TIBCO Silver® Fabric Enabler for ActiveSpaces® Enterprise Edition

User's Guide

Software Release 1.2
September 2016

Two-Second Advantage®

Contents

Preface

You can use TIBCO Silver$^{®}$ Fabric Enabler for ActiveSpaces$^{®}$ Enterprise Edition to configure ActiveSpaces as-agents as Silver Fabric components using the Silver Fabric Administration Tool. The Silver Fabric components can then be deployed on your network or cloud environment.

Topics

Related Documentation

This section lists documentation resources you might find useful.

The following documents form the TIBCO Silver Fabric Enabler for ActiveSpaces Enterprise Edition documentation set:

- *TIBCO Silver Fabric Enabler for ActiveSpaces Enterprise Edition Installation:* Read this manual for an overview of the installer and instructions on system requirements, installation requirements, installation tasks, and the uninstallation process.
- *TIBCO Silver Fabric Enabler for ActiveSpaces Enterprise Edition User's Guide:* Describes how to configure Silver Fabric components and stacks using the Enabler, the use of ActiveSpaces Enabler runtime context variables for component configuration, and how the Enabler works with shared-nothing persistence and host-aware replication.
- *TIBCO Silver Fabric Enabler for ActiveSpaces Enterprise Edition Release Notes:* Read the release notes for a list of new and changed features. This document also contains lists of known issues and closed issues for this release.

Other TIBCO Product Documentation

You might find it useful to read the documentation for the following TIBCO products:

- TIBCO ActiveSpaces documentation
- TIBCO Silver Fabric documentation

Typographical Conventions

The following typographical conventions are used in this manual.

Table 1 General Typographical Conventions

Convention	Use
ENV_NAME *TIBCO_HOME* *SF_HOME* *SFAS_HOME*	TIBCO products are installed into an installation environment. A product installed into an installation environment does not access components in other installation environments. Incompatible products and multiple instances of the same product must be installed into different installation environments. An installation environment consists of the following properties: • **Name** Identifies the installation environment. This name is referenced in documentation as *ENV_NAME*. On Microsoft Windows, the name is appended to the name of Windows services created by the installer and is a component of the path to the product shortcut in the Windows Start > All Programs menu. • **Path** The folder into which the product is installed. This folder is referenced in documentation as *TIBCO_HOME*. TIBCO Silver® Fabric installs into a directory within `TIBCO_HOME.` This directory is referenced in documentation as `SF_HOME`. The default value of SF_HOME depends on the operating system. TIBCO Silver Fabric Enabler for ActiveSpaces Enterprise Edition installs into a directory within a `TIBCO_HOME`. This directory is referenced in documentation as `SFAS_HOME`. The default value of `SFAS_HOME` depends on the operating system. For example on Windows systems, the default value is `C:\tibco\sfas\1.2`.
`code font`	Code font identifies commands, code examples, filenames, pathnames, and output displayed in a command window. For example: Use `MyCommand` to start the foo process.
`bold code font`	Bold code font is used in the following ways: • In procedures, to indicate what a user types. For example: Type **`admin`**. • In large code samples, to indicate the parts of the sample that are of particular interest. • In command syntax, to indicate the default parameter for a command. For example, if no parameter is specified, `MyCommand` is enabled: `MyCommand [`**`enable`**` | disable]`

Table 1 General Typographical Conventions (Cont'd)

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

Table 2 Syntax Typographical Conventions

Convention	Use
`[ ]`	An optional item in a command or code syntax. For example: `MyCommand [optional_parameter] required_parameter`
`|`	A logical `OR` that separates multiple items of which only one may be chosen. For example, you can select only one of the following parameters: `MyCommand para1 | param2 | param3`

Table 2 Syntax Typographical Conventions

Convention	Use

`{ }`	A logical group of items in a command. Other syntax notations may appear within each logical group.
	
	For example, the following command requires two parameters, which can be either the pair `param1` and `param2`, or the pair `param3` and `param4`.
	
	`MyCommand {param1 param2} | {param3 param4}`
	
	In the next example, the command requires two parameters. The first parameter can be either `param1` or `param2` and the second can be either `param3` or `param4`:
	
	`MyCommand {param1 | param2} {param3 | param4}`
	
	In the next example, the command can accept either two or three parameters. The first parameter must be `param1`. You can optionally include `param2` as the second parameter. And the last parameter is either `param3` or `param4`.
	
	`MyCommand param1 [param2] {param3 | param4}`

Connecting with TIBCO Resources

How to Join TIBCOmmunity

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

How to Access TIBCO Documentation

You can access TIBCO documentation here:

http://docs.tibco.com

How to Contact TIBCO Support

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

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

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

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

  https://support.tibco.com

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

Chapter 1 Introduction

This chapter provides a general overview of TIBCO Silver Fabric Enabler for ActiveSpaces Enterprise Edition. It describes how to configure Silver Fabric components and stacks using enabler, and also describes the use of ActiveSpaces Enabler runtime context variables for component configuration.

Topics

Overview

TIBCO Silver Fabric Enabler for ActiveSpaces Enterprise Edition is used to configure, start, and manage TIBCO ActiveSpaces as-agents using TIBCO Silver Fabric Administration Tool. After installing TIBCO Silver Fabric Enabler for ActiveSpaces Enterprise Edition, you can configure TIBCO Silver Fabric components and stacks according to your system architecture's use of ActiveSpaces as-agents deployed within a metaspace.

This guide assumes that you are familiar with using TIBCO Silver Fabric Administration Tool. For detailed information, see the *TIBCO Silver Fabric Cloud Administration Guide*.

To run TIBCO Silver Fabric Administration Tool, you must have a Silver Fabric Broker running and know the hostname, user name, and password. If this is not the case, see *Silver Fabric Installation Guide* or contact the administrator responsible for the Silver Fabric installation.

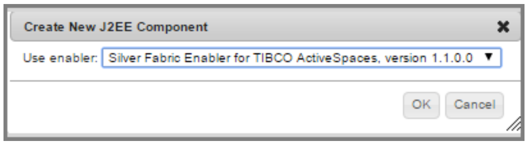
Components

TIBCO ActiveSpaces as-agents are configured as TIBCO Silver Fabric J2EE Components. When TIBCO Silver Fabric Enabler for ActiveSpaces Enterprise Edition is installed in your TIBCO Silver Fabric environment, Runtime Context Variables specific to configuring ActiveSpaces as-agents become available for configuration of a J2EE Component type. The as-agent command line options have been mapped to component Runtime Context Variables as appropriate for deployment within the TIBCO Silver Fabric environment.

To create a new Component using the ActiveSpaces Enabler, follow these steps:

1. Select **Stacks->Components**. In the **Global Actions** drop-down list, select **Create New J2EE Component**. In the Create New J2EE Component pop-up, select **Silver Fabric Enabler for TIBCO ActiveSpaces, version 1.2.0.0** and Click **OK.**

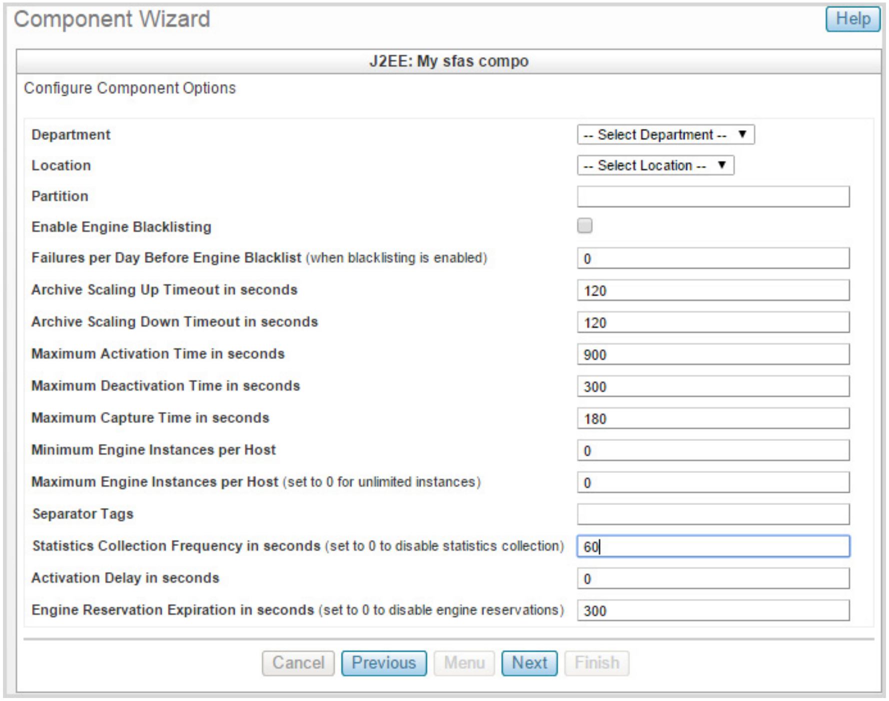
Figure 1 J2EE Active Space Component


Create New J2EE Component

Use enabler: Silver Fabric Enabler for TIBCO ActiveSpaces, version 1.1.0.0

2. In the Configure General Properties window, name your component and provide a description. Click **Next** until you come to the Configure Component Options window. Change the **Statistics Collection Frequency** `in seconds` value from 10 to 60.

Figure 2 Component Wizard



Component Wizard

J2EE: My sfas compo

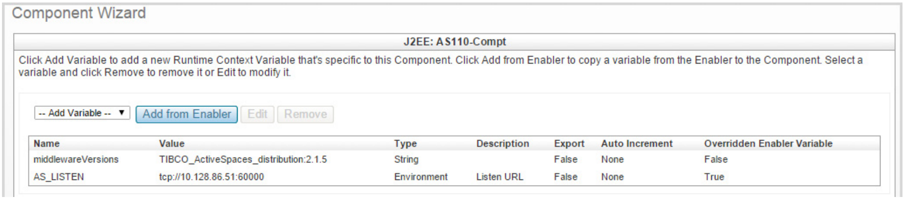
Configure Component Options

Department	-- Select Department --
Location	-- Select Location --
Partition	
Enable Engine Blacklisting	☐
Failures per Day Before Engine Blacklist (when blacklisting is enabled)	0
Archive Scaling Up Timeout in seconds	120
Archive Scaling Down Timeout in seconds	120
Maximum Activation Time in seconds	900
Maximum Deactivation Time in seconds	300
Maximum Capture Time in seconds	180
Minimum Engine Instances per Host	0
Maximum Engine Instances per Host (set to 0 for unlimited instances)	0
Separator Tags	
Statistics Collection Frequency in seconds (set to 0 to disable statistics collection)	60
Activation Delay in seconds	0
Engine Reservation Expiration in seconds (set to 0 to disable engine reservations)	300

3. Click **Next** until you come to the window in which you can add/override/edit enabler and component-specific runtime context variables. If you want to specify a distribution version which should be used, please follow below instruction, otherwise proceed to the step 5.

4. Click the **Add Variable** drop-down list and select **String**, in Add/Edit Variable dialog, enter "middlewareVersions" in the **Name** field and "`TIBCO_ActiveSpaces_distribution:<Distribution_Version>`" in the **Value** field. Then Click **OK** to save the changes.

Figure 3 Add variable to runtime context variable



Component Wizard

J2EE: AS110-Compt

Click Add Variable to add a new Runtime Context Variable that's specific to this Component. Click Add from Enabler to copy a variable from the Enabler to the Component. Select a variable and click Remove to remove it or Edit to modify it.

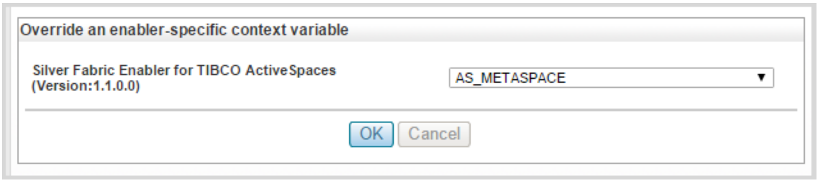
Name	Value	Type	Description	Export	Auto Increment	Overridden Enabler Variable
middlewareVersions	TIBCO_ActiveSpaces_distribution:2.1.5	String		False	None	False
AS_LISTEN	tcp://10.128.86.51:60000	Environment	Listen URL	False	None	True

Example: "TIBCO_ActiveSpaces_distribution:2.1.6

Middleware Version is optional and if it is not specified, the latest distribution version is used.

5. Click **Add from Enabler**. In the pop-up that appears, select **AS_METASPACE** in the drop down list of variables for the enabler and click **OK**.

Figure 4 Override an enabler specific context variable

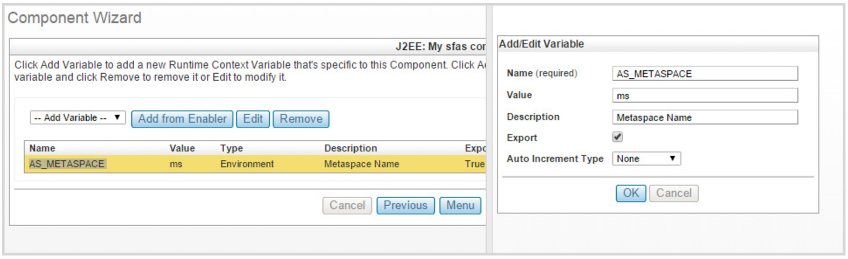


Override an enabler-specific context variable

Silver Fabric Enabler for TIBCO ActiveSpaces (Version:1.1.0.0)	AS_METASPACE

6. Highlight **AS_METASPACE** by clicking on it, and then click **Edit**. Type the desired metaspace name in the **Value** field (default: ms) and click **OK**.

Figure 5 Edit AS_METASPACE



Component Wizard

J2EE: My sfas cor

Click Add Variable to add a new Runtime Context Variable that's specific to this Component. Click A variable and click Remove to remove it or Edit to modify it.

Name	Value	Type	Description	Expo
AS_METASPACE	ms	Environment	Metaspace Name	True

Add/Edit Variable

Name (required)	AS_METASPACE
Value	ms
Description	Metaspace Name
Export	✓
Auto Increment Type	None

7. Repeat the above two steps, to add any additional Runtime Context Variables and specify their settings and click **Finish**.

8. Select the Gear icon next to the component you just configured and select **Publish Component**.

See ActiveSpaces Enabler Runtime Context Variables, page 8 to find out more about available options for configuring a component that uses the ActiveSpaces Enabler.

Stacks

TIBCO Silver Fabric Stack is used to deploy a single TIBCO ActiveSpaces metaspace environment. A stack might have only one ActiveSpaces Enabler Component, but will probably have several ActiveSpaces Enabler components configured in TIBCO Silver Fabric Stack. In TIBCO Silver Fabric Stack, ActiveSpaces Enabler Components can be configured to run on independent machines, or the same machine, as individual components or as components dependent upon other components.

To create a new Stack, follow these steps:

1. Select **Stacks->Stacks**. Click **Create New Stack**. In the **Name** field, enter a name for your Stack.
2. Click on the plus sign (**+**) next to **Components** to show a list of the Components you have configured. Click on your ActiveSpaces component under **Available Components** then click **>>** to add it to the list of Selected Components.
3. Click the **Policies** tab. Click on the plus sign (+) to display the list of components selected for this stack.

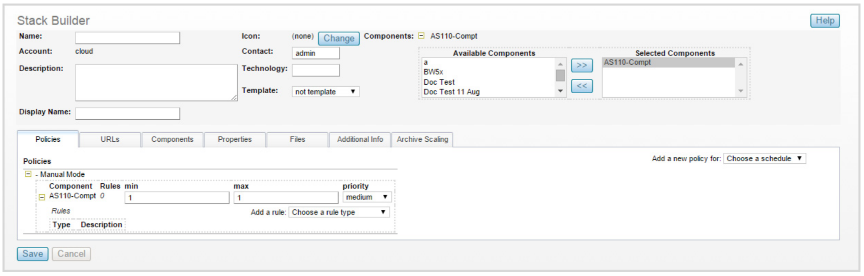


Figure 6   Stack Builder

Stack Builder

Name:
Account: cloud
Description:
Display Name:
Icon: (none)
Contact: admin
Technology:
Template: not template
Components: AS110-Compt

Available Components	Selected Components
a	AS110-Compt
BW5x	
Doc Test	
Doc Test 11 Aug	

Policies

Add a new policy for: Choose a schedule

- Manual Mode

Component	Rules	min	max	priority
AS110-Compt	0	1	1	medium

Rules

Add a rule: Choose a rule type

Type	Description

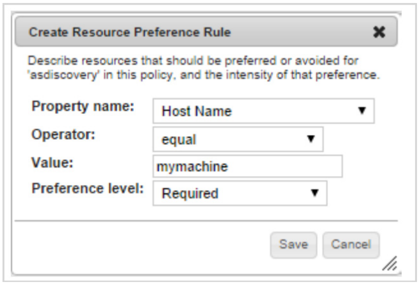
4. Set `min` and `max` to the minimum and maximum number of engines allocated to each component.
5. Set the component priority relative to other components.
6. Click **Choose a rule type** to display a drop-down list of rules to add for this component.
7. (Optional) Select **Component Dependency** to specify whether this component is dependent upon another component starting up first.
8. (Optional) Select **Resource Preference** to specify property settings to affect the running of component instances. Click **Save** when your stack configuration is complete. Select the Gear icon next to the stack you just configured, and click **Publish Stack**.

9. When you are ready to run your stack, select the Gear icon next to the stack and click **Run Stack in Manual Mode**.

Controlling Where Components Run

The TIBCO Silver Fabric Stack configuration controls which machines your as-agents run on. For scenarios where you need to run as-agents on different machines (for example, for host-aware replication), it is good practice to create separate component configurations that are then configured in the stack to run on separate machines. You can do this by adding a Resource Preference rule to the component that specifies the Host Name of the machine that the component should run on.

Figure 7 Creating Resource Reference Rule (Host)



Create Resource Preference Rule

Describe resources that should be preferred or avoided for 'asdiscovery' in this policy, and the intensity of that preference.

Property name: Host Name
Operator: equal
Value: mymachine
Preference level: Required

When running in the cloud, it might not be possible to know in advance the host name of the machine you want your component instances to run on. In this case, have your Asset Manager use a Silver Fabric property (for example, Group) and assign machines in the cloud to have different predetermined property values as they are started.

For example, when the first machine in the cloud starts, have the Asset Manager configure the machine so that it uses the Group property with a value of `Machine1`, the second machine so that it uses the Group property with a value of `Machine2`, and so on. Then add a Resource Preference rule to the component in the stack configuration that specifies the Group that engines should belong to when determining the engines on which the component should run.

In Silver Fabric Administration Tool, follow these steps to set the value of a daemon property:

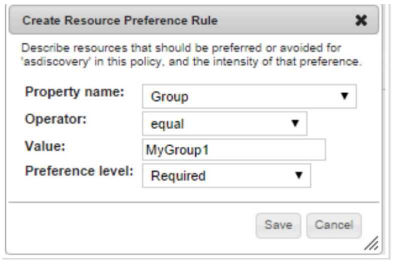
1. Select **Engines->Daemons**. Click on the **Gear** icon next to the daemon whose property you want to set.
2. Select **Edit/View Properties**. On the Engine Properties window, click **New**. Click the down arrow of the **Property** field and select the property you want to set from the list displayed.

3. In the **Value** field, enter the value for the property and click **Save**.

Any engines started for that daemon will have that property and value. You can now configure component instances to run only on engines with that particular property value by adding a Resource Preference rule to the component in your stack configuration.

For example, if you configured one of the daemons to have a Group property with a value of MyGroup1, then to constrain component instances to run only on engines of that daemon, you would configure the component in the stack with a Resource Preference rule as shown in the following figure:

Figure 8 Creating Resource Reference Rule (Group)


Create Resource Preference Rule

Describe resources that should be preferred or avoided for 'asdiscovery' in this policy, and the intensity of that preference.

Property name:	Group
Operator:	equal
Value:	MyGroup1
Preference level:	Required

ActiveSpaces Enabler Runtime Context Variables

When you run an as-agent from the command line, specify options that tell the as-agent the name of the metaspace to connect to, the discovery URL for the metaspace, the member name to use for the as-agent, and so on. To run as-agents from the TIBCO Silver Fabric environment, ActiveSpaces Enabler maps the as-agent command line options to Component Runtime Context Variables. The following is a list of the as-agent command line options and how they have been mapped to Component Runtime Context Variables.

Table 3 Enabler Runtime Context Variables

Command Line Option	Runtime Context Variable	Default
-metaspace *<metaspace_name>*	`AS_METASPACE`	“ms”
-member_name *<member_name>*	`AS_MEMBER_NAME_PREFIX`	*<component_name>*
-discovery *<url /url_list>*	`AS_DISCOVERY`	
	`AS_DISCOVERY_PORT`	52000
	`AS_DISCOVERY_MULTICAST_ADDRESS`	239.8.8.8
	`AS_MULTICAST_DISCOVERY_URL`	tibpgm://<AS_DISCOVERY_PORT>/<*engine_IP_address*>;<AS_DISCOVERY_MULTICAST_ADDRESS>
	`AS_NUM_DISCOVERY_NODES`	0
	`AS_TOTAL_DISCOVERY_NODES`	0
	`AS_SHARED_DISCOVERY_DIR`	
	`AS_FIND_DISCOVERY_MAX`	5
	`AS_FIND_DISCOVERY_WAIT`	30 seconds
	`AS_DISCOVERY_NODE_RECOVERY_TRIES`	0

Table 3 Enabler Runtime Context Variables

Command Line Option	Runtime Context Variable	Default
-listen <*url*>	AS_LISTEN AS_LISTEN_PORT	 5700
-remote_listen <*url*>	AS_REMOTE_LISTEN AS_REMOTE_LISTEN_PORT	 -32768
-data_store <*directory_path*>	AS_DATA_STORE	
-autojoin.role <role>	AS_AUTOJOIN_ROLE	
-log <*file_path*>	AS_LOG	
-log_debug <*log_level*>	AS_LOG_DEBUG	3 (INFO)
-log_limit <*limit*>	AS_LOG_LIMIT	-1 (no limit)
-log_count <*count*>	AS_LOG_COUNT	1
-log_append <*boolean*>	AS_LOG_APPEND	true
-debug <*log_level*>	AS_DEBUG	3 (INFO)
-worker_thread_count <*count*>	AS_WORKER_THREAD_COUNT	32
-rx_buffer_size <*size*>	AS_RX_BUFFER_SIZE	2 MB
-member_timeout	AS_MEMBER_TIMEOUT	30000
-cluster_suspend_threshold	AS_CLUSTER_SUSPEND_THRESHOLD	-1
-monitor_system <*boolean*>	AS_MONITOR_SYSTEM	false
-input <*script_path*>	AS_INPUT	
-security_policy	AS_SECURITY_POLICY	
-security_token	AS_SECURITY_TOKEN	
-identity_password	AS_IDENTITY_PASSWORD	

Table 3 Enabler Runtime Context Variables

Command Line Option	Runtime Context Variable	Default
-authentication_domain	`AS_AUTHENTICATION_DOMAIN`	
-authentication_username	`AS_AUTHENTICATION_USERNAME`	
-authentication_password	`AS_AUTHENTICATION_PASSWORD`	
-authentication_keyfile	`AS_AUTHENTICATION_KEYFILE`	
-site_name	`AS_SITE_NAME`	
-system_seeder	`AS_SYSTEM_SEEDER`	

For the following settings, use only characters that can be used in a file or directory name, and do not use spaces:

- `Component Name`
- `AS_METASPACE`
- `AS_MEMBER_NAME_PREFIX`

Some variables that are used internally by ActiveSpaces Enabler appear in the list of Component Runtime Context Variables. Do not add the following Runtime Context Variables to a component configuration:

- `AS_BASE`
- `AS_HOME`

Notice that most of the as-agent command line options map to a Runtime Context Variable of the same name, with "AS_" prepended. For example, the `-metaspace` command line option maps to the `AS_METASPACE` Runtime Context Variable. These directly mapped Runtime Context Variables are used and behave in the same way as if they were specified as as-agent command line options.

For some as-agent command line options, there is a mapping of the command line option to a Runtime Context Variable with a slightly different name. For other as-agent command line options, there are several Runtime Context Variables listed.

The following sections provide information about the use of these Runtime Context Variables.

Metaspace Member Names

With ActiveSpaces, each metaspace member must have a unique member name. Since multiple instances of an as-agent can be run by a single component configuration that uses ActiveSpaces Enabler, each component instance that is started has the following member name format:

<member_name_prefix>-*<component instance number>*

You can specify the member name prefix by using the `AS_MEMBER_NAME_PREFIX` Runtime Context Variable.

If a member_name_prefix is not specified, the name of the Component is used as the member name prefix.

When using ActiveSpaces Enabler, Silver Fabric reuses component instance numbers when component instances go down and come back up. So if four instances of a component are started, they have names in which 0 through 3 are appended to the member name prefix. If any of the instances go down and come back up, you will still have instances whose member names end in 0 through 3.

ActiveSpaces Enabler cannot catch member name prefixes that are not unique between components that connect to the same metaspace. If you specify a member name prefix instead of allowing ActiveSpaces Enabler to use the default member name prefix, ensure that you specify a unique member name prefix.

Depending on the scenario, it might not be possible for ActiveSpaces Enabler to provide a meaningful error to help you figure out why your component instances are failing to run.

When using shared-nothing persistence, directories and files are created based on the member name. So member name prefixes should only contain characters that are valid for directory and file names on the system you are running on.

Listen URL

When an as-agent is deployed from the command line, the following default listen URL is used:

```
tcp://<host_ip_address>:<first free TCP port beginning from port
50000>
```

When deploying as-agents using a TIBCO Silver Fabric Component, if a single listen URL is specified, errors are generated for each subsequent as-agent started on the same machine, because the first as-agent started usese the port specified in the listen URL and the listen port is no longer available.

For this reason, when configuring components with ActiveSpaces Enabler, specify only the beginning TCP listen port to use instead of the complete listen URL. When component instances are started, the complete listen URL is composed using the following information:

- The host IP address of the engine on which the as-agent is started
- A TCP port starting from the port you specify as the value for the `AS_LISTEN_PORT` Runtime Context Variable. If you do not specify a value for `AS_LISTEN_PORT`, a default value of 57000 is used.

- `AS_LISTEN` can be used to specify the entire listen URL. If a value for `AS_LISTEN` is specified, `AS_LISTEN_PORT` is ignored.
- When running multiple components on the same machine, ensure that `AS_LISTEN_PORT` specifies a different range of port numbers for each component; otherwise port conflicts occurs.

Remote Listen URL

The remote listen URL is used to specify the IP address and port on which an as-agent listens for connections from remote clients. When using ActiveSpaces Enabler, the `AS_REMOTE_LISTEN` Runtime Context Variable is used to specify the remote listen URL. If you specify a value for AS_REMOTE_LISTEN, then that same value is used for every as-agent started using that component configuration. You could run into port conflicts if these as-agents end up running on the same machine.

When running as-agents in the cloud, you might not be able to use `AS_REMOTE_LISTEN` unless you use Elastic IP addresses, because you will not know ahead of time the IP address of the engine an as-agent will be started on.

Therefore, in most cases you should use the Runtime Context Variable `AS_REMOTE_LISTEN_PORT` instead of `AS_REMOTE_LISTEN` to compose the remote listen URL.

When you use `AS_REMOTE_LISTEN_PORT`, the remote listen URL is constructed as follows:

```
tcp://<engine IP address>:<AS_REMOTE_LISTEN_PORT>
```

For each instance of a component that is started, `AS_REMOTE_LISTEN_PORT` increments to prevent port conflicts. You will notice that when you add `AS_REMOTE_LISTEN_PORT` to your component configuration, it is set to a negative number. You must set this value to a positive port number for `AS_REMOTE_LISTEN_PORT` to be recognized by the enabler.

If neither `AS_REMOTE_LISTEN_PORT` nor `AS_REMOTE_LISTEN` is added to your component configuration, then a remote listen URL is not included when starting up as-agents using the component configuration.

Metaspace Discovery URL

Several runtime context variables related to specifying the metaspace discovery URL when starting as-agents in the Silver Fabric environment. By default, Pragmatic General Multicast (PGM) multicast transport is used for discovery if none of the discovery related Runtime Context Variables are configured. The default PGM discovery URL has the form:

```
tibpgm://<AS_DISCOVERY_PORT>/<engine IP
address>;<AS_DISCOVERY_MULTICAST_ADDRESS>
```

where:

- `AS_DISCOVERY_PORT defaults to 52000`
- `AS_DISCOVERY_MULTICAST_ADDRESS defaults to 239.8.8.8`

When running in the cloud, using PGM for discovery will not work, because multicast is not supported in the cloud.

`AS_DISCOVERY` runtime context variable is used to configure a specific discovery URL that all as-agents started using this component configuration use. See the *TIBCO ActiveSpaces Developer's Guide* for more detailed information on discovery URLs.

If a component does not configure `AS_DISCOVERY` and another component on which it depends exports its `AS_DISCOVERY` setting, the exported `AS_DISCOVERY` setting is used by the component that did not configure its own `AS_DISCOVERY` setting. This scenario is likely to occur when using TCP for discovery in the cloud. See the next section for more information.

TCP Discovery in the Cloud

You must use TCP discovery when running as-agents in the cloud. A TCP discovery URL has the form:

```
tcp://<IP1[:port1]>;<IP2[:port2];...
```

When using TCP discovery for as-agents configured to run in the cloud, it is not possible to know ahead of time what the IP address of your cloud machine will be unless you use something like elastic IP addresses.

In cases where the IP address of the cloud machine is not known ahead of time, ActiveSpaces Enabler dynamically determines the TCP discovery URL in the following cases:

- `AS_DISCOVERY` is not configured
- `AS_SHARED_DISCOVERY_DIR` and `AS_NUM_DISCOVERY_NODES` are configured

`AS_SHARED_DISCOVERY_DIR` specifies a directory on a shared drive where each as-agent about to be started can register its listen address and port. When the number of as-agents that have registered reaches the amount specified in `AS_NUM_DISCOVERY_NODES`, the ActiveSpaces Enabler uses the listen addresses and ports of those as-agents to calculate and publish the discovery URL in `AS_DISCOVERY` Runtime Context Variable so that dependent components can use the dynamically determined discovery URL.

Use `AS_TOTAL_DISCOVERY_NODES` when you have multiple components in your stack and you only want a certain number of discovery nodes from each component to be used. In this case, `AS_NUM_DISCOVERY_NODES` is used to determine the number of as-agents from each component to use as discovery nodes, and `AS_TOTAL_DISCOVERY_NODES` is used to specify the total number of discovery nodes needed for the complete discovery URL.

The following configuration scenarios are useful for understanding the expected behavior of the ActiveSpaces Enabler when you configure `AS_NUM_DISCOVERY_NODES` and `AS_TOTAL_DISCOVERY_NODES`.

If a component is configured as follows:

```
AS_NUM_DISCOVERY_NODES = 0
AS_TOTAL_DISCOVERY_NODES = 4
```

then none of the as-agents started by this component are used as discovery nodes. ActiveSpaces Enabler waits until the full discovery URL has been determined by the other components in the stack before starting any as-agents using this component configuration.

If a component is configured as follows:

```
AS_NUM_DISCOVERY_NODES = 2
AS_TOTAL_DISCOVERY_NODES = 2
```

then the first two instances of this component are used to determine the discovery URL for the metaspace.

None of the other components that want to join this metaspace should set any of the discovery URL related Runtime Context Variables. AS_DISCOVERY is exported by this component and is picked up by other components in the stack for which a dependency rule on this component has been configured.

When configuring dependency rules, remember to clear the **Shutdown dependenc**y option. If it is selected and a discovery node goes down, any dependent component instances will also be shut down.

If a component is configured as follows:

```
AS_NUM_DISCOVERY_NODES = 2
AS_TOTAL_DISCOVERY_NODES = 4
```

then the first two instances of this component are used as discovery nodes. ActiveSpaces Enabler waits until the total number of discovery nodes have been registered by other components. When the total number of discovery nodes is reached, the full discovery URL can be determined and all as-agents start up using the full discovery URL.

If a component is configured as follows:

```
AS_NUM_DISCOVERY_NODES = 2
AS_TOTAL_DISCOVERY_NODES = 0
```

then the setting from `AS_NUM_DISCOVERY_NODES` is used for `AS_TOTAL_DISCOVERY_NODES` and the first two instances of this component are used to determine the discovery URL for the metaspace.

Even though ActiveSpaces Enabler dynamically determines the discovery URL when running in the cloud, if a discovery node goes down and does not come back up on the same IP address, you will lose that discovery node. If the discovery node comes back up on a different IP address, it will now be considered a member of the metaspace but not a discovery node. If you lose all of your discovery nodes, metaspace members will not be able to connect to the metaspace. If this happens, stop your stack and start it again.

Note the following about `AS_DISCOVERY_NODE_RECOVERY_TRIES`:

- Use `AS_DISCOVERY_NODE_RECOVERY_TRIES` to designate the number of times to restart a discovery node. The default value, which is 0, designates that no discovery nodes will be restarted.
- When you use `AS_DISCOVERY_NODE_RECOVERY_TRIES`, the enabler will restart the component instance and attempts to recover the discovery node on an engine using the same IP address as the engine the discovery node originally ran on. If the discovery node cannot be recovered, the metaspace member runs as a non-discovery node. However, if the discovery node restarts on an engine with a different IP address, it starts as a non-discovery node member of the metaspace.

The runtime context variables `AS_FIND_DISCOVERY_MAX` and `AS_FIND_DISCOVERY_WAIT` are used to control how long to wait for the discovery URL to be determined. `AS_FIND_DISCOVERY_MAX` is the number of times the enabler loops looking for the total number of discovery nodes to have registered themselves. `AS_FIND_DISCOVERY_WAIT` is the number of seconds the enabler waits before looking again for all of the discovery nodes.

If you are running your components on different machines and some machines run faster than others, you might need to increase `AS_FIND_DISCOVERY_WAIT` or `AS_FIND_DISCOVERY_MAX` to increase the amount of time allowed for determining the discovery URL.

Cross-site Replication

Cross-site replication is used to dynamically backup changes to the data in your metaspace to another metaspace in a different geographical location.

The runtime context variable `AS_SITE_NAME` can be used to specify the ActiveSpaces site name. You can pass an argument `-site_name <string>` to the as-agent.

The runtime context variable `AS_SYSTEM_SEEDER` is a setting, that allows you to set if the as-agent seeds any internal system spaces.You can add the variable `AS_SYSTEM_SEEDER` to component, and configure its value. If you set its value to False, the non-leach as-agents do not seed the cross-site related system spaces.

You can refer *TIBCO ActiveSpaces® Administration* for more details.

Security

You cannot use ActiveSpaces enabler to deploy as-agents that use security in the cloud, unless you use static IP addresses. These IP addresses are similar to those provided with Amazon's Elastic IP addresses.

ActiveSpaces requires the use of TCP discovery to apply security to metaspaces. The Metaspace Access List in the security policy and token files requires the metaspace name and full discovery URL. So the discovery URL must be known in advance and cannot be dynamically determined at runtime as described in TCP Discovery in the Cloud on page 13.

AS_SECURITY_POLICY specifies the full path to the security policy file. This file needs to be accessible to each as-agent being started as a security domain controller for the metaspace. If a single component configuration is being used to start multiple as-agents which will act as security domain controllers running on different machines, a shared drive should be used to store the security policy file so that all instances of as-agents being deployed will have access to the security policy file as configured in the AS_SECURITY_POLICY configuration setting.

At least one security domain controller must be a discovery node so that security can be provided as other members try to join the metaspace.

The security policy file can be uploaded to component as a content file. For more details refer Security Runtime Context Variable, page 19.

AS_IDENTITY_PASSWORD specifies the policy's security domain's identity password if the identity is encrypted.

For component configured as security domain requester, set below security runtime context variables:

- AS_SECURITY_TOKEN
- AS_IDENTITY_PASSWORD

When you are using static discovery URL for the security domain controller component, you must update the discovery URL in the policy file manually.

At least one security domain controller must be a discovery node so that security can be provided as other members try to join the metaspace.

AS_SECURITY_TOKEN specifies the full path to the security token file. This must be accessible to each as-agent being started as a security domain requester for the metaspace. If a single component configuration is being used to start multiple as-agents which are security domain requesters running on different machines, a shared drive should be used to store the security token file. Then all instances of as-agents that are deployed, have access to the security token file as configured in the `AS_SECURITY_TOKEN` configuration setting. The security token file can be uploaded to component as a content file. For more details refer Security Runtime Context Variable, page 19.

`AS_IDENTITY_PASSWORD` specifies the policy's security domain's or the token's identity password if the identity is encrypted. When security domain controller is using user authentication, add below security runtime context variables to security domain requester component:

- `AS_AUTHENTICATION_DOMAIN`
- `AS_AUTHENTICATION_USERNAME`
- `AS_AUTHENTICATION_PASSWORD`

`AS_AUTHENTICATION_DOMAIN` specifies the name of the windows domain to log into. If local/ntlm account (as per the controller), "." can be used. If not windows, it is ignored.

`AS_AUTHENTICATION_USERNAME` specifies the authentication username of the user account to be logged in as.

`AS_AUTHENTICATION_PASSWORD` specifies the authentication password of the user account to be logged in as or password for user account.

When security domain controller is using certificate authentication, add below security runtime context variables to security domain requester component:

- `AS_AUTHENTICATION_KEYFILE`
- `AS_AUTHENTICATION_PASSWORD`

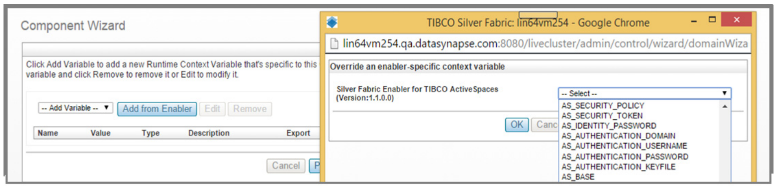
`AS_AUTHENTICATION_KEYFILE` specifies the pkcs12 keyfile location of the user to be logged in as (sasl/external x509 ldap auth). The key file can be uploaded to component as a content file. For more details refer Security Runtime Context Variable, page 19.

`AS_AUTHENTICATION_PASSWORD` specifies the password of the pkcs12 key file.

By default, identity passwords and authentication passwords are visible in the engine log as part of the command to start up the as-agent. You should always use safe passwords when configuring your components. Safe passwords can be generated using the as-admin 'create safe_password' command. For more details refer TIBCO ActiveSpaces ® Administration Guide.

To prevent the display of the command used to start up the as-agent in the engine logs, add the runtime context variable SHOW_PROCESS_COMMAND to your component configuration and set its value to false.

Figure 9   Security Runtime Context Variable



Component Wizard

Click Add Variable to add a new Runtime Context Variable that's specific to this variable and click Remove to remove it or Edit to modify it.

Name	Value	Type	Description	Export

TIBCO Silver Fabric: lin64vm254 - Google Chrome

lin64vm254.qa.datasynapse.com:8080/livecluster/admin/control/wizard/domainWiza

Override an enabler-specific context variable

Silver Fabric Enabler for TIBCO ActiveSpaces (Version:1.1.0.0)

-- Select --
AS_SECURITY_POLICY
AS_SECURITY_TOKEN
AS_IDENTITY_PASSWORD
AS_AUTHENTICATION_DOMAIN
AS_AUTHENTICATION_USERNAME
AS_AUTHENTICATION_PASSWORD
AS_AUTHENTICATION_KEYFILE
AS_BASE

Uploading Content Files for Security

The security policy, token file and authentication key file can be uploaded and deployed with component. To upload the files and to set the runtime context variables perform the following steps:

1. Set the value of `AS_SECURITY_POLICY`, `AS_SECURITY_TOKEN` and `AS_IDENTIFICATION_KEYFILE` as:

`$CONTAINER_WORK_DIR/<RELATIVE_PATH>/<FILE_NAME>`

   Where,

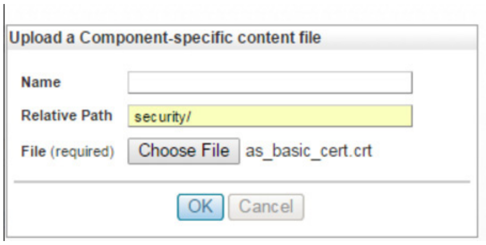
   `$CONTAINER_WORK_DIR` is a runtime context variable defined in ActiveSpaces enabler for work directory. Its default value is DSEngine/work/<hostname>-<instance>/fabric.

   `<RELATIVE_PATH>` is the relative path that is set when uploading content file to the component.

2. Navigate to the Add/override/customize Enabler and Component-specific content files page in the component creation wizard, click **Upload**.

3. Specify a string in the Relative Path field, and upload the file. If the Name field is kept blank, then the default file name is used.

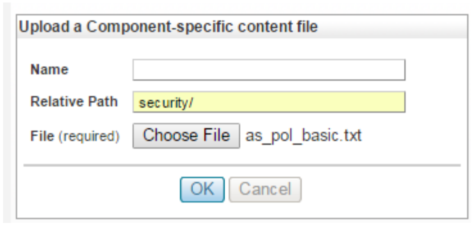
Figure 10   Uploading Component Specific Certificate File



Upload a Component-specific content file

Name	
Relative Path	security/
File (required)	Choose File as_basic_cert.crt

Figure 11 Uploading Component Specific Policy File

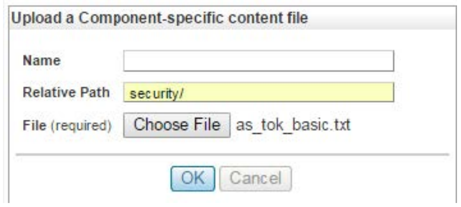

Upload a Component-specific content file

Name

Relative Path security/

File (required) as_pol_basic.txt

Figure 12 Uploading Component Specific Token File

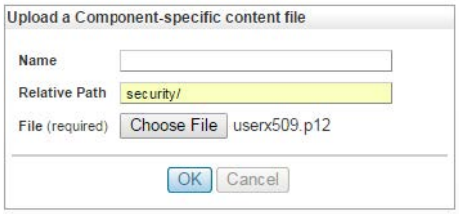

Upload a Component-specific content file

Name

Relative Path security/

File (required) as_tok_basic.txt

Figure 13 Uploading Component Specific Key File


Upload a Component-specific content file

Name

Relative Path security/

File (required) userx509.p12

Restricting Transport Access

Using TIBCO ActiveSpaces security, you can restrict transport connections within a security domain to only "trusted" nodes. The full path of trusted certificate file set in the security policy file must be accessible to each as-agent being started as a security domain controller for the metaspace. The trusted certificate file can be located in a shared drive or can be uploaded to the component.

To configure the restricted transport access and to upload the trusted certificate file to component:

1. Open the security policy file for the domain in a text editor.
2. Edit the line that reads transport_access=false;cert_file=

as

```
transport_access=true;cert_file=%CONTAINER_WORK_DIR%/<RELATIVE_PAT
```

`H>/<TRUSTED_CERTS_FILE_NAME>`

Where,

`%CONTAINER_WORK_DIR%` is the placeholder container work directory for substitution

`<RELATIVE_PATH>` is the relative path set when uploading the trusted certificates file.

`<TRUSTED_CERTS_FILE_NAME>` is the name of the trusted certificates file.

3. Upload the security policy file and trusted certificates file to Silver Fabric while creating the component.
4. In the Edit the Configuration File wizard page, add container configuration

script shown as below:

```
<containerConfig>

<configFiles baseDir="${CONTAINER_WORK_DIR}/<RELATIVE_PATH>"

include="<SECURITY_POLICY_FILE_NAME>">

<regex pattern="%CONTAINER_WORK_DIR%"

replacement="${CONTAINER_WORK_DIR}"/>

</configFiles>

</containerConfig>
```

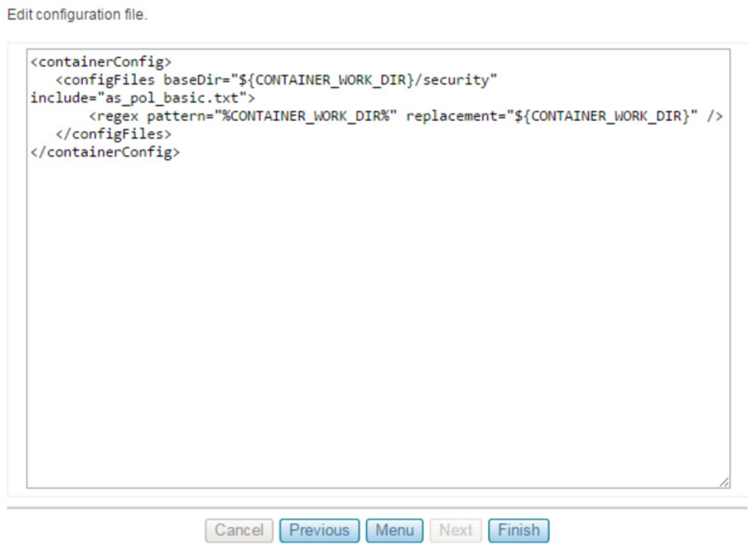
This script finds the security policy file from basiDir and substitutes the placeholder with the actual value.

Where,

`<RELATIVE_PATH>` is the relative path that is set when uploading the security policy file to component.

`<SECURITY_POLICY_FILE_NAME>` is the name of the security policy file uploaded to component.

Figure 14 Edit Configuration File


Edit configuration file.

```
<containerConfig>
   <configFiles baseDir="${CONTAINER_WORK_DIR}/security"
include="as_pol_basic.txt">
       <regex pattern="%CONTAINER_WORK_DIR%" replacement="${CONTAINER_WORK_DIR}" />
   </configFiles>
</containerConfig>
```

Alternative Memory Manager for Linux 64-bit

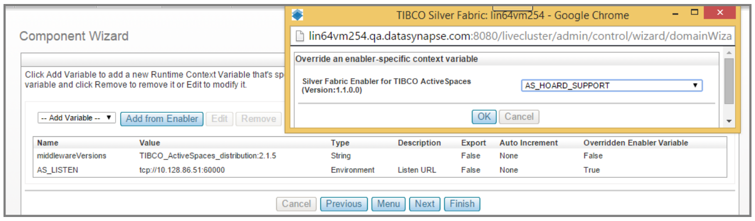
The Hoard memory manager is offered for the Linux 64-bit platform as an alternative to the operating system's native memory manager for use with ActiveSpaces. Using Hoard over the native manager can reduce memory fragmentation. Detailed information about Hoard can be found at http://.hoard.org

A runtime context variable AS_HOARD_SUPPORT is added to the TIBCO Silver Fabric Enabler for ActiveSpaces.

You can use the Add from Enabler button on the Component Wizard and select AS_HOARD_SUPPORT from the enabler specific context variable drop-down list. Click **OK** for the Hoard support.

By default, the hoard support is not provided and it will be OFF. To enable the usage of hoard, set the value to ON by adding AS_HOARD_SUPPORT variable from enabler, and updating its value to ON.

Figure 15   Hoard Support Runtime Context Variable


TIBCO Silver Fabric: lin64vm254 - Google Chrome
lin64vm254.qa.datasynapse.com:8080/livecluster/admin/control/wizard/domainWiza
Component Wizard
-- Add Variable --
Add from Enabler
Edit
Remove
OK
Cancel
Cancel
Previous
Menu
Next
Finish

Click Add Variable to add a new Runtime Context Variable that's sp
variable and click Remove to remove it or Edit to modify it.

Override an enabler-specific context variable

Silver Fabric Enabler for TIBCO ActiveSpaces (Version:1.1.0.0): AS_HOARD_SUPPORT

Name	Value	Type	Description	Export	Auto Increment	Overridden Enabler Variable
middlewareVersions	TIBCO_ActiveSpaces_distribution:2.1.5	String		False	None	False
AS_LISTEN	tcp://10.128.86.51:60000	Environment	Listen URL	False	None	True

Chapter 2 Shared-Nothing Persistence and Host Aware Replication

This chapter discusses using TIBCO Silver Fabric Enabler for ActiveSpaces Enterprise Edition to configure Silver Fabric components for use with shared-nothing persistence and host-aware replication.

Topics

ActiveSpaces Shared-Nothing Persistence

Metaspace members acting as seeders of a space use shared-nothing persistence to hold a copy of the data of the space on disk. Use shared-nothing persistence for caching of space data, where the most frequently used data is kept in memory and less used data is kept on disk.

You can also use shared-nothing persistence to recover the space members' data, when all of the members of a space have gone down for some reason; for example, software maintenance.

Persisted Data Store

When using shared-nothing persistence, specify an existing directory path to which ActiveSpaces has read and write access as the root directory for shared-nothing persisted data files.

When configuring as-agent components using the ActiveSpaces Enabler, the shared-nothing root directory is specified using `AS_DATA_STORE` runtime context variable. If a directory is not specified, the user's home directory is used.

When as-agents store data for a space where shared-nothing persistence is being used, the data the as-agent “seeds” is stored in a file with the following naming format:

```
<root_directory>/<metaspace_name>/<space_name>/<member_name>/<memb
er_name>_store_<timestamp>
```

When to Use a Shared Drive

For recovery to occur, a component instance must be able to find its local data store when it goes down and comes back up. If a stack is configured so that a component instance can be started on engines of several different machines, then the root directory for shared-nothing persistence should reside on a shared drive that is accessible by all of the machines. In that way, no matter which machine a component instance runs on, it has access to its persisted data file.

When using shared-nothing persistence, ensure that you create a component configuration, where as-agents can be started on different machines. Instead, map your component to a specific machine as described in the next section and use a local directory on that machine as your shared-nothing persistence data store.

When running in the Cloud, store the persisted data where it is available or have a means of restoring the shared-nothing persisted files on another machine brought up with the same IP address. This prevents the loss of your persisted data when your cloud machine stops. You can use a shared drive for the persisted data store or you can use shared-all persistence instead.

Mapping Components to Machines

When not running in the cloud and the use of a shared drive is not desirable, configure the stack so that each component can only be started on a specific machine. This can be done by following these steps using Silver Fabric Administration Tool:

1. Select **Stack->Stack**. Click **Create New Stack**.
2. Expand **Components**. Under **Available Components**, highlight your component created using the AS Enabler by clicking on it.
3. Click **>>.** Your component is now listed under **Selected Components**.
4. Click the **Policies** tab.Expand your stack by clicking on the plus sign (**+**) next to its name.Expand your component by clicking on the plus sign (+) next to its name.Specify the 'min' and 'max' number of instances to be started for the stack.
5. In the **Add a rule** drop-down list, select **Resource Preference**. Set the following values for the Resource Preference:
    a. Property name: Host Name
    b. Operator: equal
    c. Value: <your daemon's/machine's host name>
    d. Preference level: Required
6. Click **Save**. After configuring the stack, click **Save** at the bottom of the Stack Builder screen.

Regardless of whether or not you are running in the cloud, if you replace a machine, instead of using the `Host Name` property in the rule in step 11a, consider using a property, such as the Group property, that you set to a unique value for each machine or daemon. Then, as long as the new machine also gets configured to have the same Group property value when it replaces the original machine, you do not have to modify your stack configuration when the machines are exchanged. When running in the cloud, your asset manager should take care of setting up this property when machines are allocated.

ActiveSpaces Host-Aware Replication

Overview

Host-aware replication is used to group seeders of a space together so that the data being stored by the seeders in one group is replicated on seeders in a different group. This helps to prevent data loss when a group of seeders goes down, because their data has been replicated elsewhere.

By default, data is not replicated when it is put into a space and host-aware replication does not occur. A space must be defined with a replication count greater than 0 for replication to occur.

You must run an ActiveSpaces application to define and put data into the spaces of a metaspace. However, you cannot use ActiveSpaces Enabler to do this. Using ActiveSpaces Enabler you can configure and start as-agents. An as-agent does not define any spaces. The role of an as-agent is to automatically join any space created in a metaspace and act as a seeder for that space.

For information on replicating data in spaces, see *TIBCO ActiveSpaces Developer's Guide*. For more information about as-agents, see *TIBCO ActiveSpaces Administration*.

Specifying Member Name Prefixes for Host-Aware Replication

With host-aware replication, seeders of a space are grouped based on their member names. To organize seeders into groups, the following member naming convention is used:

<*group_name*>.<*member_name*>

AS_MEMBER_NAME_PREFIX must be unique for each component that you configure.

ActiveSpaces groups all seeders with the same *group_name* together and their data is replicated on seeders outside of that group.

To use host-aware replication with a component configured using the ActiveSpaces Enabler, prepend a <*group_name*> and '.' (dot) to the AS_MEMBER_NAME_PREFIX.

If you configured a component but did not previously specify an AS_MEMBER_NAME_PREFIX setting, the component name was used as the AS_MEMBER_NAME_PREFIX. In this case, add the AS_MEMBER_NAME_PREFIX Runtime Context Variable to your configuration and specify the following as your member name prefix:

<group_name>.<component_name>

For example, suppose you have components that are named as follows:

- NotreDame
- Purdue
- UniversityOfMichigan
- MichiganState

You have configured NotreDame and Purdue to both run-on machines physically located in Indiana, and you want to make sure that if a tornado hits Indiana and those machines go down, you do not lose any data. So you want to replicate your data on the machines in Michigan that run the components UniversityOfMichigan and MichiganState.

By adding and modifying the `AS_MEMBER_NAME_PREFIX` values of the components as follows, you ensure that any data seeded by NotreDame or Purdue component instances is replicated on component instances of UniversityOfMichigan or MichiganState:

- Indiana.NotreDame
- Indiana.Purdue
- Michigan.UniversityOfMichigan
- Michigan.MichiganState

See Controlling Where Components Run, page 6 to learn how to create stack configurations that control which machines the components are run on.

When using host-aware replication with shared-nothing persistence, the shared-nothing persistence data store still uses the same directory and file naming format for storing data:

<root_directory>/<metaspace_name>/<space_name>/<member_name>/<member_name>_store_<timestamp>

The *<member_name>* portion will include the *<group_name>* and dot (.) that are part of the member name when host-aware replication is used.

Chapter 3 Upgrading a Component

Take advantage of the new features by changing an old enabler to the latest release.

Topics

Changing the Component Enabler

Before you upgrade a component configured with a previous version of TIBCO Silver Fabric Enabler for ActiveSpaces Enterprise Edition, it is good practice to back up your engines and to decide a time to restart the component.

- Make sure to understand the ActiveSpaces concepts of redistribution and recovering data from persisted stores.
- If you restart your component instances individually, the data that is stored by the as-agent associated with each component instance is redistributed to the other running-as agents.
- You can also shut down all of your component instances at the same time and allow them to recover their persisted data after you restart them.

To change the component enabler, follow these steps:

1. Using the TIBCO Silver Fabric Administrator > Components page, identify the component you want to upgrade from a previous version of the TIBCO Silver Fabric Enabler for ActiveSpaces Enterprise Edition. Use the Enabler Version column to identify out-of-date enablers.
2. Click the **Component Action** menu icon, select **Change Enabler**, select the appropriate version of the enabler and click **OK**.
3. Click the **Component Action** menu icon again, click **Publish Changes**, and click **OK** to publish the selected component.

The component is not fully upgraded until you individually restart your component instances or stop and restart your stacks. For instructions on how to individually restart your component instances, see Individually Restarting Component Instances. For instructions on how to stop and restart your stack, see Stopping and Restarting All Component Instances.

Individually Restarting Component Instances

To restart each component instance, follow these steps:

1. Switch to the Engines page, and click the **Engine Actions** menu for the component that needs a component restart.
2. Select **Restart Component**.

Stopping and Restarting All Component Instances

To stop and restart all component instances, follow these steps:

1. Use TIBCO Silver Fabric Administator to go the Stacks page and identify the stack running the component you want to stop.
2. Click the **Stack Action** menu icon and select **Stop Stack**.
3. Restart your stack using your regular start up procedure.