatrix File Adapter projects, enforces its industry best practices, and provides elastic optimization of computing resources.

By default TIBCO Silver Fabric Enabler for Adapter for Files supports all the features of TIBCO ActiveMatrix® Adapter for Files (Unix/Win) once deployed in a TIBCO Silver® Fabric environment:

- File 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 TIBCO ActiveMatrix® Adapter for Files refer to the documentation for that product.

Main Functionalities

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

- It allows you to quickly set up a TIBCO Domain environment on multiple machines. TIBCO Administrator can then publish File 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 ActiveMatrix File 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 File 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 collects File Adapter metrics from File Adapter engines to be exploited in Silver Fabric rules. It scales up and down the number of File Adapter engines required to process the workload. Therefore, it provides elasticity and optimization of computing resources.
- Supports Archive Scaling - Gathers and reports statistics from File Adapter service instances deployed by TIBCO Silver® Fabric to support automated rule-based scaling from archive statistics.

Components

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

- TIBCO ® Adapter for Files Enabler

  It is used to configure, start, and manage File Adapter engines and its published archives.

- A set of distributions that includes all the software pieces required to run a a TIBCO ® Adapter for Files environment: TIBCO Runtime Agent™, TIBCO Administrator™, TIBCO Rendezvous ® , TIBCO Hawk ® , and TIBCO ActiveMatrix ® Adapter for Files.

Enabler for Adapter for Files

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

A TIBCO Silver Fabric Enabler for Adapter for Files Stack is an entity that runs inside TIBCO Silver® Fabric. It executes all the components necessary to publish the TIBCO® Adapter for Files platform and to publish TIBCO® Adapter for Files projects.

A TIBCO Silver Fabric Enabler for Adapter for Files Stack consists of a TIBCO Administrator component, and one or more TIBCO® Adapter for Files components.

Prerequisite for creating the TIBCO Adapter for Files 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 User's Guide* for creating and configuring a TIBCO Administrator component.

To build and run a TIBCO Silver Fabric Enabler for Adapter for Files components , you must perform the following tasks:

- Create and publish one TIBCO Administrator component. Refer to *TIBCO Silver Fabric Enabler for TIBCO Administrator User's Guide* for creating and configuring a TIBCO Administrator component.
- Create and publish one or more Adapter for Files Components. Refer to Creating ActiveMatrix Adapter for Files Components.
- Create a TIBCO Silver Fabric Enabler for Adapter for Files Stack. Refer to Creating a TIBCO Silver Fabric Enabler for Adapter for Files Stack.
- Set a dependency to the TIBCO Administrator Component for each Adapter for Files Components. Refer to Setting Dependency Requirements.
- Optionally you may set rules for TIBCO Silver Fabric Engine to publish based on engine statistics.

After completing these tasks, you can run and when necessary update a TIBCO Silver Fabric Enabler for Adapter for Files Stack. Refer section Running TIBCO Silver Fabric Enabler for Stacks for information on how to run and update a stack.

Creating ActiveMatrix Adapter for Files Components

This Component will perform the following tasks:

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

If you do not upload the ActiveMatrix Adapter for Files projects, you still can publish the Adapter for Files projects on the machine where Adapter for Files is registered to the TIBCO Domain using the TIBCO Administrator GUI or with the REST service.

To configure a Adapter for Files Component, perform these tasks:

Procedure

1. Specifying the Component Type and Name
2. Selecting the Software Product Versions
3. Specifying the Basic Configurations

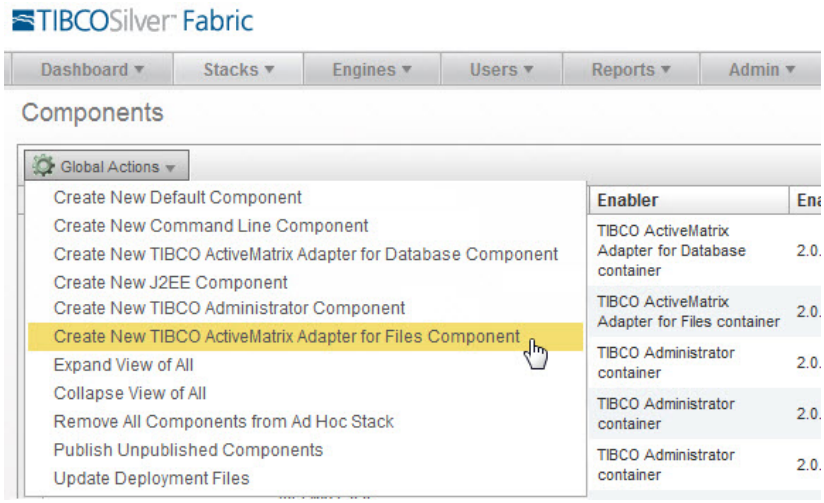
4. Uploading an Adapter for Files Project
5. Configuring Hawk Application Management Interface (AMI) Configuration (optional)
6. Configuring TIBCO Hawk Agent Running Condition
7. Adding or Editing Enabler-Specific Runtime Context Variables
8. Uploading, Ading, Customizing, or Removing a Content File
9. Removing or Adding Adapter for Files Component Statistics
10. Completing the Configuration of the Component

Specifying the Component Type and Name

Procedure

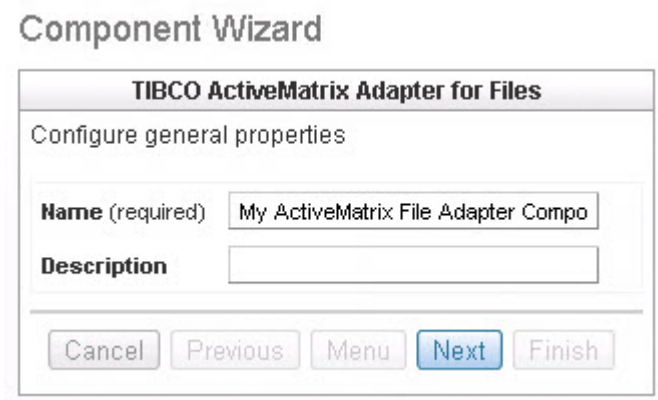
1. In TIBCO Silver Fabric Administrator GUI, select **Stacks** > **Components** .
2. In the Components page, select **Create New TIBCO ActiveMatrix Adapters for Files Component** in the Global Actions drop-down list.

   Creating a new ActiveMatrix Adapter for Files Component



3. Enter name for the Component.

   Name and describe the Component.



Selecting the Software Product Versions

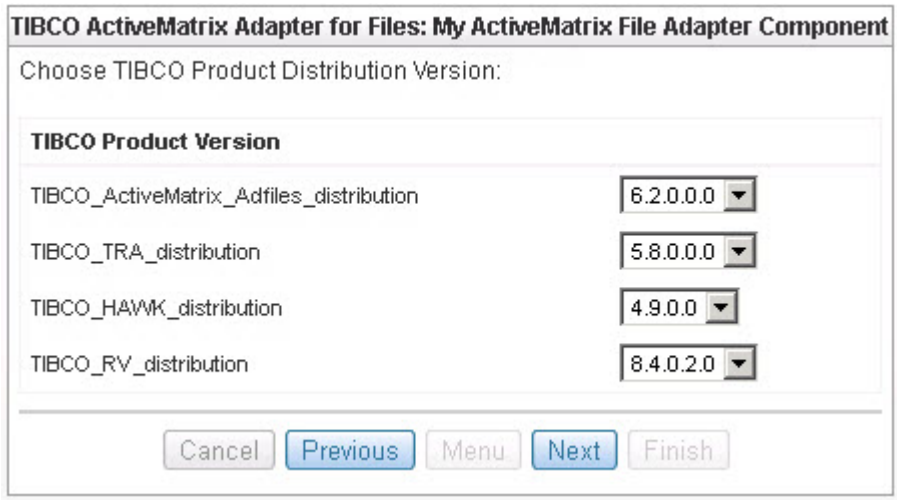
Procedure

1. You can select the versions of the distributions you want TIBCO Adapter for Files to run, as shown in Figure 19.

   You can select any versions of the installed distributions, but not all distributions are 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. Please refer to the respective product readme files for specific information on which operating systems the product distribution will support.

   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.

   Software Version

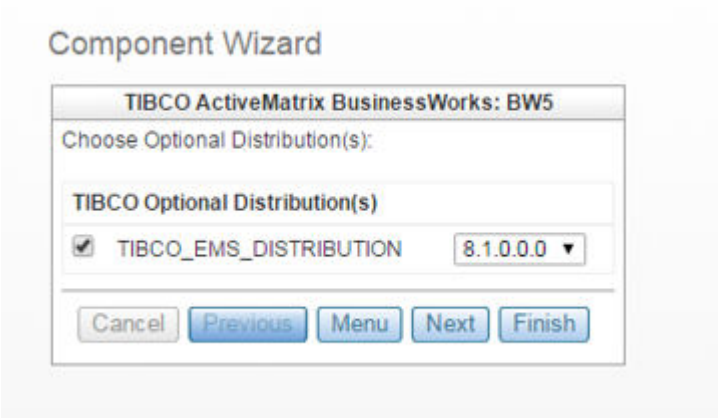


There is no EMS client embedded in TIBCO Runtime Agent since version 5.9.x. If you use EMS as a transport, you must install the EMS distribution and then select an EMS distribution version in the optional dependency screen.

You must choose Optional Distribution(s) only if your TIBCO Administrator implementation uses TIBCO Enterprise Messaging Service™ (EMS) or the deployed Adapter for Files project uses EMS. You must then upload the latest TIBCO EMS distribution to the TIBCO Silver Fabric Broker.

2. Select the optional distribution and click **Next**.

   Optional Distribution



Specifying the Basic Configuration

Each ActiveMatrix Adapter for Files Component has dependency on an instance of TIBCO Administrator.

Prerequisites

Procedure

1. **Upload a JDBC Driver (optional**).
   Specify a JDBC driver to publish with the TIBCO® Adapter for Files 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. If the Domain is not stored in a database, then don't upload a JDBC driver.

   Each Adapter for Files component is dependent on TIBCO Administrator. Refer to Setting Dependency Requirements for details.

2. Enter a value in the **TIBCO Domain machine name** field to activate use of the TIBCO Domain Machine Name. The **TIBCO Domain machine name** virtualizes the machine so that the published Components can maintain state when the targeted engine is changed for whatever reason.

   The TIBCO Domain Machine Name (which was formerly known as the TIBCO Logical Machine or TLM) provides for a singular publishing of a unique component.

   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. The component .ear (.JAR) files are republished and started on the new virtual machine hardware. When Adapter for File Applications have been previously published through TIBCO Administrator (or through the AppManage commandline interface) those previous deployments are also published again and restarted on the new hardware.

   The machine name is a virtualized name and must be unique. The TIBCO Domain machine name can use alphanumeric characters, hyphen (-), or underscore (_) characters. Do not use other special characters, including period and comma. The name length must be less than 64 characters.

   When the component instantiates multiple times, the TLM machine is named as follows: `<Your_TLM_Name>_<ComponentInstanceNumber>` , where the <ComponentInstanceNumber> is a sequential incrementing integer. When the component instance number is 0, the TLM machine name should be <Your_TLM_Name>.

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

3. **TIBCO Service state after TLM restarted** - Sets the desired services state when the TIBCO Logical Machine is restarted.
   When the TLM is restarted, service instances will be republished and either started or stopped. The stopped services state setting may be convenient for developers who are testing machines with many services that don't need to be started for every logical machine change.

4. **Specify Drive For Fast TLM Restart Configurations**
   Fast TIBCO Logical Machine Restart provides for accelerated restart of the engine so that many archives can be restarted 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. 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.

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

Domain configuration changes made in the period between the TLM stop and the TLM restart are not captured.

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

Recommendation: For those implementations that use fewer than ten Enabler for Adapter for Files deployments per component, avoid using Fast TLM restart to avoid the constraints mentioned in the preceding section. 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.

Fast TLM Restart mode is **not** fully compatible with Windows operating systems because Windows services don't have access to shared drives. It is strongly recommended that implementations requiring Fast TLM Restart mode target operating systems other than Windows based resources. This can be accomplished by defining a *Resource Preference - Allocation Rule* to specify the `OS Platform` as equal to a preferred operating system. If Windows platforms must be used then the Silver Fabric Engines can be run as applications by the desktop user so that the Silver Fabric Engine and child processes can use mapped drives.

5. Select **Delete Application Configuration at Shutdown** to remove all the existing configurations.

   Select this option to undeploy and remove all the applications deployed on the TIBCO Logical Machine at component shutdown.

   If this check box is checked, enabler deletes all the applications that are deployed on the administrator when the component is restarted. The EAR application attached with component is deployed again, and all the previous data will be lost. So if you check this check box and deploy applications through REST call, the applications will be deleted and all the data will be lost after the component is restarted.

   Use this feature with extreme caution. Do not use this feature if you are not familiar with it.

6. Select **Do Not Redeploy Existing EAR File at Startup** to prevent redepoyment of existing EAR files. Select this option to avoid redeployment of the EAR file whenever the TIBCO Logical Machine restarts.

   You must check this check box after you remove EAR applications from the component, and if you do not want these applications to run again. Use this feature with extreme caution. Do not use this feature if you are not familiar with it.

7. Select **Force Kill of Adapter for Files Process by Engine Daemon on Shutdown** If the regular stop is not succesfull.

   Select this option to forcefully kill the Adapter for Files processes at shutdown. Even though the adapter for Files engine is always stopped at shutdown, in case the Hawk is down, the adapter for Files engine stops abruptly resulting in orphan engines. To avoid such orphan engines, the engine daemon kills the adfilesagent forcefully.

Use this option only if regular stop does not work. It kills all the TIBCO Adapter for Files processes on shutdown and stops all the processes that were started by the engine daemon.

8. Select **Add the JAR file(s) in front of the Adapter for Files ClassPath (check), otherwise at the end of the ClassPath (uncheck)** as appropriate. If checked, the JAR file name will be prepended in front of the ClassPath and if it is left unchecked, the JAR file name is appended at the end of the ClassPath
   Add external JAR file(s) to the Adapter for Files Component: As an option, JAR files can be uploaded for publishing with the TIBCO Silver Fabric Enabler for Adapter for Files component. Of course, this setting does not apply if a JAR file is not uploaded for addition to the ClassPath.
9. **Upload a JRA file or JAR files in a ZIP file to the Adapter for Files 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 check box 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 might be removed with that wizard page.
10. Click **Next**.

Uploading an Adapter for Files Project

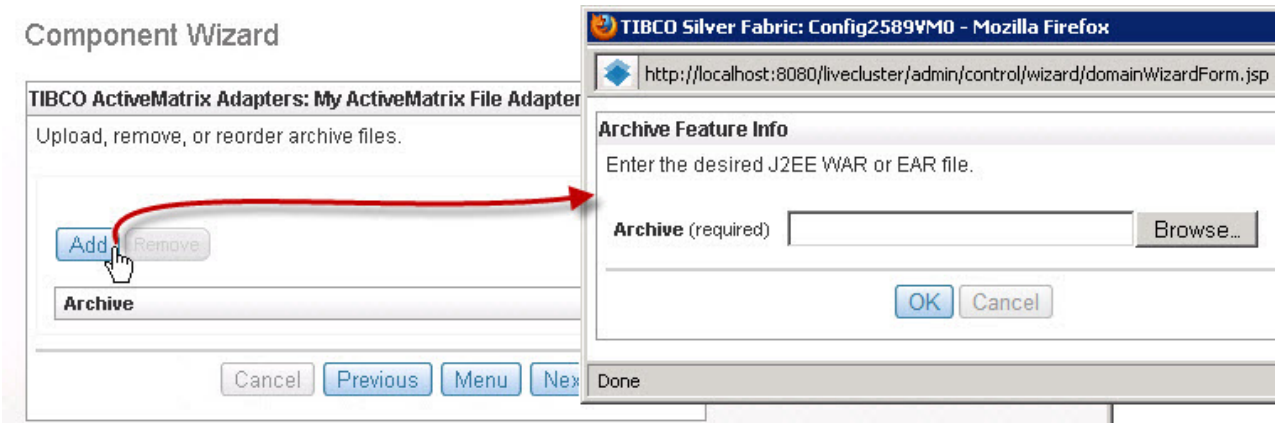
If you want TIBCO Silver Fabric Enabler for Adapter for Files to publish and run one or more Adapter for Files projects, upload one or more archive files(EAR or ZIP files) as follows.

EAR and ZIP files are briefly described in the note following Step 2: TIBCO Designer EAR, ZIP, and J2EE WAR files.

Procedure

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

   Uploading an EAR or ZIP Archive File



2. Click the **Browse** button and navigate to EAR or ZIP file and click the **OK** button.
   You can upload any one of the following archive files:

   - EAR File

     When you upload a TIBCO Adapter for Files 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.

For more information, see Creating a Deployment Configuration XML File.

Creating a Deployment Configuration XML File

Create a deployment configuration properties XML file for either an EAR deployment or for an archive continuous deployment.

Procedure

1. In *TIBCO_HOME*/tra/*tra_version*/bin, run the following command:
   ```
   AppManage -export -ear EarFile.ear -out DeploymentConfig.xml
   ```
2. Edit the file and set the value for publishing.
3. 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.

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

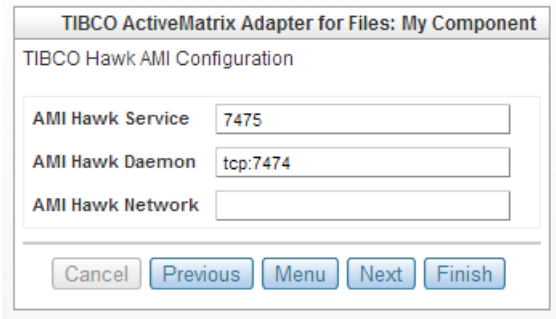
Configuring Hawk Application Management Interface (AMI) Configuration (optional)

The TIBCO Hawk AMI Configuration page defines how the Adapter for Files Component uses TIBCO Rendezvous with TIBCO Hawk Microagents (HMA). Even when EMS is used as the primary transport, HMA still uses Rendezvous as the transport.

Procedure

- In the TIBCO Hawk AMI Configuration window, enter values in each field.

  TIBCO Hawk AMI Configuration



AMI Hawk Service

It 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

It 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.

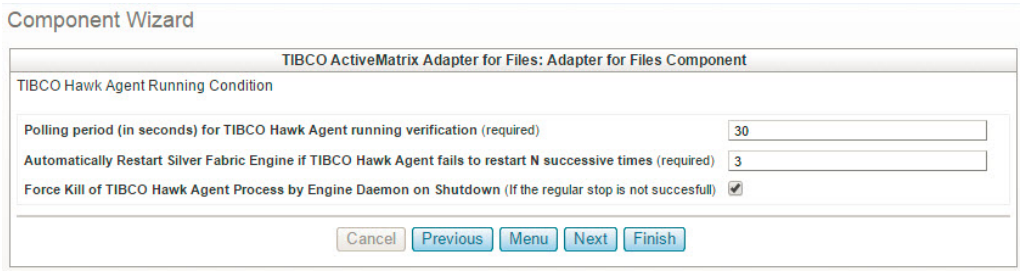
Configuring TIBCO Hawk Agent Running Condition

Set the polling period in number of seconds for checking that the TIBCO Hawk Agent is properly responding. If the TIBCO Hawk Agent fails to restart after the specified number of attempts then the TIBCO Silver Fabric Engine will be restarted automatically.

Procedure

- In the TIBCO Hawk Agent Running Condition window, enter values in each field.

  TIBCO Hawk Agent Running Conditions



The configuration parameters in the TIBCO Hawk Agent Running Condition window are as described.

- **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.

- **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 will be restarted. A successful restart will reset the count.

- **Force kill of TIBCO Hawk agent process by engine daemon on shutdown** (If the regular stop is not successful)

  Use this option only if the regular stop does not work. Select this option if you want to kill all the TIBCO Hawk Agent processes on shutdown. It stops all those processes which were started by the engine daemon.

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

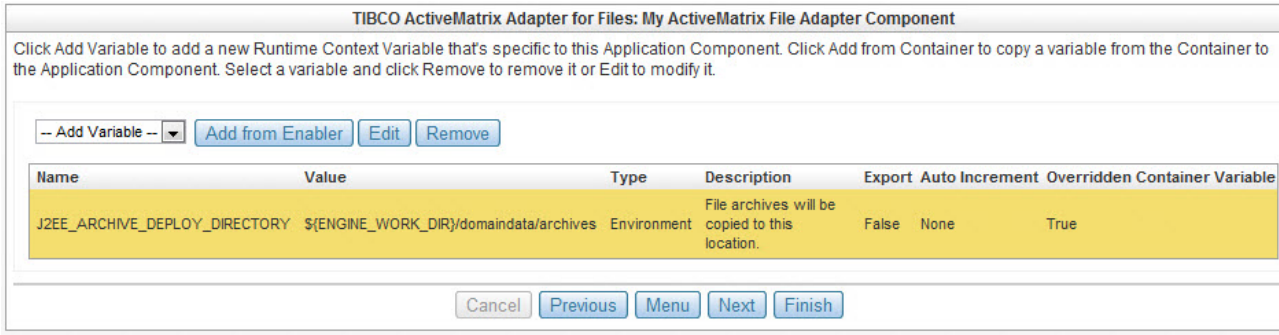
Keep all existing files and replace the existing files with the uploaded ones

Adding or Editing Enabler-Specific Runtime Context Variables

Procedure

1. Click **Add from Enabler** to copy a variable from the Enabler to the Component. To edit or remove a variable select the row first.

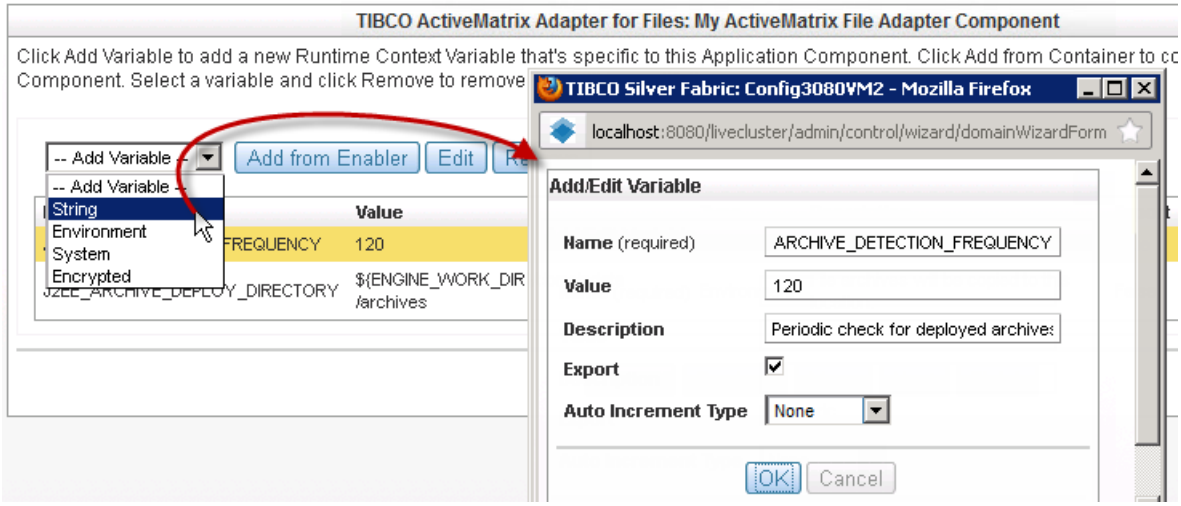
   Add/edit Enabler-specific runtime context variables



2. Select the variable and **Edit** to change its attributes or **Remove** to remove the variable.
3. You can add **String**, **Environment**, **System**, or **Encrypted** variables by first selecting a variable type from the **Add Variable** drop-down selector.

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

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.

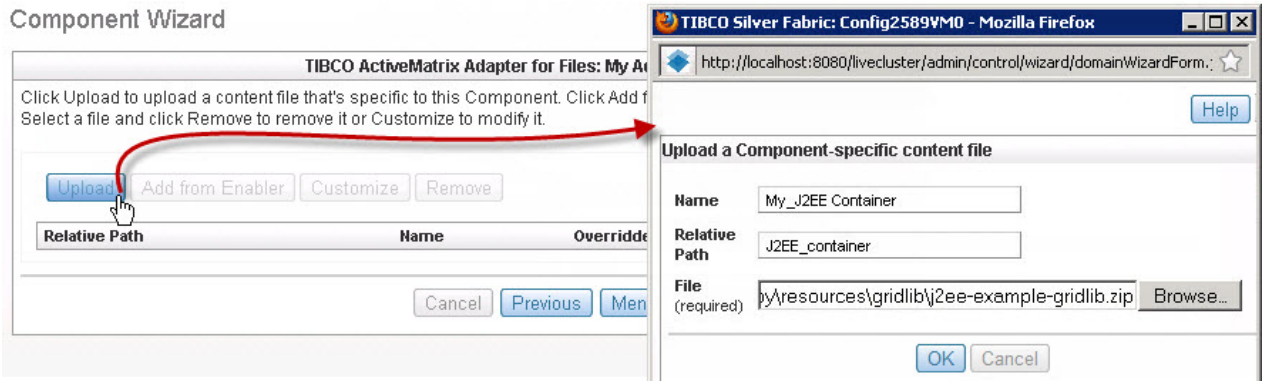
Variable values from the Enabler may be changed as well. Use the **Add from Enabler** button to change values of Enabler-specific context variables.

Adding Enabler and or Component-Specific Content Files

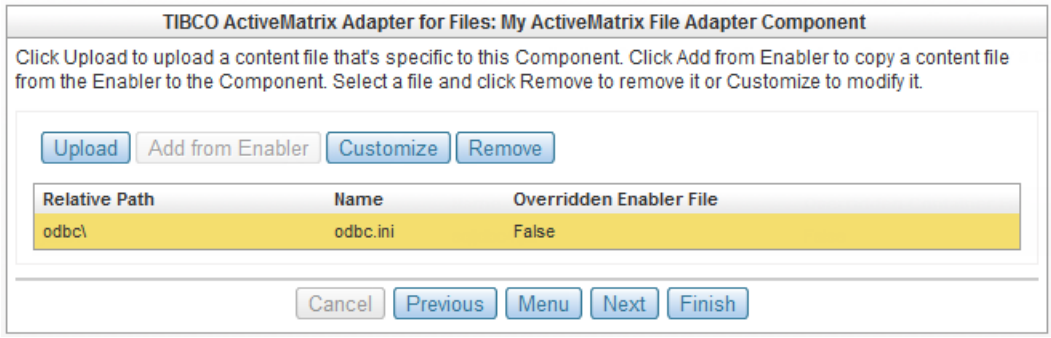
Procedure

1. Click **Upload** to upload a content file specific to the Component.
2. Select a file and click **Remove** to remove it or **Customize** to modify it.

   Add/override/customize Container and Component-specific content files



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 that were added, may be removed with that window.



Removing or Adding Adapter for Files Component Statistics

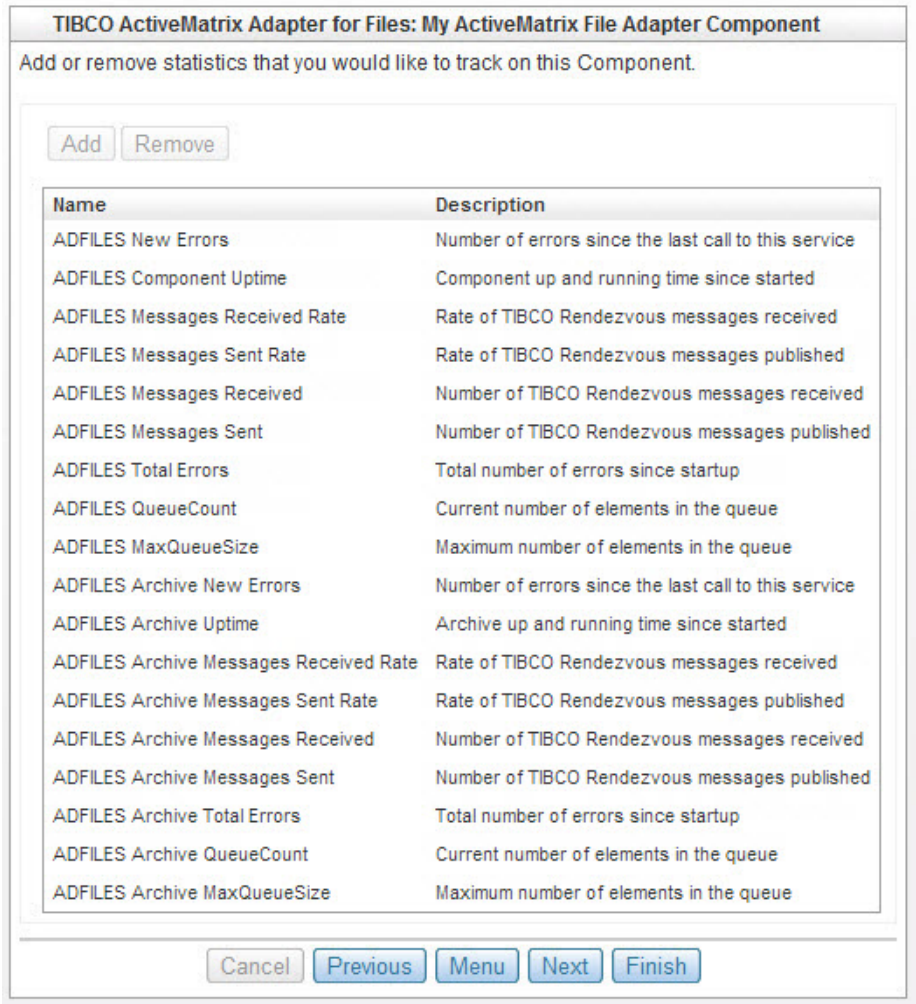
You can remove statistics that you don't want the Component to track or you can add them back again.

Procedure

- Remove the statistics that you don't want the component to track or you can add them back again..

  Add or Remove Adapter for Files Statistics

  Component scaling using statistics is not supported when TIBCO Silver Fabric publishes the Components to a single TIBCO Domain Logical Machine.



Editing Configuration File (optional)

Use the Edit the Configuration File page with extreme caution. Do not use the page unless the configuration.xml is backed up and specific knowledge about the TIBCO Silver Fabric system is being applied. This interface can be used for more advanced customizations and normally it should be left alone.

For more information, refer to "The configure.xml File" section in *TIBCO Silver® Fabric Developer's Guide*.

As an example, if a user wants to change the default maximum java heap size, the following XML example code described, accomplishes this when used in the Edit configuration file text field.

```
<containerConfig>
   <configFiles baseDir="${TIBCO_HOME}" include="adapter/adfiles/7.0/bin/
adfilesagent.tra">
      <regex pattern="java.heap.size.max=1024"
replacement="java.heap.size.max=2048" />
   </configFiles>
</containerConfig>
```

The property, baseDir, in the <configFiles> element specifies the path that includes the file to be updated. You can modify it if required.

The property, include, in `<configFiles>` element specifies which files are to be replaced. It can specify whatever files you want to change. The asterisk wild card can be used to represent a string of characters, for instance: "*.tra" to change all of the .tra files in `%baseDir%`.

The property, pattern, in the <regex> element specifies the contents that are to be replaced within the previously specified files. The value of pattern can be a regular expression. The property, replacement, in <regex> element specifies the new contents of the node specified by the pattern property value.

Completing the Configuration of the Component

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

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

Procedure

1. After you click the **Finish** button, make sure that the Component is published.
2. To do this, select **Publish Component** in the Actions drop-down list located at the left-hand side of the component you just created.

Creating a TIBCO Silver Fabric Enabler for Adapter for Files Stack

A TIBCO Silver Fabric Enabler for Adapter for Files Stack contains one (only one) TIBCO Administrator Component.

But a Stack can also be created with a single component and it could be run in standalone configurations. Each TIBCO Administrator Domain has one TIBCO Silver Fabric Enabler for Adapter for Files Stack. After starting a Stack you can update it by adding or removing Adapter for Files Components.

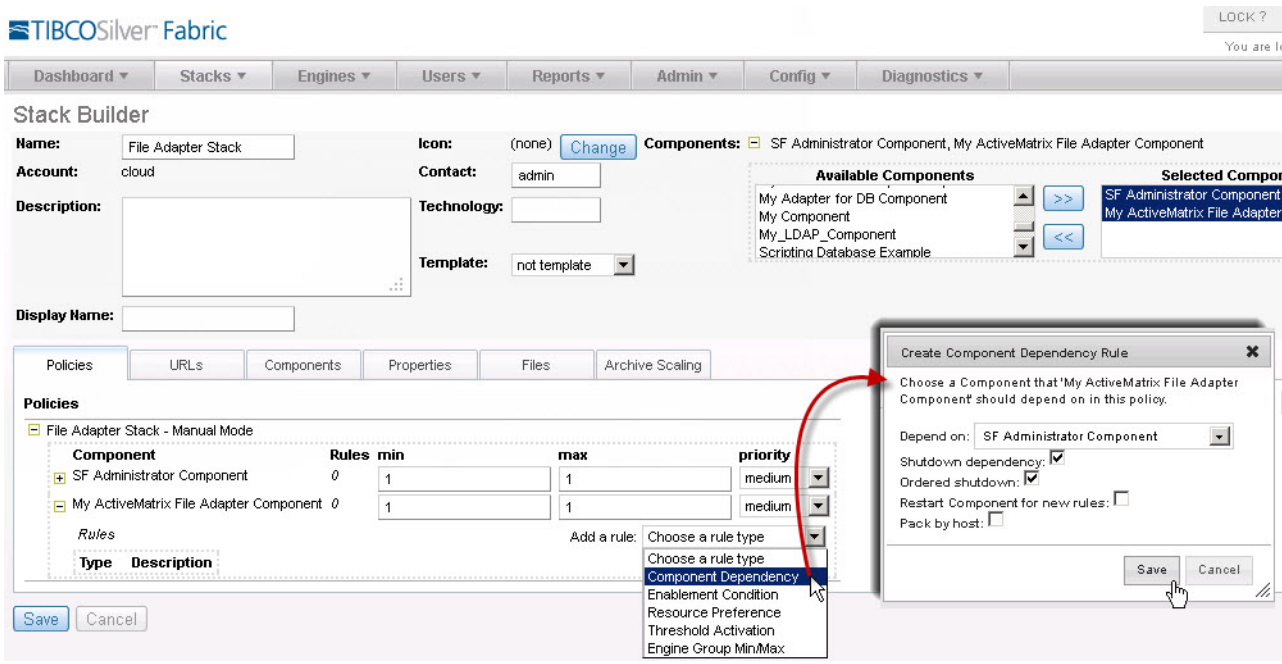
Procedure

1. In TIBCO Silver Fabric Administrator GUI, select **Stacks** > **Stacks** .
2. In the Global Actions drop-down list, select **Create New Stack** as shown in Figure 29.

   Creating a Stack

3. Enter a Stack name in the Stack Builder page as shown in Figure 30.
4. In the Components area, add one TIBCO Administrator Component and one or more Adapter for Files Components.
5. In the Policies area, expand the Component you just added to view the details of the component.

Stack Builder Page



Setting Dependency Requirements

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

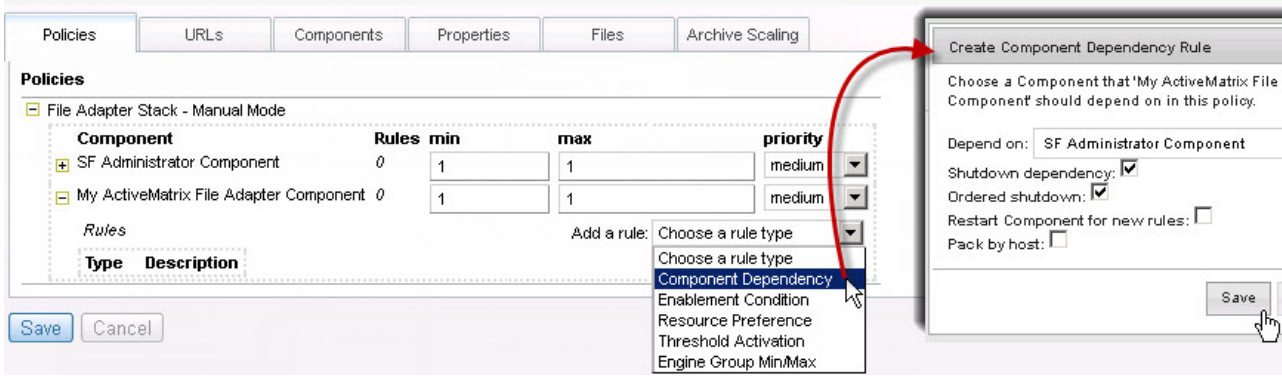
This step is required. If you do not set the dependency, TIBCO Silver Fabric Enabler for Adapter for Files will not work.

The Adapter for Files 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 for Files will start after TIBCO Administrator is up and running.

Procedure

1. During creation or edit of the Stack select **Add/edit default rule settings** from the Menu of the Component Wizard.
2. Use the **Add a Rule** pull down to select the **Component Dependency** option.

Setting an Administrator Component Dependency



In this way, the component knows all TIBCO Administrator configuration information. After setting the dependency, TIBCO Adapter for Files will start 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 for Files Components will stop if TIBCO Administrator stops working.

   If the Administrator Component was configured to "*Use dependent EMS server*" then that dependency must be set here as well.

   When using a dependent TIBCO Enterprise Message Service™ server, the dependency should be set 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.

   - **Shutdown Dependency** establishes a requirement for the parent, specified in the Depend on field, to be active to keep the dependent component running.

     When selected the stack stops component dependencies if the parent component becomes unresponsive.

     If you run TIBCO Administrator in the Fault Tolerant mode, clear the Shutdown Dependency check box. Otherwise, all Adapter for Files components will stop if TIBCO Administrator stops working.
   - **Ordered Shutdown** provides for a logical, sequential shutdown so that dependent components are shut down first. Ordered shutdown is especially important when the domain is hosted using a file structure instead of a dependent database. When you have an administrative component that uses an external database, order of shutdown is less important.
   - **Restart Component for new rules**: If new rules are defined for a component that has already been deployed, it must be restarted for the new changes to be applied. If you wish to manually restart components later to propagate changes leave this box cleared.
   - **Pack by Host**: Check the check box to specify that dependent components must run on the same host.

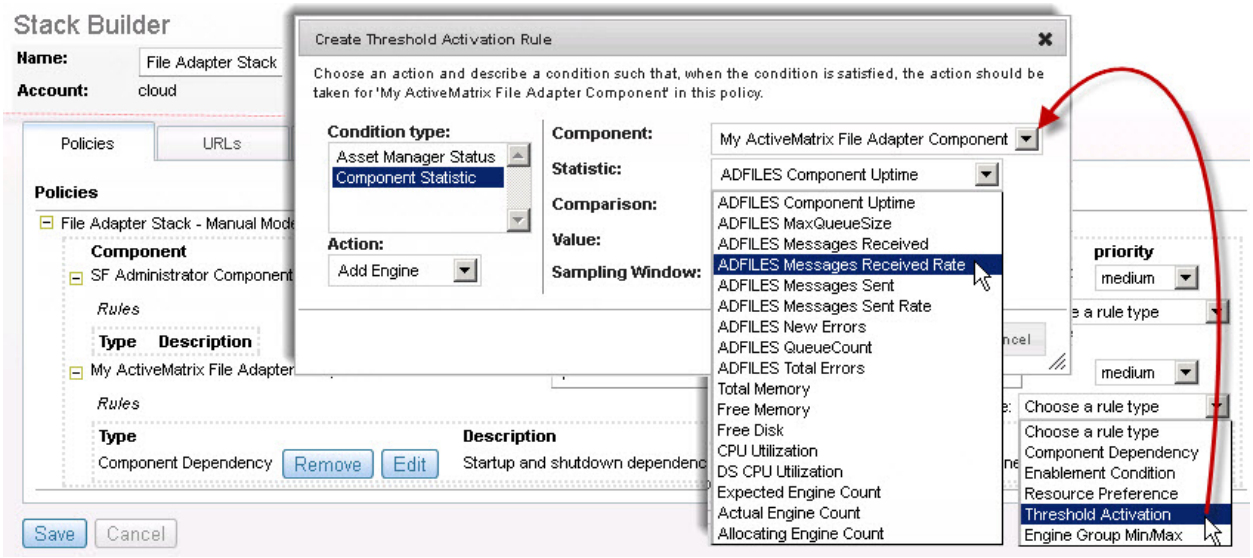
   The *TIBCO Silver Fabric User's Guide* has more information on all of these settings.

4. Save and Run your stack.

Adapter for Files Statistics

Silver Fabric Engines may be set to scale automatically based upon Adapter for Files Component statistics. On the Stack, you can define Threshold Activation rules that add or remove TIBCO Silver Fabric Engines based on Component statistics tracked during runtime.

Add an Engine based on Adapter for Files Statistics



Refer to the TIBCO Silver Fabric documentation for more details on the definition and use of Threshold Activation rules.

Adapter for Files archives can be set to scale automatically based upon Adapter for Files archive statistics. On the Stack, you can define Archive Scaling rules to add or remove Adapter for Files application archives based on Archive statistics tracked during runtime. Refer to Creating Archive Scaling Rules for more details.

From the Component Wizard choose the link to "Add/edit tracked statistics" to add or remove the Adapter for Files Component and Archive states which are tracked during runtime.

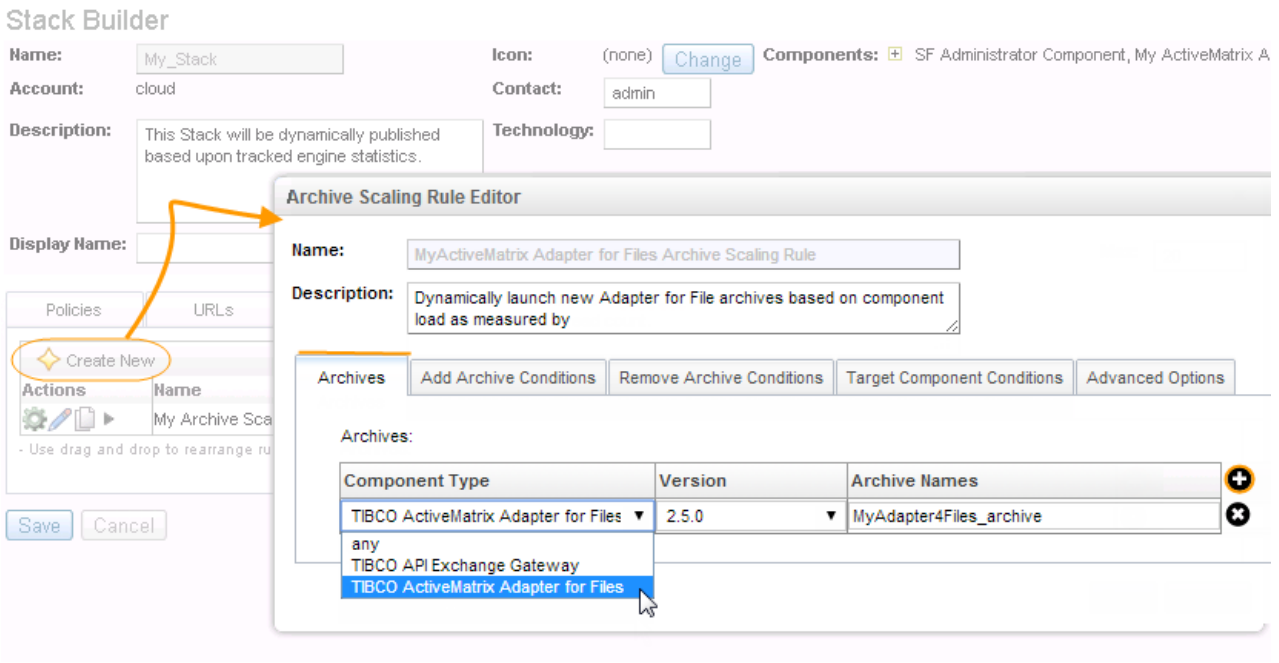
Creating Archive Scaling Rules

Procedure

1. Define Stacks that will add or remove Component Archives based on archive statistics that are monitored for triggering conditions.
2. Create Archive Scaling Rules to automate creation or removal of new Adapter for Files Archives when rules based on archive statistics meet or exceed thresholds or conditions you have set.
3. After adding Components to your Stack you can create new scaling rules by opening the **Archive Scaling** tab and clicking the **Create New** button.
4. Name your new archive scaling rule and give it a description to help you and others quickly identify the purpose and content of your archive scaling rule.

   The Archives tab defines what archive will be added or removed according to the rules you define in the other tabs.
5. Use the **Add** icon at the right of the column heading row to add one or more archives (process instances) to be scaled up or down in your Stack.

Creating a new Archive Scaling Rule

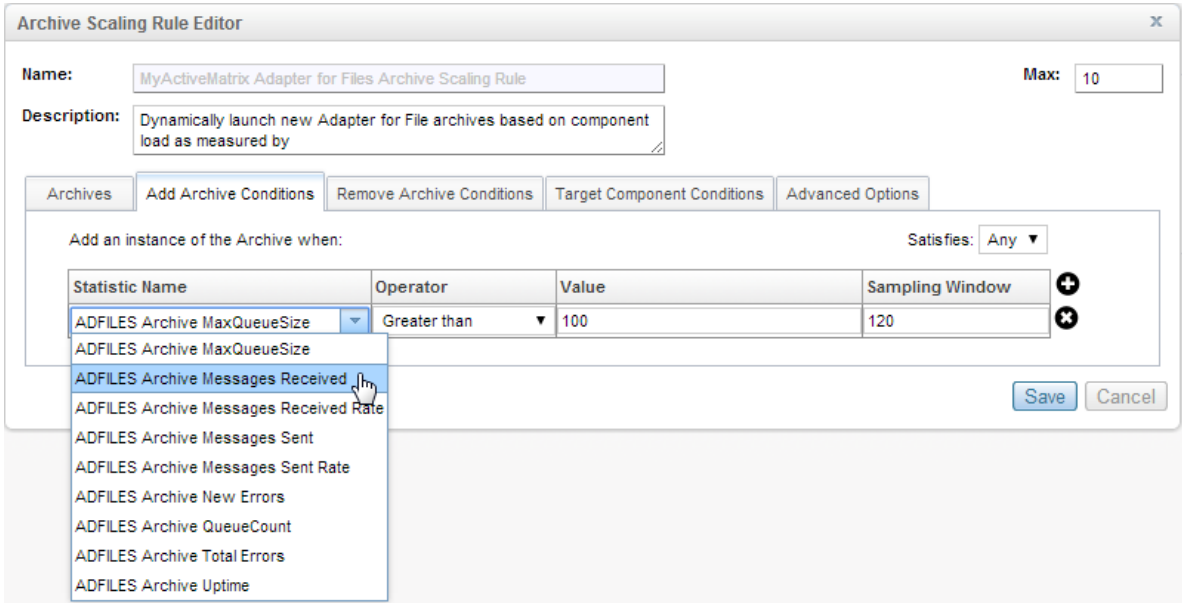


6. Select the Adapter for Files Component that was used to upload the application archive file and use the **Archive Names** field to specify what archive will be subject to the rules you set using the other tabs of the Archive Scaling Rule Editor.
7. In the **Add Archive Conditions** tab and click the **Add** icon to the right of the column heading row to create and define a new Add Archive rule.
8. Select the statistics, the operator, the value, and the sampling window period in seconds to define your condition for adding a new archive.

The Sampling Window must be a sufficiently large time period (in seconds) to allow aggregated statistics to be collected. If the Sampling Window is not big enough there is a chance that no statistics will be reported in a particular time-frame creating an inadvertent trigger condition.By default allocation statistics like "Expected Engine Count", "Client Count", "Allocating Engine Count", and "Actual Engine Count" are collected every 60 seconds and by default Component statistics are collected every 10 seconds.

9. More than one rule can be defined. With more than one rule defined, set the **Satisfies** field to specify whether all rules must be satisfied or whether any one rule may be satisfied to trigger addition of a new archive instance.

Define Archive Scaling Rule conditions



10. Optionally to set a preference for running new archives or new process instances on Component instances with favorable usage profiles. Select the statistic that is most relevant to your implementation and you can create new process instances there according to those conditions you defined.

    The tabs in the Archive Scaling Rule Editor are as described.

    The Remove Archive Conditions tab enables you to release computing resources and remove unused or idle Component Archives or process instances to scale down your Component Archives just as you scaled them up according to conditions you define on usage statistics.

    The Target Component Conditions tab allows you to restrict the start of new Archive instances to those machines that have the same set of resources that you choose. Set a rule or several rules with statistics, operators, and values as you would on the Add Archive Conditions tab and further restrict where the new Archive instances may start depending on Component Instances that have:

    - **Same Component** - will work all the time for Component Archive scaling.
    - **From the set of Components** - works only if your Component Archive is compatible with the set of Components present. For example an Adapter for Files archive cannot scale on an Adapter for Database Component.
    - **Same Component Type** - the process archive has the possibility of being scaled up on different versions of the product
    - **Same Enabler** - components that require a specific enabler should use this option.
    - **Same Middleware Version -** this selection will ensure that the Component Archive runs on machines with the appropriate middle ware: TIBCO Adapter for Files, TIBCO TRA, TIBCO Hawk, etc.
    - **Same Enabler and Middleware Version** - this selection will ensure that your Component Archive will scale up successfully, but it is the least restrictive of the target component conditions. .

      Only "From the Same Set of Components" or "Same Enabler and Same Middleware Version" will work 100% of the time.

      Provided that the Component Archive has the proper Component Type, TIBCO Silver Fabric can usually find the correct computing environment for scaling up. For example, for an Adapter for Files Component Archive, a selection of the "Same Component Type" ensures that

the Silver Fabric Broker will try and find an engine with same Adapter for Files ComponentType on which to run new Adapter for Files Component Archives.

Running TIBCO Silver Fabric Enabler for Adapter for Files Stack

After creating Stack, it should be published.

Procedure

- Select **Run Stack In Manual Mode** from the Actions list.

  Running a Stack

  If you select a policy schedule while creating a Stack, you can run the Stack in the Auto mode. The Stack runs upon the schedule defined. For more information on creating and running stacks refer to the TIBCO Silver Fabric documentation.

Updating TIBCO Silver Fabric Enabler for Adapter for Files 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 for Files (Unix/Win) Application instantiated on the Engine.

Making changes to the Stack is as easy as editing, saving, and publishing those changes to any instances that may be running. Some changes may 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 will be affected by the changes immediately.

If you want to change a Adapter for Files component, stop and restart your entire stack. To deploy, start, stop, and undeploy Adapter for Files project archives, use the Micro-scaling - Start and stop Adapter for Files archives based on your defined rules when they are already in your component. For more information refer to: Create Archive Scaling Rules.

Continuous Deployment (deploy archives directly to Adapter for Files (Unix/Win) endpoints) - publish (deploy), unpublish (undeploy), start, or stop Adapter for Files (Unix/Win) archives without having to change any Stacks, Components, or Adapter for Files (Unix/Win) engines. Deploying Adapter for Files (Unix/Win) application archives via REST and cURL commands is described in the next section.

Deployment of Archives Directly to Endpoints - Continuous Deployment

There are situations where deploying archives directly to a running TIBCO Adapter for Files (Unix/Win) 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 for Files (Unix/Win) instances already running on Silver Fabric Engines using the command line interface (CLI), Silver Fabric API, 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.

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*.

The TIBCO Silver Fabric Enabler for Adapter for files (Unix/Win)adds more REST methods to enable control of Adapter for Files (Unix/Win) archives.

Continuous Deployment Life Cycle

The continuous deployment life cycle has four REST operations that may be executed using cURL methods.

Deploy

Send an archive to a Silver Fabric Engine running an Adapter for Files (Unix/Win) 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

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

Deploy

Adapter for Files (Unix/Win) application archives may be deployed directly to an appropriate Silver Fabric Engine (TIBCO Logical Machine) by calling the REST method.

In this document cURL syntax is used to show 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/engines/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_Name"]
```

Where inputs bounded by square brackets are optional; file names, elements, variable names, and any values shown in italics should be substituted by your implementation values.

In a cURL statement all values must be URL-encoded so that special characters like spaces, for example, are converted to "`%20`". Other special characters like forward slashes "`/`" must also be converted to"`%2F`" or their respective URL encoded values so they may be sent and received properly.

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

An example cURL archives deploy statement:

```
curl -u admin:admin -X POST -H "Accept:application/json" -H "Content-
Type: multipart/form-data" -v http://MySFBroker.com:8080/livecluster/rest/v1/sf/
engines/archives -F "archiveFile=@MyProcessOrder.ear" -F "AppName=MyOrders/
MyProcOrder" -F "deploymentFile=@MyDeployCriteria.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 below:

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

archiveFile

Specifies your Adapter for Files (Unix/Win) archive file (.zip or .ear file) to upload to the Silver Fabric Broker which will then publish 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. Of course, archives may also be uploaded using the Component Wizard.

deploymentFile

Specifies the properties file that defines endpoint selection criteria described in more detail below.

```
    -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

(Optional)

Specifies the directory location where the application archive(s) will be deployed and what the application will be named.

Where your archive application will deploy and what it will be called depends on what you specify with AppName. For example when you use TIBCO Designer to create an .ear file with a name like *MyAppArchive*, varying the AppName specification will give following behavior:

- If the AppName form-data field is not specified, then *MyAppArchive* will be deployed at the top level of the Administrator directory.
- If -F "AppName=***A***" is submitted in the curl request, then *MyAppArchive* will be renamed to *A* and deployed at the top level.
- If -F "AppName=***A/***" is sent, then the directory folder *A* is used or it will be created and *MyAppArchive* is deployed within that sub-directory.
- If -F "AppName=***A/B***" is sent, then the sub-directory *A* is used or created, and *MyAppArchive* is deployed there and renamed to *B*.
- If -F "AppName=***A/B/***" is sent, then the folder *A* with a sub-folder *B* will be used or created and *MyAppArchive* will be published within sub-folder *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 don't 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 will use when deployed.

All deployment configuration cURL expressions will take the form:

-F "AppSettings.*element1.element2=SomeValue*"

Some examples:

```
 -F "AppSettings.localRepoInstance.encoding=UTF-8"
 -F "AppSettings.description=This%20application%20deployment%20is%20for
%20validation%20testing."
```

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

- `description` - A string describing the application.
- `contact` - A string to name the person responsible for the deployment.
- `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.
- `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.

- `encoding` – Specifies encoding for the repository instance. If this element is not specified, then the encoding for the admin server is used. If the admin server is not available, then 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) The Form-data parameter that specifies a comma delimited list of archives within the zip that are to be deployed. If an `archives` list is omitted then all archives in the application archive package will be deployed. Example:

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

ArchiveSettings

(Optional) The form-data parameter specifies settings for the archive.

- `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 don't have the required software. A deployment configuration cURL expressions will take the form:

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

- `av` - Specify values for archive runtime variables with a comma-separated string with each key value pair joined by an equals (=) 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 Adapter for Files. The following is a first attempt at listing them all.

- `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.

- `failureEvents.failureEvent`:

  failureType: *ANY, FIRST, SECOND, Subsequent*

  If event is used then 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 will occur 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.

- `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 will occur 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 message will be sent.

cc: a comma-separated list of email addresses to which copies of the message will be 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.

- `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 will occur 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

- `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 will occur 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 will occur 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 message will be sent.

  emailAction.cc: a comma-separated list of email addresses to which copies of the message will be 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 will occur 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

- `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 will occur 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 will occur 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 message will be sent.

emailAction.cc: a comma-separated list of email addresses to which copies of the message will be 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 will occur 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

- `failureCount`: The value in this field defines how many restarts should be attempted before resetting the error counter to 0.
- `failureInterval`: The value in this field defines how much time should expire before resetting the error counter to 0.

InstanceSettings

(Optional) Some syntax examples:

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

- `description`: Specify any pertinent information about the binding.
- `contact`: Name the person responsible for this application instance.
- `startOnBoot`: When *true* the service instance starts when the computer is restarted. Default value is *false*.
- `enableVerbose`: When *true*, sets the Enabler for verbose tracking for service instances. Default value is *false*.
- `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.
- `startUpType`: Specifies the instance service startup type as either: *Automatic, Manual*, or *Disabled*.

- `login`: Specifies the login account for the service, if any. The domain name must be specified as well.
- `password`: Sets the password for that service, if any.
- `iv` : This element uses a comma-separated string with name-value pairs with each key value pair joined by an equals (=) sign. For example:

configurationFile

(Optional) The 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 the 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"
```

GV

(Optional) Sets global variables for use on the targeted application endpoint by the archive. The GV form-data field lets you define a comma delimited list of declarative name equals value statements.

```
-F "GV=globalVariableA=123,globalVarB=SomeString"
```

For example to change the JMS SSL and Rv Service ports:

```
-F "GV=JmsSslProviderUrl=ssl://localhost:7555,RvService=7222"
```

If global variables are not defined with explicit values in the cURL statement then those values you may have set in the deployment *configurationFile*`.xml` will apply.

If global variables with the same name are set by both REST statement and a specified variable provider then the value set by REST statement will overwrite and take precedence over the value set in the variable provider.

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.

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

See Creating a Variable Provider for use by a REST call.

NoDeploy

(Optional) The default value is false which means that the archive(s) are uploaded to the Silver Fabric Broker *and* they are deployed to the application endpoints. When NoDeploy is set to true, the archives are uploaded with associated service Enabler bindings created in TIBCO Administrator, but the archives are not deployed to a Silver Fabric Engine and the application endpoint.

```
-F "NoDeploy=true"
```

NoStart

(Optional) The default value is false, meaning that the archives are both deployed and started by default. If NoStart is set to true, the application will be deployed but not started.

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

Deployment File

When deploying an archive by REST you must include a deployment file, which will specify at least one selection criteria for determining which Engine and Component will receive 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, the following 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*"

Criteria - The following criteria are supported:

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.

Name	Definition
EngineProperty(*name*)	A named Engine property. The following engine properties may be used: • id: The numeric ID unique to each Engine, generated upon installation. • guid: The globally unique identifier for the Engine (network card address). • instance: The instance of the Engine, starting with zero. Multi-CPU hosts may have multiple instances of Engines running concurrently. • IP: The IP address of the Engine. • username: The username of the Engine, which will be the hostname of the Engine unless installed with tracking properties. • hostname: The hostname of the Engine. • cpuNo: The number of CPUs on the Engine's host computer. • cpuCoreCount: The total number of CPU cores on the Engine's host computer, if available. A value of -1 indicates that a value could not be determined. • cpuSocketCount: The total number of physical CPUs on the Engine's host computer, if available. A value of -1 indicates that a value could not be determined. • cpuThreadCount: The total number of hardware threads on the Engine's host computer, if available. A value of -1 indicates that a value could not be determined. • cpuIdString: The CPUID string of the first physical CPU on the Engine's host computer, if available. • cpuTotal: The amount of processing power on the Engine's host computer, measured in millions of floating-point arithmetic operations per second. • Calculated on the Engine Daemon using a Linpack performance calculation and reported when the Engine Daemon logs into the Manager and updated on a frequency that is set in the Engine Configuration. • totalMemInKB: The amount of total physical memory on the Engine's host computer, in kilobytes. • freeMemInKB: The amount of free physical memory on the Engine's host computer, in kilobytes, updated when any Engine instance is launched. • freeDiskInMB: The amount of available disk space on the Engine's host computer, in megabytes, updated when any Engine instance is launched. • os: The operating system platform. This corresponds with the Engine installation. Available platforms can be viewed on the Engine Install page.

Name	Definition
	• osVersion: The version of the operating system. • homeDir: Home directory, the installation directory on the Engine's host computer. • dataDir: Data directory, the directory which is home to the DDT (direct data transfer) data. • dataUrl: Data URL, the URL corresponding to the DDT directory. • workUrl: Work URL, the URL of work directory on the Engine's fileserver that stores log and temporary files. • sharedDirID: A number which identifies a group of Engines running from the same home directory. • osUsername: The owner of the process running the Engine. • configurationName: The current Engine Configuration name. The value is [os]:[name] like on the Engine Configuration page. • pid: The process ID of the Engine process.
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.
Statistic(*name*)	A named Enabler for Adapter for Files (Unix/Win) statistic like: Total Memory, Free Memory, Free Disk, CPU Utilization, DS CPU Utilization, Expected Engine Count, Actual Engine Count, or Allocating Engine Count
ArchiveStatistic(*statName*, *archiveName*)	A named statistic for a specified archive.
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 Deployment File

```
# Sample deployment file
ComponentType = "TIBCO ActiveMatrix Adapter for Files:2.5.0"
Statistic(CPU Utilization) < 80
ActivationInfoProperty(ClusterName) matches dev_cluster.*
```

Successful Deployment

REST will return 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 will return an error of Status 500 or some other appropriate error status depending on the cause.

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.AbstractVariableProvider;
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/variableProviders
```

3. Create an XML file with the properties shown below 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.

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

```
SILVERFABRIC_HOME/webapps/livecluster/deploy/config/variableProviders
```

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

Continuous Deployment Timeout

Continuous deployment transfers can timeout due to one or more 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 were deployed but not started. You must know the Engine-Id, Engine-Instance, and the full application ArchiveName.

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/engines/{engine-
id}/{engine-instance}/archives/{full_archive-name}/start"
```

Expressions in the generic form cURL expression above were separated by line breaks to help with readability, but normally an unbroken string is submitted in the execution as in the example shown in the tip below.

The values of *{engine-id}* and *{engine-instance}* may 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.

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**.

{full_archive-name} :

The {full_archive-name} is the full name of the application including the parent directories in the TIBCO Administrator UI.

The *{full_archive-name}* must be 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 in appropriately.

A cURL example that starts an Archive.

The full archive name shown below had to be URL encoded:

/Silver Fabric/SFFL 22E5/DomainMachineName/processOrder/Orders/MyProcOrder

...to be passed in the cURL statement below:

```
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%2FSFFL
%2022E5%2FDomainMachineName%2FprocessOrder%2FOrders%2FMyProcOrder/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/engines/{engine-
id}/{engine-instance}/archives/{full_archive-name}/{archive-id}/stop"
```

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

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/engines/{engine-
id}/{engine-instance}/archives/{full_archive-name}/undeploy"
     -F "DeleteApp=true"
```

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

DeleteApp

(Optional - for use with undeploy only) The default value is false and it can be omitted. When the DeleteApp parameter is false then 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.

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*.

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.properties

```
#properties for the build.xml
# stack.name=Adfiles-Stack
stack.description=
Adfiles-Compt.component.name=Adfiles-Compt
Adfiles-Compt.component.description=
Adfiles-Compt.component.type=TIBCO ActiveMatrix Adapter for Files\:3.0.0
Adfiles-Compt.enabler.name=TIBCO ActiveMatrix Adapter for Files container
Adfiles-Compt.enabler.version=3.0.0.0
Adfiles-Compt.utility=false
Adfiles-Compt.blacklisting=false
Adfiles-Compt.blacklist.failures=0
Adfiles-Compt.archive.scale.up.timeout=120
Adfiles-Compt.archive.scale.down.timeout=120
Adfiles-Compt.deactivation.timeout=3600
Adfiles-Compt.activation.timeout=3600
Adfiles-Compt.maximum.capture.time=180
Adfiles-Compt.max.instances.per.host=0
Adfiles-Compt.stats.collection.frequency=10
Adfiles-Compt.activation.delay=0
Adfiles-Compt.engine.reservation.expiration=300
Adfiles-Compt.min=1
Adfiles-Compt.max=1
Adfiles-Compt.priority=3
Adfiles-Compt.relative.url=
Adfiles-Compt.routing.prefix=
Adfiles-Compt.https.enabled=false
Adfiles-Compt.http.enabled=true
Adfiles-Compt.route.directly.to.endpoints=false
Adfiles-Compt.archive=ContainerReader2Writer2.ear
Adfiles-Compt.start.archives.on.activation=true
Adfiles-Compt.checkpoint.frequency.in.seconds=300
middleware.versions=TIBCO_ActiveMatrix_Adfiles_distribution:
7.0.0.0.0,TIBCO_TRA_distribution:5.10.0.0.1,TIBCO_HAWK_distribution:
```

```
5.2.0.0.0,TIBCO_RV_distribution:8.4.4.0.0
optional.middlware.versions=TIBCO_EMS_DISTRIBUTION:8.3.0.0.0
Adfiles-Compt.archive.application.logs=true
Adfiles-Compt.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
Adfiles-Compt.content.dir=Adfiles-Compt/content/
Adfiles-Compt.config.dir=Adfiles-Compt/configure/
Adfiles-Compt.archive.0.path=Adfiles-Compt/archives/ContainerReader2Writer2.ear
Adfiles-Compt.contextvar.nb.hawk.restart.before.restart.engine=3
Adfiles-Compt.contextvar.hawk.pollperiod=30
Adfiles-Compt.contextvar.is.prepand=false
Adfiles-Compt.contextvar.jdbc.driver.file=/jdbc/sqljdbc4.jar
Adfiles-Compt.contextvar.external.jar.file=
Adfiles-Compt.contextvar.plugin.config.file=
Adfiles-Compt.contextvar.logical.machine.name=ADFiles
Adfiles-Compt.contextvar.domaindata.dir=/opt/qa/adfiles-compt
Adfiles-Compt.contextvar.tibco.services.state.after.tlm.moved=Started
Adfiles-Compt.contextvar.use.full.archive.name=true
Adfiles-Compt.contextvar.ami.hawk.service=7475
Adfiles-Compt.contextvar.ami.hawk.daemon=tcp\:7474
Adfiles-Compt.contextvar.ami.hawk.network=
Adfiles-Compt.contextvar.hawk.plugin.config.file=
Adfiles-Compt.contextvar.action.to.perform.for.hawk.file=Delete_The_Files
Adfiles-Compt.contextvar.delete.application.conf.at.shutdown=false
Adfiles-Compt.contextvar.do.not.redeploy.ear.file.at.startup=false
Adfiles-Compt.contextvar.managed.process.hawk.agent.enabled=true
Adfiles-Compt.contextvar.managed.process.service.enabled=true
Adfiles-Compt.contextvar.save.delete.application.conf.at.shutdown=false
Adfiles-Compt.contextvar.save.do.not.redeploy.ear.file.at.startup=false
Adfiles-Compt.contextvar.save.is.prepand=
Adfiles-Compt.contextvar.save.managed.process.service.enabled=true
Adfiles-Stack.policy.0Adfiles-Compt.min=1
Adfiles-Stack.policy.0Adfiles-Compt.max=1
Adfiles-Stack.policy.0Adfiles-Compt.priority=3
Adfiles-Stack.policy.0.Adfiles-Compt.rule.0.component.name=Admin-294
Adfiles-Stack.policy.0.Adfiles-Compt.rule.0.shutdown=true
Adfiles-Stack.policy.0.Adfiles-Compt.rule.0.pack.by.property=null
Adfiles-Stack.policy.0.Adfiles-Compt.rule.1.property.name=hostname
Adfiles-Stack.policy.0.Adfiles-Compt.rule.1.operator=matches
Adfiles-Stack.policy.0.Adfiles-Compt.rule.1.property.value=lin64vm324
Adfiles-Stack.policy.0.Adfiles-Compt.rule.1.affinity.pos=0
Adfiles-Stack.policy.0.Adfiles-Compt.rule.2.property.name=instance
Adfiles-Stack.policy.0.Adfiles-Compt.rule.2.operator=not matches
Adfiles-Stack.policy.0.Adfiles-Compt.rule.2.property.value=3
Adfiles-Stack.policy.0.Adfiles-Compt.rule.2.affinity.pos=0
stack.mode=stopped
```

build.xml

```
<project name="Adfiles-Stack-build"
default="release" basedir="." xmlns:sf="antlib:com.datasynapse.fabric.ant">

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

    <sf:connection-props brokerurl="${DSPrimaryDirector}" username="${DSUserName}"
    password="${DSPassword}"
    clientssltrustfile="${DSSSLTrustFile}" />

    <target name="release" depends="create-all-components, release-Adfiles-Compt-
component,release-stack" />

    <target name="clean" depends="clean-stack, clean-Adfiles-Compt-component" />
```

```
    <target name="create-all-components">
        <sf:component action="create" name="${Adfiles-Compt.component.name}"
        description="${Adfiles-Compt.component.description}" type="${Adfiles-
Compt.component.type}"
        enablername="${Adfiles-Compt.enabler.name}" enablerversion="${Adfiles-
Compt.enabler.version}"
        utility="${Adfiles-Compt.utility}" />
    </target>


<target name="release-Adfiles-Compt-component">
 <echo message="Building ${Adfiles-Compt.component.name}" />
 <sf:option name="${Adfiles-Compt.component.name}" action="replace">
 <sf:property name="Engine Blacklisting" value="${Adfiles-Compt.blacklisting}"/
>
 <sf:property name="Failures Per Day Before Blacklist" value="${Adfiles-
Compt.blacklist.failures}"/>
  <sf:property name="Archive Scale Up Timeout" value="${Adfiles-
Compt.archive.scale.up.timeout}"/>
  <sf:property name="Archive Scale Down Timeout" value="${Adfiles-
Compt.archive.scale.down.timeout}" />
  <sf:property name="Maximum Deactivation Time" value="${Adfiles-
Compt.deactivation.timeout}" />
  <sf:property name="Maximum Activation Time" value="${Adfiles-
Compt.activation.timeout}"/>
  <sf:property name="Maximum Capture Time"value="${Adfiles-
Compt.maximum.capture.time}"/>
  <sf:property name="Maximum Instances Per Host" value="${Adfiles-
Compt.max.instances.per.host}" />
  <sf:property name="Statistics Collection Frequency" value="${Adfiles-
Compt.stats.collection.frequency}" />
  <sf:property name="Activation Delay" value="${Adfiles-Compt.activation.delay}" /
>
  <sf:property name="Engine Reservation Expiration"
               value="${Adfiles-Compt.engine.reservation.expiration}" />
   <sf:property name="Middleware Versions" value="${middleware.versions}"/
>
   <sf:property name="Optional Middlware Versions" value="$
{optional.middlware.versions}"/>
 </sf:option>


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


<sf:feature name="${Adfiles-Compt.component.name}" action="add" feature="HTTP
Support"/>
<sf:feature name="${Adfiles-Compt.component.name}"action="update" feature="HTTP
Support">
    <sf:property name="Relative Url" value="${Adfiles-Compt.relative.url}" /
>
    <sf:property name="Routing Prefix" value="${Adfiles-Compt.routing.prefix}" /
>
    <sf:property name="HTTPS Enabled" value="${Adfiles-Compt.https.enabled}" /
>
    <sf:property name="HTTP Enabled" value="${Adfiles-Compt.http.enabled}" /
>
    <sf:property name="Route Directly To Endpoints" value="${Adfiles-
Compt.route.directly.to.endpoints}"/>
    </sf:feature>


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

```
Compt.start.archives.on.activation}" />        </sf:feature>


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

<sf:content-file type="component" name="${Adfiles-Compt.component.name}"
action="add">
    <sf:contentset dir="${Adfiles-Compt.content.dir}">
     <include name="**/*.*" />
     </sf:contentset>
</sf:content-file>


 <sf:config-file type="component" name="${Adfiles-Compt.component.name}"
                      action="update" configfile="${Adfiles-Compt.config.dir}/
configure.xml"/>
    <sf:archive-file name="${Adfiles-Compt.component.name}" action="add" >
      <path>
         <pathelement path="${Adfiles-Compt.archive.0.path}"/>
      </path>
   </sf:archive-file>


 <sf:context-variable type="component" name="${Adfiles-Compt.component.name}"
action="update">
    <sf:contextvar name="NB_HAWK_RESTART_BEFORE_RESTART_ENGINE" type="string"
              value="${Adfiles-
Compt.contextvar.nb.hawk.restart.before.restart.engine}"
              export="false" autoincrement="none" description="null"/>
    <sf:contextvar name="HAWK_POLLPERIOD" type="string"
                   value="${Adfiles-Compt.contextvar.hawk.pollperiod}"
                   export="false" autoincrement="none" description="null"/
>
    <sf:contextvar name="IS_PREPAND" type="environment"
                   value="${Adfiles-Compt.contextvar.is.prepand}" export="false"
                   autoincrement="none" description="null" />
    <sf:contextvar name="JDBC_DRIVER_FILE" type="environment"
                   value="${Adfiles-Compt.contextvar.jdbc.driver.file}"
                   export="false" autoincrement="none"
                   description="Only if the Domain is stored in a database"/>
    <sf:contextvar name="EXTERNAL_JAR_FILE"
                   type="environment"
                   value="${Adfiles-Compt.contextvar.external.jar.file}"
                   export="false"
                   autoincrement="none"
                   description="null"/>
    <sf:contextvar name="PLUGIN_CONFIG_FILE" type="environment"
                 value="${Adfiles-Compt.contextvar.plugin.config.file}"
                export="false" autoincrement="none" description="null"/>
    <sf:contextvar name="LOGICAL_MACHINE_NAME"
                   type="environment"
                   value="${Adfiles-Compt.contextvar.logical.machine.name}"
                   export="false"
                   autoincrement="none"
                   description="Machine name. If empty the value will be the
Component Name"/>
    <sf:contextvar name="DOMAINDATA_DIR"
                   type="environment"
                   value="${Adfiles-Compt.contextvar.domaindata.dir}"
                   export="false" autoincrement="none"
                   description="Share drive where deployment configuration will be
created"/>
```

```
    <sf:contextvar name="TIBCO_SERVICES_STATE_AFTER_TLM_MOVED"
                   type="environment"
                   value="${Adfiles-
Compt.contextvar.tibco.services.state.after.tlm.moved}"
                   export="false" autoincrement="none"
                   description="On logical machine restart, all services are either
started or stopped"/>
    <sf:contextvar name="USE_FULL_ARCHIVE_NAME"
                   type="environment"
                   value="${Adfiles-Compt.contextvar.use.full.archive.name}"
                   export="false"
                   autoincrement="none"
                   description="null"/>
   <sf:contextvar name="AMI_HAWK_SERVICE" type="environment"
                  value="${Adfiles-Compt.contextvar.ami.hawk.service}"
                  export="false"
                  autoincrement="none"
                  description="null"/>
  <sf:contextvar name="AMI_HAWK_DAEMON"
                 type="environment"
                 value="${Adfiles-Compt.contextvar.ami.hawk.daemon}"
                 export="false" autoincrement="none" description="null"/>
  <sf:contextvar name="AMI_HAWK_NETWORK"
                  type="environment"
                  value="${Adfiles-Compt.contextvar.ami.hawk.network}"
                  export="false" autoincrement="none" description="null"/>
  <sf:contextvar name="HAWK_PLUGIN_CONFIG_FILE"
                 type="environment"
                 value="${Adfiles-Compt.contextvar.hawk.plugin.config.file}"
                 export="false" autoincrement="none" description="null"/>
  <sf:contextvar name="ACTION_TO_PERFORM_FOR_HAWK_FILE"
                 type="environment" value="${Adfiles-
Compt.contextvar.action.to.perform.for.hawk.file}"
                 export="false" autoincrement="none" description="null"/
>
  <sf:contextvar name="DELETE_APPLICATION_CONF_AT_SHUTDOWN"
              type="environment" value="${Adfiles-
Compt.contextvar.delete.application.conf.at.shutdown}"
              export="false" autoincrement="none" description="null"/>

   <sf:contextvar name="DO_NOT_REDEPLOY_EAR_FILE_AT_STARTUP"
                  type="environment"
                  value="${Adfiles-
Compt.contextvar.do.not.redeploy.ear.file.at.startup}"
                 export="false" autoincrement="none" description="null"/>
   <sf:contextvar name="MANAGED_PROCESS_HAWK_AGENT_ENABLED"
                   type="environment"
                   value="${Adfiles-
Compt.contextvar.managed.process.hawk.agent.enabled}"
                   export="false" autoincrement="none"
                   description="If the regular stop is not succesfull"/>
   <sf:contextvar name="MANAGED_PROCESS_SERVICE_ENABLED"
                   type="environment"
                   value="${Adfiles-
Compt.contextvar.managed.process.service.enabled}"
                   export="false" autoincrement="none"
                   description="If the regular stop is not succesfull"/>
   <sf:contextvar name="SAVE_DELETE_APPLICATION_CONF_AT_SHUTDOWN"
                   type="environment"
                   value="${Adfiles-
Compt.contextvar.save.delete.application.conf.at.shutdown}"
                   export="false" autoincrement="none"
                  description="null"/>
   <sf:contextvar name="SAVE_DO_NOT_REDEPLOY_EAR_FILE_AT_STARTUP"
type="environment"
                   value="${Adfiles-
Compt.contextvar.save.do.not.redeploy.ear.file.at.startup}"
                   export="false" autoincrement="none"
                   description="null"/>
   <sf:contextvar name="SAVE_IS_PREPAND"
                   type="environment"
                   value="${Adfiles-Compt.contextvar.save.is.prepand}"
```

```
                  export="false"
                  autoincrement="none"
                  description="null"/>
  <sf:contextvar name="SAVE_MANAGED_PROCESS_SERVICE_ENABLED" type="environment"
                  value="${Adfiles-
Compt.contextvar.save.managed.process.service.enabled}"
                  export="false" autoincrement="none"
                  description="If the regular stop is not succesfull"/>
    </sf:context-variable>
    <sf:publish type="component" name="${Adfiles-Compt.component.name}"/>
 </target>



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

<!--  uncomment this section to create schedules
-->
<sf:stack-policy action="update" name="${stack.name}" >
   <sf:policy manualpolicy="true">
    <sf:compalloc component="${Adfiles-Compt.component.name}"
                  min="${Adfiles-Stack.policy.0Adfiles-Compt.min}"
                  max="${Adfiles-Stack.policy.0Adfiles-Compt.max}"
                  priority="${Adfiles-Stack.policy.0Adfiles-
Compt.priority}">
        <sf:allocationrule type="Component Dependency" >
        <sf:property name="component"
                value="${Adfiles-Stack.policy.0.Adfiles-Compt.rule.
0.component.name}"/>

        <sf:property name="shutdown"
                value="${Adfiles-Stack.policy.0.Adfiles-Compt.rule.0.shutdown}"/
>
        <sf:property name="packByProperty"
                 value="${Adfiles-Stack.policy.0.Adfiles-Compt.rule.
0.pack.by.property}"/>
   </sf:allocationrule>
      <sf:allocationrule type="Resource Preference" >
      <sf:property name="propertyName"
                   value="${Adfiles-Stack.policy.0.Adfiles-Compt.rule.
1.property.name}"/>

    <sf:property name="operator" value="${Adfiles-Stack.policy.0.Adfiles-Compt.rule.
1.operator}"/>
      <sf:property name="propertyValue"
       value="${Adfiles-Stack.policy.0.Adfiles-Compt.rule.1.property.value}"/
>

      <sf:property name="affinityPos"
              value="${Adfiles-Stack.policy.0.Adfiles-Compt.rule.1.affinity.pos}"/
>
   </sf:allocationrule>

     <sf:allocationrule type="Resource Preference" >
     <sf:property name="propertyName"
         value="${Adfiles-Stack.policy.0.Adfiles-Compt.rule.2.property.name}"/
>
     <sf:property name="operator"
         value="${Adfiles-Stack.policy.0.Adfiles-Compt.rule.2.operator}"/
```

```
>

     <sf:property name="propertyValue"
                  value="${Adfiles-Stack.policy.0.Adfiles-Compt.rule.
2.property.value}"/>
      <sf:property name="affinityPos"
          value="${Adfiles-Stack.policy.0.Adfiles-Compt.rule.2.affinity.pos}"/>
    </sf:allocationrule>
   </sf:compalloc>
 </sf:policy>

 </sf:stack-policy>
 <sf:publish type="stack" name="${stack.name}"/>
 <sf:stack-mode name="${stack.name}" action="run" mode="${stack.mode}"/>
</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"/>

<!--  uncomment this section to remove schedules on clean-->
  </target>

</project>
```

Retrieving Log Files

You can retrieve Adapter for Files log files from the TIBCO Silver Fabric Administrator GUI.

Procedure

1. In TIBCO Silver Fabric Administrator GUI, select **Engines** > **Log Search** .
2. Select the component of which you want to see the log files, as shown in Figure 36.
3. Optionally type a regular expression you want search in the Expression field.
4. Select the Start Time to see the logs since that time.

Log Files

Retained Log Files

In addition to the Engine log file, some more log files are retained.

- TIBCO Hawk Agent Log Files
- Adapter for Files Log Files

TIBCO Hawk Agent Log Files

For Adapter for Files Components, TIBCO Silver Fabric Enabler for Adapter for Files uses TIBCO Hawk and Hawk Agent. Table 2 lists the retained Hawk log files.

Retained 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 Adapter for Files Log Files

Table 4 lists the retained File Adapter Log Files.

Retained TIBCO Adapter for Files Log Files

Name	Location	Purpose of the Log
`ApplicationManagement.log`	`$ENGINE_WORK_DIR/domaindata/tra/`*`TIBCO_DOMAIN`*`/logs`	Generated by Appmanage, which publishes File Adapter Applications.
`domainutility.log`	`$ENGINE_WORK_DIR/tibco/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 for Files 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 File Adapter and trace all activities that have happened.