TIBCO ActiveSpaces®

Administration

Release 2.1.2 Add-On
January 2014

Two-Second Advantage®

Important Information

Contents

Preface

TIBCO ActiveSpaces® is a distributed peer-to-peer in-memory data grid, a form of virtual shared memory that leverages a distributed hash table with configurable replication.

TIBCO ActiveSpaces® combines the features and performance of databases, caching systems, and messaging software to support large, highly volatile data sets and event-driven applications. It lets you off-load transaction-heavy systems and allows developers to concentrate on business logic rather than the complexities of developing distributed fault-tolerance.

TIBCO ActiveSpaces is available in three versions:

- **TIBCO ActiveSpaces® Enterprise Edition**—Provides C, Java, and .NET API sets and enables full cluster functionality. To enable remote clients, you must purchase licenses for the TIBCO ActiveSpaces Remote Client Edition.
- **TIBCO ActiveSpaces® Remote Client Edition**—Can be purchased in addition to the Enterprise Edition. Allows you to set up remote clients. Applications running on the remote clients can access the data grid and perform most ActiveSpaces operations.
- **TIBCO ActiveSpaces® Developer Edition**—A developer version of the product.. This version is downloadable from TIBCO Developer Network at http://tap.tibco.com.

Topics

Related Documentation

This section lists documentation resources you may find useful.

TIBCO ActiveSpaces Documentation

The following documents form the TIBCO ActiveSpaces documentation set:

- *TIBCO ActiveSpaces Installation* Read this manual for instructions on site preparation and installation.
- *TIBCO ActiveSpaces Administration* Read this manual to gain an understanding of the product that you can apply to the various tasks you may undertake.
- *TIBCO ActiveSpaces Developer's Guide* Read this manual for instructions on using the product to develop an application that manages data grids.
- *TIBCO ActiveSpaces C Reference* Read this manual for reference information on the C functions used to develop an application that manages data grids.
- *TIBCO ActiveSpaces 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.

Typographical Conventions

The following typographical conventions are used in this manual.

Table 1 General Typographical Conventions

Convention	Use
TIBCO_HOME *ENV_HOME* AS_*HOME*	Many TIBCO products must be installed within the same home directory. This directory is referenced in documentation as *TIBCO_HOME*. The default value of *TIBCO_HOME* depends on the operating system. For example, on Windows systems, the default value is `C:\tibco`. Other TIBCO products are installed into an *installation environment*. Incompatible products and multiple instances of the same product are installed into different installation environments. An environment home directory is referenced in documentation as *ENV_HOME*. The default value of *ENV_HOME* depends on the operating system. For example, on Windows systems the default value is `C:\tibco`. TIBCO ActiveSpaces installs into a directory within *TIBCO_HOME*. This directory is referenced in documentation as *AS_HOME*. The default value of *AS_HOME* depends on the operating system. For example on Windows systems, the default value is `C:\tibco\as\2.0`.
`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, a place to share and access the collective experience of the TIBCO community. TIBCOmmunity offers forums, blogs, and access to a variety of resources. To register, go to http://www.tibcommunity.com.

How to Access All TIBCO Documentation

After you join TIBCOmmunity, you can access the documentation for all supported product versions here:

http://docs.tibco.com/TibcoDoc

How to Contact TIBCO Support

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

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

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

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

  https://support.tibco.com

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

Chapter 1 Overview of ActiveSpaces Administration and Deployment

This chapter provides an overview of the TIBCO ActiveSpaces® administration tools and guidelines for deploying ActiveSpaces in the network.

Topics

Overview of Administration

ActiveSpaces provides three administrative tools:

- **as-admin** The main administration utility for ActiveSpaces. Provides a command line interface (CLI) that allows you to deploy and manage nodes in the ActiveSpaces data grid. Provides commands that you use to:
  - Connect to a metaspace
  - Create a space
  - Join a space
  - Display information about existing spaces and members
- **as-agent** A pre-built agent process that you can run on any host to connect to a metaspace, join all distributed spaces in the specified metaspace as a seeder, and keep the space active.
- **ASMM** ActiveSpaces Monitoring and Management (ASMM) provides a GUI that lets you connect to a metaspace, display information about various ActiveSpaces system resources, for example, metaspaces, metaspace members, spaces, entry statistics, and so on, and browse space entries.

In addition, ActiveSpaces provides an `as-dump` utility that you can use to display information about shared-nothing persistence data files.

Deployment Modes

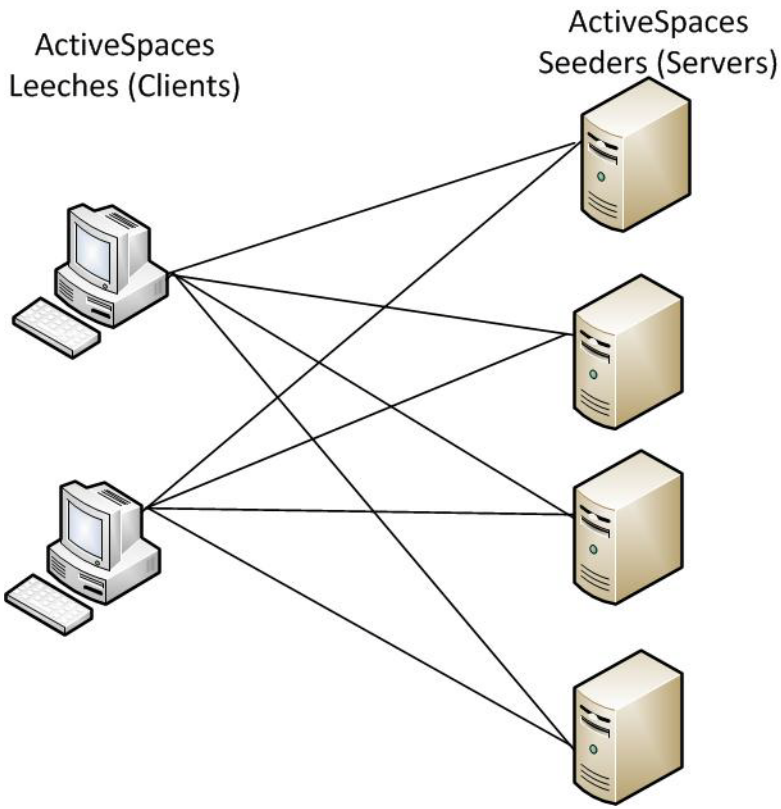
This section describes the basic deployment modes for ActiveSpaces.

Clients (Leeches) and Servers (Seeders)

When a member joins a space, it can join as a server (seeder) or as a leech (client). This allows you to distribute the node deployment between seeders and leeches:

- *Seeders* play an active role in maintaining the space by providing CPU and RAM.
- *Leeches* play a passive role. They have access to space data but provide no resources.

Figure 1 Client-Server Deployment



For more information on deploying seeders and leeches in the ActiveSpaces network, see When to Join the Space as a Seeder or a Leech in the *TIBCO ActiveSpaces Developer's Guide*.

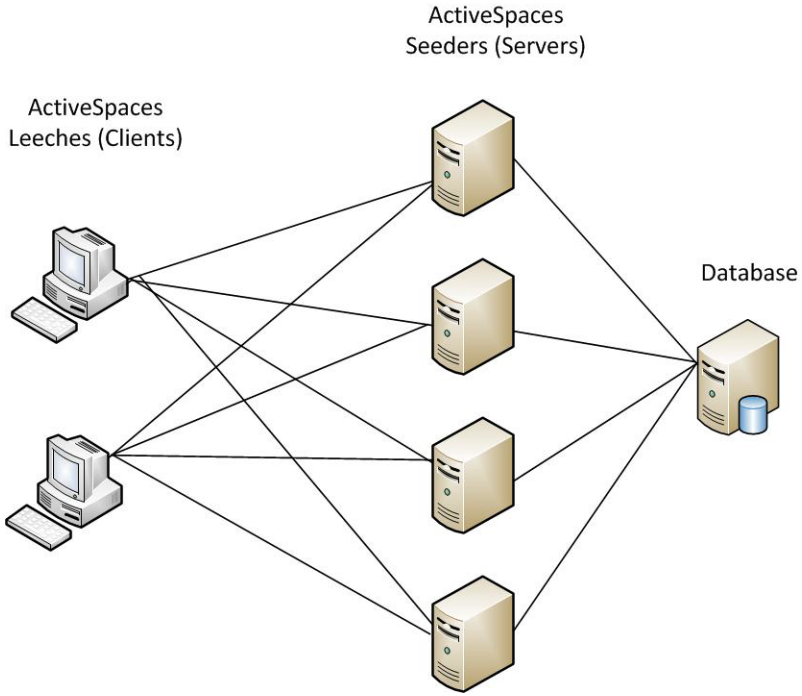
Shared-All Persistence

ActiveSpaces allows you to persist data to disk storage and recover it if data loss occurs or there is a problem with cluster startup. For detailed information on persistence, see Persistence in the *TIBCO ActiveSpaces Developer's Guide*.

With shared-all persistence, all seeder nodes share a single persister or a set of persisters. Your application must provide an implementation of the persistence interface and interface to the shared persistence layer of choice.

Figure 2, Deployment with Shared-All Persistence shows all of the seeder nodes in an ActiveSpaces sharing a single database for persistence and data recovery.

Figure 2 Deployment with Shared-All Persistence

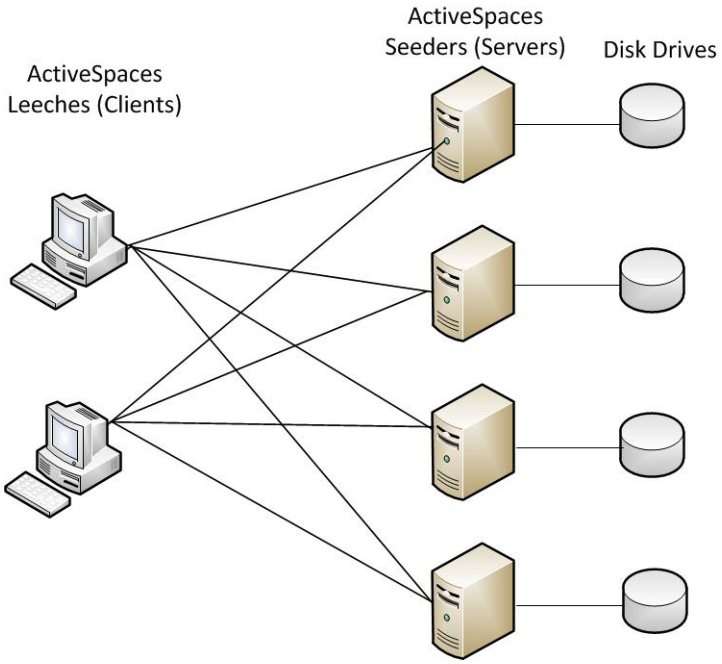


Shared-Nothing Persistence

With shared-nothing persistence, each node that joins a space as a seeder maintains a copy of the space data on disk. Each node that joins as a seeder writes its data to disk and reads the data when needed for recovery and for cache misses.

Figure 3, Deployment with Shared-Nothing Persistence shows a group of seeder nodes that persist data to local hard disk.

Figure 3 Deployment with Shared-Nothing Persistence



Host-Aware Replication

Host-aware replication allows you to ensure that the data for seeders running on the same machine is always replicated on a different machine than the one running the original seeders. If there is only one seeder running on a machine, then host-aware replication is not needed, because data could not be replicated onto the same machine if there are no other seeders running on that machine.

To set up host-aware replication for the seeders on a machine, you logically group the seeders together using the following member naming convention:

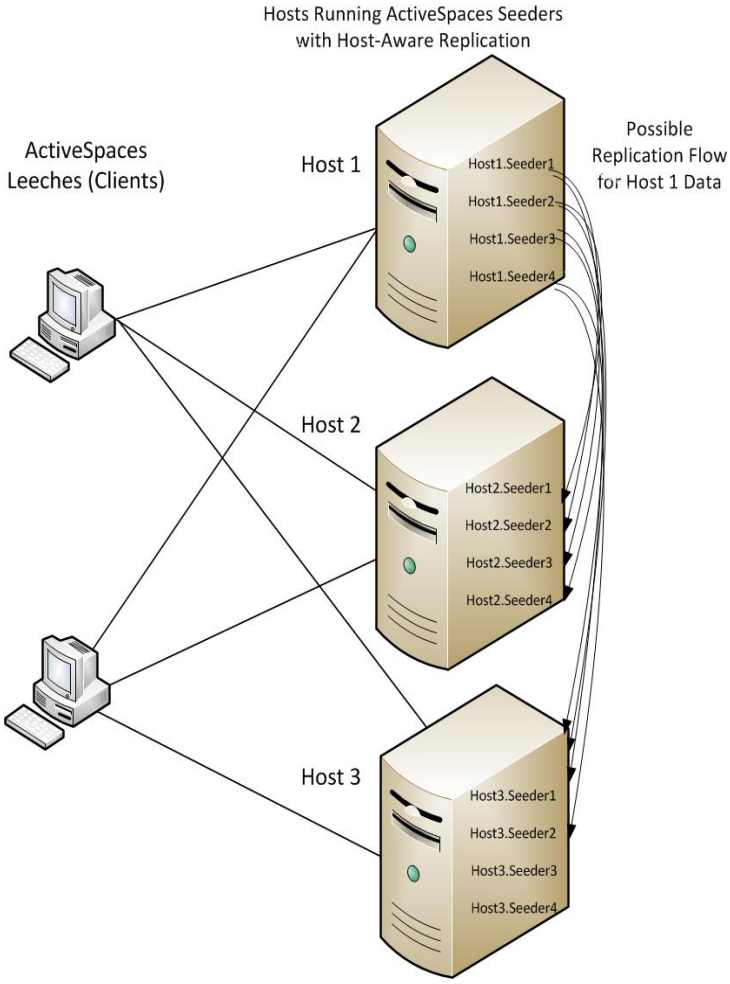
<group_name>.<member_name>

where *group_name* can be any name which helps you to identify the grouping of seeder members. For example, the *group_name* can be the name of the machine on which the seeders are running or some other arbitrary name that logically identifies the group of seeders.

Host-aware replication is purely based on the *group_name* part of the member name of the seeder, and not on the IP address of the physical host the process is running on.

Figure 4, Sample Deployment with Host-Aware Replication shows a topology for host-aware replication in which each host is running four seeders and a seeder grouping is defined for each host.

Figure 4 Sample Deployment with Host-Aware Replication



The first host is named `Host1`, and runs four seeders, `Seeder1`, `Seeder2`, `Seeder3`, and `Seeder4`. The second host is named Host2, and the third host is named `Host3`. Each of these hosts also runs four seeders.

If you name the seeders according to host-aware replication member naming, then ActiveSpaces always replicates data that is seeded on one of the seeders on a given host to a seeder that is running on another host. Therefore, if a host or device goes down, the data that was seeded by any of the seeders on that host is not lost, because it is automatically backed up on a seeder that resides on another host.

You can set up this type of topology in several ways:

1. By using the TIBCO ActiveSpaces API programs in an application program that implements <*group_name*>.<*member_name*> member names.

   For information on the API functions to implement host-aware replication, see Using the ActiveSpaces API Set to Implement Host-Aware Replication on page 62 in the *TIBCO ActiveSpaces Developer's Guide*.

2. By using `as-agent` and specifying the member names using the **`-name`** parameter.

   For example, you might use `as-agent` with the **`-name`** parameter to start four seeders on each host. For example, on host 1, run as-agent as follows:

   ```
   java -jar as-agent.jar -metaspace ms -discovery
   tcp://127.0.0.1:50000  -listen tcp://127.0.0.1:50000 -name
   Host.1.Seeder1
   ```

   Then run two additional instances, with the **`-name`** parameter specifying `Host1.Seeder2`, and `Host1.Seeder3`.

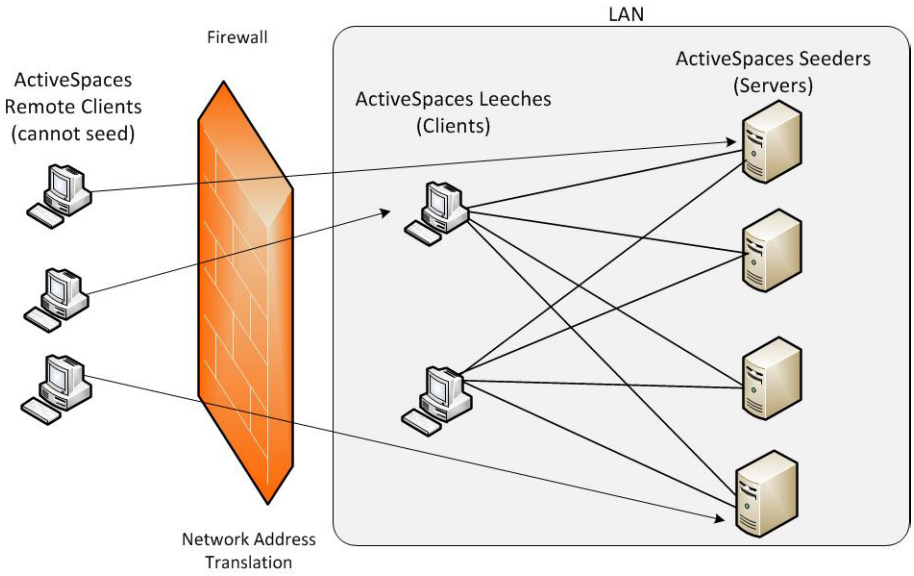
3. By using as-`admin` to connect to a metaspace. The member name is specified using the **`membername`** option of the **`connect`** command.

Remote Client Architecture

In some situations, where nodes are unable to become full peers in the data grid, for example, when there is a firewall protecting a LAN, you can deploy ActiveSpaces nodes as remote clients. Remote clients can perform puts and gets, but cannot act as seeders or assume a management role in the core cluster.

Using remote clients also has the advantage of saving cluster processing overhead.

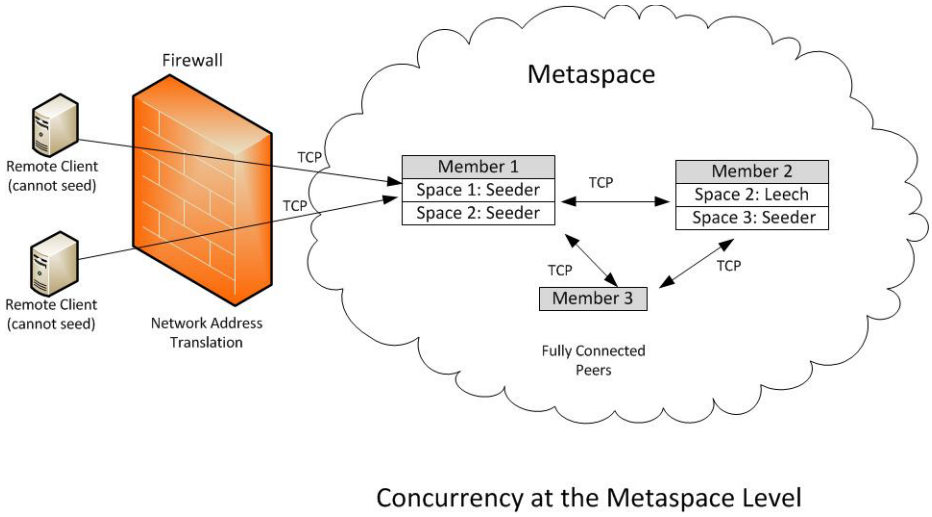
Figure 5 Remote Client Deployment



When you use remote clients, the remote clients can update the entries in the database, and active spaces seeders ensure that concurrency is maintained. Remote clients connect using TCP connections.

Figure 6, Concurrency with Remote Clients, shows remote clients updating to a seeder (Member 1). The seeders in the metaspace ensure that the data replicated on Member 2 synchronized with the data on Member 1.

Figure 6 Concurrency with Remote Clients



Concurrency at the Metaspace Level

Best Practices for Node Discovery

This section describes how to specify discovery parameters in several deployment scenarios:

Specifying Discovery When Using ActiveSpaces Security

If you are implementing ActiveSpaces security:

- The discovery transport type must be TCP.
- The discovery list on both the requestor and controller members for the given metaspace binding must be the same.

Also, when you implement ActiveSpaces security, you do not need to specify the discovery URL for a member that is joining the metaspace, because the discovery URL is retrieved from the security policy file or security token file used to start a node. If you do specify the discovery URL, it is ignored and ActiveSpaces uses the value for the metaspace as specified in the security configuration files.

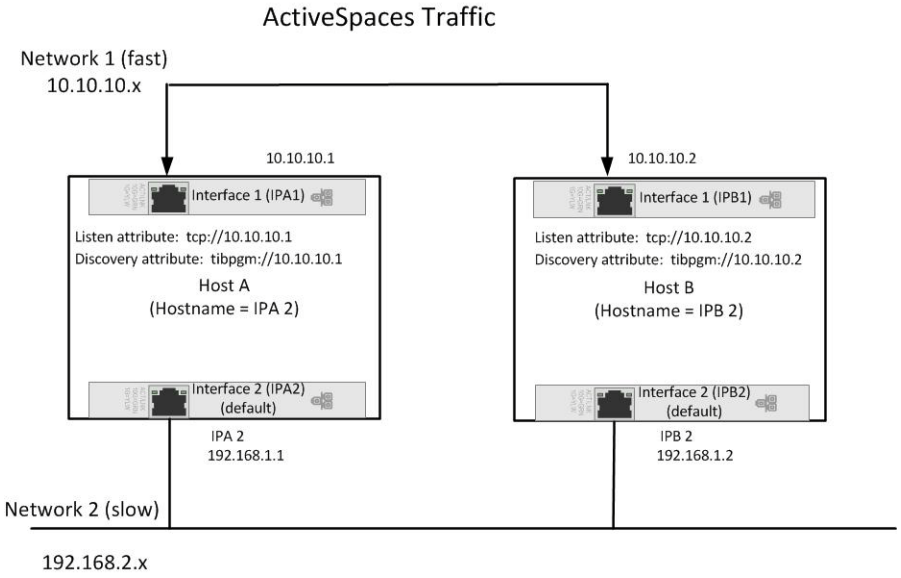
For more information on ActiveSpaces security, see Chapter 4, Implementing ActiveSpaces Security in the *TIBCO ActiveSpaces Developer's Guide*.

Choosing the Right Discovery Point

When you specify the discovery attribute (either in the `discovery` attribute of the `MemberDef` object for the space member or in the `discovery` parameter for the `as-admin` **connect** command), you can specify the IP address of the network interface to use for discovery.

It is important to specify the optimum discovery IP address. For example, in a network where you have several subnetworks, or where you have a relatively fast network and also a slower network, performance will be best if you choose the faster network. Figure 7 shows a network topology that has a fast network and a slower network.

Figure 7 Selecting an Interface and Network for Optimum Discovery



In the network shown in Figure 7, there are two networks: network 1 is fast and network 2 is slow.

To ensure that you use the interface connected to the faster network for discovery, specify the interface for the faster network. In the example, we specify the tibpgm discovery protocol and the IP address of the interface for the faster network:

```
tibpgm://10.10.10.1
```

This ensures that discovery uses the faster network.

Specifying Multiple TCP Discovery Nodes for Fault Tolerance

If you are using the TCP discovery method, you can specify multiple TCP discovery nodes to provide fault tolerance.

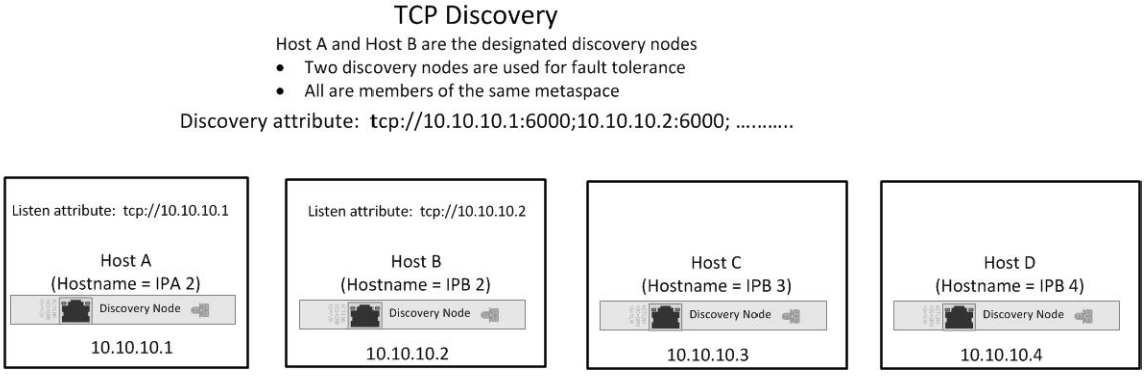
Figure 8, Specifying Multiple TCP Discovery Nodes illustrates how to specify multiple TCP discovery nodes in a network with four TCP hosts.

In the example, we specify the tcp discovery method and two node address/port pairs:

```
tcp://10.10.10.1:6000;10.10.10.2:6000
```

When setting up the discovery attribute in the MemberDef object for the space or with the **discovery** parameter for the `connect` command, you can specify additional IP addresses and ports, separated by semicolons.

Figure 8 Specifying Multiple TCP Discovery Nodes



Chapter 2 Administering ActiveSpaces with the Admin CLI

Administrative tasks for TIBCO ActiveSpaces are performed through a utility called `as-admin`, or the *Administration Command Line Interface* (Admin CLI). These tasks include connecting to a metaspace, creating a space, and displaying information about existing spaces and members. This chapter lists and describes the commands available in the Admin CLI.

Topics

Starting the Admin CLI

To start the TIBCO ActiveSpaces Admin CLI, you must:

- Set environment variables for TIBCO ActiveSpaces
- Launch the Admin CLI, in a command prompt window

Setting the Required Environment Variables

To be able to run the Admin CLI, you must set environment variables to point to:

- **JAVA_HOME** The location of the Java installation on your computer.
- **AS_HOME** The location of the ActiveSpaces installation on your computer.
- **PATH** Defines the path to `AS_HOME` and `JAVA_HOME`.

The easiest method to set the variables is in a script file (a command file in MS Windows or a shell script on UNIX platforms).

The following Windows command script sets the required environment variables for starting the Admin CLI in a command window.

Example 1 Windows Command File for Setting Environment Variables for the Admin CLI

```
@echo off

set AS_HOME=c:\tibco\as\2.1
set PATH=%AS_HOME%\lib;%AS_HOME%\bin;%PATH%

set JAVA_HOME=%DRIVE_C%\Program Files\Java\jdk1.6.0_24
set PATH=%PATH%;%JAVA_HOME%\bin;%JAVA_HOME%\lib
```

Make sure that you prepend the `AS_HOME\lib` PATH setting and the `AS_HOME\bin` PATH setting to the PATH setting; for example:

```
set PATH=%AS_HOME%\lib;%AS_HOME%\bin;%PATH%
```

Launching the Admin CLI

The default location of the Admin CLI program on Windows is:

`C:\tibco\as\2.1\lib`

To launch the CLI:

1. Open a command window.

2. Make sure that you have set the required environment variables as described in the previous section, Setting the Required Environment Variables, page 17.
3. Enter the following commands:

```
cd C:\tibco\as\2.1\lib
java -jar as-admin.jar
```

If you are running Windows on a 64-bit platform, the command to launch the Admin CLI is:

```
java [-d64] -jar as-admin.jar
```

The Admin CLI prompt appears:

```
as-admin>
```

You can now enter Admin CLI commands.

The examples in this section use Windows conventions.

Using a script file to pass arguments when launching the Admin CLI

Using a script file, you can pass command line arguments when you launch the Admin CLI, as follows:

```
java -jar as-admin.jar -i admin-cmd-filename
```

The above usage information for `as-admin` can be obtained by invoking **`java -jar as-admin.jar-h`** from the ActiveSpaces `lib` directory in a command line window.

Comments can be inserted into the script file by starting a line with `#`, as follows:

```
# this is a comment
```

This mechanism allows execution of commands in a batch mode through the Admin CLI. There are several limitations on the structure of the file:

1. No special characters are allowed.
2. Each line is executed sequentially by the Admin CLI. Therefore, each individual command must be entered on one line.
3. The admin tool will not automatically exit and return to the shell after executing the commands of the file unless the script file contains a quit command

Here is an example.

```
######
connect name "UserMetaspace" discovery "tibpgm"
show members
export metaspace to "export.txt"
quit
######
```

This will connect to a metaspace, display members, export the metaspace, and quit

Using the Admin CLI command **`export metaspace [to <filename>]`** (see export metaspace on page 44) will generate a file containing the schema for the existing metaspace. The `admin-cmd-filename` value can be the path to this generated file. In this way, a new metaspace can be created using the exported metaspace schema.

Example

1. Create a text file that invokes the Admin CLI.

   Here is sample of the contents of such a file. In this example, the file is named `example01.txt`, and is located in the directory `C:\temp2.`

   ```
   connect name 'ms' discovery 'tibpgm' listen 'tcp';
   define space name 'testspace' (field name 'key' type 'integer'
   field name 'value' type 'string') key (type 'hash' fields
   ('key'));
   ## This defines the key index to be of index type "hash" with
   fields "key"
   ```

2. Launch the Admin CLI using the following syntax:

   `java -jar as-admin.jar-i c:\temp2\example01.txt.`

Additional Characteristics of the Admin CLI Tool:

- When running the tool, hitting the question mark key (?) at any time brings up context-sensitive help.
- Arrow keys and the tab key function in the command window as in typical advanced shells.
- You can invoke a shell command by using the escape character'!', for example, **`!dir`** or **`!ls`** to list the files in the current directory.
- Field names and literals must be enclosed in single or double quotes.

- In most cases, the Admin CLI will start by using the `connect` command to connect to a metaspace.
- The default discovery mechanism is PGM. If RV discovery or TCP unicast discovery is needed, you must specify the appropriate discovery URL.

  For more information on discovery, see “Connecting to the Metaspace “in the *TIBCO ActiveSpaces Developer's Guide*.

The Execute method

Admin CLI administrative commands (for example defining a space) can be executed directly from within an application. This is done by using the Metaspace object's `execute` method and passing it a string representing the Admin CLI command. A string is returned containing the output resulting from executing the command.

alter space

Admin CLI Command

Syntax

```
alter space name <string> add (field name <string> type <string>
[nullable <boolean>] (, field name <string> type <string> [nullable
<boolean>])*)
|
alter space name <string> add index ( name <string> [type <string>]
fields (<string> (, <string>)*))
|
alter space name <string> drop index (<index_name> (,
<index_name>)*)
```

Purpose

Use `alter space` to add a field to an existing space definition, or to add or drop an index from a space definition.

Parameters

The following table describes the parameters for this command.

Table 3 alter space Parameters

Parameter	Description
name	The name of the space to be modified.
add	Specifies that a field is to be added: • `name` specifies the name of the field to be added. • `type` specifies the data type for the field. Must be one of the following: `boolean`, `char`, `short`, `integer`, `long`, `float`, `double`, `string`, `datetime`, `blob`. • `nullable` is optional when adding a field. If you do not specify the `nullable` parameter, the field is by default not nullable. If you enter `nullable`, you must enter `nullable true`.
add index	Specifies that an index is to be added: • `name` specifies the name of the index. • `type` specifies the type of the index, which can be hash or tree. • `fields` specifies the fields to be used in the index.  When you add an index, the new index cannot have fields from an existing index.

Table 3 alter space Parameters

Parameter	Description
drop index	Specifies that one or more indexes are to be removed. index must be followed by a list of one or more index names specifying the indexes to remove.

Examples

Examples for add field:

```
alter space name “myspace” add (field name “average” type “double”)
alter space name “myspace” add (field name “average” type “double” nullable true)
alter space name “myspace” add (field name “average” type “double”, field name “total” type “long” nullable true)
```

Examples for add index:

```
alter space add index (name “index1” type “hash” fields(“a”, “b”, “c”))
alter space add index (name “index1” type “hash” fields(“a”, “b”, “c”)) index (name “index2” type “hash” fields(“a”, “b”, “c”))
```

Examples for drop index:

```
alter space drop index (“index1”)
alter space drop index (“index1”, “index2”)
```

Example for adding a field and an index:

```
alter space name “myspace” add (field name “average” type “double”) index (name “index1” type “hash” fields(“a”, “b”, “c”))
```

The parameters for fields and index use the following format:

- **fields:** `(field name “average” type “double”)`
- **index:** `index (name “index1” type “hash” fields(“a”, “b”, “c”))`

You can perform consecutive updates by adding one field after another.

You cannot add and drop an index in the same **alter space** command with `as-admin`. However, you can do this using the API operations. For an example showing how to do this with the Java API set, see Adding and dropping an index using the Java API on page 72 in the *TIBCO ActiveSpaces Developer’s Guide*.

clear

Admin CLI Command

Syntax `clear`

Purpose `clear` is used to remove all lines of text from the command window except for a single Admin CLI prompt.

Parameters None.

clear | set password

Admin CLI Command

Syntax

```
clear | set password domain_name <string> policy_file <string>
[new_policy_file <string>]
clear | set password token_file <string> [new_token_file <string>]
```

Purpose

Use the **`clear password`** command to create a new policy file without a password or a new token file without a password.

Use the **`set password`** command to create a a new policy file or a new token file with a new password.

When you issue the set password command, you are prompted to enter and verify the new password for the domain.

Parameters

The following table lists the parameters for this command with a description of each parameter.Because there are two versions of the command, there are two sets of parameters.

Table 4 clear password Parameters

Parameter	Description
domain_name	Specify the domain name for the domain for which you want to set or clear the password.
policy_file	Specify the policy file to use when creating the a new policy file without a password.
new_policy_file	This optional parameter specifies the name of a new policy file that is created based on the policy file that you specify with the **`policy_file`** parameter. If you do not specify a new policy filename, then a new policy file created is with the current date and time appended to the filename. For example, if you specify an existing policy filename of “policy.txt,” ActiveSpaces creates a policy file name; for example, “policy.txt.2013_01_16_20_38_00.'  The policy filename cannot contain a forward slash character (“/”).
token_file	If you are using the clear password command to clear the password in a token file or the set password command to set the password for a token file, specify the token filename.

Table 4 clear password Parameters

Parameter	Description
new_token_file	This optional parameter specifies the name of a new token file that is created based on the token file that you specify with the **token_file** parameter. If you do not specify a new token filename, then a new token file created is with the current date and time appended to the filename. For example, if the token file is named "my token," ActiveSpaces creates a token filename; for example, "mytoken.2013_01_16_20_38_00.'

Example

```
clear password domain_name 'AS-DOMAIN' policy_file 'policy.txt'
new_policy_file 'newdeal.txt'
```

Parameter values must be enclosed in either single or double quotes.

The following examples illustrate the syntax of the **clear password** command:

- `clear password domain_name 'AS-DOMAIN' policy_file 'policy.txt'`
- `clear password domain_name 'AS-DOMAIN' policy_file 'policy.txt' new_policy_file 'newdeal.txt'`
- `clear password token_file 'mytoken'`
- `clear password token_file 'mytoken' new_token_file 'yourtoken'`

connect

Admin CLI Command

Syntax

```
connect [name <string>] [discovery <string>] [listen <string>]
[membername <string>] [security_token <string>] [security_policy
<string>]
```

Purpose

`connect` is used to connect to a metaspace. Connecting to a metaspace is a necessary initializing step for the Admin CLI in order to begin working with ActiveSpaces, just as it is for an ActiveSpaces application. The Admin CLI can only be connected to one metaspace at a time.

You cannot specify a token file and a policy file at the same time.

Parameters

The table lists the parameters for this command with a description of each parameter.

Table 5 connect Parameters

Parameter	Description
name	Optional. If a metaspace with this name does not exist, it will be created as a result of this command. If no name is specified, the Admin CLI will connect to the default metaspace, called `ms`.

Table 5 connect Parameters

Parameter	Description
discovery	Optional. The discovery URL can take one of three forms: NOTE: If you are using ActiveSpaces security, you MUST use TCP discovery. • If Unicast discovery is used, then a list of 'well known' IP addresses and ports must be passed in a URL with the following syntax: `tcp://ip1:port1;ip2:port2;...` • If multicast discovery is to be used then the URL must be one of the following depending on which reliable multicast transport is to be used: The tibrv (multicast) URL takes three parameters: service, network, and daemon. In many cases, the default values are sufficient. Syntax: `tibrv://service=`*service*`/network=`*network*`/daemon=`*daemon* or tibpgm://destination port/interface;discovery group address/optional transport arguments See "PGM (Pragmatic General Multicast) URL Format" in the *TIBC0 ActiveSpaces Developer's Guide* for more information on PGM discovery URLs. See "TIBCO Rendezvous Discovery URL format" in the *TIBC0 ActiveSpaces Developer's Guide* more information on TIBCO Rendezvous discovery URLs.  If you are connecting as a security domain controller or as a security domain requestor, do not specify the discovery parameter. If you do specify a discovery parameter, it will be overwritten by the discovery parameter specified in the security policy file or the security token file specified with the connect command.

Table 5 connect Parameters

Parameter	Description
listen	Optional. Specifies which interface and port the administrative process should create its listening TCP socket on. Syntax: `tcp://interface:port` See "Listen URL Format" in the *TIBC0 ActiveSpaces Developer's Guide* for more information on listen URLs.
membername	Optional. Specifies a member name for the member. This helps to identify which member name is associated with which member ID. The show members command displays the member name if one has been assigned; otherwise, a default member name is assigned that is constructed from the member ID.
security_token	Optional. Specifies the token file for a security domain requestor that must be authenticated by a security domain controller. If TIBCO ActiveSpaces security is implemented and you are connecting from a requestor node, and the metaspace to which you are connecting requires a token file, specify the `security_token` parameter and provide the directory path and filename for the token file. If you specify a token file, do not specify the `security_policy` parameter.
security_policy	Optional. If TIBCO ActiveSpaces security is implemented and you are connecting from a domain security controller node, specify the `security_policy` parameter and provide the directory path and filename for the policy file. If you specify a policy file, do not specify the `security_token` parameter.

Example `connect name 'ms' discovery 'tcp://192.168.1.10'`

Parameter values must be enclosed in either single or double quotes.

The following examples illustrate the syntax of the `connect` command:

- connect
- connect name <metaspace_name>
- connect discovery <discovery_url>
- connect listen <listen_url>
- connect discovery <discovery_url> listen <listen_url>
- connect name <metaspace_name> discovery <discovery_url>
- connect name <metaspace_name> discovery <discovery_url> listen <listen_url>
- connect name <metaspace_name> discovery <discovery_url> listen <listen_url> membername <member_name>
- connect security_policy 'mypolicy.txt' name 'ms' listen 'tcp://127.0.0.1:50000'
- connect security_token 'mytoken.txt' name 'ms' listen 'tcp://127.0.0.1:50000'

define | create security_policy

Admin CLI Command

Syntax

```
(define | create) security_policy [policy_name <string>]
[encrypt <boolean>][validity_days <integer>] policy_file <string>
```

Purpose

Use the **`define | create security_policy`** command to create a security policy file.

Parameters The following table lists the parameters for this command with a description of each parameter.

Table 6 define | create security_policy Parameters

Parameter	Description

policy_name	Optional. Specifies the name of the policy to be created. If you do not specify a policy name, the policy is given the default name AS-POLICY.
	  You cannot specify a policy file and a security token for the same connection.
	You can also specify one or more domains that the policy is associated with:
	1. To specify that the policy is associated with one domain, specify the policy name and the domain as follows:
	`(define | create) security_policy policy_name <policy_name>/<domain name> policy_file <string>.`
	For example:
	`create security_policy policy_name "OUR_POLICY/OUR_DOMAIN" policy_file "ourpolicy.txt"`
	If you enter the command in this way, the `encrypt` setting defaults to false: then if you specify one domain, you are prompted to enter and verify the password for that domain. If you specify multiple domains, you are prompted to enter and verify the password for each domain.
	  If you specify `encrypt=false`, then ActiveSpaces creates all domains is created with an unencrypted ID, which requires no password, and you are not prompted for a password.
	2. To create multiple domains associated with the policy, specify the policy name and a list of domains that the policy is associated with. Specify the domains separated by commas:
	`(define | create) security_policy policy_name "<string/string, string, string ...>" policy_file <string>`
	For example:
	`create security_policy policy_name "NEW_POLICY/MD1,MD2,MD3" policy_file "newpolicy.txt"`

Table 6 define | create security_policy Parameters

Parameter	Description
encrypt	Optional. Indicates whether the private key for the policy is to be encrypted. The default is `encrypt true`. If you specify encryption, as-admin prompts you to specify and verify a new domain password and creates an encrypted private key in the Domain Identity section of the policy file. If you specify `encrypt false`, the domain does not require a password, and as-admin creates an unencrypted private key in the policy file.
validity_days	An integer that specifies how long the domain ID that the command creates remains valid. The default value is 365 days. Policies can have more than one domain, where (in theory) each of them can have different validity days if the domain definitions are moved between policy files manually.
policy_file	Enter the name of the policy file that is to be created for the policy.  You cannot specify a policy file and a security token for the same connection.  The policy filename cannot contain a forward slash character (“/”).

Example

```
create security_policy policy_name 'mypolicy' policy_file
'policy.txt'

create security_policy policy_name 'mypolicy' encrypt false
policy_file 'policy.txt'
```

Parameter values must be enclosed in either single or double quotes.

The following examples illustrate the syntax of the `define | create security_policy` command:

- `create security_policy policy_name 'mypolicy' policy_file 'policy.txt'`
- `create security_policy policy_name 'mypolicy' encrypt false policy_file 'policy.txt'`
- `define security_policy "MY_POLICY/MY_DOMAIN" policy_file 'policy.txt'`

define | create security_token

Admin CLI Command

Syntax

```
(define | create) security_token domain_name <string>
policy_file <string> [create_identity [common_name <string>]
[encrypt <boolean>][validity_days <integer>]]token_file <string>
```

Purpose

Use the **`define | create security_token`** command to create a security token for deployment to ActiveSpaces requestor nodes.

When you enter the command, you are prompted to enter and verify a new token password for the security token. Enter and verify the password.

Parameters

The following table lists the parameters for this command with a description of each parameter.

Table 7 define | create security_token Parameters

Parameter	Description
domain_name	Specifies the name of the domain for which the security token is to be created.
policy_file	Specifies the name of the policy file that is to be used to create the token.
create_id	Optional. Enter the **`create_id`** parameter if you want to create a private key to verify the identify of connecting nodes.
common_name	Optional. If you enter the `create_id` parameter and you want to provide an X.509 common name to identify the private key, specify a common name. If you do not specify a common name, ActiveSpaces generates a common name that contains the domain name plus a random number; for example "/CN=AS-REQUESTOR-FEF3A467." If there is no common name associated with the token, then node connections use a temporary name generated by ActiveSpaces. If you provide a common name for the token file, this name is always used.
encrypt	Optional. If you enter the `create_id` parameter and you want to encrypt the private key, enter `encrypt true` (the default setting). If you do not want to encrypt the private key, enter `encrypt false`. Using `encrypt false` eliminates having to enter the password each time the node is started.

Table 7 define | create security_token Parameters

Parameter	Description
validity_days	Optional. To specify the number of days that the private key is valid for, enter the number of days. The default setting is 365 days.
token_file	Provide the name of the token file that is to be created.

Example

```
create security_token domain_name 'AS-DOMAIN' policy_file
'policy.txt' create_identity common_name 'MyRequestor-123' encrypt
true validity_days 90 token_file 'mytoken'
```

Parameter values must be enclosed in either single or double quotes.

The following examples illustrate the syntax of the **`define | create security_token`** command:

- create security_token domain_name 'AS-DOMAIN' policy_file 'policy.txt' create_identity common_name 'MyRequestor-123' encrypt true validity_days 90 token_file 'mytoken'
- `create security_token domain_name 'AS-DOMAIN' policy_file 'policy.txt' create_identity token_file 'newtoken'`
- `create security_token domain_name 'AS-DOMAIN' policy_file 'policy.txt' create_identity common_name 'MyRequestor-123' encrypt true validity_days 100 token_file 'mysecurity_token'`

define | create space

Command

Syntax

```
(define | create) space name <string>
(field name <string> type <string> [nullable <boolean>] (, field
name <string> type <string> [nullable <boolean>] [encrypted
<boolean>)*)
     key ( [type <string>] fields (<string> (, <string>)*))
     distribution_def ('KEY','field0','field1','field2' ...)
     (index ( name <string> [type <string>] fields (<string>
     (, <string>)*)))*
     [distribution_policy <string>]
     [persistence_type <string>]
     [persistence_policy <string>]
     [update_transport <string>]
     [replication_policy <string>]
     [replication_count <integer>]
     [min_seeders <integer>]
     [capacity <long>]
     [eviction_policy <string>]
     [ttl <long>]
     [lock_ttl <long>]
     [lock_wait <long>]
     [lock_scope <long>]
     [space_wait <long>]
     [write_timeout <long>]
     [read_timeout <long>]
     [phase_count <int>]
     [phase_interval <long>]
```

Purpose Used to create a space.

Remarks The supported data types for fields are:

- **Field types** boolean, char, short, integer, long, float, double, blob, string, datetime
- **Distribution policies** non_distributed, distributed
- **Persistence types** none, share_all, share_nothing
- **Persistence policies** sync, async

- **Update transports** unicast, multicast
- **Replication policies** sync, async
- **Eviction policies** none, lru
- **Lock scopes** thread, process

Parameters The parameters for this command are listed and described in Table 8, define space Parameters.

Table 8 define space Parameters

Parameter	Description
`name`	Required.
`field`	Required. The data type for a field must be one of the following: `boolean`, `char`, `short`, `integer`, `long`, `float`, `double`, `string`, `datetime`, `blob`.
`nullable`	Optional. Can be either true or false (no quotes). By default is equal to false. If a field has nullable set to true, tuples put into the space do not need to contain a field with that name.
`encrypted`	Optional. If the field is not a key field or an index field and you have enabled ActiveSpaces security, you can specify that the data in the field is encrypted. Each (non-key, non-index) field can be made encrypted, as long as the policy for the corresponding domain allows it.
`key`	Required. Identifies one or more fields (already specified with the `field` parameter) that will serve as a unique key for the space. When you enter the key parameter, you can optionally specify the index type of the key field by including the type keyword. For example: `key (type "hash" fields (...))` The `fields` keyword is required. The `type` keyword is optional. The default index type is the "hash" index type.

Table 8 define space Parameters

Parameter	Description

`index`	Optional. Identifies one or more fields already specified with the "field" parameter that will serve as a secondary index. You can specify an index name and index type to be used by entering:
	`index (name "index1" type "tree" fields (...))`
	The name keyword is required.
	The `type` keyword is optional. The default is index type is "tree."
	The `fields` keyword and fields are required
	You can specify as many indexes as desired by specifying the indexes, one by one, after the `key` parameter. You just need to put them one after other after the key field. For example:
	`key (...) index(name "index1" ...) index(name "index2" ...) index (name "index3" ...)`

`distribution_def`	Defines one or more fields as distribution fields. If a field is defined as a distribution def field, then all tuples that have an identical data value for the field are stored on the same seeder.  The distribution fields must be a subset of the key fields. Otherwise, ActiveSpaces throws an exception. The following example shows how to set up distribution def fields: `key (fields ('KEY','field0','field1','field2')) distribution_def ('KEY','field0','field1','field2') distribution_policy 'distributed' replication_count 0` In the example, `field0`, `field1`, and `field2` are defined as key fields and also as distribution def fields.  If you define fields as distribution def fields, then you must also set `distribution_policy` to `distributed`.

Table 8 define space Parameters

Parameter	Description
`distribution_policy`	Optional. Determines whether management of entries in the space is shared among the seeders that have joined the space (**`distributed`**) or a single seeder is responsible for all entries in the space (**`non_distributed`**). The default value is **`distributed`**.  If you define fields as distribution def fields, then you must also set `distribution_policy` to `distributed`.
`persistence_type`	Optional. Specifies whether persistence is enabled for the space, and if so, what type of persistence to use. To specify no persistence, specify **none**. To specify shared all persistence (space members designated as persisters maintain data on disk), specify **share-all**. To specify shared-nothing persistence (each member maintains data on disk), specify **share_nothing**.
`update_transport`	The transport protocol used to distribute notifications of updates to the data stored in the space. Can be either **`unicast`** or **`multicast`**, signifying whether unicast or multicast is used.
`replication_policy`	Optional. A value of **`sync`** specifies that replication is done in synchronous mode for the space, so that when an operation modifies one of the entries in the space, the operation only returns an indication of success when that modification has been positively replicated up to the degree of replication required for the space. A value of **`async`** specifies that replication is asynchronous.
`replication_count`	Optional. Specifies the degree of replication for the space, i.e., the number of replicates required.
`persistence_policy`	Optional. Specifies the what type of communication is used to maintain persistence: synchronous (**sync**) or asynchronous (**async**).
`min_seeders`	Optional. Specifies the minimum number of seeders that should be joined to the space before the space becomes ready to accept operations. The default value is 1.

Table 8 define space Parameters

Parameter	Description
`capacity`	Optional. Specifies a maximum number of entries per seeder for the space. When the capacity is reached the result of any additional request to put (insert) a new entry in the space will depend on the value of the **`eviction_policy`** attribute. The default value is `-1` (no capacity).
`eviction_policy`	Optional. If a `put` operation on a space would cause a seeder to exceed the space's **`capacity`** attribute, then the value of this attribute will dictate the result of this operation: if the value is 'none' (in quotes) then there will be no eviction and the operation will fail because the seeder is already at capacity. If the value is 'lru' (in quotes) then the seeder will evict another entry from the space using the 'least recently used' eviction algorithm. The default value is 'none' (no eviction).
`ttl`	Optional. Time to live in milliseconds. The default is -1 (forever).
`lock_ttl`	Optional. Specifies in milliseconds the duration of a lock placed on the space. The default is -1 (forever).
`lock_wait`	Optional. For a space that is locked, specifies how long a member process will wait for it to become unlocked. The default is -1 (forever). Other valid values are 0 or any positive value. The unit of measure is milliseconds.
`lock_scope`	Optional. Specifies the lock scope to be used for each operation that includes locking. You can specify the following: • **`thread`** The lock applies to the current thread only. • **`process`** The lock applies to the entire application. The default value is `thread`.
`space_wait`	Specifies the space wait for the specified space. The space wait value is a timeout that applies to operations that cannot be processed because the space is not in the READY state, i.e., the space in the INITIAL, LOADING, RECOVER, or SUSPEND state.

Table 8 define space Parameters

Parameter	Description
`write_timeout`	Specifies the write timeout value that is set for the space. The write timeout value applies to Put, Take, Lock, and Unlock operations.
`read_timeout`	Specifies the read timeout value for a specified SpaceDef. The read timeout value applies to Get operations.
`phase_count`	An integer that specifies a phase count value. The phase count value specifies how many phases are carried out when an operation is performed. Phase count is used in conjunction with the `phase_interval` parameter, which specifies how long each phase lasts, in milliseconds. If you do not specify a phase count, the system uses the default phase count value (-1), which specifies that the phase count is calculated internally based on the number of entries in the space. If the phase count is small, more entries are redistributed in each phase, and will take longer for clients to operate. If the phase count is large, fewer entries are redistributed and it will take less time to complete operations.
`phase_interval`	A value that specifies a phase interval, in milliseconds. The phase interval specifies the duration of each processing phase. The default value is 200 milliseconds. Phase interval is used in conjunction with the `phase_count` parameter. For example, to specify that there will be 10 phases and each phase duration is 200 milliseconds, you would enter: **`create space name "test" (field name "k" type "blob") key(fields("k")) phase_count 200 phase_interval 10`**

Examples

Simple example

```
define space name 'myspace' (field name 'key' type 'integer' field
name 'value' type 'string') key (fields ('key'))
```

With additional parameters

```
define space name 'testspace' (field name 'key' type 'double' field
name 'value' type 'blob') key (fields ('key')) distribution_policy
'non_distributed' replication_count 1 replicated_policy 'sync'
capacity 10000 eviction_policy 'none' persistence_type
'share_nothing' persistence_policy 'sync'
```

With distribution key parameters

```
define space name 'usertable3' (field name 'KEY' type 'string',
field name 'field0' type 'string', field name 'field1' type
'string',field name 'field2' type 'string',field name 'field3' type
'string',field name 'field4' type 'string', field name 'field5'
type 'string', field name 'field6' type 'string', field name
'field7' type 'string', field name 'field8' type 'string', field
name 'field9' type 'string') key (fields
('KEY','field0','field1','field2')) distribution_def
('KEY','field0','field1','field2') distribution_policy
'distributed' replication_count 0
```

disconnect

Command

Syntax `disconnect`

Purpose Disconnects the Admin CLI from the metaspace to which it is currently connected.

Parameters None.

drop space

Command

Syntax `drop space <name>`

Purpose Drops the named space.

Remarks It is required that no members be connected to the space when the `drop space` command is executed.

Note that by definition, the `as-agent` process joins all user spaces as a seeder. Therefore, the `drop space` command might fail if you are running `as-agent` in the same cluster.

Parameters The following table shows parameters for this command.

Table 9 drop space Parameters

Parameter	Description
name	Name of the space to be dropped.

export metaspace

Command

Syntax `export metaspace [to <filepath>]`

Purpose Exports the definitions for the metaspace and its spaces (if any are currently defined) to a designated text file or to the screen.

Remarks The file that is created can be used as a parameter when invoking as-admin with the `-i` parameter, as shown below.

Example using java -jar lib/as-admin.jar -i

1. Export a metaspace to a file by invoking `export metaspace`, including '**`to`**' and a filepath parameter. Use single- or double-quotes around the filepath:

   ```
   as-admin> export metaspace to 'C:\temp3\saved_metaspace.txt'
   ```

2. Quit as-admin:

   ```
   as-admin> quit
   ```

3. Launch as-admin with the `-i` flag and the filepath (no quotation marks):

   ```
   C:\tibco\as\2.0\lib>java -jar as-admin.jar-i
   C:\temp3\saved_metaspace.txt
   ```

4. The output will indicate that metaspace has been recreated and as-admin is now connected to this newly-created metaspace:

   ```
   C:\tibco\as\2,0\lib> java -jar as-admin.jar -i
   C:\temp3\saved_metaspace.txt admin.jar -i
   C:\temp3\saved_metaspace

   Connected to metaspace ms
   ```

Parameters The table shows parameters for this command.

Table 10 export metaspace Parameters

Parameter	Description
filepath	Optional. The path to a location where the text file will be saved. If no filepath is given, the metaspace information is displayed on the screen.

help

Command

Syntax `help`

Purpose Provides a complete list of Admin CLI commands and their syntax.

Remarks The `help` command also displays the keyboard shortcuts available within the Admin CLI. Context-sensitive help (such as the syntax for the current command) is displayed when the question mark (**'?'**) character is typed.

The syntax for the as-admin command, with all possible parameters, is displayed by invoking:

`java -jar as-admin.jar-h`:

Usage: **`java -jar as-admin.jar-h - -i admin-cmds-filename`**

Parameters None.

quit | exit | bye

Command

Syntax `quit | exit | bye`

Purpose Exits the Admin CLI.

Parameters None.

recover space

Admin CLI Command

Syntax `recover space <string> (with | without) data`

Purpose If shared-nothing persistence is implemented for the space, specifies whether, when the space is recovered, data is written from disk to the in-memory grid. If you specify **with data**, data is recovered from disk.

Parameters The following table lists the parameters for this command with a description of each parameter.

Table 11 recover space Parameters

Parameter	Description
data	You can specify **with data** or **without data**. If you specify **with data**, then if shared-nothing persistence is set for the space, data is written from disk to the in-memory grid. If you specify **without data**, recovery without data is performed (no data is read from the old file).

Example

```
recover space 'myspace' with data
```

Parameter values must be enclosed in either single or double quotes.

resume space

Admin CLI Command

Syntax `resume space <string>`

Purpose This command is used when shared-nothing persistence is specified for the space. Specifies that the specified space is resumed.

Parameters The following table lists the parameters for this command with a description of each parameter.

Table 12 resume space Parameters

Parameter	Description
<string>	Specifies the name of the space to resume. If shared-nothing persistence is set for the space and the space loses one of its persisters, then the space is set to the SUSPENDED state, which means that no writes to persistence files can occur. You can issue the resume space command to put the space back in a normal state.

Example

```
resume space 'myspace'
```

Parameter values must be enclosed in either single or double quotes.

show | describe member

Command

Syntax `show | describe member <name>`

Purpose Outputs information to the screen about the member specified with the `name` parameter.

Parameters The table shows parameters for this command.

Table 13 show member Parameters

Parameter	Description
`name`	Name (member ID) of the member for which information is desired.

Output The following example shows output from the `show member` command.

```
show member 'test_member'
------------------------------------------------------------------
 Member id         : a62c8ce-c350-4febaa34-35e
 Member name       : test_member
 Member ip:port    : 10.98.200.206:50000
 Member join time  : 2012-06-28T00:49:56GMT
 Member role       : manager
 Member context    : none
------------------------------------------------------------------
No spaces
```

Member Attributes

Member id The system-assigned member ID for the member.

Member name If a name was specified when the member was created, shows the member name.

Member ip:port Indicates the IP address and port number for the member.

Member join time Indicates the time that the member joined the space.

Member role Indicates the member role. Can be `member` or `manager`.

Member context If the application that is managing the member has set up a thread context, displays the thread context; otherwise, indicates `none`.

show | describe members

Command

Syntax `show | describe members`

Purpose Outputs to the screen a list of all members currently running in the metaspace.

Remarks In addition to the name (member ID) of each space in the metaspace, the output includes the *role* of the member in the space, that is, whether the member is a metaspace manager (for the metaspace group membership protocol) or just a metaspace member.

If remote members are active, the output shows the remote members.

If you have started an as-agent using the -name parameter, for example:

```
as-agent -name agent1
```

then you can issue the show member command and specify the agent name on the command line, as follows:

```
show member "agent1"
```

This will output information about the as-agent that has been launched with the specified member name.

Parameters None.

show | describe space

Command

Syntax `show | describe space '<name>'`

Purpose Outputs information to the screen about the space specified with the `name` parameter.

Remarks See the Output section below for information on the output of the `show space` command.

Parameters Table 14 shows parameters for this command.

Table 14 show space Parameters

Parameter	Description
name	The name of the space for which information is desired. The name value must be in single or double quotes.

Output

The output of the `show space` command is displayed in the form of three tables as described in this section.

Space definition attributes

The first section of the output provides information about the space's attributes, arranged in the following columns:

Space Name The name of the space, as given when the space was defined.

Space State The current state of the space. The value can be **`initial`** (meaning that the Space needs more seeders or more persisters), **`loading`** (meaning the space is being loaded from the persistence layer), or **`ready`** (meaning the space has enough seeders, persisters, and has been loaded)

Distribution Policy Can be either **`distributed`** (meaning the space is distributed) or **`non_distributed`**.

Replication count The replication count is displayed. Default is 0, meaning there is no replication.

Replication Policy Can be **`sync`** or **`async`**.

Capacity The capacity value for the space in number of entries per seeder. **`-1`** indicates that there is no capacity limit.

Eviction policy The policy of the eviction to be applied when a space operation would cause the capacity to be exceeded. Can be **none** (no eviction) or **LRU** (least recently used eviction).

Min seeders The minimum number of seeders that need to be joined to the space before the space becomes ready.

Persistence Type Can be **none**, **share_all**, or, **share_nothing**.

Persistence Policy Can be **none**, **sync** or **async**.

Update transport The transport protocol used to distribute notifications of updates to the data stored in the space. Can be **unicast** or **multicast**.

Entry TTL The TTL (time-to-live) of the entries stored in the spaces in milliseconds. Default is **-1** (forever).

Lock wait The **Lock wait** defined for the space is displayed. Default is **-1** (forever).

Lock TTL The **Lock TTL (time-to-live)** defined for the space is displayed. Default is **-1** (forever).

Lock scope Indicates the lock scope for the space, if set. Can be **thread** or **process**.

Space wait Indicates the space wait value for the space, if a space wait value has been set.

Write timeout Indicates the write timeout value for the space, if a write timeout has been set.

Read timeout Indicates the read timeout value for the space, if a read timeout has been set.

Phase count Indicates the phase count value in effect for the space.

Version num Indicates how many time the space definition has been changed since it was defined initially. This value is incremented for each alter space operation. The initial value is 0.

Fields

For each field, the name and type of the field, and whether or not the field is nullable (optional), is displayed.

Below this list of field definitions are names of all the fields that make up the space's key.

Key and Index

The display shows any primary keys that have defined for the space and secondary indexes. For each key or index, the output indicates the index type (HASH or TREE) and the fields for that key or index definition.

```
Key : {KeyDef index_type=HASH, fields=[KEY, field0, field1,
field2]}
```

In addition, the display indicates any distribution fields that have been set up:

```
Distribution Fields: {[KEY, field0, field1, field2] }
```

Distribution fields are key fields that have been configured for affinity—for these fields, all tuples that have the same value are stored on the same seeder.

Members

List the members of the space.

Statistics

The count of put, get, take, seeded, and replicated entries is displayed, as well the number of expires, evictions, locks, unlocks and invokes. These counts are displayed per member and the cumulative count for all members connected to this space.

show | describe spaces

Command

Syntax `show | describe spaces`

Purpose Lists all spaces defined for the metaspace.

Parameters None.

validate policy_file

Admin CLI Command

Syntax `validate [policy_name <string>] policy_file <string>`

Purpose Use the validate policy_file command to validate the syntax and settings in a specified security policy file.

Parameters The following table lists the parameters for this command with a description of each parameter.

Table 15 validate policy_file Parameters

Parameter	Description
policy_name	Optional. Specifies the name of a specific security policy to be validated.
policy_file	Specifies the name of the security policy file to be validated.

Example `validate policy_file 'policy.txt'`

Parameter values must be enclosed in either single or double quotes.

The following examples illustrate the syntax of the **`validate policy_file`** command:

- `validate policy_file 'policy.txt'`
- `validate policy_name 'mypolicy' policy_file 'policy.txt'`

validate token_file

Admin CLI Command

Syntax `validate token_file <string>`

Purpose Use the **`validate token_file`** command to validate a security token file that you have generated by using the **`define | create token_file`** command.

Parameters The following table lists the parameters for this command with a description of each parameter.

Table 16 validate token_file Parameters

Parameter	Description
token_file	Specifies the name of the token file to be validated.

Example

```
validate policy_file "policy.txt"
```

Parameter values must be enclosed in either single or double quotes.

The following examples illustrate the syntax of the **`validate token_file`** command:

- `validate token_file "mytoken"`

validate truststore

Admin CLI Command

Syntax `validate truststore cert_file <string>`

Purpose Use the validate truststore file to validate a specified security certificate file.

ActiveSpaces validates the specified truststore and outputs error messages if errors are found. If the certificate file is validated successfully, the output indicates the X509 information and validity time period.

Parameters The following table lists the parameters for this command with a description of each parameter.

Table 17 validate truststore Parameters

Parameter	Description
cert_file	Specifies the name of the certificate file that is to be validated.

Example `validate truststore cert_file "mycert.pem"`

Parameter values must be enclosed in either single or double quotes.

The following examples illustrate the syntax of the `validate truststore` command:

- `validate truststore cert_file "mycert.pem"`

The following example shows sample output from the command:

Example 2 Sample Output from the validate truststore command

```
as-admin> validate truststore cert_file "mycert.pem"

 [1] /CN=AS-DOMAIN-FEF3A467 [serial: A9CD21B4A05FB244]
 [valid after: Jan 17 03:10:34 2013 GMT, before: Jan 17 03:10:34
2014 GMT]
Found [1] requestor trust anchor(s)
Truststore validated successfully: mycert.pem
```

Chapter 3 Using the as-dump Utility

This chapter describes how to use the `as-dump` utility to view backup files that are used with ActiveSpaces shared-nothing persistence.

Topics

Overview of as-dump

The `as-dump` program is a utility that can you can run offline to examine shared-nothing persistence files and get information such as the ActiveSpaces version the file was created with, the number of entries in the file, and optionally, the data in the file entries. As-dump is useful for examining the data store files created when you are using shared-nothing persistence and detecting possible problems.

The `as-dump` utility takes a file name as and/or a specific directory name as input. You must provide the full directory path.

If you provide the directory name, `as-dump` reads all files in the directory or in its subdirectory and prints out information on each of the files. If you specify a filename, `as-dump` outputs information for the specified file only.

as-dump Syntax

The as-dump utility has the following syntax:

```
as-dump [-vhm] <filepath/directory>
 -h help
 -v verbose, verbose output
 -m <int>, number of entries to print at once
```

Table 18, as-dump Parameters describes the syntax for as-dump.

Table 18 as-dump Parameters

Parameter	Description
filepath/directory	Required.
`-h`	Displays the command syntax for as-dump.
`-v`	Verbose. Displays detail for each tuple in the file.
`-m <int>`	Specifies output of *int* entries.

Sample Output

The output from `as-dump` indicates:

- The location of the persistence file (or files if you specify the directory name only).
- The filename for each persistence file.
- The ActiveSpaces version that created the files.
- The number of entries in the file.
- If you specify the verbose option, the contents of each tuple.

The following example shows output from the `as-dump` command with the verbose option and a specified filename specified.

```
as-dump -v c:\tmp\ms\shared_nothing_persisted\
a62c937-c350\a62c937-c350_store_1352146870

[2012-11-05 14:29:23:101][4896][7040][INFO][asdump]
fileloc=c:\tmp\ms\shared_nothing_persisted\a62c937-c350\a62c937-c3
50_store_1352146870
[2012-11-05 14:29:23:101][4896][7040][INFO][asdump] <?xml
version="1.0" encoding="UTF-8"?>
[2012-11-05 14:29:23:201][4896][7040][INFO][asdump] <Dump>
[2012-11-05 14:29:23:301][4896][7040][INFO][asdump]
<Filter></Filter>
[2012-11-05 14:29:23:401][4896][7040][INFO][asdump]     <File>
[2012-11-05 14:29:23:501][4896][7040][INFO][asdump]
<FilePath>c:\tmp\ms\shared_nothing_persisted\a62c937-c350\a62c937-
c350_store_1352146870</FilePath>
[2012-11-05 14:29:23:601][4896][7040][INFO][asdump]
<Version>2.0.2</Version>
[2012-11-05 14:29:23:701][4896][7040][INFO][asdump]
<ViewNumber>2</ViewNumber>
[2012-11-05 14:29:23:801][4896][7040][INFO][asdump]
<Entries>
[2012-11-05 14:29:23:901][4896][7040][INFO][asdump] <Tuple>
        <value type="int32" pos="0" name="key"> 1 </value>
        <value type="string" pos="1" name="value"> one </value>
        <value type="date_time" pos="2" name="time">
2012-11-05T20:21:33.000GMT </value>
</Tuple>
[2012-11-05 14:29:24:1][4896][7040][INFO][asdump] <Tuple>
        <value type="int32" pos="0" name="key"> 2 </value>
        <value type="string" pos="1" name="value"> two </value>
        <value type="date_time" pos="2" name="time">
2012-11-05T20:21:41.000GMT </value>
</Tuple>
[2012-11-05 14:29:24:101][4896][7040][INFO][asdump] <Tuple>
        <value type="int32" pos="0" name="key"> 3 </value>
        <value type="string" pos="1" name="value"> three </value>
```

```
        <value type="date_time" pos="2" name="time">
2012-11-05T20:21:54.000GMT </value>
</Tuple>
[2012-11-05 14:29:24:201][4896][7040][INFO][asdump] <Tuple>
        <value type="int32" pos="0" name="key"> 4 </value>
        <value type="string" pos="1" name="value"> four </value>
        <value type="date_time" pos="2" name="time">
2012-11-05T20:22:02.000GMT </value>
</Tuple>
[2012-11-05 14:29:24:301][4896][7040][INFO][asdump] <Tuple>
        <value type="int32" pos="0" name="key"> 5 </value>
        <value type="string" pos="1" name="value"> five </value>
        <value type="date_time" pos="2" name="time">
2012-11-05T20:22:11.000GMT </value>
</Tuple>
[2012-11-05 14:29:24:424][4896][7040][INFO][asdump]
</Entries>
[2012-11-05 14:29:24:524][4896][7040][INFO][asdump]
<EntryCount>5</EntryCount>
[2012-11-05 14:29:24:624][4896][7040][INFO][asdump]
<InvalidBlockCount>0</InvalidBlockCount>
[2012-11-05 14:29:24:724][4896][7040][INFO][asdump]
<ExceptionCount>0</ExceptionCount>
[2012-11-05 14:29:24:824][4896][7040][INFO][asdump]     </File>
[2012-11-05 14:29:24:924][4896][7040][INFO][asdump] </Dump>
```

Chapter 4 Using as-agent

This chapter describes how to use the as-agent program to administer distributed spaces from a host.

Topics

Overview of as-agent

ActiveSpaces provides an agent process called `as-agent`. The main purpose of `as-agent` is to join all distributed spaces in the specified metaspace as a seeder. You can run as-agent on any host.

You can also use as-agent to:

- Ensure that the desired degree of replication specified for a space is achieved.

  Because the amount of data that can be stored in a space depends on the number of seeding members of the space, you might need to add seeders to a space to scale it up. Although any process using the ActiveSpaces API can become a seeder on a space, there are situations where applications may not always be running and joined to the space, or where not enough applications have joined the space.

- Ensure data integrity.

  Because the data contained in the space disappears along with the last seeder in the space, this can be a problem. You can use the as-agent process to ensure that there are always one or more seeders for each space in the metaspace.

  Agents remain connected to the metaspace, and always seed the system spaces, keeping the metaspace's space definitions alive, even when all other applications have quit the metaspace.

Starting as-agent

To start `as-agent`:

1. Go to the ActiveSpaces `\bin` directory:

   Windows:

   Default location: `C:\tibco\as\2.0\bin`

   UNIX/Linux:

   Default location: `/opt/tibco/as/2.0/bin`

2. Enter **`as-agent.`**

You can specify a number of optional parameters. Each parameter specifies a parameter name and a value.

The syntax for launching as-agent with parameters is as follows:

```
as-agent -metaspace <metaspace_name> -discovery <discovery_url>
 -listen <listen_url> -name <member_name> -remote_listen
  <remote_listen_url> -log <log_file> -debug <log_level> -name
  <member_name> -admin -data_store <directory path>
```

If you start as-agent with the `-member` parameter; for example **`as-agent -name agent1`**, then when you run the Admin CLI you can issue the `show member` command and specify the member name of the agent to display the attributes of the member; for example, `show member -name "agent1"`

3. To display usage help for as-agent, enter **as-agent -help**

The output of the help command is shown here. It includes the default values used if no parameter is provided by the user:

```
Usage:
-metaspace <metaspace_name> default ms
-discovery <url> / <url_list> default tibpgm://
-listen <url> default tcp://
-remote_listen <url>
-log <log_file>
-debug <log_level> default 3 (INFO)
-name <membername>
-admin
-security_token <token_filename>
-data_store <directory path>

Discovery url format:
  tcp://interface:port;interface2:port2;interface3:port3
  tibpgm://dport/interface;multicast/key1=value1;key2=value2;
  key3=value3
  tibrv://service/network/daemon
Listen url format: tcp://interface/listen_port
Remote listen url format: tcp://interface/remote_listen_port
```

The member name that you specify with the -name parameter can specify simply the member name; or, if you are implementing host-aware replication, can specify a membername in the form *a.b*, where *a* specifies the name of a region, for example *region1*, and *b* specifies the name of a seeder running in that region, in effect, on the same host.

For information on deploying host-aware replication, see Host-Aware Replication, page 5.

There is also a Java version of `as-agent` available in the `lib` directory that can be launched using **`java -jar as-agent`**, and which takes the same argument as the C version of the agent described above. The Java version of the `as-agent` behaves exactly as the C version, but will be able to service requests to remotely invoke methods as triggered by other applications using the space (obviously as long as it has the classes being invoked present in it's `CLASSPATH`).

For more details about discovery and listen URLs, see Chapter 3, "Performing Basic ActiveSpaces Tasks," in the *TIBCO ActiveSpaces Developer's Guide*.

- Connecting to the Metaspace
- Discovery Attribute
- Listen Attribute

Starting as-agent with Security Enabled

If you are using `as-agent` to start a requestor node that is authorized using a security domain controller that uses a security token file, the command to start `as-agent` must include the **`-security_token`** parameter. This parameter specifies the name of the security token file that has been placed on the controller node.

The following example shows how to start as-agent to start a requestor node with security enabled:

```
java -jar as-agent.jar -metaspace ms -security_token
"exdomain_token.txt"
```

Command Parameters

This section gives examples of the as-agent command with additional parameters specified.

Running as-agent to Include a Command Console

To run as-agent and display an as-admin console that allows you to enter Admin CLI commands, specify the **`-admin`** parameter as follows:

```
java - jar as-agent.jar -admin
```

When `as-agent` starts, an `as-agent` command prompt appears, which allows you to enter Admin CLI commands.

Log File Example

If you include the parameter **-log <***log_file***>**, then the log filename will be log_file-<*processid*>.log. For example, if you enter -log as, then the log filename is **as-<processid>.log**.

Using the -debug Parameter

To specify a log level, specify:

-debug <*log_level***>**

The log levels are as follows:

```
1 = ERROR_LEVEL
2 = WARNING_LEVEL
3 = INFO_LEVEL
```

The default is 3 (INFO). The log information displayed on the console is minimal and cannot be controlled through this parameter. This parameter is only for log files. If a log file is not specified, then the debug (log level) value is ignored.

Using the -data_store Parameter

The `-data_store` parameter specifies the directory path where shared-nothing persistence files are stored.

If you are using shared-nothing persistence, specify the `-data_store` parameter when you run as-agent; for example:

as-agent -data_store <*directory_path***>**

where *directory_path* specifies the directory path for the shared nothing persistence data store.

Remote Invocation

To provide necessary class files to support remote invocation, start your Java as-agent as follows:

```
java -Djava.ext.dirs=$LOCATION_OF_JARS$ -jar as-agent.jar
```

Otherwise, running java as-agent in a cluster that includes remote members will cause invoke methods to fail with a “ClassNotFound” exception.

Chapter 5 Administering ActiveSpaces Security

This chapter describes the main administration tasks for administering TIBCO ActiveSpaces® security.

Topics

Main Tasks for Setting Up Security

This section describes the basic entities involved in ActiveSpaces security and provides an overview of the basic tasks needed to implement security.

Basic Entities Involved in Security

Configuring and maintaining security involves the following elements:

- **as-admin utility** Sets discovery parameters, generates and maintains security configuration files.
- **policy files** Specifies security settings across metaspaces, binds metaspaces to security domains.
- **token files** Define connection parameters to secured metaspaces.
- **ActiveSpaces API** Sets up and manages access to secured metaspaces.

Table 19, Tasks for Setting Up Security lists the main tasks for setting up ActiveSpaces security.

Table 19 Tasks for Setting Up Security

Task	See
Create a Policy File	Creating a Security Policy File, page 74
Edit the Policy file	Editing a Security Policy File, page 75
Set up Data Encryption	Setting up Data Encryption, page 78
Validate the Security Policy file	Validating a Security Policy File, page 79
Create a Security Token	Creating a Security Token, page 80
Validate a Security Token	Validating a Security Token File, page 82
Set up Authorization	Setting Up Authorization, page 83

Table 19 Tasks for Setting Up Security

Task	See
Start Security Domain Controllers	Starting Security Domain Controllers, page 84
Start Security Domain Requesters	Starting Security Domain Requesters, page 86

Creating a Security Policy File

To create a security policy file:

1. Start the **as-admin** utility:

   ```
   java -jar as-admin.jar
   ```

2. At the `as-admin` command prompt, enter:

```
create security_policy policy_name "mypolicy/mydomain"
[encrypt <boolean>][validity_days <integer>] policy_file
<policy_filename>
```

```
For example
create security_policy policy_name "mypolicy/mydomain" encrypt
true validity_days 100 policy_file "acme_policy.txt"
```

where:

- **policy_name** Is an optional parameter that specifies the name of the security policy and security domain that is created. If you do not specify this parameter, a policy named `AS-POLICY` and a domain named `AS-DOMAIN` are created.
- **encrypt** Is an optional parameter that indicates whether the identity for the domain is to be encrypted. Each policy can have one or more domains.The default is `encrypt true`.
- **validity_days** An integer that specifies how long the domain ID that the command creates remains valid. The default value is 365 days.
- **policy_filename** Specifies the name of the policy file that is created.

You are prompted to enter a domain password:

```
New domain password [mydomain]:
```

3. Enter a password for the domain.
4. You are prompted to verify the domain password.

```
Verifying - New domain password [mydomain]:
```

5. Re-enter the password to confirm it.

A message appears indicating that the policy was created; for example:

```
Policy created at: acme_policy.txt
```

Editing a Security Policy File

After you have created a policy file for your security domain, you must edit the settings in the file to specify the security configuration for the domain.

For detailed information on editing security policy files, refer to Chapter 4, Implementing ActiveSpaces Security in the *TIBCO ActiveSpaces Developer's Guide*.

Table 20, Security Policy File Sections indicates the sections in the policy file. Some sections are optional and might not need to be modified from the default values or specified in your security policy file. These sections are marked as "optional."

Table 20 Security Policy File Sections

Setting	Requirements	Description
Metaspace Access List	Required	Specifies the metaspaces in the security domain and the discovery URL to be used to discover each metaspace. You must specify at least one metaspace and discovery URL. For detailed information on the settings, see Metaspace Access List on page 111 in the *TIBCO ActiveSpaces Developer's Guide*.
Transport Security	Required	Specifies the type of Transport security used when ActiveSpaces data is transmitted. For detailed information, see Transport Security on page 113 in the *TIBCO ActiveSpaces Developer's Guide*.
Restricted Transport Access	Required	Specifies whether transport access is restricted to members with specific token identities For detailed information, see Restricting Transport Access on page 114 in the *TIBCO ActiveSpaces Developer's Guide*.

Table 20 Security Policy File Sections

Setting	Requirements	Description
Data Encryption	Optional	Specifies whether tuple data is encrypted in shared memory and when it is persisted on local storage. The default setting, `data_encryption=false`, specifies that data is not encrypted. For detailed information, see Setting up Data Encryption on page 78 in this guide and Data Encryption on page 116 in the *TIBCO ActiveSpaces Developer's Guide*.
Authentication	Optional	Species whether user authentication is required, and if so, what type of authentication is used. The default value, `authentication=none`, specifies there is no authentication. For detailed information, see Metaspace Access List on page 111 in the *TIBCO ActiveSpaces Developer's Guide*.
Security Domain Access Control	Optional	Specifies whether access control is required. The default value `access_control=false`, specifies that there is no access control. For detailed information, see User Access Control on page 123 and Metaspace Access List on page 111 in the *TIBCO ActiveSpaces Developer's Guide*.
Access Control Groups	Optional	Allows you to specify what groups or users are granted access to specified to ActiveSpaces operations. The default policy file contains a `groups` keyword with no groups defined. For detailed information, see the following sections of the *TIBCO ActiveSpaces Developer's Guide:* • Metaspace Access List on page 111 • Access Control Groups on page 124

Table 20 Security Policy File Sections

Setting	Requirements	Description
Access Control Permissions	Optional	Allows you to specify which ActiveSpaces operations are permitted for specified groups or users. The default policy file contains a `permissions` keyword with no permissions defined. For detailed information on setting permissions, see the following sections of the *TIBCO ActiveSpaces Developer's Guide*. • User Access Control on page 123 • Access Control Permissions on page 125

Setting up Data Encryption

TIBCO ActiveSpaces allows you to specify encryption of tuple data for fields that have been defined as secure data fields.

Data encryption is set up in the policy file for each domain and by using the TIBCO ActiveSpaces security API functions.

For detailed information on implementing data encryption, see (Data Encryption, page 116 in the *TIBCO ActiveSpaces Developer's Guide*.

Validating a Security Policy File

To validate a security policy file:

At the as-admin command prompt, enter:

```
validate [policy_name <string>] policy_file <string>
```

where

- **policy_name** specifies the name of the policy to be validated
- **policy_file** specifies the the name of the policy file to be validated.

For example

```
validate policy_name 'mypolicy' policy_file 'policy.txt'
```

Creating a Security Token

Complete this task if you want to create a token. A token is an optional configuration file that can be deployed on nodes that have access to or create secured ActiveSpaces resources. The token is created from the security parameter values set in a specified policy file.

If not used, the keyword “none” is provided for the token file location. In such a case, requestors will trust any controller and these requestors cannot connect to a secured metaspace where transport level authentication is required.

When you create a token, you can specify that it is encrypted: in this case, a requestor can only be started if the password is typed when the node starts.

1. Create a policy file and set the policy parameters required for a token.
2. Run the create security_token command to create a token file.

The create security token command has the following syntax:

```
(define | create) security_token domain_name <string>
policy_file <string> [create_identity [common_name <string>]
[encrypt <boolean>][validity_days <integer>]]token_file <string>
```

For a complete description of the create security_token command, see define | create security_token, page 33.

For example:

```
as-admin> create security_token
domain_name "mydomain"
policy_file "mypolicy.txt"
token_file "mytoken.txt"
```

3. Copy the token file to requestor nodes as needed.
4. Verify OS-level access privileges on the security tokens.

Perform Additional Programming Tasks to Process Authentication Requests

To process authentication requests, once you have set up authentication using the ActiveSpaces CLI, you can code a callback routine for client authentication on requestors and develop code to process authentication requests.

Implementing the API's authentication callback is optional. If authentication is required on a metaspace, a default authenticator is always provided, which prompts the user for the username/account-password or keyfile/keyfile-password. If more sophisticated credential feeding implementations are required, the callbacks can be implemented to customize this behavior.

The ActiveSpaces API provides a sample Java program, ASUserAutenticator.java, that demonstrates the use of a callback routine to process user authentication information.

For a general description of user authentication, see User Authentication on page 118 in Chapter 4 in the *TIBCO ActiveSpaces Developer's Guide*," Implementing ActiveSpaces Security."

For a description the `ASUserAuthenticator` sample program, see ASUserAuthenticator on page 186 in Chapter 5 of the *TIBCO ActiveSpaces Developer's Guide*," Using the Example Code."

Validating a Security Token File

To validate a security token file:'

At the as-admin command prompt, enter:

```
validate token_file <token_filename>
```

where *token_filename* is the name of the token file to be validated.

For example, enter:

```
validate token_file "mytoken"
```

Setting Up Authorization

If you want to provide granular authorization, ActiveSpaces allows you to use using Access Control Lists (ACLs) to set up authorization scopes, rights, and privileges.

For information on setting up authorization, see Enabling User Access Control, page 124 in the *TIBCO ActiveSpaces Developer's Guide.*

Starting Security Domain Controllers

To start a security domain controller:

1. Make sure that the paths to the required system variables are set.

   See Setting the Required Environment Variables, page 17 for information on setting the environment variables.

2. Make sure that you have a valid policy file for the domain.

3. Start the **as-admin** utility:

   ```
   java -jar as-admin.jar
   ```

4. Issue the following CLI command to start the domain controller node:

   ```
   connect security_policy <string> [name <string>] [membername
   <string>]
   ```

   For example:

   ```
   connect security_policy 'mypolicy.txt' name 'ms' membername
   'secure1' listen 'tcp://127.0.0.1:50000'
   ```

   where:

- **security_policy** Specifies a string indicating the name of the security policy file for the security domain.
- **name** Specifies a string containing the name of the metaspace that is specified in the Metaspace Access List within the security policy file
- **membername** Is an optional parameter that specifies a space member name.
- **listen** Specifies a string indicating the listen parameter for the metaspace.

Do not specify the `discovery` parameter for the **`connect`** command when starting either a security domain controller or a security domain requestor. When you start these nodes, the **`connect`** command picks up the discovery setting specified in the security policy file or the security token file for the node.

On the UNIX platform, if a controller is started in background mode, it issues a password error without prompting for a password.

There are several possible workarounds:

- Start the controller normally and provide the password; when authentication is completed, type CTRL-Z and then enter the command **`bg`** to run in the background.
- Alternatively, if customized authentication is required, implement a custom callback function; for example, an application can send its credentials from its command line if needed and the custom callback will use them.

Starting Security Domain Requesters

You can start a security domain requestor with a token file, if you have deployed token files for your security installation, or you can start a requestor without a token file if you have implemented security without a token file.

Starting a Security Domain Requestor with a Token File

To start a security domain controller with a token file:

1. Make sure that the paths to the required system variables are set:

   See Setting the Required Environment Variables, page 17 for information on setting the environment variables.

2. Make sure that you have a valid token file for the domain
3. Start the `as-admin` utility:

   ```
   java -jar as-admin.jar
   ```

4. Issue the following CLI command to start the requestor node:

   ```
   connect security_token <string> [name <string>] [membername
   <string>] [listen <string>]
   ```

   For example:

   ```
   as-admin> connect security_token 'mytoken.txt' name 'ms'
   membername 'client1' listen 'tcp://127.0.0.1:50000'
   ```

   where

- **security_token** Specifies a string indicating the name of the token file for the security domain.
- **name** Specifies a string containing the name of the metaspace that is specified in the Metaspace Access List within the security policy file.

  The metaspace name that you use to start the requestor must be the same metaspace name that you used to start the security domain controller.
- **membername** Is an optional parameter that specifies a space member name.
- **listen** Specifies a string indicating the listen parameter for the metaspace.

Do not specify the `discovery` parameter for the `connect` command when starting either a security domain controller or a security domain requestor. When you start these nodes, the **`connect`** command picks up the discovery setting specified in the security policy file or the security token file for the node.

Starting a Security Domain Requestor Without a Token File

You can start the domain requestor without specifying a security token filename.

For example:

```
connect name "ms" discovery "tcp://127.0.0.1:50000" listen
"tcp://127.0.0.2:50000" security_token "none"
```

Chapter 6 Using ActiveSpaces Monitoring and Management

TIBCO ActiveSpaces provides an administrative GUI, called ActiveSpaces Monitoring and Management (ASMM) that lets you connect to a metaspace, display information about various ActiveSpaces system resources, for example, metaspaces, metaspace members, spaces, entry statistics, and so on, and browse space entries.

Topics

Starting ASMM and Connecting to a Metaspace

To start ASMM:

1. Navigate to the folder where you deployed ActiveSpaces Monitoring and Management.
2. Open a Windows command window.
3. Navigate to the directory where ASMM is installed (normally this is `C:\tibco\as\2.0\asmm`).
4. Enter the following command:

   ```
   java -jar asmm.jar
   ```

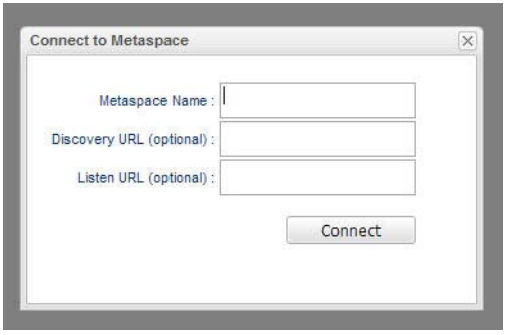
5. Open a browser, and in the address field, enter the URL for ASMM. The default value is:

   ```
   http://localhost:8686
   ```

You can change the port number by editing the deployer properties file for ASMM. This file is located in the directory where you installed ASMM.

ASMM starts and the Connect to Metaspace dialog appears.

Figure 9 Connect to Metaspace Dialog



6. Enter the name of the metaspace to connect to.
7. If you want to enter a Discovery URL or a Listen URL, enter them as required.

The start page for ASMM appears. You are now ready to use ASMM to view the metaspace and configured spaces.

Connecting and Disconnecting from a Metaspace

To select a connection option:

1. On the main page of ASMM, click **Connections**.

   The Connections menu appears.

2. From the pull-down menu, select an option.

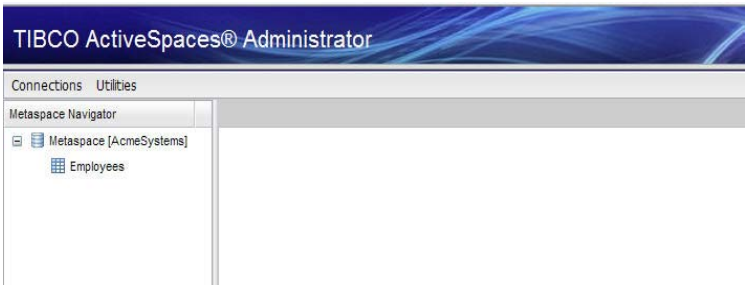
   — To connect to a metaspace, choose **Connect To** and enter the name of the metaspace when prompted.

   — To disconnect from a metaspace, choose **Disconnect From**, and enter the name of the metaspace when prompted.

   — To disconnect from all metaspaces, choose **Disconnect All.**

When you connect to a metaspace, the Metaspace Navigator panel is populated at the left of the ActiveSpaces Administrator page, as shown in the following figure.

Figure 10 Metaspace Manager



The Metaspace Navigator in the figure shows the AcmeSystems metaspace, for which one space has been defined—the Employee space.

Viewing Space Information

ASMM lets you view basic information about each space attached to each metaspace in the data grid. You can view:

- **Definition** Shows the definition for the space.
- **Schema** Shows the schema for the space and the metadata that is active for each tuple field.
- **Members** Shows information about the currently defined space members.
- **Indices** Shows primary keys and secondary indexes for the selected space member.

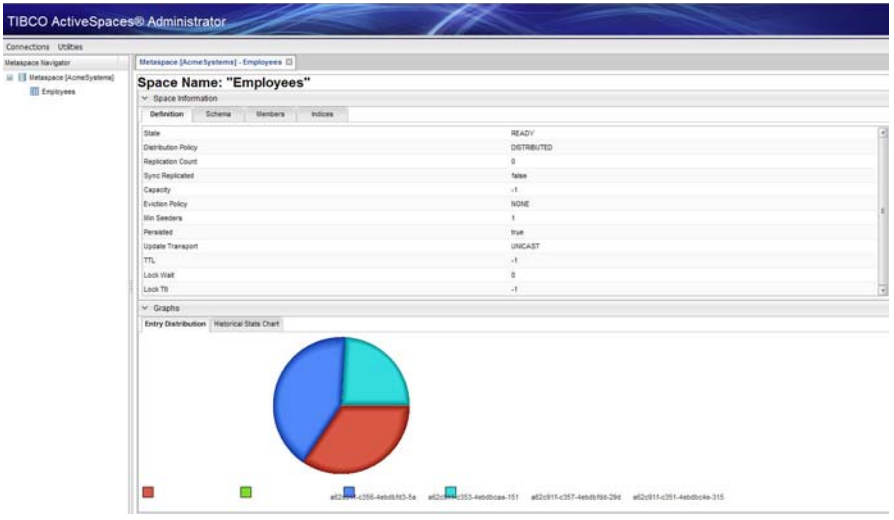
To view information about a space:

1. Connect to a metaspace.
2. Click on the icon for a space defined on the metaspace.

   The Space Information page for the selected space appears.

   The following figure shows the Space Information page for the Employees space that is defined for a metaspace called AcmeSystems. The Employees space contains information that describes the various attributes of an employee at AcmeSystems (for example, name, age, and salary).

Figure 11 Space Information Page



Viewing a Space Definition

To view a space definition, on the Space Name page, click the Definition tab.

For the selected space, ASMM displays:

State The current state of the space. The value can be **`initial`** (meaning that the Space needs more seeders or more persisters), **`loading`** (meaning the space is being loaded from the persistence layer), or **`ready`** (meaning the space has enough seeders, persisters, and has been loaded)

Distribution Policy Can be either **`distributed`** (meaning the space is distributed) or **`non_distributed`**.

Replication count The replication count is displayed. Default is 0, meaning there is no replication.

Replication Policy Can be **`sync`** or **`async`**.

Capacity The capacity of the space in number of entries per seeder. **`-1`** indicates no capacity.

Eviction policy The policy of the eviction to be applied when a space operation would cause the capacity to be exceeded. Can be **`none`** (no eviction) or **`LRU`** (least recently used eviction).

Min seeders The minimum number of seeders that need to be joined to the space before the space becomes ready.

Persistence Type Can be **none**, **share_all**, or, **`share_nothing`**,

Persistence Policy Can be **sync** or **`async`**,

Update transport The transport protocol used to distribute notifications of updates to the data stored in the space. Can be **unicast** or **multicast**.

Entry TTL The TTL (time-to-live) of the entries stored in the spaces in milliseconds. Default is **`-1`** (forever).

Lock wait The **Lock wait** defined for the space is displayed. Default is **`-1`** (forever).

Lock TTL The **Lock TTL (time-to-live)** defined for the space is displayed. Default is **`-1`** (forever).

Lock scope The lock scope defined for the space.

Viewing the Space Schema

To view the schema for the selected space, click the Schema tab.

ASMM displays the schema for the space.

Viewing the Space Members

To view information on the members connected to the selected space, click the **Members** tab.

ASMM displays a list of the space members, as shown in the following figure.

Figure 12 Members Page

For the each space member, the display shows:

- Role—Indicates seeder or leech.
- Puts
- Replicates
- Takes
- Gets
- Expires

Viewing Indices

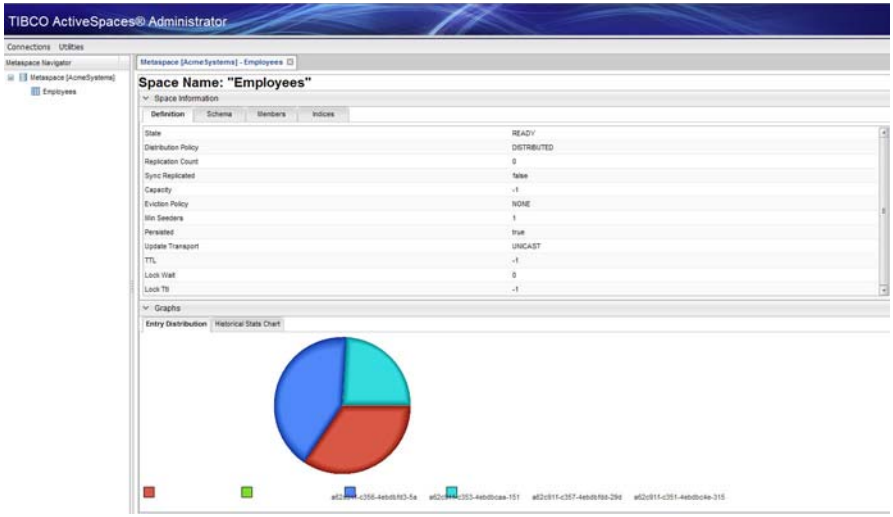
To view the primary keys and secondary indexes defined for a space, click the Indices tab.

ASMM displays the primary key, and any secondary indexes that are defined. For each primary key or index, the index type (HASH or TREE) and the key fields are shown.

Viewing Space Distribution

When you view a space using ASMM, the program shows you how the data for the space is distributed between members. The bottom of the Space Name page shows a pie chart indicating the distribution of entries amongst space members, as shown in the following figure.

Figure 13 Space Name Page with Entry Distribution



You can enlarge the Graphs section by clicking on the Space Properties. This will enlarge the Space Properties area and make the graph appear larger.

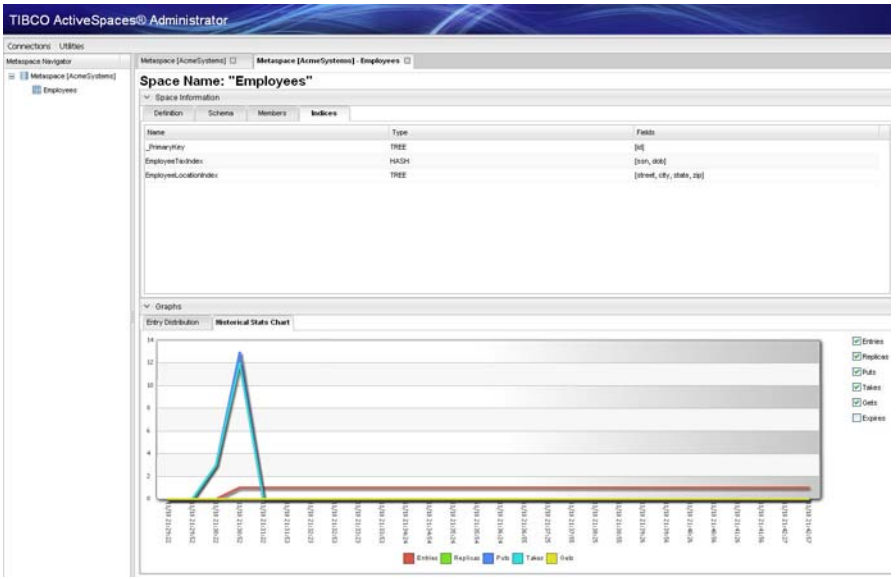
Viewing Historical Statistics

The Space Name page allows you to view historical statistics for space entries in graphical format.

To view historical statistics, in the Graphs section at the bottom of the Space Name page click **Historical Stats Chart**.

ASMM displays a graph of the activity on the space over a specified interval, as shown in the following figure.

Figure 14 Historical Stats Graph Display



To control which actions are shown in the Historical Stats display, check or uncheck the boxes at the right of the graph area for Entries, Replicas, Puts, Takes, Gets, and Expires.

Using the Space Browser

ASMM provides a space browser that lets you browse through the entries defined in a selected space.

The space browser shows you entries that match filter criteria that you specify using the Filter Builder.

To browse the entries in a space:

1. Click **Utilities** at the top of the ASMM page.
2. Click **Space Browser**.

   A list of metaspaces appears.
3. Click on a metaspace name.

   A list of the spaces defined for the metaspace appears.
4. Click on a space name.

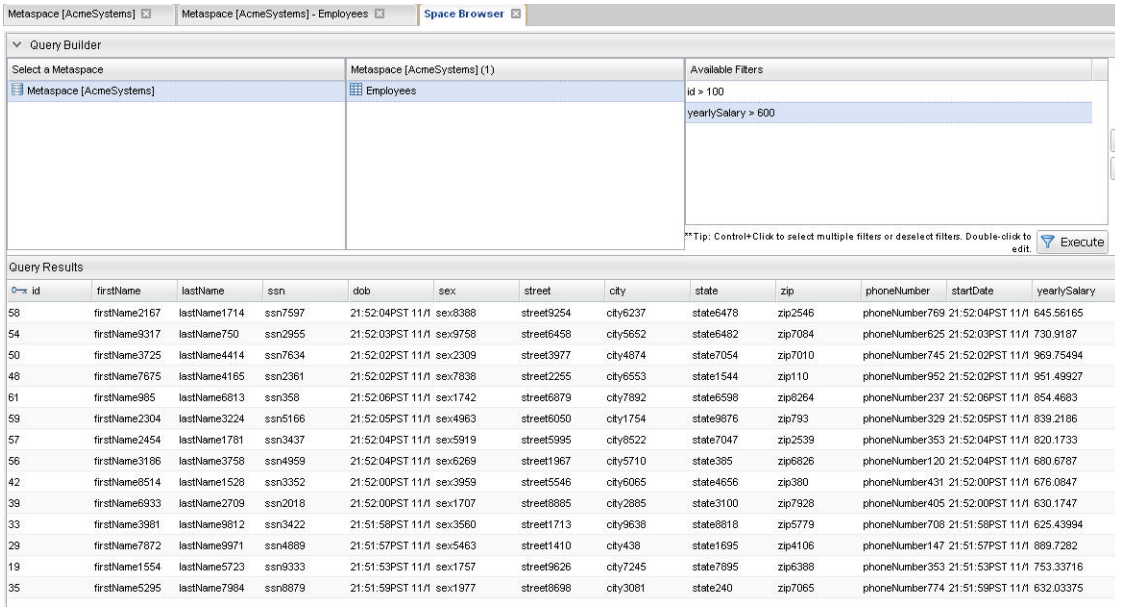
   A list of Available Filters appears at the right of the Filter Builder.
5. Specify a filter definition to use for browsing the space.
   a. Click the plus sign (+) to add a filter.
   b. In the Filter Entry dialog, enter a filter; for example Age < 50.
   c. Continue entering filters until you have built the filter that you want to use.
   d. To remove a filter element, click the delete button (X).
6. To run the query, click the **Execute** button.

   A list of the matching entries appears at the bottom of the display.

Holding down the control key and clicking on filters in the list allows you to select multiple filters or deselect filters. Also, double-clicking on a filter in the list allows you to edit it.

The following figure shows a sample query definition and the resulting entry display.

Figure 15 Query Builder with Tuple Display



id	firstName	lastName	ssn	dob	sex	street	city	state	zip	phoneNumber	startDate	yearlySalary
58	firstName2167	lastName1714	ssn7597	21:52:04PST 11/1	sex8388	street9254	city6237	state6478	zip2546	phoneNumber769	21:52:04PST 11/1	645.56165
54	firstName9317	lastName750	ssn2955	21:52:03PST 11/1	sex9758	street6458	city5652	state6482	zip7084	phoneNumber625	21:52:03PST 11/1	730.9187
50	firstName3725	lastName4414	ssn7634	21:52:02PST 11/1	sex2309	street3977	city4874	state7054	zip7010	phoneNumber745	21:52:02PST 11/1	969.75494
48	firstName7675	lastName4165	ssn2361	21:52:02PST 11/1	sex7838	street2255	city6553	state1544	zip110	phoneNumber952	21:52:02PST 11/1	951.49927
61	firstName985	lastName6813	ssn358	21:52:06PST 11/1	sex1742	street6879	city7892	state6598	zip8264	phoneNumber237	21:52:06PST 11/1	854.4683
59	firstName2304	lastName3224	ssn5166	21:52:05PST 11/1	sex4963	street6050	city1754	state9876	zip793	phoneNumber329	21:52:05PST 11/1	839.2186
57	firstName2454	lastName1781	ssn3437	21:52:04PST 11/1	sex5919	street5995	city8522	state7047	zip2539	phoneNumber353	21:52:04PST 11/1	820.1733
56	firstName3186	lastName3758	ssn4959	21:52:04PST 11/1	sex6269	street1967	city5710	state385	zip6826	phoneNumber120	21:52:04PST 11/1	680.6787
42	firstName8514	lastName1528	ssn3352	21:52:00PST 11/1	sex3959	street5546	city6065	state4656	zip380	phoneNumber431	21:52:00PST 11/1	676.0847
39	firstName6933	lastName2709	ssn2018	21:52:00PST 11/1	sex1707	street8885	city2885	state3100	zip7928	phoneNumber405	21:52:00PST 11/1	630.1747
33	firstName3981	lastName9812	ssn3422	21:51:58PST 11/1	sex3560	street1713	city9638	state8818	zip5779	phoneNumber708	21:51:58PST 11/1	625.43994
29	firstName7872	lastName9971	ssn4889	21:51:57PST 11/1	sex5463	street1410	city438	state1695	zip4106	phoneNumber147	21:51:57PST 11/1	889.7282
19	firstName1554	lastName5723	ssn9333	21:51:53PST 11/1	sex1757	street9626	city7245	state7895	zip6388	phoneNumber353	21:51:53PST 11/1	753.33716
35	firstName5295	lastName7984	ssn8879	21:51:59PST 11/1	sex1977	street8698	city3081	state240	zip7065	phoneNumber774	21:51:59PST 11/1	632.03375

Index