Addendum to Readme for TIBCO ActiveSpaces 2.1.2 Add-On

This addendum describes new software features introduced with the TIBCO ActiveSpaces 2.1.2 Add-On.

Topics

- Setting Up ActiveSpaces Routing, page 2
- C Admin Commands, page 5
- TIBCO ActiveSpaces Hawk MicroAgent, page 7
- Cross-Site Replication, page 12

Setting Up ActiveSpaces Routing

The ActiveSpaces routing feature is implemented by means of a callback function, similar to the callback function used with shared-all persistence. Using routing, your application forwards updates to another site, and receives a status message in return. The operation is blocked until the status is returned.

A space does not require persistence to be enabled to enable routing. If persistence is also enable, both the persister and the router receive callbacks before completing an operation.

This section describes:

- Implementing ActiveSpaces Routing, page 2
- Freeing a Router, page 4

Implementing ActiveSpaces Routing

Implementing routing for a space consists of these steps:

1. Enabling Routing in the Space Definition, page 2
2. Creating the Router Object, page 3
3. Declaring a Callback Function, page 3
4. Setting the Router Object on the Space, page 4

Enabling Routing in the Space Definition

To enable routing for a specified space, connect to a metaspace, create a SpaceDef object, and then call the `tibasSpaceDef_SetRouted()` function (C API) or the `setRouted` method (Java API).

The `tibasSpaceDef_SetRouted()` function is defined as follows:

```
tibas_status tibasSpaceDef_SetRouted(
    tibasSpaceDef spaceDef,
    tibas_boolean routed);
```

where:

- **spaceDef** Specifies the `SpaceDef` object returned to your application by the `tibasSpaceDef_Create()` function.
- **routed** Specifies whether the node data is routed. To route data, specify `TIBAS_TRUE`; otherwise, specify `TIBAS_FALSE`.

Creating the Router Object

To create a Router Object using the C API, call the `tibasRouter_Create()` function. The `tibasRouter_Create()` function is defined as follows:

```
tibas_status tibasRouter_Create(
    tibasRouter* router,
    tibas_onRoute onOpen,
    tibas_onRoute onWrite,
    tibas_onRoute onClose,
    tibas_onRoute onAlter);
```

where:

- **router** Returns a router object that can be associated with a space.
- **onOpen** Specifies the function to be invoked when the tibasSpace_SetRouter() function is called to set the router for a space.Your application is responsible for making the necessary connections to the routed site.
- **onWrite** Specifies the function to be invoked when the there is a Put or Take operation on the node to which data is routed. You function is responsible for performing the Put or Take actions that take effect on the other site.
- **onClose** Specifies the function to be invoked when the connection to another node is terminated (the `tibasRouter_Free()` function is called).
- **onAlter** Specifies the function to be invoked when a space definition is altered.

The callback definition for a router does not provide an `OnRead` function. You cannot perform Get operations over a routed connection.

Declaring a Callback Function

Declare a callback that conforms to the `tibas_onRoute` typedef (C API) or the Router interface (Java API).

The `tibas_onRoute` callback has the following function prototype:

```
typedef void (TIBAS_CALL_API *tibas_onRoute) (
    tibasRouter router,
    tibasAction action,
    tibasActionResult result);
```

where:

- **router** Specifies the router object returned by the `tibasSpace_RouterCreate()` function.
- **action** Returns the action that occurred over the routed connection.

- **result** Returns the result of the action.

In Java there is no need to declare a function prototype: the Router interface provides for methods referenced in the callback.

Setting the Router Object on the Space

After you have configured routing, you must set the Router object for the space.

To set the Router object for the space, call the `tibasSpace_SetRouter()` function (C API) or the `setRouter` method (Java API).

The `tibasSpace_SetRouter()` function is defined as follows:

```
tibas_status tibasSpace_SetRouter(
    tibasSpace space,
    tibasRouter router);
```

where:

- **space** Is a valid space object.
- **router** Specifies the router object returned by the `tibasRouter_Create()` function.

Java Implementation:

The Java `setRouter` method has the following signature:

```
Router setRouter (Router router) throws ASException;
```

Freeing a Router

When the routed connection has been terminated, you should free the router object.

To free the router object using the C API, call the `tibasRouter_Free()` function. The tibasRouter_Free() function is defined as follows:

```
tibas_status TIBAS_COMMON_API TIBAS_CALL_API tibasRouter_Free(
    tibasRouter* router);
```

where `router` specifies the Router object that was used to create the router.

Java Implementation:

Using the Java API, you can free the router by calling the stopRouter method. The stopRouter method has the following definition:

```
void stopRouter (Router router) throws ASException;
```

C Admin Commands

ActiveSpaces 2.1.2 introduces a new C version of the as-admin utility—`as-admin.exe`.

This utility is located in the `AS_HOME/bin` directory and can be invoked by navigating to the /bin directory and entering:

./as-admin

The C Admin commands can also be invoked programmatically by calling the tibasAdmin_Execute() C API function and specifying the command with the admin parameter.

The TibasAdminExecute() function is define as follows:

```
tibas_status tibasAdmin_Execute(
    tibasAdmin admin,
    char** result,
    tibasMetaspace metaspace,
    const char* cmd);
```

Command List

The following new commands have been added to the C Admin command set.

Metaspace Commands

- **commit transaction [all | member_name <string> | transaction_id <string>]**
- **rollback transaction [all | member_name <string> | transaction_id <integer>]**
- **(show | describe) metaspaces**
- **(show | describe) transactions [member_name <string>| state ('uncommitted' | 'committed' | 'rolled_back')]**

Space Commands

- **join space <string>**
- **leave space <string>**
- **(show | describe) space <string> entries [filter <string>] [entry_scope ('all' | 'seeded')] [time_scope ('snapshot' | 'current')] [prefetch <integer>] [query_limit <integer>]**
- **(show | describe) space <string> locks [count] [member_name <string>] [filter <string>]**

- `(show | describe) space <string> size [filter <string>]`
- `unlock space <string> (entry_id <string>| all [member_name <string>] [filter <string>])`

Admin Routing Commands:

- `execute on member <string> <admin_command>`
- `execute on members <admin_command>`
- `execute on proxy <admin_command>`

Logging Commands

- `set event_log log_level <string> [log_file <string>]`
- `set file_log log_level <string> [log_file <string>]`
- `show console_log`
- `show event_log [log_file <string>]`
- `show file_log [log_file <string>]`

Member Listen Port Connection Commands

- `close`
- `open name <string> listen <string>`

TIBCO ActiveSpaces Hawk MicroAgent

The TIBCO ActiveSpaces Hawk Microagent is an ActiveSpaces managed application instrumented with the TIBCO Hawk Application Management Interface (AMI) protocol and which exposes a set of methods that allow the connected Metaspace to be monitored and controlled by TIBCO Hawk tools. The ActiveSpaces Hawk Microagent communicates with the TIBCO Hawk Agent by using TIBCO Rendezvous as the transport for exchanges of monitored and management data to interact with the Metaspace.

With the TIBCO ActiveSpaces Hawk Microagent, you can perform interactive monitoring through the Hawk display or automate the execution of the methods by creating rulebases to raise alerts and actions performed by the TIBCO Hawk Agents. Refer to the Hawk documentation on the concept of a Hawk Microagent.

You can use the TIBCO ActiveSpaces Hawk Microagent to capture:

- The size, throughput, and growth of a Space monitored by a Hawk rulebase.
- The event on definition or the drop of a Space.
- The joining and leaving of members to the Metaspace as well as member to a Space so alerts and corrective actions can be taken.

  Combining the Hawk rules with other TIBCO AMI instrumented applications allows you to build a monitoring and management system for all Spaces and Members of a Metaspace cluster.

Getting Started

The TIBCO ActiveSpaces Hawk Microagent application is packaged and installed as a jar file under the following lib folder:

```
<TIBCO_HOME>/as/2.1/lib/as-hawk-agent.jar
```

Before you start the TIBCO ActiveSpaces Hawk Microagent, ensure that the TIBCO Rendezvous daemon and the Hawk Agent have already been started on the machine. Refer to the TIBCO Hawk documentation for more details.

Start your TIBCO ActiveSpaces application before starting the TIBCO ActiveSpaces Hawk Microagent application to monitor your Metaspace cluster. The Hawk Microagent application primarily collects statistics information for Spaces and Members of your Metaspace by joining the System Spaces as Leech. There is no direct access to the data of the User Spaces.

To start the TIBCO ActiveSpaces Hawk Microagent, the classpath must include the lib folder of both the TIBCO Rendezvous and the Hawk installations. From the command line, it accepts the same set of Metaspace connection arguments as in the as-agent, such as Metaspace name, Discovery and Listen URL.

Additionally, it accepts parameters to establish connection to the TIBCO Rendezvous for communications with the Hawk Agent. The default is at port=7474 when it is not specified. Refer to the ActiveSpaces documentation for details on the command line arguments of the as-agent.

Here is a typical Unix shell script to start the TIBCO Hawk Microagent:

```
#!/bin/sh
export LD_LIBRARY_PATH=<AS_HOME>/lib:<RV_HOME>/lib

java -Djava.ext.dirs=<RV_HOME>/lib:<HAWK_HOME>/lib -jar
as-hawk-agent.jar -metaspace <metaspace>
```

Help information can also be displayed by including with the `-help` argument:

```
java -jar as-hawk-agent.jar -help
```

Once TIBCO ActiveSpaces Hawk Microagent is successfully started, you can use the TIBCO Hawk Display to perform interactive monitor activities on the connected.

From the Hawk display, gather all Hawk Microagent registered and discovered on the machine. The TIBCO ActiveSpaces Hawk Microagent is displayed as:

```
ASRuntimeInfo:<metaspace>:<version>
```

Selecting the TIBCO ActiveSpaces Hawk Microagent displays all the methods exposed for this Metaspace connection. Note that you can monitor multiple Metaspaces by starting multiple instances of the TIBCO Actives Hawk Microagent. Unlike `as-agent`, where you can start multiple instances to perform seeding and operations on User Spaces, you should only have one instance of the TIBCO ActiveSpaces Hawk Microagent per Metaspace connection.

ActiveSpaces Hawk Microagent Methods Overview

The TIBCO ActiveSpaces Hawk Microagent provides the following categories of monitoring methods:

- Space and Member Statistics
- Space and Member Distributions
- Space Operation Throughput at Application Level
- Space Definition Events
- Metaspace Member and Space Member Events

From the TIBCO Hawk display, select a TIBCO Hawk Microagent. The microagent displays all the methods exposed for monitoring the Metaspace cluster. You can typically *invoke* for on-demand request to get the result interactively. You can also *subscribe* to the exposed methods specifying the data delivery polling interval. Data delivery interval for Space Definition and Member events are not required as it is instrumented based on ActiveSpaces Event Listener. In the case of a *subscription*, it allows you to create a monitoring rule for building a Hawk rulebase.

Space and Member Statistics Methods

The following table lists the space and member statistics methods.

Method	Argument
getSpaceStatisticsByName	SpaceName
getSpaceStatisticsAll	none
getSpaceStatisticsByFilter	SpaceNameFilter with wildcard matches
getMemberStatisticsAll	none

The Space Member statistics methods return the result either on a single row or multiple rows.

The information reported is equivalent to the statistics information reported by the as-admin 'show space' command, which includes Entries, Puts, Gets, and etc. Subscribing on these methods from the Hawk console also allows you to chart the change history of all the reported attributes.

Space Distribution Methods

Method	Argument
getSpaceDistributionByName	SpaceName
getSpaceDistributionAll	none
getSpaceDistributionByFilter	SpaceNameFilter with wildcard matches
getMemberDistributionAll	none

The Space and Member Distribution methods return the percentage distribution of the data either grouped by the Space Members of a Space or grouped by Spaces of a Member. You can monitor the distribution of data across Members of a Space. Or you can monitor the distribution of Space entries stored grouped by Member.

Space Operation Throughput at Application Level Methods

Method	Argument
getSpaceThroughputByName	SpaceName
getSpaceThroughputAll	none
getSpaceThroughputByFilter	SpaceNameFilter with wildcard matches
getMemberThroughputAll	none

The Space Operation Throughput methods provide the application level throughput based on the changes of the result attributes over the data delivery interval. For example, it reports the number of Puts per Second by collecting the number of Put operations performed on a Space and divided by the time elapsed over the subscription time interval. You can subscribe and create a rule to create an alert action when the Put throughput spikes or falls below a min/max threshold on a Hawk Rulebase.

Space Definition Events Methods

There is one Space Member events method, as shown in the following table.

Method	Argument
onSpaceDefEvent	none

The Definition Event methods provide an event driven subscription service that allows the monitoring of new Space defined or when a Space is dropped or altered. It reports all the Spaces Definitions in multiple result rows when any Space event.

Metaspace Member and Space Member Events Methods

The following table lists the Metaspace Member and Space Member Events Methods

Method	Argument
onMemberEvent	none
onSpaceMemberEvent	SpaceName

The Metaspace Member and Space Member Events methods provide an event driven subscription service that allows the monitoring of Members joining and leaving the Metaspace or a Space. It reports all the Members of the Metaspace or all the Space Members of a Space along with the attributes of the reported Members when any Member event happens. The additional member attributes also allows you to monitor when a Metaspace Member changes the Manager Role or when a Space Member changes its Distribution Role between Seeder and Leech.

Cross-Site Replication

The ActiveSpaces 2.1.2 Add-on release introduces a new feature called cross-site replication. This feature lets you deploy ActiveSpaces across multiple data centers and have the data replicated between them. Cross-site replication provides a single view of all the data, fault tolerance at the data-center level, and faster peformance through the reduction of round-trip times for clients located near the data centers.

To configure cross-site replication, you should configure each site with the other sites' name and IP address. Sites connect to each other through routers and receivers, where routers push updates out from the local site to the receivers on the remote sites. Multiple routers and receivers can be registered for load balancing and fault tolerance.

Once configured, cross-site replication takes care of connecting and replicating updates from the local site to all remote sites. Replication is guaranteed and automatic. If a remote site is down, updates are recorded in a transaction log and later replayed when the site is back up.