TIBCO ActiveSpaces® - Enterprise Edition

Administration

Version 5.0.0 | February 2025

Copyright © 2009-2025. Cloud Software Group, Inc. All Rights Reserved.

Contents

Contents ... 2

Who Should Read This Document ... 8

About This Product ... 9

Administrative Concepts ... 10

Copysets ... 13

State Keeper ... 14

Realm Service ... 14

Development Environment ... 16

Building a Docker Image ... 18

Production Environment ... 19

String Encoding ... 21

Running Processes as a Service ... 22

Recommended Minimum Configuration ... 22

Logging ... 23

State Keeper ... 24

Administration Service ... 26

Proxy ... 28

Administration Tool ... 30

Administration Tool Reference ... 32

Environment Variables for the Administration Tool ... 33

tibdg Status ... 33

tibdg Table Stats ... 36

tibdg Grid Generate and tibdg Table Generate ... 37
The tibdg Commands That Support Interaction ... 39
Using tibdg grid mode to Put a Data Grid into Maintenance Mode ... 41
tibdg proxy shed ... 42
Using the Proxy Shed Command and the Balanced Binding Strategy ... 43
tibdg purge ... 44
tibdgadmind ... 44
Stop the tibdg Daemon ... 46

Designing a Data Grid ... 47

Starting a Realm Service ... 49

Defining a Data Grid ... 52
Grid Create Configuration Options ... 54
Memory Usage Considerations with the node_read_cache_size Option ... 62
Configuration Options to Use Specific Ports and Network Interfaces ... 63
Configure Ports ... 63
Configuration Options when the Proxy and Client are on Different Subnets ... 64
Configure Network Interfaces ... 66
Configure Internal Subnet Masks ... 66

Starting the Data Grid Processes ... 68
Component Command-Line Parameters ... 68

Starting a State Keeper ... 71
Keeper Reference ... 71

Starting a Node ... 73
Node Reference ... 74
Starting Multiple Nodes ... 74
three_copysets.tibdg ... 75
one_copyset_two_replicas.tibdg ... 76

Starting a Proxy 77
Proxy Reference 78
Starting a Proxy with an External Host and Port 78

Methods of Selecting a Proxy for a Client 80

Adding Copysets 85
Data Redistribution 86

Removing Copysets 88

Defining a Table 91
Table Create Configuration Options 92
Column Names 94
Special Characters in Column Names 95
Secondary Indexes 95
Enabling Statistics 96
Row Expiration 96
Defining a Table with Row Expiration 97
Overriding the Default TTL for a Single Row 98
Deletion of Expired Rows 98

Defining a Table by Using SQL DDL Commands 100
Creating a New Table 100
Dropping a Table 102
Creating an Index 103
Dropping an Index 104
SQL Data Type Mapping 104

Security 106
Authentication and Authorization 106
Authorization Groups 107
Password File 108

Starting Realm Services with Authentication 109
Starting Data Grid Processes With Authentication 110
Using User-Defined TIBCO FTL Certificates 111
Enabling Transport Encryption on a Data Grid 117
Trust File (TIBCO FTL-Generated Certificates) 118
Using Trust Files with Primary Realm Service 119
Using Trust Files with the Disaster Recovery Feature 120

Grid and Table Permissions 121
Enabling Permission Checking on Data Grids and Tables 121
ActiveSpaces Custom Roles 122
Enabling Permission Checking when Creating or Modifying a Data Grid 123
The tibdg Commands to Set Permissions on a Table 124

ActiveSpaces Monitoring Service 127
Using ActiveSpaces Monitoring Service 128

Installing or Uninstalling ActiveSpaces Processes as Windows Services 130

Installing ActiveSpaces Processes as Windows Services 131

Uninstalling ActiveSpaces Processes as Windows Services 133

Deployment Scenario for Running ActiveSpaces Processes as Windows Services 134
Preparing for Installation 135
Installing TIBCO FTL Server as a Windows Service 137
Creating the ActiveSpaces Data Grid 138
Installing the ActiveSpaces State Keeper as a Windows Service 140
Installing the ActiveSpaces Node as a Windows Service 142
Installing the ActiveSpaces Proxy as a Windows Service 144
Installing the ActiveSpaces tibdgadmind as a Windows Service 146
Running an ActiveSpaces Sample 147
Uninstalling the Sample Windows Services 147

Stopping a Data Grid Gracefully 149
Selecting a Secondary Node to be Promoted as the Primary Node 149
Best Practices for Node Synchronization 150
Timeouts During Maintenance 150

Clearing a Data Grid Definition 152

Checkpoints 153
Creating a Checkpoint 153
- Creating a Manual Checkpoint 153
- Creating a Periodic Checkpoint 154

Listing Checkpoints 155
Listing Tables in a Checkpoint 156
Deleting Checkpoints 156
- Automatically Deleting Old Checkpoints 157

Validating Checkpoints 157
Checkpoint Properties 158
Checkpoint Best Practices 159

Caching Rows in a Proxy 160

Live Backup and Restore 163
Restoring a Data Grid 167
- Realm Service Database Restore 168
- Realm Service Checkpoint Restore 169
- Restoring State Keepers 170
- Restoring a tibdg Node 171
- Removing a Rollback Record 172

Disaster Recovery 174
Suggested Deployment Model for Disaster Recovery 175
A Quick Look at Setting Up Disaster Recovery 176
Gridset Configuration 178

- Getting Help on the gridset Command 178
- Creating a Gridset 179
- Adding Data Grids to a Gridset 179
- Modifying a Gridset 180
- Permission Checking in Disaster Recovery Gridsets 180
- Configuring a Proxy with Static Mirroring Host and Port 181

Activating the Mirror Grid as the Primary Grid 181

Preventing Data Loss by Using the Maintenance Mode 184

Retention Limits 185

Automatic Mirroring 185

Recovery Objectives 186

- Recovery Point Objective 186
- Recovery Time Objective 186

Capacity and Sizing 187

- Disk Space Used by the Checkpoint Metadata 187
- Query Capacity 187

Security in a Disaster Recovery Setup 187

Disaster Recovery Playbook 188

- Setting Up a Planned Cutover to a Mirror Grid 188
- Disaster Recovery at a Mirror Grid 189
- Multiple Mirror Sites 190
- Read Replicas 191

TIBCO Documentation and Support Services 192

Legal and Third-Party Notices 193

Who Should Read This Document

The document is primarily focused on administrators. However, some portions of this document cater to the needs of a developer. In such scenarios, the roles are identified at the beginning of a section. Unless specified otherwise, the procedures in the document are meant for administrators.

About This Product

The TIBCO ActiveSpaces® - Enterprise Edition software is a distributed in-memory data grid product. Some features of ActiveSpaces® - Enterprise Edition include the use of familiar database concepts, high I/O capacity, and network scalability. It is ideal for all application development projects, and for deploying and managing applications in the production environment of an enterprise.

TIBCO FTL® is an embedded and bundled component of ActiveSpaces® - Enterprise Edition.

Administrative Concepts

These concepts and definitions pave the way to a more detailed understanding of ActiveSpaces administration.

Data Grid

A set of cooperating processes that distribute data across a set of host computers.

Three kinds of cooperating processes implement a data grid: nodes, proxies, and state keepers.

Copyset

A data grid partitions the complete set of data into copysets. Each copyset contains a portion of the full data set.

Each table row resides within only one copyset.

Partitioning

The data grid horizontally partitions the rows of a table across copysets. So, a query or a transaction can span many copysets.

Node

Nodes are processes that implement a copyset. Administrators define nodes and assign them to copysets.

Each copyset requires a primary node. Secondary nodes can provide optional backup protection.

Each node of a copyset maintains one copy of the data (that is, one copy of all the rows in that copyset).

Each node is part of only one copyset.

Replica

The number of replicas in a copyset is identical to the number of nodes that implement that copyset. Replicas provide fault tolerance and protect data against hardware failures. More replicas yield greater protection.

- In a prototyping or testing environment, you can implement a copyset using only one node.
- In most production environments, two nodes provide adequate protection.
- For even stronger fault tolerance, you can use three nodes.

Replication

The replication feature, when used, provides fault tolerance by preventing data loss when a node (or the computer running the node) fails and cannot be accessed.

All nodes in a given copyset are replicas of each other and they all have the same set of data.

There is a single primary replica in a copyset and the other nodes in that copyset are secondary replicas.

Every copyset in the data grid is organized to ensure that the slice of data owned by that copyset is stored on as many replicas as desired.

Reconciling Nodes of a Copyset

When a node of a copyset is brought back online, the data for the node is reconciled with the primary node. After reconciliation, the node being brought back online resumes as a secondary node of the copyset.

For more information, see Copysets.

Using Multiple Nodes

There are several reasons for using multiple nodes:

- Nodes in different copysets are created with the goal of scaling horizontally.

  As a result, multiple copysets are created, each with a slice of the data.
- Nodes in the same copyset are created to provide multiple replicas for fault tolerance.

  These contain identical copies of the data.
- In a product environment, a combination of the previously described use cases can be used.

  For example, you might choose to have two replicas per copyset and multiple copysets (say three) to scale horizontally.

In this example, your environment would have a total of six nodes.

Proxy

Proxies are processes that mediate data grid operations on behalf of application programs.

Application programs connect to proxies, which in turn connect to nodes.

Proxy processes are independent of one another and do not require persistent state, so you can share the load of operations among multiple proxies.

State Keeper

Fault-tolerant state keeper processes determine and record the data grid's run time state information by which a data grid operates, and supply this information to the proxies and copyset nodes.

A set of fault-tolerant state keeper processes protect this crucial information and ensure nonstop access to it. One of the state keepers is designated the lead state keeper and supplies this information to the proxies and copyset nodes. If the lead state keeper goes down, one of the secondary state keepers takes over as the lead. In a fault-tolerant set of three state keepers, a quorum of two state keepers must always be running to ensure data consistency in split brain scenarios. If a state keeper is restarted when a quorum is running, one of the running state keepers updates the state of the restarted state keeper. If the number of running state keepers falls below the quorum and the state of a copyset changes (for example, a node goes down), operations on the data grid continue to fail until a quorum of state keepers are running again. Until a copyset state change occurs, live operations may still continue working. However, it is critical that a quorum of state keepers is running to provide the full grid functionality.

For more information, see State Keeper.

Service

You can configure a TIBCO FTL server to run one or more processes as services. When a process is run as a service, the TIBCO FTL server is responsible for starting and stopping the process when the TIBCO FTL server starts up and shuts down. The TIBCO FTL server continuously monitors the liveness of the process and restarts it if the process is not running. On starting, the YAML configuration file of the TIBCO FTL server provides the parameters to a service. For a standalone process, you add the process parameters by using command line. For the processes running as a service, add those parameters to the YAML configuration file of TIBCO FTL server. For example, if you used the following

command to start a standalone state keeper:

```
tibgdkeeper --name k1
```

When the state keeper is running as a service, add the following parameters to the YAML configuration file of the TIBCO FTL server:

```
- tibdgkeeper:
name: k1
```

You can run state keeper, administration service, and proxy as services.

ActiveSpaces Core Server

When you configure a TIBCO FTL server to run one or more ActiveSpaces processes, it is known as an ActiveSpaces Core Server.

Copysets

A data grid partitions the complete set of data into copysets. Each copyset contains a portion of the full data set.

The data grid horizontally partitions each table, assigning each row to one specific copyset. This partitioning is transparent to application programs.

Programs explicitly interact with tables, but do not refer to copysets.

Tables and Copysets

Tables and copysets are independent concepts.

Tables organize data in a way that makes sense to users of the data. Tables consist of rows, structured by columns.

Copysets *store* table rows, distributing them across a network in a way that facilitates fast access, fault tolerance, data replication, and flexibility.

State Keeper

The state keeper determines and records the data grid's run time state information by which a data grid operates, and supplies this information to the proxies and copyset nodes.

Runtime Information Stored in the State Keeper

Primary Nodes

Within each copyset, one node is the primary copy, which both stores data and provides read access. Other nodes are secondary nodes that store back-up copies of the data. The state keeper records which node is the primary.

Data Distribution Mapping

The state keeper determines the mapping that assigns each table row to a copyset.

State Keeper Fault Tolerance

A set of fault-tolerant state keeper processes protect this crucial information and ensure nonstop access to it. One state keeper process supplies this information to the proxies and copyset nodes.

In production environments, use three processes. In a prototyping or testing environment, only one process suffices.

For added protection, each state keeper process also maintains a copy of the governing decisions in a disk file.

Realm Service

A data grid is run inside a TIBCO FTL realm. A TIBCO FTL realm serves as a repository for data grid configuration information and provides communication services that enable all data grid processes to communicate with each other. A client application accesses the data grid by using the realm service URL.

The realm service URL is the URL of the TIBCO FTL server. The realm service offers the following capabilities:

- Stores data grid definitions

- Communicates with the administrative tools to store and retrieve data grid definitions
- Communicates with all the processes running in the data grid and updates the internal configuration if anything changes
- Collects monitoring data from all processes

For more information, see "Processes in ActiveSpaces" section in *TIBCO ActiveSpaces® - Enterprise Edition Concepts*.

Development Environment

In many enterprises, programmers act as administrators during the development and test phases of a project. To develop and test application programs that use ActiveSpaces software, deploy the following processes.

- **Realm service** One realm service
- **State keeper** One process or service
- **Node** One process
- **Proxy** One process
- **Administration Daemon or Administration Service** One process or one service
- **Your application programs** One or more processes, as appropriate

In a development environment, you can run all of these processes on the same host computer.

Sample Scripts

Refer to the `TIBCO_HOME/as/<version>/samples/readme.md` before using the sample scripts.

The following scripts are available:

- `TIBCO_HOME/as/<version>/samples/scripts/as-start` defines a simple data grid and starts its component processes.
- `TIBCO_HOME/as/<version>/samples/scripts/as-stop` stops those component processes.

Sample Docker Environment

A sample docker-compose environment is provided to demonstrate how to deploy an ActiveSpaces data grid in Docker. For more information, see `TIBCO_HOME/as/<version>/samples/docker/README.md`.

Sample Kubernetes Environment

A sample Kubernetes manifest file and Helm chart are provided to demonstrate how to deploy an ActiveSpaces data grid in Kubernetes. For more information, see `TIBCO_HOME/as/<version>/samples/kubernetes/README.md` and `TIBCO_HOME/as/<version>/samples/kubernetes-helm/README.md`.

Note: The installation environment of ActiveSpaces is referenced as TIBCO_HOME. For example, on Microsoft Windows, TIBCO_HOME might be `C:\tibco`.

Building a Docker Image

You can build your own Docker images for ActiveSpaces. You can also build an image that includes a sample client program.

A script and Dockerfile are available for building Docker images at `scripts/build-images` location.

For assistance on building Docker images, see `scripts/build-images/README.md`.

For examples of how these docker images can be used, see `samples/docker`, `samples/kubernetes`, and `samples/kubernetes-helm` directories.

Note: The included Dockerfile and script, when used in accordance with the instructions here, downloads and installs the third-party components to create Docker images. We recommend that you review the script to identify the websites from which the components are downloaded to ensure that you understand which license terms apply to these components, what is required to comply with those terms, to track their security status, and determine if and when to update or replace them for security purposes.

For more information about Docker containerization in FTL, see "Docker Containerization for FTL" in *TIBCO FTL® Administration*.

Production Environment

To use ActiveSpaces software in a production environment, deploy the following processes.

- **Realm service**: A realm service is a cluster of TIBCO FTL servers that provide realm services. If one realm service goes down, any of the other services can take over for it provided the applications have included them in their pipe-separated connection URL. For fault tolerance, they must not all be on the same computer. Run either one TIBCO FTL server, or a group of three or five or seven TIBCO FTL servers.
- **State keeper**: The minimum production arrangement consists of three state keeper processes. To ensure high availability during a network partition or hardware failure, each state keeper process must run on a separate host computer. Not doing so might result in grid-wide data loss.

  At any given time, you must maintain a quorum of running state keepers. To run more than one state keeper, configure three state keepers and ensure you have at least two running state keepers.

  As the quorum requirements for the realm service and the state keeper are the same, TIBCO recommends that every TIBCO FTL server must run both a realm service and a state keeper.
- **Administration Daemon or Administration Service**: The minimum number of administration daemons or services is one, but to ensure high availability, you must configure more. TIBCO recommends using the administration service rather than the administration daemon. If you use the administration service, configure every TIBCO FTL server to run an administration service.
- **Node**: The minimum production arrangement consists of two node processes per copyset.

  Optional. For greater data protection, you can run three nodes per copyset.

Note: Additional copies can become expensive in two ways:

- Increasing the node count by one adds one complete copy of all the data.
- Every node process must run on a separate host computer. Usually, this requirement determines the number of host computers that you must maintain. For example, a data grid with three copysets and two nodes per copyset requires six nodes, all on separate hosts. Increasing to three nodes per copyset would require nine nodes, all on separate hosts.

- **Proxy**: The minimum production arrangement consists of one proxy process.

  Optional. You can run additional proxies to increase the capacity for client programs and to improve response time. For best results, run proxy processes on a separate host computer.

- **Your application programs**: Run processes as appropriate.

Components Sharing a Host Computer

You can reduce the number of host computers in a production environment by running more than one component per host.

For example, you can run a realm service, a state keeper, a node, and a proxy, all on one host. (In contrast, do not run two state keepers on the same host.)

For effective fault tolerance, run the nodes of each copyset on separate host computers.

Warning: Combining component processes on a host computer increases the risk that a single point of failure on the host can disrupt all those processes simultaneously. Assess the risk tolerance of your enterprise.

String Encoding

To preserve interoperability throughout your enterprise, all strings must use UTF-8 encoding.

- When the TIBCO FTL Java libraries send messages, all strings are automatically UTF-8 encoded.
- C programs must treat strings in inbound messages as UTF-8 encoded strings.
- C programs must send only UTF-8 encoded strings.
- With the Golang API, strings are automatically UTF-8 encoded.

Note: Strings cannot include embedded null characters.

Running Processes as a Service

The state keeper can run as a service hosted by a TIBCO FTL server. In this model, each TIBCO FTL server runs one state keeper as a service. These TIBCO FTL servers that are running ActiveSpaces components as services are called ActiveSpaces Core Servers.

To simplify deployments, you can also replace the administration daemon by an administration service, hosted by an ActiveSpaces Core Server. Such configuration is required in deployments that use security.

Although proxies can also be run in an ActiveSpaces Core Server, it is typically better to run them as standalone processes (outside the ActiveSpaces Core Server). This is because you might need to scale the number of proxies independently, driven by the number of client processes that need to connect and the load they impose on their proxy.

To configure an ActiveSpaces Core Server, start with a TIBCO FTL server YAML file and add the definitions of the grid components that you want the server to host. When the ActiveSpaces Core Server is configured, it is responsible for starting and stopping the grid components when it starts up and shuts down. The ActiveSpaces Core Server monitors the liveness of the grid components that it hosts and attempts to restart a component if it detects that it is not running. Combining a TIBCO FTL server, a state keeper and an administration daemon into one ActiveSpaces Core Server reduces the administrative overhead of running a grid.

Add each grid component to be hosted by an ActiveSpaces Core Server to the corresponding server item in the servers section of the TIBCO FTL YAML file and the values that would have been used as command-line parameters. In addition, indicate the location of the grid component binary file by using the value of the `exepath` property.

Recommended Minimum Configuration

For a recommended minimum configuration, there are three ActiveSpaces Core Servers and each server hosts a state keeper and an administration server.

```
globals:
  core.servers:
```

```
    SRV1: localhost:8080
    SRV2: localhost:8081
    SRV3: localhost:8082

servers:
  SRV1:
  - realm:
      data: TIBCO_HOME/activespacesdata/realm_data
  - tibdgkeeper:
      exepath: /opt/tibco/as/current-version/bin/tibdgkeeper
      name: k1
      logfile: TIBCO_HOME/activespacesdata/logs/_default-k1.log
  - tibdgadminsvc:
      exepath: /opt/tibco/as/current-version/bin/tibdgadminsvc
      logfile: TIBCO_HOME/activespacesdata/logs/tibdgadminsvc1.log
  SRV2:
  - realm:
      data: TIBCO_HOME/activespacesdata/realm_data
  - tibdgkeeper:
      exepath: /opt/tibco/as/current-version/bin/tibdgkeeper
      name: k2
      logfile: TIBCO_HOME/activespacesdata/logs/_default-k2.log
  - tibdgadminsvc:
      exepath: /opt/tibco/as/current-version/bin/tibdgadminsvc
      logfile: TIBCO_HOME/activespacesdata/logs/tibdgadminsvc2.log
  SRV3:
  - realm:
      data: TIBCO_HOME/activespacesdata/realm_data
  - tibdgkeeper:
      exepath: /opt/tibco/as/current-version/bin/tibdgkeeper
      name: k3
      logfile: TIBCO_HOME/activespacesdata/logs/_default-k3.log
  - tibdgadminsvc:
      exepath: /opt/tibco/as/current-version/bin/tibdgadminsvc
      logfile: TIBCO_HOME/activespacesdata/logs/tibdgadminsvc3.log
```

Logging

For all components, the `logfile` property is optional, and if it is omitted, logs of the component are merged with the logs of ActiveSpaces Core Server. When the logs are merged with the logs of ActiveSpaces Core Server, the name of the component is added as a prefix to the log. Non-log output that a component sends to `stderr` or `stdout` always appears in the ActiveSpaces Core Server logs.

You can define the logging parameters that are common to all the grid components in the `globals` section. For example, to set a maximum log file size for all grid components hosted by the ActiveSpaces Core Server (and all other TIBCO FTL components hosted by it), specify the log level only one time.

```
globals:
  core.servers:
    SRV1: localhost:8080
    SRV2: localhost:8081
    SRV3: localhost:8082
  max.log.size: 10485760
```

If a property is defined in both the `globals` section and the section for a particular component, the definition from the component section takes precedence.

State Keeper

This topic provides an example of the configuration for a cluster containing three ActiveSpaces Core Servers that each runs a state keeper.

```
globals:
  core.servers:
    SRV1: localhost:8080
    SRV2: localhost:8081
    SRV3: localhost:8082

servers:
  SRV1:
  - realm:
      data: TIBCO_HOME/activespacesdata/realm_data
  - tibdgkeeper:
      exepath: /opt/tibco/as/current-version/bin/tibdgkeeper
      name: k1
      logfile: TIBCO_HOME/activespacesdata/logs/_default-k1.log
  SRV2:
  - realm:
      data: TIBCO_HOME/activespacesdata/realm_data
  - tibdgkeeper:
      exepath: /opt/tibco/as/current-version/bin/tibdgkeeper
      name: k2
      logfile: TIBCO_HOME/activespacesdata/logs/_default-k2.log
  SRV3:
```

```
  - realm:
      data: TIBCO_HOME/activespacesdata/realm_data
  - tibdgkeeper:
      exepath: /opt/tibco/as/current-version/bin/tibdgkeeper
      name: k3
      logfile: TIBCO_HOME/activespacesdata/logs/_default-k3.log
```

In this example, the following information is specified for each state keeper:

- The location of the state keeper binary
- The name of the state keeper
- The location where its logs are written

Live Backup and Restore

When you are performing a live backup and restore, add the `recovery.file` property to the YAML file of ActiveSpaces Core Servers and start the Core Server before starting the restore operation, that is before running the following command:

```
tibdg grid load --rollback
```

Properties

The following table lists the properties that you can configure for the state keeper when it is run in an ActiveSpaces Core Server and the equivalent command-line arguments. Other command-line arguments such as the URL for the realm and any security-related arguments are not required because the connection to the realm and authentication is handled by the hosting Core Server.

YAML Property	Command-Line Argument	Required/Optional
exepath	None	Required
name	--name or -n	Required
grid	--grid or -g	Optional

YAML Property	Command-Line Argument	Required/Optional
loglevel	--trace or -t	Optional
logfile	--logfile or -k	Optional
max.log.size	--max-log-size or -q	Optional
max.logs	--max-logs or -w	Optional
health.server	--health-server	Optional
recovery.file	--recovery-file	Optional

Administration Service

Configure the ActiveSpaces Core Servers to run administration services rather than running standalone `tibdgadmind` administration daemons. Important points to note:

- Any environment that is using secure transports, except the one using TIBCO FTL-generated certificates, must run the administration daemon as a service.
- The `exepath` property must refer to the location of the `tibdgadminsvc` binary and not the `tibdgadmind` binary.
- If one of the Core Servers is running an administration service, all the Core Servers must run the administration service. It is done to ensure that whenever the realm is functioning, an administration service is available.

Here is an example of the configuration for a cluster containing three ActiveSpaces Core Servers that each runs an administration service.

```
globals:
  core.servers:
    SRV1: localhost:8080
    SRV2: localhost:8081
    SRV3: localhost:8082
  max.log.size: 10485760

servers:
```

```
SRV1:
- realm:
    data: TIBCO_HOME/activespacesdata/realm_data
- tibdgadminsvc:
    exepath: /opt/tibco/as/current-version/bin/tibdgadminsvc
    logfile: TIBCO_HOME/activespacesdata/logs/tibdgadminsvc1.log
SRV2:
- realm:
    data: TIBCO_HOME/activespacesdata/realm_data
- tibdgadminsvc:
    exepath: /opt/tibco/as/current-version/bin/tibdgadminsvc
    logfile: TIBCO_HOME/activespacesdata/logs/tibdgadminsvc2.log
SRV3:
- realm:
    data: TIBCO_HOME/activespacesdata/realm_data
- tibdgadminsvc:
    exepath: /opt/tibco/as/current-version/bin/tibdgadminsvc
    logfile: TIBCO_HOME/activespacesdata/logs/tibdgadminsvc3.log
```

In this example, each ActiveSpaces Core Server hosts an administration service, the `max.log.size` property is configured globally but each administration service has its own `logfile`.

Properties

The following table shows the properties that can be configured for the administration service and their corresponding `tibdgadmind` command-line arguments. Other command-line arguments such as the URL for the realm and any security-related arguments are not required because the connection to the realm and authentication is handled by the hosting ActiveSpaces Core Server.

YAML Property	Command-Line Argument	Required/Optional
exepath	None	Required
loglevel	--trace	Optional
logfile	--logfile	Optional
max.log.size	--max-log-size	Optional

YAML Property	Command-Line Argument	Required/Optional
max.logs	--max-logs	Optional
http.timeout	--http-timeout	Optional

Proxy

Proxies are not typically run in ActiveSpaces Core Servers, however, you can run them in the Core Servers if need be. For development environments, it might be appropriate to configure an ActiveSpaces Core Server that hosts a state keeper, an administration service, and a proxy. In such an environment, starting the grid involves starting the ActiveSpaces Core Server and the node.

The following example provides a simple configuration suitable for a development environment. In this example, there is one ActiveSpaces Core Server that hosts a state keeper, an administration service, and a proxy. The only other component that you must start to create a running grid is a node.

```
globals:
  core.servers:
    SRV1: localhost:8080

servers:
  SRV1:
  - realm:
      data: TIBCO_HOME/activespacesdata/realm_data
  - tibdgkeeper:
      exepath: /opt/tibco/as/current-version/bin/tibdgkeeper
      name: k1
      logfile: TIBCO_HOME/activespacesdata/logs/_default-k1.log
  - tibdgadminsvc:
      exepath: /opt/tibco/as/current-version/bin/tibdgadminsvc
      logfile: TIBCO_HOME/activespacesdata/logs/tibdgadminsvc.log
  - tibdgproxy:
      exepath: /opt/tibco/as/current-version/bin/tibdgproxy
      name: p1
      logfile: TIBCO_HOME/activespacesdata/logs/_default-p1.log
```

Properties

The following table lists the properties that you can configure for the administration service and their corresponding proxy command-line arguments. Other command-line arguments, such as the URL for the realm and any security-related arguments are not required because the connection to the realm and authentication is handled by the hosting ActiveSpaces Core Server.

YAML Property	Command-Line Argument	Required/Optional
exepath	None	Required
name	--name or -n	Required
grid	--grid or -g	Optional
loglevel	--trace or -t	Optional
logfile	--logfile or -k	Optional
max.log.size	--max-log-size or -q	Optional
max.logs	--max-logs or -w	Optional
health.server	--health-server	Optional
external.hostport	--external-hostport or -e	Optional

Administration Tool

`tibdg` is an administrative command-line tool for ActiveSpaces. You can use it to define data grid components, tables, and indexes; to see the status of data grid components; and to save and restore the definitions of a data grid.

Usage Help

To see a summary of commands, run the administration tool with the help command:

```
tibdg help
```

To see information about a specific command or command area, run the administration tool with the help command and the command as an argument. For example:

```
tibdg help copyset
```

```
tibdg help table
```

```
tibdg help status
```

Realm Service Interactions

Administration tool commands interact with the realm service:

- Storing definitions in the realm service
- Retrieving definitions from the realm service
- Retrieving status information from the realm service

Every interaction command requires the location of the realm service, either as an argument or as the value of an environment variable.

Modes of Operation

You can use the administration tool in two ways:

- **Immediate command execution**: When you run `tibdg`, the tool changes the realm service workspace, and immediately deploys that change to the realm service's clients (namely, data grid component processes).

  This mode is convenient for changes to a running data grid (such as adding a table), for saving the data grid definition to a file, and for requesting status information about a running data grid.

- **Command script**: Alternatively, you can create a command script file containing several commands. Then `tibdg` runs that batch of commands, accumulating those changes in the realm service workspace. Finally, the tool deploys all the workspace changes to the realm service's clients before exiting.

  This mode is convenient for a series of related changes, such as defining a data grid or creating a table and its columns.

Consider the following two examples, which accomplish the same goal: defining a data grid. The first example runs five separate command-lines, deploying each change immediately.

```
tibdg grid create
tibdg copyset create my_copyset
tibdg node create my_node
tibdg keeper create my_keeper
tibdg proxy create my_proxy
```

In contrast, the second example consists of five commands in a script file, `my_script_file.tibdg`:

```
grid create
copyset create my_copyset
node create my_node
keeper create my_keeper
proxy create my_proxy
```

Then it runs the script with one command line, deploying all the changes at the end.

```
tibdg -s my_script_file.tibdg
```

For more information about the realm service and its *workspace*, see *TIBCO FTL® Administration*.

Administration Tool Reference

Administrators use `tibdg` to configure and monitor a data grid.

Syntax

```
tibdg [-r realm_service_URL] [-g grid_name] [-c path] [-s path] [-m
message] [Command Command_Args]
[ConfigurationOption=value]
```

Note:

- If the `-r` command-line option is not specified, the default realm service URL of http://localhost:8080 is used.
- If the `-g` command-line option is not specified, the default grid name of `_default` is used. The `-g` option is ignored for commands that take the grid name as an argument. For example, `grid rebuild <grid name>`.
- If the `-m` command-line option is specified, then the message provided is included in the TIBCO FTL deployment and is visible on the Deployments page of the TIBCO FTL UI. This helps you store the purpose of the deployment.

Command-Line Parameters

See also Environment Variables for the Administration Tool.

Parameter	Description
`-h` `--help`	Output help text about `tibdg` and its command-line parameters.

Environment Variables for the Administration Tool

The following environment variables can be used with the `tibdg` command-line administrative tool.

Values on the command line override the values of these environment variables.

Environment Variable	Description
`TIBDG_FTL`	The administration tool contacts the realm service at this URL (host and port).
`TIBDG_PARAM_FILE`	The administration tool reads parameters from this file path. Values in this file override the tool's default values. Individual values on the command-line override values in this file. If this variable is not set, the tool reads the parameters from the file it finds in either of these two default locations: • `./.tibdg` • `~/.tibdg`
`TIBDG_FTL_USER`	The username and password used to connect to the secure realm service.
`TIBDG_FTL_PASSWD`	
`TIBDG_FTL_TRUSTFILE`	The value of this property is the location of the trust file.

tibdg Status

Run the administrative tool with the `status` command to view the status of the data grid components.

The following information is displayed when you run the `tibdg status` command:

```
PROCESSES

    TYPE  NAME                 HOST   PID   REV  TXNS  REQS  COPYSET     ROLE  EST SIZE                                         DATA DIR
    node    s1  bender3.na.tibco.com  3619  1061     3     3     set1  primary   29.7MiB  /var/tmp/jgeltner/store/C_TIBDG_BANK_1M_REIDX/s1_data
    node    s2  bender3.na.tibco.com  3624  1061     1     2     set2  primary   29.8MiB  /var/tmp/jgeltner/store/C_TIBDG_BANK_1M_REIDX/s2_data
    node    s3  bender3.na.tibco.com  3629  1061     0     0     set3  primary   29.8MiB  /var/tmp/jgeltner/store/C_TIBDG_BANK_1M_REIDX/s3_data


      TYPE  NAME                 HOST   PID   REV    ROLE                                              STATE DIR
    keeper    k1  bender3.na.tibco.com  3525  1061  leader  /var/tmp/jgeltner/store/C_TIBDG_BANK_1M_REIDX/k1_data


     TYPE  NAME                 HOST   PID   REV  CLIENTS      REQ  TXN  ITER  STMT  QRY  LSNR
    proxy    p1  bender3.na.tibco.com  3615  1061        9  1600465    1     0     0    0     0


REINDEXING

    COPYSET  TABLE  STATUS  COUNT
       set1  noise  ACTIVE  68677
       set2  noise  ACTIVE  63791
       set3  noise  ACTIVE  63973
```

The PROCESSES section lists the status of the `tibdgnode`, `tibdgkeeper`, and `tibdgproxy` processes.

The value in the EST SIZE column represents how much data that node has written to the disk.

Note: EST SIZE is updated infrequently and must be interpreted as an approximate value.

The REINDEXING section displays information for any table that is being reindexed or is pending a reindex. If there is no reindexing in progress or pending, this section is not displayed.

To get a more detailed status from a specific process, include a process type and a process name when running the status command.

For example:

```
tibdg node status t1
```

The following is an example of a `tibdg node status`:

```
user@user-mbp:[~/home]:tibdg -r http://users-mbp.na.tibco.com:7715 node
status s1
Node Name: s1
Node ID: 6B97B5D0-EA30-4A63-AD64-781378D5848B
Data Dir: /Users/home/grid1/asnodedb
Copyset Name: set1
Copyset ID: 796878AB-5BE8-4905-B4B3-2FDE19A64292
Running: true (1 instances)
Instance 1:
  Host:                            users-mbp.na.tibco.com
  IP:                              10.97.128.112
```

```
  PID:                          31344
  Is Primary:                   true
  Active Transactions:          0
  Active Requests:              0
  Epoch:                        0
  Live Data Size (est):         0.0B
  Reindexing Operations:        0
  Redistribution Operations:    0
```

The following is an example of `tibdg proxy status`:

```
user@users-mbp:[~/home]:tibdg -r http://users-mbp.na.tibco.com:7715
proxy status p1
Proxy Name: p1
Proxy ID: 4E50717C-C942-4AD9-876A-F79F688236E1
Running: true (1 instances)
Instance 1:
  Host:           users-mbp.na.tibco.com
  IP:             10.97.128.112
  PID:            31335
  Clients:        0
  Client Ops:     502
  Iterators:      0
  Statements:   0
  Queries:        0
  Listeners:      0
```

The following is an example of `tibdg keeper status`:

```
user@users-mbp:[~/home]:tibdg -r http://users-mbp.na.tibco.com:7715
keeper status k1
State Keeper Name: k1
State Keeper ID: E9BFE22A-4271-4279-9007-5A7603BC449E
State Dir: /Users/home/grid1/k1_data
Running: true (1 instances)
Instance 1:
  Host:               users-mbp.na.tibco.com
  IP:                 10.97.128.112
  PID:                31331
  Is Leader:          true
  Grid Status Vote:   true
  Primaries:          1
                      Copyset ID                            Primary Node
ID
                      796878AB-5BE8-4905-B4B3-2FDE19A64292  6B97B5D0-
```

```
EA30-4A63-AD64-781378D5848B
  Copysets:          1
                     Copyset ID                              Epoch
Primary                               Bins    Nodes
                     796878AB-5BE8-4905-B4B3-2FDE19A64292  0
6B97B5D0-EA30-4A63-AD64-781378D5848B  0-4095  6B97B5D0-EA30-4A63-AD64-
781378D5848B (alive)
```

tibdg Table Stats

Run the `tibdg table stats <table-name>` command to view statistics such as row counts or overall table size for a table and all of its indexes.

The following is an example that shows statistics for a table named `t1`:

```
$ tibdg table stats t1
Table 't1' statistics:
  Rows:  10  (exact)
  Size by Index:
    primary: 103.0B (exact)
```

In this example, `t1` contains 10 rows. The `(exact)` after the row count indicates that this is an exact count. The table has one index called `primary` that takes up 103 bytes of space, which is also exact. The size reported for the primary index is the size of data in all rows in the table, which includes all the data for the primary index. The size reported for a secondary index is the additional data size of that index. The total data size for a given table is the sum of the sizes of all its indexes.

The sizes reported by this command are the sizes of the uncompressed data, so they might not reflect the disk usage for the table due to other factors such as compression of the data when it is written to the disk.

If a table or index does not have statistics enabled, the values for row count and sizes are 0, and `(off)` is displayed next to the values. See Enabling Statistics to set the row_counts attribute during table or index create to exact.

tibdg Grid Generate and tibdg Table Generate

The `tibdg grid generate` command can be used to generate a sequence of commands that can later be run to create a specific grid configuration. The `tibdg table generate` command is similar to the `tibdg grid generate` command except it only generates the commands required to create a single table.

When you run the `tibdg grid generate` command, you are asked a series of questions regarding the design of the grid. After you have provided the necessary input values, the command generates the sequence of necessary commands to create that data grid. These commands can either write output to the console or to a file. Then you can modify the commands and run them either one at a time or by using the `-s` option to run all the commands in a file.

tibdg Grid Generate Example

The following example shows a record using the `tibdg grid generate` command to generate the necessary commands to create a grid with five copysets, three nodes per copyset, three state keepers, five proxies, and two tables:

```
user:install user$ ./bin/tibdg grid generate my_grid.tibdg
Enter the number of copysets[1]: 5
Enter the number of nodes per copyset[1]: 3
Enter the number of statekeepers[1]: 3
Enter the number of proxies[1]: 5
Create a table (y|n) [y]: y
Table name: customers
Enter the name and type for column 1 in the primary index (columnName
columnType): cust_id long
Create more columns to be used in the primary index (y|n) [n]: n
Create more columns (y|n) [y]: y
Enter the name and type for the column (columnName columnType): name
string
Create more columns (y|n) [y]:
Enter the name and type for the column (columnName columnType): address
string
Create more columns (y|n) [y]:
Enter the name and type for the column (columnName columnType): phone
long
Create more columns (y|n) [y]: n
Create a secondary index (y|n) [n]: y
Enter index name: phone_index
Columns defined:
       1. cust_id
       2. name
```

```
       3. address
       4. phone
Select the ids of the columns in the order that is used in the index (#
# #...): 4
Create another secondary index (y|n) [n]:
Create another table (y|n) [n]: y
Table name: orders
Enter the name and type for column 1 in the primary index (columnName
columnType): order_id long
Create more columns to be used in the primary index (y|n) [n]: n
Create more columns (y|n) [y]: y
Enter the name and type for the column (columnName columnType): cust_id
long
Create more columns (y|n) [y]:
Enter the name and type for the column (columnName columnType): date
datetime
Create more columns (y|n) [y]:
Enter the name and type for the column (columnName columnType): value
long
Create more columns (y|n) [y]:
Enter the name and type for the column (columnName columnType):
description string
Create more columns (y|n) [y]: n
Create a secondary index (y|n) [n]: y
Enter index name: cust_index
Columns defined:
       1. order_id
       2. cust_id
       3. date
       4. value
       5. description
Select the ids of the columns in the order that is used in the index (#
# #...): 2
Create another secondary index (y|n) [n]:
Create another table (y|n) [n]:
35 commands written to my_grid.tibdg
```

This example session would write the following commands to the `my_grid.tibdg` file:

```
user:install user$ cat my_grid.tibdg
grid create copyset_size=3
copyset create cs_01
copyset create cs_02
copyset create cs_03
copyset create cs_04
copyset create cs_05
node create --copyset cs_01 --dir ./cs_01.n_1_data cs_01.n_1
```

```
node create --copyset cs_01 --dir ./cs_01.n_2_data cs_01.n_2
node create --copyset cs_01 --dir ./cs_01.n_3_data cs_01.n_3
node create --copyset cs_02 --dir ./cs_02.n_1_data cs_02.n_1
node create --copyset cs_02 --dir ./cs_02.n_2_data cs_02.n_2
node create --copyset cs_02 --dir ./cs_02.n_3_data cs_02.n_3
node create --copyset cs_03 --dir ./cs_03.n_1_data cs_03.n_1
node create --copyset cs_03 --dir ./cs_03.n_2_data cs_03.n_2
node create --copyset cs_03 --dir ./cs_03.n_3_data cs_03.n_3
node create --copyset cs_04 --dir ./cs_04.n_1_data cs_04.n_1
node create --copyset cs_04 --dir ./cs_04.n_2_data cs_04.n_2
node create --copyset cs_04 --dir ./cs_04.n_3_data cs_04.n_3
node create --copyset cs_05 --dir ./cs_05.n_1_data cs_05.n_1
node create --copyset cs_05 --dir ./cs_05.n_2_data cs_05.n_2
node create --copyset cs_05 --dir ./cs_05.n_3_data cs_05.n_3
keeper create k_0
keeper create k_1
keeper create k_2
proxy create p_00
proxy create p_01
proxy create p_02
proxy create p_03
proxy create p_04
table create customers cust_id long
column create customers name string address string phone long
index create customers phone_index phone
table create orders order_id long
column create orders cust_id long date datetime value long description
string
index create orders cust_index cust_id
```

The tibdg Commands That Support Interaction

Certain `tibdg` commands, such as `tibdg rollback`, `tibdg gridset remove` can result in the reset of a grid and an inadvertent deletion of all the data within the grid. Therefore, such commands now require interactive confirmation from the user before they are run. To run any of these commands in an unattended environment, use the `[-f|--force] flag` to run without confirmation.

If you do not specify the `[-f|--force]` flag, these commands prompt to confirm their execution. If you specify the `[-f|--force]` flag, these commands are run by force. The following commands support interaction by using the `[-f|--force]` flag:

1. `tibdg rollback create`

2. `tibdg gridset remove`
3. `tibdg gridset setPrimary`

tibdg rollback create

Creates a rollback record.

Usage:

```
tibdg rollback create [-f|--force] <checkpoint id>
```

Example of not using the -f flag:

```
tibdg --grid myGrid -r http://user-mbp:1234 rollback create A38F799B-
2FC3-A800-B31B-3EDB0247FE9C
Enter yes to confirm rollback create. yes (the "yes" must be typed in by
the user)
Rollback record 2CA87BE5-246D-48A1-9A19-9672A0BDF26A created for
checkpoint A38F799B-2FC3-A800-B31B-3EDB0247FE9C
```

Example of using the -f flag:

```
tibdg --grid grid1 -r http://user-mbp:1234 rollback create -f 145B29B0-
73D9-A900-9BDB-177A18B91594
Rollback record F740B3B0-B316-4736-9130-1C7CB049F06B created for
checkpoint 145B29B0-73D9-A900-9BDB-177A18B91594
```

tibdg gridset remove

Removes a member grid from a gridset.

Usage:

```
tibdg gridset remove [-f|--force] [-p|--makePrimary] <gridset> <grid>
```

Example of not using the -f flag:

```
tibdg --grid grid1 -r http://user-mbp:1234 gridset remove -makePrimary
gridset1 grid2
Enter yes to confirm gridset remove. yes (the "yes" must be typed in by
the user)
Grid grid2 removed from gridset gridset1
```

Example of using the -f flag:

```
tibdg --grid grid1 -r http://user-mbp:1234 gridset remove -f -
makePrimary gridset1 grid2
Grid grid2 removed from gridset gridset1
```

The `gridset remove` command has an important change in behavior. In ActiveSpaces 4.0, `gridset remove` removed a mirror grid from the gridset, without deleting any of its data. From ActiveSpaces 4.1.0, the mirror grid removed from the gridset is cleaned of all the data. Remember that a mirror grid removed from the gridset does not have any data in it after it is removed.

tibdg gridset setPrimary

Sets the primary grid in a gridset.

Usage:

```
tibdg gridset setPrimary [-f|--force] <gridset> <grid>
```

Example of not using the -f flag:

```
tibdg --grid grid1 -r http://user-mbp:1234 gridset setPrimary gridset1
grid2
Enter yes to confirm gridset setPrimary. yes (the "yes" must be typed in
by the user)
Grid grid2 is now primary for gridset gridset1
```

Example of using the -f flag:

```
tibdg --grid grid1 -r http://user-mbp:1234 gridset setPrimary -f
gridset1 grid2
Grid grid2 is now primary for gridset gridset1
```

Using tibdg grid mode to Put a Data Grid into Maintenance Mode

The `tibdg grid mode` command can be used to put a data grid into maintenance mode that prevents data from being written into your data grid.

Putting your data grid into maintenance mode can be useful when:

- Performing data grid backups.
- Transitioning primary grids to mirror grids for disaster recovery.
- Performing system software upgrades.

Syntax

```
tibdg grid mode [-r realm_service_URL] [-g grid_name] grid mode
maintenance|normal
```

The following operations are allowed in the maintenance mode:

- read operations

Warning: Write operations are not allowed and result in an exception.

In the 'normal' mode, both read and write operations are allowed.

tibdg proxy shed

The `tibdg proxy shed` command is used to unbind one or more clients from a given proxy to force them to go through the binding process again. When the clients use the balanced binding strategy, this command is used to rebalance the clients across the running proxies.

The command has two forms:

```
tibdg proxy shed <proxy_name> connection <id>
```

This command notifies a specific client connection to unbind and rebind by using the configuration that was already configured on the client. The client connection is identified by the `connection id` (a number). You can use the `tibdg proxy clients <proxy_name>` command to list the clients by their `connection id`.

The second form notifies a specific number of clients at the proxy to unbind and then rebind.

```
tibdg proxy shed <proxy_name> clients <n>
```

This command notifies the proxy to disconnect `n` clients.

For an example of using the `shed` command with the balanced binding strategy, see Using the Proxy Shed Command and the Balanced Binding Strategy.

Using the Proxy Shed Command and the Balanced Binding Strategy

The following example shows how you can use the `shed` command and the balanced binding strategy to balance a grid.

Assuming you had three proxies running and there were 20 clients connected to them, all using the balanced binding strategy. The proxies (P1, P2, P3) might have the following numbers of clients:

P1: 7 clients

P2: 7 clients

P3: 6 clients

Now you provision a new Proxy, P4, and start it. All the clients remain bound to their current proxies until either the clients or the proxies restart. Assuming all the clients impose an equal load on their proxies, the ideal distribution across the proxies would be to have five clients per proxy. This distribution can be achieved by the following commands:

```
tibdg proxy shed P1 clients 2
tibdg proxy shed P2 clients 2
tibdg proxy shed P3 clients 1
```

Note that if you notify a proxy to disconnect from all its clients, it briefly has zero clients bound to it. Therefore, when the clients attempt to rebind to the proxy, their old proxy appears to have the lowest load so they rebind back to where they came from. The way to avoid this binding is to unbind only from the number of clients that you want to move, as was done in the example above.

tibdg purge

When tables and indexes are deleted, they are not automatically removed from the grid configuration. Over time, the data stored for those tables and indexes are deleted by the nodes but their existence is still recorded in the grid configuration. In environments where tables and indexes are frequently created and destroyed, the grid configuration can grow quite large. This results in operations that change the grid configuration taking progressively longer and longer. The `tibdg purge` commands are designed to address this issue.

A table or index can only be purged once all the nodes in all the copysets have deleted the data for that node or index.

All the nodes in all the copysets must be running for these purge commands to work.

tibdg table purge

The `tibdg table purge` command removes all the deleted tables and their indexes that had their data deleted from every node in every copyset.

tibdg index purge

The `tibdg index purge` command removes all the deleted indexes that had their data deleted from every node in every copyset. This command does not purge indexes from deleted tables. To purge those indexes, use the `tibdg table purge` command.

tibdgadmind

`tibdgadmind` is an administrative daemon for ActiveSpaces. The SQL `ExecuteUpdate` command requires `tibdgadmind` running in the data grid.

Syntax

```
tibdgadmind [-r realm_service_URL] [-l listen_URL][--logfile
<file>][--max-log-size <bytes>] [--max-logs <num-files>][--trace
<level>]
```

By default, `tibdgadmind` listens on http://localhost:7171.

If more than one `tibdgadmind` needs to run on the same host or in a production environment where processes on other hosts must be able to communicate with the `tibdgadmind`, the listen URL must be specified and must be something other than the default value `localhost:7171`. The value can be changed by specifying `-l listen_URL`.

More than one realm service URL can be specified by separating the URLs with the pipe (|) character when starting the tibdgadmind process.

After connecting to the realm service, `tibdgadmind` can process requests for table configuration changes such as creating a table, dropping a table, creating an index, and deleting an index.

To make table and index configuration updates to your data grid, you must run a realm service and an active data grid, a `tibdgadmind` process, and you must use the `ExecuteUpdate` API of the `tibdgSession` object. For more information, see Defining a Table by Using SQL DDL Commands.

Use the `--logfile <file>` command-line option to specify a file name or prefix to log to.

Use the `--max-log-size <bytes>` command-line option to specify the maximum size of a log file (bytes). This option is ignored if a log file is not set. The default size is `9223372036854775807 bytes`.

Use the `--max-logs <num-files>` command-line option to specify the maximum number of log files. This option is ignored if a log file is not set. The default size is `1`.

Use the `--trace <level>` command-line option to set the log level. The valid values are `severe`, `warn`, `info`, or `debug`. The default value is `severe`.

Note: The `trace` command-line option differs slightly from the equivalent option for the other processes. Also, the other processes have logging modules that allow for finer control of logging whereas `tibdgadmind` does not provide finer control. You can only specify a log level as documented in the `--trace` level section.

To provide fault tolerance, multiple `tibdgadmind` processes can be run.

Stop the tibdg Daemon

You can stop the `tibdg` daemon by using the following command:

```
tibdg -r  <URL> -t <adminURL> admind stop
```

For example, `tibdg -r "http://localhost:8280" -t http://localhost:7171 admind stop`.

Designing a Data Grid

This task guides you through the design decisions that characterize the structure of a data grid.

Fundamental Decisions

The decisions you make in the following steps define the fundamental characteristics of the data grid. After completing this task, you cannot change these parameters *except* by deleting the data grid definition and starting over again.

As you make these design decisions, record them for later reference.

Procedure

1. Determine the number of copysets in the data grid.

   The amount of data that the grid can contain depends on the capacity of the host computers and the number of copysets.

   A single copyset can suffice for prototyping and development.

2. Determine the number of nodes per copyset.

   - For development, use one node per copyset.
   - For fault tolerance, use two nodes per copyset.
   - For stronger fault tolerance protection, use three nodes per copyset.

   Each copyset consists of the same number of nodes.

3. Determine the number of state keeper processes.

   - For development, use one state keeper process.
   - For fault tolerance, use three state keeper processes.

4. Determine the number of proxy processes.

5. Determine unique process names.

   Assign a unique name to each component process of the data grid. You can use these unique names to address the individual processes as you monitor and manage them.

a. Compose a name for each copyset.

   For example, `DG.CS-A`, `DG.CS-B`, `DG.CS-C`.

b. Compose a name for each node, incorporating the copyset name.

   For example, `DG.CS-A.N1`, `DG.CS-A.N2`.

c. Compose a name for each state keeper process.

   For example, `DG.SK-1`, `DG.SK-2`, `DG.SK-3`.

d. Compose a name for each proxy process.

   For example, `DG.PX-1`, `DG.PX-2`, `DG.PX-3`.

What to do next

Starting a Realm Service

Starting a Realm Service

Each ActiveSpaces data grid depends on a TIBCO FTL realm service to supply configuration data to its components. The realm service is a process that is run by the TIBCO FTL server. The TIBCO FTL server can also be configured to run the state keeper, administration service, and proxy. When it is configured in this way, it is called an ActiveSpaces Core Server.

Dedicate a separate realm for each data grid. If your application programs also use TIBCO FTL communications, arrange a separate realm for them. Run either one TIBCO FTL server or an ActiveSpaces Core Server, or a group of three or five or seven TIBCO FTL servers or core servers.

If you choose to run some ActiveSpaces components as services, follow the instructions in the Starting a Core Server section. Else, follow the instructions in the Starting a TIBCO FTL Server section.

Starting a Core Server

Before you begin

TIBCO FTL and ActiveSpaces software must already be installed on all computers hosting a realm service. Complete the steps mentioned in Designing a Data Grid.

Procedure

1. Create a YAML configuration file as described in the Running Processes as a Service section. Use the component names that you decided in the Designing a Data Grid section.
2. Copy the YAML configuration file to the `TIBCO_HOME/activespacesdata` directory on all the computers where the core server runs.
3. Start each core server from the `TIBCO_HOME/activespacesdata` directory by running the following command:

```
tibftlserver -c <yaml_config_file> -n <server_name>
```

where

`<server_name>` is a unique name for the core server as defined in the YAML file, for example, SRV1.

Starting a TIBCO FTL Server

Before you begin

TIBCO FTL software must already be installed on all computers hosting a realm service.

Procedure

1. Navigate to the realm configuration data directory.

   ```
   cd my_data_dir_1
   ```

   The realm service uses the current directory as the default location to store its working data files.

   - The first time you start a realm service for a data grid, navigate to an empty directory. When the realm service detects an empty working directory, it begins with a default realm definition. As you configure the realm definition, in subsequent tasks, the realm service stores that definition in its data directory.
   - If you have already begun to configure the realm definition, then navigate to the same data directory. The realm service reads the realm definition from the working directory.

2. Run the realm service executable.

   ```
   tibftlserver -n <name>@<host>:<port>
   ```

   where

   `<name>` is a unique name for the TIBCO FTL server, for example, ftl1.

   The port must not be bound by any other process.

   ActiveSpaces component processes initiate contact with the realm service at this address.

Note: Application programs must supply this realm service URL (*host*:*port*) to the data grid connect call.

Defining a Data Grid

To define and configure a data grid, complete the steps in this task.

This task implements decisions about the structure of your data grid, creating a data grid definition within a TIBCO FTL realm service. The realm service delivers the information to the component processes of the data grid and your application processes that use the grid.

The examples in these steps illustrate adding commands to a configuration script. When the script is complete, the administration tool executes the script to define the data grid.

Alternatively, you can execute each step immediately as a separate administration tool command, instead of accumulating them in a script.
You have already completed the task Designing a Data Grid. This task refers to decisions you recorded during that task.

Before you begin

A realm service must be running and reachable.

Procedure

1. In a text editor, begin editing a script file.

   Follow the convention of naming your script with the `.tibdg` file name extension.

2. Add a script command to create the data grid by using the syntax: `grid create [option=value]... [<grid_name>]`. For example:

   ```
   grid create statekeeper_count=1 copyset_size=1 mydevgrid
   ```

   Note: For more information on `grid create` option, see Grid Create Configuration Options.

   You can run the following command for a list of all the options for the `grid create` script command:

```
tibdg help grid create
```

Define the Component Processes of the Data Grid

3. For each copyset, add a script command to create that copyset. For example:

```
copyset create copyset_name
```

4. For each node, add a script command to create that node. For example:

```
node create --copyset copyset_name node_name
```

By default, the node's data directory is created at the `<grid_name>/<process_name>_data` . However, you can also specify the custom data directory path by using `-d` option. For example:

```
node create [(-d|-dir) dir_name] (-cs|-copyset) copyset_name node_name
```

5. For each state keeper, add a script command to create that state keeper. For example:

```
keeper create keeper_name
```

By default, the state keeper's data directory is created at the `<grid_name>/<process_name>_data` directory. However, you can also specify the custom data directory path by using `-d` option. For example:

```
keeper create [(-d|-dir) dir_name] keeper_name
```

6. For each proxy, add a script command to create that proxy. For example:

```
proxy create proxy_name
```

7. Optional. Run the script to create the data grid.

Alternatively, you might postpone this step until you have defined the tables of the data base (see the task Defining a Table).

```
tibdg -s script_file_path -r http://<host>:<port>
```

where <*host*> and <*port*> refer to the realm service URL.

What to do next

Starting the Data Grid Processes

Grid Create Configuration Options

The following configuration options can be used with the `tibdg grid create` command.

Warning: Properties that affect only a specific process type might only require restarting of that process type, but in general TIBCO recommends that you restart a grid whenever you update a property. For example, updating a proxy property does not require restarting a grid. In this case, it would suffice to restart only the proxy.

Option	Description	Default Value	Valid Values
`checkpoint_interval`	The interval, in seconds, between periodic checkpoints. The default value of 0.0 seconds disables periodic checkpoints. **Warning:** Checkpoints require additional space on disk, so care must be taken to avoid taking checkpoints frequently, as this can lead to a rapid increase in disk usage.	`0.0`	
`checkpoint_list_compression`	Enabling this option causes a reduction in the size of the list of checkpoint-related metadata written to disk by the state keepers and nodes. By default, this option is enabled for new grids and requires no change in checkpoint recovery	`enabled`	`enabled` `disabled`

Option	Description	Default Value	Valid Values
	procedures. For existing grids that are upgrading to this version, this feature is not enabled by default. To use this feature on existing grids, an administrator must first make sure that all the grid processes have been upgraded. After all the grid processes are upgraded, run the following command: `tibdg grid modify checkpoint_list_compression=enabled` You do not need to restart the grid processes to see this change. Only checkpoints taken after this property is modified apply compression to the metadata.		
`checkpoint_retention_limit`	The number of checkpoints (manual and periodic) to keep at a time. When the total number of checkpoints (manual and periodic) on disk exceeds the value of `checkpoint_retention_limit`, the oldest checkpoint is deleted. The default value of `0` indicates that all checkpoints must be kept. To determine the proper setting for this option, multiply the `checkpoint_retention_limit` by the `checkpoint_interval`. This value indicates the duration (in	`0`	Minimum: `0`

Option	Description	Default Value	Valid Values
	seconds) for a checkpoint is retained. This option must typically be set to a small number to avoid excessive disk usage.		
`client_req_timeout`	The time (in seconds) the client API synchronously waits for completion of a request (such as GET or PUT operation), before timing out.	`5.0`	Minimum: `0.0`
`compaction`	A value less than six indicates more emphasis on performance and less on the compaction of the disk space. Conversely, a higher value indicates more emphasis on the compaction of the disk space than performance.	`7`	Minimum: `1` Maximum: `10`
`consistent_query_limit`	The maximum number of iterators and statements (queries) that a node can handle concurrently.	`64`	Minimum: `1`
`copyset_size`	The number of nodes in a copyset.	`2`	Minimum: `1`
`encrypted_connections`	Specifies which connections in the data grid get encrypted.	`none`	`all` or `none`
`expiration_scanner_max_rows`	Determines the maximum number of rows that are expired each time a table is scanned for expired rows.	`1000000`	Minimum: `1`
`expiration_scanner_wakeup`	Determines how frequently the leader of each copyset scans a table for rows to expire. The unit of measurement is in seconds.	`5`	Minimum: `1`

<table>
<tr><th>Option</th><th>Description</th><th>Default Value</th><th>Valid Values</th></tr>
<tr><td><code>full_table_
delete</code></td><td>Defines the behavior when a SQL DELETE statement is created which does not contain a WHERE clause. Execution of a SQL DELETE statement without a WHERE clause deletes all rows of a table. This option takes one of the following values:
<code>warn</code> (default): A warning is logged by the proxy when a SQL DELETE statement without a WHERE clause is created. All rows of any user tables in the grid can be deleted.
<code>enabled</code>: A debug message is logged by the proxy when a SQL DELETE statement without a WHERE clause is created. All rows of any user tables in the grid can be deleted.
<code>disabled</code>: Prevents creation of any SQL DELETE statement that does not contain a WHERE clause.</td><td><code>warn</code></td><td><code>warn</code>
<code>enabled</code>
<code>disabled</code></td></tr>
</table>

Option	Description	Default Value	Valid Values
	Note: The `full_table_scans=disabled` setting prevents running of a SQL DELETE statement without a WHERE clause, regardless of the setting for `full_table_delete`. The setting also prevents the execution of a SQL DELETE statement with a WHERE clause when an index cannot be found for the columns in the WHERE clause.		

`full_table_scans`	Defines the behavior when processing a query that requires a full table scan. This option takes one of the following values:
	• `warn` (default): A warning is logged when a query performs a full table scan.
	• `enabled`: Logs a debug message when a query performs a full table scan. The behavior of this option is similar to that in the previous versions of ActiveSpaces.
	• `disabled`: Prevents a query from running a full table scan. An exception is generated if queries try to perform a full table scan.
		`warn`	`warn`
			`enabled`
			`disabled`

Option	Description	Default Value	Valid Values
	Note: The `full_table_scans=disabled` setting prevents the execution of SQL DELETE and UPDATE statements without a WHERE clause. The setting also prevents the execution of SQL DELETE and UPDATE statements with a WHERE clause when an index cannot be found for the columns in the WHERE clause.		
`grid_internal_subnet_mask`	See Configure Internal Subnet Masks.	none	See Configure Internal Subnet Masks.
`iter_inactivity_timeout`	The time, in seconds, taken by the proxy to wait for the next client request on a table iterator or statement query before automatically closing the table iterator or statement query.	`600.0`	Minimum: `0.0`
`minimum_replication_factor`	The minimum number of nodes (including the primary and any secondary nodes) in a copyset that must be in the `Alive` state before `WRITE` operations are allowed.	`1`	Minimum: `1`
`mirroring_max_batch_size_rows`	The maximum rows in a batch that is mirrored collectively to the mirror grid. This size must be an integer >=1. It can typically be left at the default value unless transport loss is seen during mirroring operations. If transport loss is experienced during mirroring, this value must be reduced.	`256`	

Option	Description	Default Value	Valid Values
mirroring_ interval	The default mirroring interval (in seconds). This option determines how frequently a mirror grid checks for new checkpoints to be mirrored. Setting this option to 0 disables mirroring.	30.0	
node_detailed_ stats_ collection	Retrieves detailed statistics of a node configuration. The option can take one of the following values: enabled or disabled. When this option is enabled, the nodes enable extra statistics collection around all disk operations. The statistics can be retrieved by using the tibdg node status command. Additionally, these statistics are logged to the node log at the status:verbose level once every 60 seconds. Enabling node_detailed_stats_ collection results in a 5-10% performance penalty. node_ detailed_stats_collection is disabled by default.	disabled	enabled disabled
node_read_ cache_size	Every node stores a read cache that holds uncompressed blocks of data in memory. The size of a read cache is specified in bytes. There are some memory usage considerations to be made when using this option. For details, see Memory Usage Considerations with the node_read_cache_size Option.	1073741824 (1 gigabyte)	Minimum: 0 Maximum: 9223372036854775806 (LLONG_MAX - 1)

Option	Description	Default Value	Valid Values
node_storage_timeout	The time (in seconds) a node waits for a successful response from a READ or WRITE operation to complete before timing it out.	60	Minimum: 0
permissions	You can enable or disable permissions on tables in a data grid to control who has access to the data in the tables. For details, see Grid and Table Permissions.	disabled	enabled disabled
proxy_checkpoint_cache_size	See Caching Proxy Rows by Using Checkpoints.	0	Minimum: 0
proxy_client_listen_external_host	The host name or the IP address that external clients connect to when attempting to reach a proxy. See Configuration Options when the Proxy and Client are on Different Subnets.	none	
proxy_client_listen_external_port	This is the default port that external clients connect to when attempting to reach a proxy. See Configuration Options when the Proxy and Client are on Different Subnets.	none	
proxy_client_listen_subnet_mask	See Configure Network Interfaces	none	See Configure Network Interfaces.
proxy_client_listen_port	See Configure Ports.	none	See Configure Ports.
statekeeper_count	The number of state keeper processes that are expected to be	3	Minimum: 1

Option	Description	Default Value	Valid Values
	run. Due to the requirement that state keepers must be run in a quorum, the supported values are 1, 3, 5, 7, and 9.		

Memory Usage Considerations with the node_read_cache_size Option

The `tibdg grid create` command comes with the `node_read_cache_size` option. This property governs the size of the read cache allotted to a node.

Here are some memory usage considerations when you are assigning a size to the read cache:

- If the RAM on a node's host exceeds the amount of data persisted by the node, increasing the `node_read_cache_size` value to match the size of the uncompressed data set yields an improved random read performance.
- If the data set is larger than the host's RAM, increasing the `node_read_cache_size` value can negatively affect the random read performance. In such instances, the read cache of the application comes at the expense of the page cache of the operating system.
- If the value of `node_read_cache_size` is a significant percentage of the host RAM, the operating system must be configured to prevent swapping of the node application for optimal performance. Example:
  - On Linux, you might have to reduce the value of `/proc/sys/vm/swappiness` or disable `swap`.
  - In Docker, you can modify the `--memory-swappiness run` parameter to prevent swapping.

Configuration Options to Use Specific Ports and Network Interfaces

ActiveSpaces uses TIBCO FTL internally to aid in connecting the different ActiveSpaces processes so that they can communicate over the network. In certain cases, you can specify additional configuration information when initially setting up the data grid processes to help influence what ports and network interfaces must be chosen when the processes are started.

For example, an ActiveSpaces client application might connect to an ActiveSpaces `tibdgproxy` that is running on a server, which has a firewall. In that case, a network administrator might open a specific port or ports in the firewall so that the TCP connections can be made to the `tibdgproxy` running behind the firewall. After the `tibdgproxy` is started, it reads from its data grid configuration what port it must bind to when listening for incoming TCP connections to ensure that traffic can pass through the firewall.

Configure Ports

Regarding ports, the configuration option `proxy_client_listen_port` can be specified when creating a grid or proxy. This is the port that a proxy binds to when listening for incoming TCP connections. When specified at the grid level, all proxies inherit this value and listen for clients on this port.

For example:

```
grid create copyset_size=1 proxy_client_listen_port=8890
proxy create p1
```

This works as long as there is only one proxy running on each computer. If two proxies are started on the same computer and both try to bind to port 8890, the second proxy fails to start due to an error. To avoid this, override the configuration option at the proxy level when creating a specific proxy to override the listen port specified at the grid level.

For example, in the following grid configuration, p1 inherits and listens on port 8890, p2 overrides its listen port with the specified port 8891, and both proxies can run on the same computer:

```
proxy create proxy_client_listen_port=8891 p2
```

Configuration Options when the Proxy and Client are on Different Subnets

When the realm service and proxy are running on a subnet that is different from the one on which the client application is running, the proxy might need additional configuration. This is a requirement when you are dealing with a Network Address Translation (NAT) setup as is common with cloud environments or Docker.

For example, if the realm service and proxy are on the subnet `10.0.75.0/24` and the ActiveSpaces client application is on the subnet `192.168.1.0/24`, the client application often cannot route to the `10.0.75.0/24` subnet (the proxy's subnet). In such a situation, in addition to `proxy_client_listen_port`, configure the `proxy_client_listen_external_host` option.

If the port that the proxy is listening on is mapped to a different external port, use `proxy_client_listen_external_port` to specify the correct external port.

Note: If you are in a dynamic environment, use the `-e <ip:port>` command-line parameter when you run the `tibdgproxy` process. For more information, see Starting a Proxy with an External Host and Port.

Configuring a Proxy That Can Be Accessed from Different Subnets

The Network Address Translation (NAT) is possible with the use of the options, `proxy_client_listen_external_host` and `proxy_client_listen_external_port`.
The steps to configure the external host and port differ based on your environment. Treat this procedure as a general guideline to help client applications connect to a proxy.

Procedure

1. Set `proxy_client_listen_port` to the internal port for the proxy.
2. Determine the external host IP address that a client can use to connect to the proxy.

a. Set `proxy_client_listen_external_host` to this external host IP address.

3. Determine if the internal port to which the proxy is listening is the same as the one that is exposed externally.

> **Note:** If a different port is exposed externally, you must set up port forwarding outside of ActiveSpaces to map the external port to the internal port.

a. If the external port is different from the internal port, set `proxy_client_listen_external_port` to the external value.

> **Note:** If you are in a dynamic environment, use the command-line parameter `-e <ip:port>` when you run the `tibdgproxy` process. For more information, see Starting a Proxy with an External Host and Port.

An Example of Creating a Proxy That Can Be Accessed from a Different Subnet

```
tibdg -r http://localhost:8080 proxy create proxy_client_listen_
port=8999
proxy_client_listen_external_host=192.168.1.136 p2
```

An Example of Creating a Proxy with an External Port

```
tibdg -r http://localhost:8080 proxy create proxy_client_listen_
port=8999
proxy_client_listen_external_host=192.168.1.136 proxy_client_listen_
external_port=7999 p2
```

In this example, when the client application attempts to connect, it first connects to the realm service (which needs ports 8080 and 8083 opened or forwarded). The realm service notifies the client about the proxies that are running and includes the external host and port (if configured) so that the client can connect to the proxy that is on a different subnet. For the connection to succeed, the administrator must set up port forwarding correctly. In Docker, this might be with the `-p 8999:8999` syntax or `-p 7999:8999` to forward a port on the host to a port in a specific container.

Configure Network Interfaces

You can use the configuration option `proxy_client_listen_subnet_mask` to configure network interfaces. This can be specified at both the grid and proxy level to control which network interface the proxy binds to when listening for connections from clients.

If a computer has multiple network interfaces, a specific subnet mask can be provided in a standard CIDR notation to control which interface must be selected. When specified at the grid level, all proxies inherit this value and attempt to use the specified subnet mask.

For example, in the following grid configuration, all proxies attempt to use the subnet mask 10.0.1.0/24:

```
grid create copyset_size=1 proxy_client_listen_subnet_mask=10.0.1.0/24
proxy create p3
```

To override this value for a specific proxy, it can also be specified at the proxy level. The options for proxy ports and proxy listen subnet masks can be combined at both the grid and proxy level.

For example:

```
proxy create proxy_client_listen_subnet_mask=10.0.2.0/24 p4
proxy create proxy_client_listen_subnet_mask=10.0.2.0/24 proxy_client_
listen_port=8892 p5
```

Configure Internal Subnet Masks

For internal communication between the ActiveSpaces server processes (`tibdgproxy`, `tibdgnode`, and `tibdgkeeper`), you can specify the subnet mask to be used for this internal communication between server processes by using the configuration option `grid_internal_subnet_mask`. This option is different from the other configuration options described because it does not influence the client-to-proxy communication. It can also only be specified at the grid level.

For example:

```
grid create copyset_size=1 grid_internal_subnet_mask=10.0.10.0/24
```

Setting this configuration option causes all communication between proxy, node, and state keeper processes to occur on the specified interface. This option can also be combined with the two proxy configuration options `proxy_client_listen_subnet_mask` and `proxy_client_listen_port`.

The following is an example of specifying all three options at the grid level:

```
grid create copyset_size=1 proxy_client_listen_subnet_mask=10.0.1.0/24
proxy_client_listen_port=8890 grid_internal_subnet_mask=10.0.10.0/24
```

Starting the Data Grid Processes

To start the data grid, start its component processes in this order.

For details, see the "Sample Scripts" section in Development Environment.

Before you begin

- The realm service must be running and reachable.
- The data grid components must be defined.
- It is not required to define your tables before starting the data grid processes. However, your tables must be defined before they can be used by a client application.

Procedure

1. Start the state keeper processes.

   See Starting a State Keeper.

2. Start the node processes.

   See Starting a Node.

3. Start the proxy processes.

   See Starting a Proxy.

Result

The data grid is ready to use. You can start application processes.

Component Command-Line Parameters

All three executable components -- state keeper, node, and proxy -- accept the same set of command-line parameters, as documented here.

Parameter	Description
-n *name* --name *name*	Required. Process name. Supply one of the names you assigned in Defining a Data Grid.
-r *realm_service_URL* -rs *realm_service_URL* --realmserver *realm_service_URL*	Required. Realm service location. Supply the realm service URL in the form `http://`*`host`*`:`*`port`*. Use the values of *host* and *port* that you supplied as the `-n` arguments in Starting a Realm Service. If running a secure realm service, use `https` instead of `http`. When you are running a cluster of FTL servers, the *realm_service_URL* can be a list of URLs separated by a | (pipe) character. For example: `-r realm_service1_URL [ | realm_service2_URL [ | realm_service3_URL] ]`
-g *name* --grid *name*	Optional. Required when a data grid has been configured with a name. The name of the data grid as specified in Defining a Data Grid.
-k *file_name* --logfile *filename*	Optional. Setting this parameter enables rotating log files that start with the specified filename as the prefix.
-q *integer* --max-log-size *integer*	Optional. Specify the maximum size of a single log file before rotating.
-w *integer* --max-logs *integer*	Optional. Specify the maximum number of log files to keep. The default is `1`.

Controlling Data Grid Access

Access to tables in the data grid can be controlled by setting permissions on tables.

For more information about starting data grid processes when table permissions are used, see Grid and Table Permissions.

Starting a State Keeper

Start state keeper processes first, because all other ActiveSpaces component processes depend on them. If you have configured ActiveSpaces Core Servers to host the state keepers, you can skip this section because the Core Server manages the lifecycle of the state keeper it is hosting.

Before you begin

- The realm service must be running and reachable.
- The data grid definition in the realm service must be complete and valid.
- When permissions are enabled on the data grid, a user with the `tibdg-internal` role must be used to run the state keeper. For more information about table permissions, see Grid and Table Permissions. For more information about roles, see ActiveSpaces Custom Roles.

Procedure

1. Start the state keeper process.

   ```
   tibdgkeeper -n name -r realm_service_URL [-g grid_name]
   ```

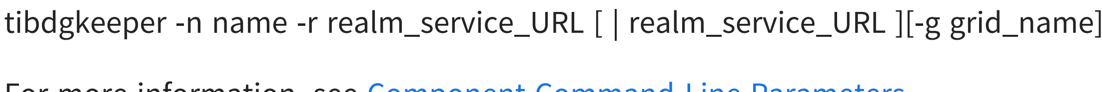
2. Repeat the previous step for all the state keeper names assigned in the data grid definition.
3. Verify that the state keeper processes are ready.

   Check the status by using the administration tool.

Keeper Reference

Administrators use `tibdgkeeper` to start a keeper process.

Syntax



For more information, see Component Command-Line Parameters.

Starting a Node

Start the node processes that implement the copysets.

Before you begin

- The realm service must be running and reachable.
- The data grid definition in the realm service must be complete and valid.
- The state keeper must be running and reachable.
- When permissions are enabled on the data grid, a user with the `tibdg-internal` role must be used to run the node. For more information about table permissions, see Grid and Table Permissions. For more information about roles, see ActiveSpaces Custom Roles.

Procedure

1. Start the node process.

```
tibdgnode -n name -r realm-service_URL [-g grid_name]
```

2. Verify that the node process is synchronized and ready.

   If the number of nodes per copyset is greater than one, and the node you have started is not the primary node, then wait for the node to synchronize with the primary.

   To verify synchronization, check the status by using the administration tool.

   For the fastest and most efficient start sequence, it is important to start only one node process at a time, and wait for it to synchronize before starting the next node process.

3. Repeat the previous steps for all the node names assigned in the data grid definition.
4. Verify communication.

   If the number of nodes per copyset is greater than one, verify that the node processes within each copyset can communicate with one another.

   One of the nodes reports in its console output that it is *active*.

Node Reference

Administrators use `tibdgnode` to start a node.

Syntax

tibdgnode -n name -r realm_service_URL [| realm_service_URL][-g grid_name]

For more information, see Component Command-Line Parameters.

Starting Multiple Nodes

There are several use cases for using multiple nodes.

- Additional nodes are created with the goal of scaling horizontally.

  For example, if you have three copysets, start the components in the following sequence:

  ```
  tibrealmserver
  tibdg -r http://localhost:8080 -s /<path>/three_copysets.tibdg
  tibdgkeeper -r http://localhost:8080 -n k1
  tibdgproxy -r http://localhost:8080 -n p1
  tibdgnode -r http://localhost:8080 -n s1
  tibdgnode -r http://localhost:8080 -n s2
  tibdgnode -r http://localhost:8080 -n s3
  ```

  Here *<path>* refers to the location where `three_copysets.tibdg` is stored. For a sample script, see three_copysets.tibdg.

- Nodes are created as replicas of the copysets.

  For example, if you have one copyset and two replicas, start the components in the following sequence:

  ```
  tibrealmserver
  tibdg -r http://localhost:8080 -s /<path>/one_copyset_two_
  replicas.tibdg
  ```

```
tibdgkeeper -r http://localhost:8080 -n k1
tibdgproxy -r http://localhost:8080 -n p1
tibdgnode -r http://localhost:8080 -n s1
tibdgnode -r http://localhost:8080 -n s2
```

Here <*path*> refers to the location where `one_copyset_two_replicas.tibdg` is stored. For a sample script, see one_copyset_two_replicas.tibdg.

Note: The sample scripts three_copysets.tibdg and one_copyset_two_replicas.tibdg create nodes and state keepers by using the `--dir` option, for example:

```
node create --copyset set1 --dir ./s1_data s1
statekeeper create --dir ./k1_data k1
```

Note: The nodes store the data that applications put in the data grid in the `s1_data` folder and the state keeper stores state information about the primary and secondary nodes in the `k1_data` folder.

three_copysets.tibdg

```
# Data grid where each copyset has 1 replica
grid create copyset_size=1

table create t1 key long
column create t1 col2 string col3 opaque

copyset create set1
copyset create set2
copyset create set3
node create --copyset set1 --dir ./s1_data s1
node create --copyset set2 --dir ./s2_data s2
node create --copyset set3 --dir ./s3_data s3

statekeeper create --dir ./k1_data k1
```

```
proxy create p1

# Show results
status
table list
copyset list
```

one_copyset_two_replicas.tibdg

```
# Data grid where each copyset has 2 replicas
grid create copyset_size=2

table create t1 key long
column create t1 col2 string col3 opaque

copyset create set1
node create --copyset set1 --dir ./s1_data s1
node create --copyset set1 --dir ./s2_data s2
statekeeper create --dir ./k1_data k1
proxy create p1

# Show results
status
table list
copyset list
```

Starting a Proxy

Start the proxy processes that mediate between application processes and the data grid. If you have configured ActiveSpaces Core Servers to host the proxies, you can skip this section because the Core Server manages the lifecyle of the proxy is it hosting.

Before you begin

- The realm service must be running and reachable.
- The data grid definition in the realm service must be complete and valid.
- The state keeper must be running and reachable.
- At least one node of each copyset must be running and reachable.
- When permissions are enabled on the data grid, a user with the `tibdg-internal` role must be used to run the proxy. For more information about table permissions, see Grid and Table Permissions. For more information about roles, see ActiveSpaces Custom Roles.

Procedure

1. Start the proxy process.

   ```
   tibdgproxy -n name -r realm_service_URL [-g grid_name]
   ```

   The proxy process name is required. Supply one of the proxy names you assigned in Defining a Data Grid.
2. Verify whether the proxy process is ready by checking the status by using the administration tool.
3. Repeat the previous steps for all the proxy names assigned in the data grid definition.

What to do next

The data grid is ready to support data operations. You can start application program processes.

Proxy Reference

Administrators use `tibdgproxy` to start a proxy.

Syntax

tibdgproxy -n name -r realm_service_URL [| realm_service_URL][-g grid_name]

For more information, see Component Command-Line Parameters.

Starting a Proxy with an External Host and Port

When the realm service and proxy are running on a subnet that is different from the one on which the client application is running, you must configure the external host and port settings in the configuration file.

You can use a command-line parameter with the `tibdgproxy` process to specify the external host and port. The `-e` option followed by the IP address and the port number helps the client reach the proxy on a different subnet. If you specify the command-line option, it overrides the values set for `proxy_client_listen_external_host` and `proxy_client_listen_external_port` in the configuration file. For more information about setting `proxy_client_listen_external_host` and `proxy_client_listen_external_port`, see Configuration Options when the Proxy and Client are on Different Subnets.

Before you begin

- The following components must be running and reachable:
  - The realm service
  - The state keeper
  - At least one node of each copyset
- The data grid definition in the realm service must be complete and valid.

Procedure

1. Start the proxy process.

```
tibdgproxy -n name -r realm_service_URL  -e External IP Address:Port
```

The name of the proxy process is required. Supply one of the proxy names you assigned in Defining a Data Grid. The URL of the realm service is required. If the client is trying to reach a proxy on a different subnet, provide the `-e` option followed by the IP address and the port number. Here is an example to start the proxy and set the external host and port:

```
tibdgproxy -r http://localhost:8080 -e "192.168.1.136:7999" -n p3
```

Warning: The command overrides any configuration value that is already defined for this proxy.

If the proxy is on one of several known external IP addresses, use a semicolon-separated list of IP addresses in the command-line parameter. Also, see the `docker-compose` file in the `TIBCO_HOME/as/<version>/samples/docker` folder for an example on using different proxy configuration options that allow client applications on the host to communicate with a proxy running in a docker container.

2. Verify whether the proxy process is ready by checking the status by using the administration tool.

3. Repeat the previous steps for all the proxy names assigned in the data grid definition.

What to do next

The data grid is ready to support data operations. You can start application program processes.

Methods of Selecting a Proxy for a Client

ActiveSpaces client communication with the data grid always goes through an ActiveSpaces proxy (`tibdgproxy`). When multiple proxies exist in a data grid, you might want to control the proxy to which your clients connect. The following methods are available to select a proxy for a client:

- Random binding strategy
- Named binding strategy
- Balanced binding strategy
- Random pattern binding strategy
- Balanced pattern binding strategy

By default, the random binding strategy is used. If you want more control over proxy selection, use the named binding strategy. To balance the available client connections across the number of proxies, use the balanced binding strategy. Use the pattern binding strategies to filter proxies by using a regular expression before binding.

Random Binding Strategy

By default, the random binding strategy is used to select a proxy for a client connection. Here, a proxy is chosen at random from the proxies that respond when the client first connects to the data grid. By default, the client connection waits for the `CONNECT_WAIT_TIME` to expire before selecting a proxy. Use the `early cutoff` property to notify the client connection to wait only until a certain number of proxy responses are received. Using this property reduces the amount of time taken for the client to establish a connection with the data grid.

In this method, specify the following properties:

```
TIBDG_CONNECTION_PROPERTY_LONG_BINDSTRATEGY = TIBDG_CONNECTION_
BINDSTRATEGY_RANDOM
TIBDG_CONNECTION_PROPERTY_LONG_CONNECT_NUMRESPONSES = n
```

Here, `n` is the number of proxy responses.

If you are using the C API, pass the `Properties` object as an argument to `tibdgGrid_Connect()`. If you are using the Java API, pass the `Properties` object as an argument to `DataGrid.Connect()`.

After receiving the specified number of proxy responses, the client stops waiting for more proxy responses and chooses randomly between the responses already received.

If the `TIBDG_CONNECTION_PROPERTY_LONG_CONNECT_NUMRESPONSES` value is set higher than the number of active proxies, it has no noticeable effect. The value must be set high enough to get a good distribution of clients among proxies. A general guideline would be to set the value to be approximately 50%-80% of the number of proxies in the system. Without using the `early cutoff` property, the proxy response time does not directly affect the binding process, unless the `CONNECT_WAIT_TIME` property is configured so low that no responses reach the client in time. If you need a response from all proxies, you must wait for the slowest proxy to respond before a proxy is selected for the client connection.

The following are the possibilities of an early cutoff considering `CONNECT_NUMRESPONSES=8`:

- By default, the `CONNECT_WAIT_TIME` property value is 100 ms and a client waits for that interval before collecting responses from proxies that are running. Then, the client randomly chooses one of those proxies to establish a connection.
- If you have eight proxies and you set `CONNECT_NUMRESPONSES=8`, there is a probability of having an early cutoff. If all eight proxies respond within 5 ms, the connection can be established within 5 ms. If one proxy is down or is busy and does not respond until 200 ms, the client waits for 100ms, and then selects a proxy for the connection. This is why specifying `CONNECT_NUMRESPONSES=8` is a good practice, especially if you know the total number of proxies that are configured.
- You can specify the `CONNECT_NUMRESPONSES=8` to some percentage of the total proxies. As per the example, if you consider 50% of the total proxies, there would be four proxies. The client stops waiting after four proxy responses have been received. Maybe the first four responses come back in 3 3 ms and then the other ones range from 5 ms to 200 ms. The client stops waiting at 3 ms after the first four responses, which is better than waiting for 100 ms.

Named Binding Strategy

In the named binding strategy, the client can choose from a predefined list of proxies.

Consider a scenario where you have two applications - one that runs GET, PUT, and DELETE operations in response to business logic and the other that performs administrative operations such as creating or deleting tables. You want them to be bound to separate proxies so the important business logic is not delayed by the less predictable,

more expensive administrative operations. In such a scenario, use the named binding strategy to bind the business application on proxy1 or proxy2 and the administrative application on proxy3 or proxy4. In most cases, a primary and fail-over proxy must be enough.
To enable the named binding strategy, set the following properties:

```
TIBDG_CONNECTION_PROPERTY_LONG_BINDSTRATEGY = TIBDG_CONNECTION_
BINDSTRATEGY_NAMED
TIBDG_CONNECTION_PROPERTY_STRING_CONNECT_PROXYNAMES =
"proxy1|proxy2|proxy3"
```

Here, `proxy1|proxy2|proxy3` is a list of proxies. The delimiter used to separate the list of proxies is a | (pipe) symbol. The highest priority proxy is specified first in this list followed by the others. The last proxy mentioned has the lowest priority.

If you are using the C API, pass the `Properties` object as an argument to `tibdgGrid_Connect()`. If you are using the Java API, pass the `Properties` object as an argument to `DataGrid.Connect()`.

When the named binding strategy is configured, the client binds to the proxy with the highest priority. Usually, the highest priority proxy responds within the duration of the waiting time specified for a connection. If the highest priority proxy responds, the client stops waiting so the operations can begin.

If the proxy does not respond in time, the client waits out the `CONNECT_WAIT_TIME` value and binds with the next highest priority proxy. The named binding strategy can be used to ensure that loads between proxies and clients can be balanced more carefully if the default random matching is not preferred.

Balanced Binding Strategy

Based on the number of current connections, you can use the balanced binding strategy to balance the client connections across the available proxies. When all proxies have the same number of connections, the balanced binding strategy works like the random binding strategy. For example, when the grid starts, all proxies have zero client connections. As a result, the balanced binding strategy randomly assigns clients to proxies. By default, the client connection waits until the wait time for a connection expires and then selects a proxy. Optionally, you can use the `early cutoff` property to notify the client connection to wait only until a certain number of proxy responses are received. Using this property reduces the amount of time taken for the client to establish a connection with the data grid.

To enable the balanced binding strategy, set the BINDSTRATEGY property and optionally set the NUMRESPONSES property.

```
TIBDG_CONNECTION_PROPERTY_LONG_BINDSTRATEGY = TIBDG_CONNECTION_
BINDSTRATEGY_BALANCED

TIBDG_CONNECTION_PROPERTY_LONG_CONNECT_NUMRESPONSES = n
```

Where n is the number of proxy responses that you want the client to wait for before selecting a proxy. The property is set on the Connection object's properties.

The default CONNECT_WAIT_TIME is 100 ms, which means that the client waits for this time before selecting a proxy. If you want the client to wait longer, then set the following property:

```
TIBDG_CONNECTION_PROPERTY_DOUBLE_CONNECT_WAIT_TIME = m
```

Where m is the number of seconds that you want the client to wait.

When the balanced binding strategy is enabled, the client performs the following tasks:

- Collects the responses from the proxies that responded in time
- Chooses to bind to the proxy with the fewest clients
- Notifies the remaining proxies that they have been rejected.

Note: If several clients connect to the grid at the same time, they might all choose the same proxy leading to an unbalanced distribution across the proxies. For more information about how to rebalance the clients, see the tibdg proxy shed command.

Pattern Binding Strategy

There are two pattern binding strategies - random and balanced that are used to bind clients to proxies by using PCRE2 regular expression to match the proxy names. You can use these strategies to bind clients to proxies based on a pattern. For example, you can bind clients to proxies based on the location of the client or the type of operation that the client performs. Alternatively, you can use a pattern to specify a list of named proxies as a regular expression just like the named binding strategy. However, unlike the named

binding strategy, each matching proxy is given equal weight instead of a priority-ordered list.

The difference between the pattern-based and non-pattern based binding strategies is how a proxy is chosen from the list of proxies that match the pattern. The pattern random binding strategy selects a proxy at random and selects the proxy with the fewest clients just like the balanced binding strategy. The pattern binding strategies also honor the `CONNECT_WAIT_TIME` and `CONNECT_NUMRESPONSES` properties. For example, consider a grid where proxies are spread among different availability zones, and the zone name is part of the proxy name. For example, zone1-p1, zone1-p2, zone2-p1, and zone2-p2. To enable a client in zone1 to bind to a random proxy in zone1 only, use the pattern random binding strategy. Set the `BINDSTRATEGY` property to `TIBDG_CONNECTION_BINDSTRATEGY_PATTERN_RANDOM` and the `PROXYPATTERN` property to `'^zone1-.*'`. The pattern `'^zone1-.*'` matches all the proxies in zone1 and the client binds to a random proxy in zone1.

To enable a pattern binding strategy, set the following properties:

```
TIBDG_CONNECTION_PROPERTY_LONG_BINDSTRATEGY = TIBDG_CONNECTION_
BINDSTRATEGY_PATTERN_RANDOM # or TIBDG_CONNECTION_BINDSTRATEGY_PATTERN_
BALANCED
TIBDG_CONNECTION_PROPERTY_STRING_CONNECT_PROXYPATTERN = "pattern" #
mandatory if the pattern random or pattern balanced binding strategy is
used
TIBDG_CONNECTION_PROPERTY_LONG_CONNECT_NUMRESPONSES = n # optional
TIBDG_CONNECTION_PROPERTY_DOUBLE_CONNECT_WAIT_TIME = m # optional
```

Adding Copysets

To scale an existing data grid horizontally, you can create additional copysets and create nodes to assign to these copysets.

Important: Data grids created by using earlier versions of ActiveSpaces must be upgraded by following the procedures detailed in "Upgrading from an Earlier Version" in *TIBCO ActiveSpaces® - Enterprise Edition Installation*.

Use the `copyset create` command in the `tibdg` tool to create the copysets and nodes. Once created, the nodes can be started but they do not yet receive any data. Use the `grid redistribute` command in the `tibdg` tool to start the data redistribution process.

The `tibdg grid redistribute` command can be invoked at any time to assign data to newly added copysets. If the nodes involved in the redistribution are not running, they begin redistributing data once they are started.

For example, to grow a data grid from three copysets to four copysets, the following commands must be used:

```
tibdg -r http://host:port copyset create cset4
tibdg -r http://host:port node create --copyset cset4 --dir ./s4_data s4

tibdgnode -r http://host:port -n s4

tibdg -r http://host:port grid redistribute
```

Note: Before starting any nodes in the new copyset, all members of the copyset must be added by using the `tibdg node create` command. For example, if each copyset has two nodes (as defined by the `copyset_size` parameter supplied to the `tibdg grid create` command), you must create two nodes in the new copyset before starting the `tibdgnode` processes.

View the status of the redistribution by running the `status` command in the `tibdg` tool.

```
tibdg -r http://host:port status
```

Caution: If you are using a secure realm server, use `https://host:port` not `http://host:port`.

Data Redistribution

Data redistribution is done in the background and does not block ongoing operations when data is being transferred.

When the sending copyset completes its migration of data, it briefly delays live operations when assigning ownership to the new copyset. During this interval, transactions that were started during the migration process might fail, and iterator creation and query execution might fail. If a row moves during data redistribution, transacted reads might become invalid during a transaction, meaning that the transaction might fail to commit. Given that possibility, an application must avoid taking action on a transacted read until it learns that the transaction commit has returned successfully. After this, there is a period of time where other processes (nodes and proxies) in the data grid begin to learn about the change in ownership of the data now at the new copyset. So, it is expected that operations occurring during that window can experience a timeout error at the client while the processes learn about the new configuration.

Statements and table listeners created before the data redistribution are out of date once the data redistribution is completed and receives an `invalid resource` error at the client. The object must be destroyed in the client application and re-created.

An existing copyset that sends data to a new copyset retains its data on disk until the data redistribution process is complete. The rows previously owned by the copyset are deleted as a background operation.

Note: For capacity planning purposes, it is possible that the portion of data being contributed by a copyset at the moment the redistribution is completed exists at both the old copyset and new copyset. In other words, in a one to two copyset redistribution scenario where the one existing copyset contains 100GB of data and is contributing 50GB to a new copyset, there would be a time where the total aggregate disk usage would be 150GB (this does not account for any additional disk usage by background activities like compaction).

Caution:
The following are important considerations to manage copysets and optimize your ActiveSpaces deployments:

- The first copyset defined in the grid is responsible for global transaction coordination, and cannot be removed.
- Carefully consider how the load on the `tibdgnode` processes changes when data is redistributed. When moving from 10 copysets to five, every `tibdgnode` process has approximately twice the load as before the redistribution.
- Checkpoint data is not redistributed. If checkpoints are in use, a copyset might service checkpoint requests even if all other data has been redistributed. Removing a copyset might impact checkpoint availability. (Use the `tibdg checkpoint list` to know the checkpoints that are no longer available due to copyset removal.)
- Ensure that data redistribution is complete before running the `tibdg copyset remove` command. If ActiveSpaces is in the process of redistributing data, it does not allow you to remove a copyset.
- Monitor the redistribution process using the `tibdg status` command.
- Exercise caution when removing copysets, as this action permanently alters the grid configuration.

Removing Copysets

The Remove Copyset feature enhances the ActiveSpaces data grid scalability by enabling scaling down of resources.

Procedure

1. Initiate deactivation of a copyset by running the following command:

   ```
   tibdg copyset deactivate <copyset_name>
   ```

   This command marks the copyset as inactive, but does not begin redistribution of data to other copysets. It enables more efficient data redistribution when you are removing multiple copysets.

2. Redistribute data by running the following command:

   ```
   tibdg grid redistribute
   ```

   When redistribution starts, any copysets marked as deactivated transfer data to the remaining active copysets. To determine when redistribution is complete, check the `tibdg` status.

3. When the status indicates there are no redistributions in progress, remove the copyset from the grid configuration by running the following command:

   ```
   tibdg copyset remove <copyset_name>
   ```

   When a copyset is removed, all `tibdgnode` processes associated with that copyset exit. When the processes have stopped, their data directories can be removed. If the data redistribution fails, this command fails. Removing a copyset that still owns data would result in immediate data loss, so the copyset remove command returns failure until the copyset can be safely removed.

   Note: If the grid is currently using checkpoints (such as, Live Backup and Restore or DR/Mirroring), take a new checkpoint immediately.

Checkpoints and Copyset Removal

When data is being redistributed, checkpoint data is not moved with the live data. This means that removing copysets might remove access to some checkpoints. Checkpoints that are not available due to copyset removal are displayed with an exclamation mark (!) in the output of the `tibdg checkpoint list` command. If using checkpoints, it is necessary to take a new checkpoint after redistribution completes to provide a target for the `latest` checkpoint name. After removing a copyset, all recent (since the last redistribution) checkpoints are marked as unavailable and cannot be used for checkpoint reads, checkpoint queries, or DR mirroring.

```
$tibdg -r http://myhost:8080 checkpoint list
                                    ID                 NAME
                                     DIRECTORY
TIMESTAMP   STATUS  NOTES
  607F019E-56C1-2400-9845-7647EC31E14D  only_original_nodes
20250115T223824.452Z_00000000_00000001_only_original_nodes  2025-01-15
22:38:24.452383000Z  Success
  93B69141-F773-3100-6D0C-F1B7A43BB817     with_set3_inst_0
20250115T223842.124Z_00000000_00000002_with_set3_inst_0  2025-01-15
22:38:42.124752000Z  Success      !
  AC1CA7B1-4C9C-EC00-C6D3-DAC3AC364F29    after_set3_inst_0
20250115T223902.937Z_00000000_00000003_after_set3_inst_0  2025-01-15
22:39:02.937931000Z  Success
  98C53D7E-907D-7F00-79C0-386DD8E790EE     with_set3_inst_1
20250115T223950.214Z_00000000_00000004_with_set3_inst_1  2025-01-15
22:39:50.214332000Z  Success      !
  DC4E8BF6-A499-5600-58C7-34FC7BC5CA95    after_set3_inst_1
20250115T224013.527Z_00000000_00000005_after_set3_inst_1  2025-01-15
22:40:13.527030000Z  Success
  93A3B9E9-F2E4-0C00-28D6-160EFE2BF2EC     with_set3_inst_2
20250115T224102.214Z_00000000_00000006_with_set3_inst_2  2025-01-15
22:41:02.214492000Z  Success      !
  B75E0AB0-E80E-3000-BE50-CD69907ADDBE    after_set3_inst_2
20250115T224124.836Z_00000000_00000007_after_set3_inst_2  2025-01-15
22:41:24.836348000Z  Success

! - No Longer Available because a required copyset is not present in the
grid config.
```

To use this information programmatically, specifying the `-j` flag provides the same information in JSON format.

Similarly, attempting to verify a checkpoint that requires removed copysets fails:

```
$tibdg checkpoint validate with_set3_inst_2
Checkpoint with_set3_inst_2 (epoch: 0, sqn: 6) failed validation on the
following copysets:
        <removed> (E9F1FA52-F50D-4F7E-BC67-5BFF791F3AAE)
*** ERROR OCCURRED OR CMD TIMED OUT RC = 1***
```

Defining a Table

Administrators define tables as needed to structure data. To define a table within the data grid, complete this task.

The examples in these steps illustrate adding commands to a configuration script. When the script is complete, the administration tool executes the script to define the table.

Alternatively, you can execute each step immediately as a separate administration tool command, instead of accumulating them in a script.

Note: Statistics for a table or an index must be enabled at creation time and cannot be enabled or disabled afterward. For more information, see Enabling Statistics.

Before you begin

A realm service must be running and reachable.

Either the realm must contain a valid data grid definition, or your configuration script file must contain commands to create a valid data grid definition.

Procedure

1. In a text editor, either begin editing a script file, or continue adding commands to an existing script.

   Follow the convention of naming your script with the `.tibdg` file name extension.

2. Add a script command to create the table. For example:

```
table create table_name key_column_name key_column_type
```

```
table create table_name key_col_1 col_1_type key_col_2 col_2_type
```

   Every table requires a primary key, which can consist of one or more columns. The first example creates a key with one column. The second example creates a key with

two columns.

The data type of key columns must be either long or string.

3. Define additional columns.

   For each column in the table, add a script command to create the column. For example:

   ```
   column create table_name column_name column_type
   ```

   Only the following FTL datatypes are valid as column types:

   - Long
   - Double
   - String
   - DateTime
   - Opaque

4. Optional. Define secondary indexes.

   For each index in the table, add a script command to create the index. For example:

   ```
   index create table_name index_name column_name
   ```

   ```
   index create table_name index_name column_1 column_2 column_3
   ```

   The data type of index columns must be either long or string.

5. Run the script to create the tables in the data grid.

   ```
   tibdg -s script_file_path
   ```

You can repeat this task to define additional tables. After creating the table, grant users or roles permissions to read from or write to the table.

Table Create Configuration Options

The command to create a table using tibdg has the following format:

```
tibdg table create [option=value]... table-name column-name column-type [column-
name column-type] ...
```

The following configuration options can be used with the `tibdg table create` command:

Option	Description	Default Value	Valid Value
`default_ttl`	Defines the number of seconds a row exists in a table before it is automatically removed from the table. A value of 0 seconds means that the row is never automatically removed from the table.	0	Minimum 0
`expiration_scan_period`	Defines how often, in seconds, the data grid should scan for rows that have surpassed their `default_ttl` setting.	3600	Minimum 1
`full_table_delete`	Defines the behavior when a SQL DELETE statement is created which does not contain a WHERE clause. Overrides the grid's full_table_delete setting for all tables. Execution of a SQL DELETE statement without a WHERE clause deletes all rows of a table. This option takes one of the following values: • inherited (default): The grid's full_table_delete setting is applied to the table. • enabled: A debug message is logged by the proxy when a SQL DELETE statement without a WHERE clause is created. All rows of the table are allowed to be deleted. • disabled: Prevents creation of SQL DELETE statements that do not contain a WHERE clause. • warn: A warning is logged by the proxy	inherited	inherited enabled disabled warn

Option	Description	Default Value	Valid Value
	when a SQL DELETE statement without a WHERE clause is created. All rows of the table are allowed to be deleted. **Note:** The `full_table_scans=disabled` option prevents the execution of a SQL DELETE statement without a WHERE clause regardless of the setting for `full_table_delete`. The option also prevents the execution of a SQL DELETE statement with a WHERE clause when an index cannot be found for the columns in the WHERE clause.		
row_counts	Enables statistics for a table. It cannot be modified after the table is created.	off	off exact

Column Names

Choose column names that follow these rules for SQL identifiers.

- Begin with a letter character.

  ActiveSpaces reserves column names that begin with an underscore character for internal use.
- Subsequent characters can be letters, digits, or underscore characters.
- Do not use SQL keywords as column names.
- Column names are not case-sensitive.
- The maximum length for column names is 256 bytes.

Invalid column names can cause errors when starting `tibdgnode`.

Special Characters in Column Names

Column names with special characters require special treatment.

It is good practice for administrators to define column names that follow the SQL identifier rules. (See "Column Names" in *TIBCO ActiveSpaces Administration*.)

Nonetheless, in some situations, a table might contain non-standard column names. For example, a table copied from a legacy data base might have columns with names that contain a space character.

If you must refer to non-standard column names in a filter expression, surround the column name with any of the following escape characters:

Technique	Example
Double quotes	"column name"
Escaped double quotes	\"column name\"
Square brackets	[column name]
Back ticks (accent grave)	\`column name\`

Secondary Indexes

A secondary index can increase query efficiency by reducing the number of rows to examine.

A secondary index can span one or more columns, and can include columns that are part of the primary key.

You can use the same column in more than one index.

Limit the number of secondary indexes, because each index increases the size of the information stored in the grid, and increases the overhead for each write operation.

As an administrator, create indexes that improve the performance of frequent query patterns. Delete indexes that no longer serve that purpose. For information about query performance, see "Efficiency of Filters" in *TIBCO ActiveSpaces® - Enterprise Edition Concepts*.

Enabling Statistics

Set the `row_counts` attribute to `exact` while creating a table or index.

Note: Statistics for a table or an index must be enabled at the creation time and cannot be enabled or disabled afterward.

The following is an example to create a table named `t1` with exact statistics enabled:

```
$ tibdg table create row_counts=exact t1 key long
```

Indexes on a table inherit the value of their table's `row_counts` setting by default, but can be explicitly configured differently when they are created.

The following is an example to add a secondary index to the table `t1` created above with statistics disabled.

Create the index with `row_counts` set to `off`:

```
$ tibdg index create row_counts=off t1 index2 myfield
```

See tibdg Table Stats to run a command to view statistics such as row counts or overall table size for a table and all of its indexes.

Note: As there is a minor performance impact associated with enabling statistics on a table or index, you can enable statistics on a table itself to maintain accurate row counts, but explicitly disable statistics on that table's secondary indexes.

Row Expiration

Rows are considered expired when they have exceeded their configured time-to-live (TTL) value. Ordinarily, rows in a table are not deleted until a client explicitly deletes them. In some situations, however, the data in the rows might only remain valid or relevant for a short span of time. Leaving these rows in the table indefinitely consumes disk space and in some situations, can slow down your queries. The row expiration feature is designed to remove outdated rows.

An application developer can override the default TTL by setting the TTL on a specific row before it is inserted in a table. Developers can use the C or Java API to override the TTL set for a table. For more information, see Overriding the Default TTL for a Single Row.

When a row is expired, it becomes available for deletion from the table. Note that becoming available for deletion does not mean it gets deleted immediately. Deletion takes place only after the table is scanned for expired rows. For more information, see Deletion of Expired Rows.

Defining a Table with Row Expiration

When defining a table, you can set the `default_ttl` property to a non-zero value to enable row expiration. The time interval is specified in seconds. Unless a different TTL was set on a particular row, rows inserted in a table inherit the default TTL value set on the table and expires after their TTL interval has elapsed. If the `default_ttl` property is not set or is set to `0` (zero), the rows of the table never expire and row expiration is not enabled for the table.

Procedure

1. Open the command prompt, and run `tibdg` using the following syntax to create a table in which rows expire:

   ```
   table create [default_ttl=<time interval in seconds>] table_name key_column_
   name key_column_type
   ```

   For example, if rows expire after 1 hour, the table would be created with the following `tibdg` command:

   ```
   tibdg table create default_ttl=3600 t1 key long value string
   ```

   For the same example, you can use the following SQL DDL command:

   ```
   CREATE TABLE t1 (key INT PRIMARY KEY, value VARCHAR) default_
   ttl=3600
   ```

Note: To ensure that a table is scanned for expired rows periodically, use the option `expiration_scan_period`. The default value is `3600` seconds and the minimum value is `1` second.

Result

All rows in the table expire after the specified TTL.

If you are using the C or Java APIs for ActiveSpaces, you can override the TTL for a table by setting the TTL property at a row level. For more details, see Overriding the Default TTL for a Single Row.

Overriding the Default TTL for a Single Row

When row expiration has been enabled for a table, developers can use the client APIs to override the default TTL set on a table.

Before performing the `PUT` operation on a row to insert it into a table, use the following code snippet to specify the TTL:

```
tibdgRow_SetTTL(row, <time_interval in seconds>);
tibdgTable_Put(table, row);
```

For example, if you want the row to expire after an hour, use the following code snippet:

```
tibdgRow_SetTTL(row, 3600);
tibdgTable_Put(table, row);
```

If you do not want the row to expire, set the TTL of the row to a very large value. For example, 10 years.

For more information on the APIs used to override the TTL at the row level, see *TIBCO ActiveSpaces® - Enterprise Edition C API Reference* or *TIBCO ActiveSpaces® - Enterprise Edition Java API Reference*.

Deletion of Expired Rows

When a row expires, it is available for deletion, but it is not deleted immediately. Deletion is a background process that scans the tables for expired rows and then deletes them.

Therefore, the application can retrieve rows that have expired but not deleted.

By default, tables are scanned for expired rows once every hour. The frequency of the scan is governed by the `expiration_scan_period` property. This property is specified in seconds. If the default TTL interval for a row is short or if you want the scanner to run more frequently to delete the expired rows, set the `expiration_scan_period` property when creating the table. After scanning the table, all expired rows are identified and deleted. The rows that expire first are deleted first.

Here is an example that uses the `tibdg` command to Defining a Table with a default TTL of 1 hour and a scan interval of 5 minutes:

```
tibdg table create default_ttl=3600 expiration_scan_period=300 t1 key
long value string
```

Defining a Table by Using SQL DDL Commands

Instead of using the `tibdg` command-line tool to define tables and index columns, you can use SQL Data Definition Language (DDL) command strings from within an application. When you pass these commands to the `ExecuteUpdate` API of the `tibdgSession` object, you can dynamically create and drop tables, and secondary table indexes, in a running data grid.

Before you begin

1. A realm service must be running and reachable.
2. The realm service must contain a valid data grid definition.
3. The `tibdgadmind` process must be running.
4. The data grid processes must be running (such as state keeper, nodes, and proxies).
5. If permissions are enabled on a data grid, ensure that you have the `tibdg-ddl` role. For more information, see ActiveSpaces Custom Roles.

Creating a New Table

Use the SQL DDL command `CREATE TABLE` to create a table in the data grid.

Before you begin

If the permissions are enabled on a table, you must have the `tibdg-ddl` role to create or modify the table. For more information about roles, see ActiveSpaces Custom Roles. For more information about table permissions, see Grid and Table Permissions.

Procedure

1. Compose a string with the following format:

```
CREATE TABLE [ IF NOT EXISTS ] <table_name> (
            <column_name> <column_type> <column_constraint>
            [, <column_name> <column_type> <column_constraint>]...
            [, <table_constraint>] )
            [ <property_name>=<property_value>
            [, <property_name>=<property_value>]... ]
```

Where:

```
<column_constraint> = [ [CONSTRAINT <constraint_name>] NOT NULL |
NULL | PRIMARY KEY ]
<column_type> = (see SQL Data Type Mapping)
<table_constraint> =
        [CONSTRAINT <constraint_name>] PRIMARY KEY (<column_name>
        [, <column_name>]...)
```

2. Pass the string to the `ExecuteUpdate` method of the `tibdgSession` object.

   For example:

```
CREATE TABLE mytable (col1 INT PRIMARY KEY, col2 VARCHAR)
CREATE TABLE IF NOT EXISTS mytable (col1 INT, col2 VARCHAR
CONSTRAINT col2_pk PRIMARY KEY)
CREATE TABLE table2 (col1 INT PRIMARY KEY, col2 VARCHAR) row_
counts=exact
```

   Note the following points when using the `CREATE TABLE` command:

   - Only columns with data types that map to ActiveSpaces long, string, and datetime data types can be primary key columns. For more information, see SQL Data Type Mapping.

   - Specifying a PRIMARY KEY column constraint and a PRIMARY KEY table constraint causes an error.
   - Specifying multiple columns with a PRIMARY KEY constraint causes an error. Use the PRIMARY KEY table constraint instead.
   - Specifying NOT NULL for primary key columns is optional. Primary key columns are implied to be NOT NULL.
   - Specifying NULL for primary key columns causes an error.

- For non-primary key columns, specifying NOT NULL causes an error. ActiveSpaces treats all non-primary key columns as nullable.
- Specifying a length for string columns is ignored (for example, VARCHAR(255)). ActiveSpaces does not support limiting the length of string columns.
- Object names (for example, <column_name>, <index_name>, and <table_name>) are case insensitive. ActiveSpaces converts all object names to lowercase before running the command.
- The property names that can be specified are the same as those used when defining a table by using the administration tool. For example, to enable statistics for a table, use `row_counts=exact`.

Dropping a Table

Use the SQL command `DROP TABLE` to remove a table and the data from the data grid.

Procedure

1. Compose a string with the following format:

```
DROP TABLE [ IF EXISTS ] <table_name>
```

2. Pass the string to the ExecuteUpdate method of the tibdgSession object.

   For example:

```
DROP TABLE myTable
DROP TABLE IF EXISTS myTable
```

Note: When using the `DROP TABLE` command, object names (for example, <table_name>) are case insensitive. ActiveSpaces converts all object names to lowercase before running the command.

Creating an Index

Use the SQL command `CREATE INDEX` to create a secondary index for a table in the data grid.

Procedure

1. Compose a string with the following format:

```
CREATE INDEX [ IF NOT EXISTS ] <index_name> ON <table_name> (
             <column_name> [, <column_name>]...)
             [ <property_name>=<property_value>
             [, <property_name>=<property_value>]... ]
```

2. Pass the string to the `ExecuteUpdate` method of the `tibdgSession` object.

   For example:

```
CREATE INDEX index1 ON table1 (col1, col5)
CREATE INDEX IF NOT EXISTS index2 ON table1 (col1, col5, col7)
CREATE INDEX index3 ON table1 (col2, col3) row_counts=exact
```

   Note the following points when using the `CREATE INDEX` command:

   - Only columns with data types that map to ActiveSpaces long and string data types can be secondary index columns. For more information, see SQL Data Type Mapping.
   - Escaped names are supported.

     Note: In SQL, an escaped name is a sequence of one or more characters enclosed within SQL escape characters. Trailing spaces are insignificant.

     Special characters used within the escaped name must themselves be escaped.

   - Object names (for example <column_name>, <index_name>, and <table_name>) are case insensitive.
   - ActiveSpaces converts all object names to lowercase before running the command.

Dropping an Index

Use the SQL command DROP INDEX to remove a secondary index from a table in the data grid.

Procedure

1. Compose a string with the following format:

```
DROP INDEX [IF EXISTS] <table_name>.<index_name>
```

2. Pass the string to the ExecuteUpdate method of the tibdgSession object.

   For example:

```
DROP INDEX table1.index1
DROP INDEX IF EXISTS table1.index1
```

   Note the following points when using the `DROP INDEX` command:

   - Primary keys cannot be dropped. Only secondary indexes can be dropped.
   - Object names (for example, <index_name>, and <table_name>) are case insensitive. ActiveSpaces converts all object names to lowercase before running the command.

SQL Data Type Mapping

ActiveSpaces uses a small set of data types for storing data in the data grid. Several SQL data types have been mapped to each of the ActiveSpaces data types. The following table lists each data type and the SQL data types that have been mapped to it:

ActiveSpaces Data Type	SQL Data Types
string	char varchar longvarchar

ActiveSpaces Data Type	SQL Data Types
	text character varying
long	int bit bigint tinyint smallint integer boolean
opaque	blob binary varbinary longvarbinary
double	float real double
datetime	datetime date time timestamp

Security

ActiveSpaces security is based on the security features of TIBCO FTL.

The following security features are provided and must be used together:

- Transport encryption
- Authentication and authorization

Transport encryption is used to encrypt any network communication between the processes of your data grid to protect that communication from packet sniffing. For more information, see Enabling Transport Encryption on a Data Grid.

TIBCO recommends that you configure a secure data grid with transport encryption and authentication and authorization.

Authentication and authorization use usernames and passwords to authenticate the users of the data grid and prevent unwanted users from accessing the data grid. When authentication and authorization are enabled, each ActiveSpaces process authenticates itself to a secure realm service by using the password files credentials.

For more information, see Authentication and Authorization.

Authentication and Authorization

ActiveSpaces authentication is based on the authentication support of TIBCO FTL.

FTL supports the following forms of authentication:

- LDAP using JAAS
- Flat-file authentication which runs inside of the realm service and reads usernames and passwords from a flat file.
- Flat-file authentication that runs in a container external to the realm service.

Authorization Groups

A username may belong to several authorization groups (also known as roles). Authorization groups can be configured in either the JAAS file or the flat-file.

The following are examples of users and authorization groups defined in a flat-file with the required authorization groups for running a data grid with authentication and authorization:

- Admin User - A user for authenticating the `tibftladmin`, `tibdg`, and `tibdgproxy` processes that has the `ftl-admin` role and the `tibdg-internal` role. The following statement is an example:

  ```
  admin: adminpw, tibdg-internal,ftl-admin,ftl
  ```

- Realm Service User - A single user with roles for authenticating the primary TIBCO FTL realm and all satellite realms. This single user facilitates switching between a backup realm service and its primary realm service, or a satellite realm service and its primary realm service. The following statement is an example:

  ```
  rs: rspw, ftl-internal,ftl-admin
  ```

- `tibdgadmind` User - A user for authenticating the `tibdgadmind` process that requires the `ftl-internal` and `ftl-admin` roles. The following statement is an example:

  ```
  tibdgadmind: tibdgadmindpw, ftl-internal,ftl-admin
  ```

- `tibdg` User - An internal `tibdg` user for starting and authenticating internal grid processes like the `tibdgkeeper` and `tibdgnode`, which require the basic `ftl` role and the `tibdg-internal` role. The following statement is an example:

  ```
  tibdguser: tibdguserpw, tibdg-internal,ftl
  ```

- Client SQL DDL Users - Additional users as required for ActiveSpaces clients who also have permission to run SQL DDL statements such as CREATE TABLE. See the section on Grid and Table Permissions. The following is an example:

  ```
  user3: user3pw, tibdg-ddl,ftl
  ```

- Client Users - Additional users as required for ActiveSpaces clients. These clients only

need the basic role of `ftl`. The following statement is an example:

```
user1: user1pw, ftl
user2: user2pw, ftl
```

When running a disaster recovery data grid with a satellite realm service, an authorization file must include the following users as described above:

- Realm Service User - use the same name and password in all authentication files used by affiliated realm services
- Admin User
- `tibdg` user

The client users listed in an authentication file can vary between primary and satellite realm services. A `tibdgadmind` user is only required for the primary realm service. However, it is a good practice to include a `tibdgadmind` user in all authentication files so that it does not have to be added later when a mirror data grid needs to become the primary data grid. For more information about disaster recovery, see Disaster Recovery.

Note: No spaces are allowed between the comma-separated list of authorization groups. For example:

```
ftl-satellite,ftl-admin  (correct)
ftl-satellite, ftl-admin (incorrect)
```

Password File

If the realm service enables authentication and authorization, then you must configure credentials for ActiveSpaces processes such as `tibdgnode`, `tibdgkeeper`, `tibdgproxy`, and `tibdgadmind`.

Each service process authenticates itself to the realm service by using credentials that it reads from an ASCII password file. Specify the name and location of that file by using the client's `--user-password-file` command-line parameter.

The password file consists of two lines. The first line contains the username. The second line contains the password string. On all platforms (including Windows) the lines must be separated by the new line character `\n`.

You can use the masking of a password feature from TIBCO® FTL.

For more information about how to generate a masked password, see the `--mask` option available in the "FTL Administration Utility" section in *TIBCO® FTL Administration*.

Starting Realm Services with Authentication

To use a data grid with authentication, secure realm services by using transport encryption plus authentication must be used.
The remainder of this section contains examples of using a flat-file for authentication. You must alter the steps as required for the type of authentication you intend to use with your data grid.

Before you begin

Use secure realm services that have enabled transport encryption and authentication.

Procedure

1. Determine the type of authentication that you need by reading the section on "Authentication Service" in the document *TIBCO FTL® Administration*.
2. Perform the authentication setup tasks required before starting up the realm service. For flat-file authentication, create a flat-file with usernames, passwords, and authentication groups. For details, see Authorization Groups.
3. Start a secure primary realm service as described in step 1 under Transport Encryption with the additional authentication options required for the type of authentication that you are going to use.

```
tibrealmserver -http <host>:<port> --data <rs_db_path> --secure
pass:<keystore_pwd> --tls.trust.file
<trust_file_path> --auth.url file://<flat_file_path> --server.user
<rs_user_name> --server.password <rs_user_pwd>
```

Note: When using TIBCO FTL 6.0 or later, use `tibftlserver` instead of `tibrealmserver`. Refer to *TIBCO FTL Administration* for information on converting TIBCO FTL 5.x `tibrealmserver` command-line options into the appropriate TIBCO FTL 6.x configuration file options, most of which use the same name. For example, `--server.user` is the `server.user` configuration file option.

4. Ensure that the trust file from the primary realm service has been copied to locations where any affiliated realm service (for example, backup, satellite), each of the data grid's processes, and any client processes can access a copy of it.

5. Start the affiliated realm services (for example backup, satellite) and enable transport encryption.

6. Set the following authentication options:

```
--secure pass:<keystore_pwd>
--tls.trust.file <path>
--server.user <rs_user_name>
--server.password <rs_user_pwd>
--auth.url file://<flat-file path>
```

   For more information about running secure realm services, realm service authentication, realm service command-line options, and realm service configuration properties, see *TIBCO FTL® Administration*.

Starting Data Grid Processes With Authentication

Before starting your data grid processes with authentication, define your data grid and its component processes as described in starting affiliated realm services and secure realm services in Enabling Transport Encryption on a Data Grid.

Procedure

1. For the `tibdg` and `tibdgadmind` tools to authenticate a secure realm service specify the following properties:

   - `--trust-file <path>`
   - For providing the user credentials, use one of the following methods:
     - `--user-password-file <path>`: cannot be used with `-user` and `-password` options. For details, see Password File.

       or

     - `-user <user_name> -password <pwd_option>`: cannot be used with `--user-password-file` option. `<pwd_option>` can take one of the following values:

Options	Usage Example
pass:<password>	-password pass:mypassword
env:	env:<environment variable>
stdin	You are prompted for the password when the command runs.

Note: `tibdg` requires the user to have the `ftl-admin` role. `tibdgadmind` requires the user to have `ftl-satellite` and `ftl-admin` roles.

For example:

```
tibdg -r %REALM_URL% -s my_script_file.tibdg --user-password-file
/path/to/my/user-password.txt --trust-file %TRUST_FILE%

    tibdg -r %REALM_URL% -s my_script_file.tibdg -user admin -
password pass:password --trust-file %TRUST_FILE%
```

2. For the `tibdgnode`, `tibdgkeeper`, and `tibdgproxy` processes to authenticate with a secure realm service specify the following properties:
   - `--trust-file <path>`. For details, see Trust File (TIBCO FTL-Generated Certificates).
   - `--user-password-file <path>`. For details, see Password File.

Note: `tibdgnode` and `tibdgkeeper` processes require a set of username and password with the `ftl` role. `tibdgproxy` processes require a set of username and password with the `ftl-admin` role.

Using User-Defined TIBCO FTL Certificates

ActiveSpaces grid processes and client applications can use the security capabilities of TIBCO FTL, which include user-defined certificates and OAuth2 or mTLS authentication providers.

User-Defined Certificates

Configure a grid using your own user-defined certificates instead of TIBCO FTL-generated certificates. For more information, see Enabling TLS for FTL Server. If you are using these certificates, TIBCO FTL server, all ActiveSpaces grid processes, and all ActiveSpaces client applications must be updated and properly configured with the correct command-line parameters and connection properties to communicate successfully.

If you want, you can load the required trust file for each process to the system trust store rather than configuring the trust file by using command line or connection properties. For specific information about loading certificates into the system trust store, consult your operating system documentation.

You must configure an authentication provider when you enable TLS connections in TIBCO FTL.

ActiveSpaces includes a TLS sample directory with configuration and a README file to highlight additional parameters needed for this setup. Additionally, in the `samples/scripts` directory, the `as-certs` script is provided to demonstrate how to generate the certificates correctly.

Note the following key differences in ActiveSpaces grid processes when you configure them with user-defined certificates:

- After generating the certificates, the TIBCO FTL server must set the following additional properties in the `ftl.yaml` file:
    - `tls.server.cert`
    - `tls.server.private.key`
    - `tls.server.private.key.password`
    - `tls.client.trust.file`
- You must configure `tibdgadminsvc` as service in the TIBCO FTL server and not as a standalone process. Although you can configure state keepers as a standalone process, TIBCO recommends you to configure the state keeper in the same way (as a service). When you run processes as services, you do not need security-related command-line parameters because the TIBCO FTL server provides the local, secure connections. Every TIBCO FTL server must host a `tibdgadminsvc` service.
- To connect to the TIBCO FTL server, the `tibdg` admin tool needs the following files:
    - a trust file (`--trust-file`)

    - a user password file (--user-password-file)

- The tibdgnode and tibdgproxy grid processes connecting to the TIBCO FTL server must use the following command-line parameters:
    - --trust-file
    - --user-password-file
- As tibdgnode and tibdgproxy grid processes act as servers for other processes, they must use the following command-line parameters:
    - --server-cert-file
    - --server-private-key
    - --server-private-key-pwd
    - --server-host
- The tibdgnode and tibdgproxy grid processes must specify a --server-host parameter that matches their certificate and the host on which they are running.
- ActiveSpaces client applications must set the connection property TIBDG_CONNECTION_PROPERTY_STRING_TRUST_TYPE to TIBDG_CONNECTION_HTTPS_CONNECTION_USE_SPECIFIED_TRUST_FILE and set the TIBDG_CONNECTION_PROPERTY_STRING_TRUST_FILE property value to the location of the user-generated trust file. For more information, see the samples directory.

Authentication Providers

You can configure the following authentication providers with user-defined certificates:

- File-based authentication provider
- OAuth2 authentication provider
- mTLS authentication provider

File-based Authentication Provider

The file-based authentication provider in the FTL server is supported for both FTL-generated and user-defined certificates. Client applications use the TIBDG_CONNECTION_PROPERTY_STRING_USERNAME and TIBDG_CONNECTION_PROPERTY_STRING_USERPASSWORD properties to set the authentication values that must match the values provided in the authentication file provided to the TIBCO FTL server.

OAuth2 Authentication Provider

Configure the TIBCO FTL server with an OAuth2 authentication provider, which is then used by ActiveSpaces grid processes and client applications. Grid connections are authenticated by obtaining and using a signed JWT token issued by a separate OAuth2 server. For more information, see Using the Built-In OAuth 2.0 Based Authentication Service in TIBCO FTL® *Security*.

ActiveSpaces includes an oauth2 sample directory that includes configuration and a README to provide you with the parameters that are needed with this setup. Additionally, the `as-certs` script is provided that helps you generate the certificates correctly.

Note the following key differences in ActiveSpaces grid processes when you configure them with an OAuth2 authentication provider:

- After the certificates are generated, the TIBCO FTL server sets several additional properties in the `ftl.yaml` file including:
    - `tls.server.cert`
    - `tls.server.private.key`
    - `tls.server.private.key.password`
    - `tls.client.trust.file`
    - `oauth2.validation.key`
    - `oauth2.svr.client.id`
    - `oauth2.svr.client.secret`
    - `oauth2.svr.endpoint.token`
    - `oauth2.provider.trust.file` (required only if the OAuth2 server uses https)
- You must configure `tibdgadminsvc` as service in the TIBCO FTL server and not as a standalone process. Although you can configure state keepers as a standalone process, TIBCO recommends you to configure the state keeper in the same way (as a service). When you run them as service, you do not need the security-related command-line parameters because the TIBCO FTL server provides the local, secure connections. Every TIBCO FTL server must host a `tibdgadminsvc` service.
- To connect to the TIBCO FTL server, the `tibdg` admin tool needs the following files:
    - a trust file (`--trust-file`)
    - oauth2 token (`--oauth2-token`)

- To connect to the TIBCO FTL server, the `tibdgnode` and `tibdgproxy` grid processes must use the following command-line parameters:
  - `--trust-file`
  - `--oauth2-server-url`
  - `--oauth2-client-id`
  - `--oauth2-client-secret`
  - `--oauth2-server-trust-file` (when connecting via https to the OAuth2 server)
- As `tibdgnode` and `tibdgproxy` grid processes act as servers for other processes, they must use the following command-line parameters:
  - `--server-cert-file`
  - `--server-private-key`
  - `--server-private-key-pwd`
  - `--server-host`
- ActiveSpaces client applications must set the following connection properties:
  - `TIBDG_CONNECTION_PROPERTY_STRING_OAUTH2_SERVER_URL`
  - `TIBDG_CONNECTION_PROPERTY_STRING_OAUTH2_CLIENT_ID`
  - `TIBDG_CONNECTION_PROPERTY_STRING_OAUTH2_CLIENT_SECRET`
  - `TIBDG_CONNECTION_PROPERTY_STRING_OAUTH2_SERVER_TRUST_FILE` (needed to connect to an https-based OAuth2 server URL)

mTLS Authentication Provider

Configure the TIBCO FTL server with an mTLS authentication provider, which is then used by ActiveSpaces grid processes and client applications. Clients can authenticate with the TIBCO FTL server with specifically formatted certificates. For more information, see the Using the Built-In mTLS Based Authentication Service section in TIBCO FTL® *Security*.

ActiveSpaces includes the `mtls sample` directory that includes configuration and a README to highlight the parameters that are needed with this setup. Additionally, the `as-certs` script is provided that helps you generate the certificates correctly.

Note the following key differences in ActiveSpaces grid processes when you configure them with user-defined certificates and an mTLS authentication provider:

- After the certificates are generated, the TIBCO FTL server sets several additional properties in the `ftl.yaml` file including:
    - `tls.server.cert`
    - `tls.server.private.key`
    - `tls.server.private.key.password`
    - `tls.client.trust.file`
- You must configure `tibdgadminsvc` as service in the TIBCO FTL server and not as a standalone process. Although you can configure state keepers as a standalone process, TIBCO recommends you to configure the state keeper in the same way (as a service). When you run them as service, you do not need the security-related command-line parameters because the TIBCO FTL server provides the local, secure connections. Every TIBCO FTL server must host a `tibdgadminsvc` service.
- To connect to the TIBCO FTL server, the `tibdg` admin tool must use a trust file and client certificates:
    - `--trust-file`
    - `--client-cert-file`
    - `--client-private-key`
    - `--client-private-key-pwd`
- To connect to the TIBCO FTL server, the `tibdgnode` and `tibdgproxy` grid processes must use the following command-line parameters:
    - `--trust-file`
    - `--client-cert-file`
    - `--client-private-key`
    - `--client-private-key-pwd`
- As `tibdgnode` and `tibdgproxy` grid processes act as servers for other processes, they must use the following command-line parameters:
    - `--server-cert-file`
    - `--server-private-key`

  - `--server-private-key-pwd`
  - `--server-host`
  - `--server-trust-file`
- The `tibdgnode` and `tibdgproxy` grid processes must specify a `--server-host` parameter that matches their certificate and the host on which they are running.
- ActiveSpaces client applications must set the following connection properties:
  - `TIBDG_CONNECTION_PROPERTY_STRING_CLIENT_CERT`
  - `TIBDG_CONNECTION_PROPERTY_STRING_CLIENT_PRIVATE_KEY`
  - `TIBDG_CONNECTION_PROPERTY_STRING_CLIENT_PRIVATE_KEY_PASSWORD`

Samples

In the installation directory of ActiveSpaces, in the `samples/security` directory, security configurations and README files are provided to explain how to configure and use the TIBCO FTL 7.0.0 security capabilities. Refer to these samples to understand how to configure grid processes and client applications.

In the `sample/scripts` directory, the `as-certs` script is provided that you can use as an example to understand how to generate properly formatted user-defined certificates.

Enabling Transport Encryption on a Data Grid

Before you begin

ActiveSpaces transport encryption is based on the transport encryption of TIBCO FTL. If the computer on which you run the TIBCO FTL server has multiple network interface cards, ensure that the host name is mapped to the IP address that you use to start your TIBCO FTL server. Otherwise, the certificate generated by the TIBCO FTL server might use one of the other available IP addresses. As a result of the IP address mismatch, ActiveSpaces processes would not be able to connect to the realm service.

Procedure

1. Generate a trust file by using a TIBCO FTL server. For instructions, see "Securing FTL

Servers" in *TIBCO FTL® Administration*.

2. Supply copies of the keystore file and trust file to every TIBCO FTL server.
3. Supply a copy of the trust file to locations that can be accessed by any of the data grid's processes and client processes.
4. Configure the TIBCO FTL servers to use TLS security in their configuration files.

```
globals:
        tls.secure: <keystore_password>
```

5. Start the TIBCO FTL servers.

```
tibftlserver -c <config_file> -n <server_name>
```

6. After the secure realm services have been started, create the data grid configuration by using the encrypted_connections option and set its value to `all`.

```
grid create copyset_size=1 statekeeper_count=3 encrypted_
connections=all mygrid
```

7. Define the component processes of your data grid. For more information, see Defining a Data Grid.

Trust File (TIBCO FTL-Generated Certificates)

A trust file is generated by using the `--init-security` command-line option of `tibftlserver`. The content of the trust file instructs clients to trust the realm service's certificate. Administrators and developers coordinate to supply the trust file to application programs.

A secure realm service generates the trust file in its data directory. The trust file is named `ftl-trust.pem`. The file contains one or more PEM-encoded public certificates, each of which is typically 1 - 2 KB of data.

Realm administrators give the trust file to the clients: that is, developers and application administrators coordinate so that client programs can access the trust file at run time.

Administrators also supply the trust file directly to ActiveSpaces processes such as `tibdgnode`, `tibdgkeeper`, `tibdgproxy`, and `tibdgadmind`.

Users can load the trust file into a web browser’s trust store.

Affiliated Realm Services and the Trust File

An affiliated realm service uses the same trust file as its primary realm service. That is, even if you create a different private key for a backup or satellite realm service, a primary server signs that key, so the primary's trust file is still valid for the satellites and their clients. As a consequence, you do not distribute separate trust files to clients of a family of affiliated servers: one trust file suffices for the whole family.

Regeneration and Redistribution of the Trust File

If a realm service cannot access its TLS data files, or it cannot decrypt the keystore file, then it generates new TLS data files. The newly generated data files replace any existing data files.

If a primary realm service generates new TLS data files, you must redistribute the new trust file to all clients, including affiliated realm services, other TIBCO FTL components, application programs, and browsers that access the realm service GUI.

Two scenarios can trigger this requirement:

- **No Access**: A primary realm service restarts and cannot access its TLS data files: for example, they have been deleted or moved, or their file access permissions have changed.
- **New Password**: An administrator restarts the primary realm service, supplying a different password. The server cannot decrypt the existing keystore file by using the new password.

If a *secondary* realm service generates new TLS data files, do not redistribute its trust file.

Using Trust Files with Primary Realm Service

Procedure

1. When using the `tibdg` administration tool to communicate with the primary realm service, always specify the path to the trust file from the primary realm service. For example, to use `tibdg` to define your data grid run the following command:

```
tibdg -r <realm_service_url> --trust-file <path> grid create
copyset_size=1 statekeeper_count=3 encrypted_connections=all mygrid
```

2. Start your data grid processes as described in Starting the Data Grid Processes with the following additional command-line option to indicate the location of the trust-file on the primary realm service:

```
--trust-file <path>
```

The following statement is an example of starting the data grid processes by specifying the location of the trust file located on the primary realm service:

```
tibdgnode -r <realm_service_URL> -g mygrid -n <node_name> --trust-
file <path>
```

Using Trust Files with the Disaster Recovery Feature

If using the disaster recovery feature, note the following points:

- Start your realm services and data grid processes as described in Enabling Transport Encryption on a Data Grid.
- The data grid to run on the Disaster Recovery site must be configured and deployed by using the primary realm service and not the satellite realm service. Configuration changes can only occur on the primary realm service.
- The `tibdgadmind` tool is not run on the Disaster Recovery site as configuration changes can only occur on the primary realm service.

> **Note:** Realm services can be started by using configuration files or different formats of the command-line options specified in this section. For alternatives to using the realm service command-line options, see the document *TIBCO FTL® Administration*.

Grid and Table Permissions

Data grid access control can be extended to apply permissions on tables to control who has access to the data in the tables.

If you are not familiar with using authentication and authorization with ActiveSpaces, it is recommended that you read Authentication and Authorization. Enabling table permissions helps you grant read or write access to users or authorization groups (also known as roles). With the read permission on a table, a user or role can read the data in the table. With the write permission on a table, a user or role can modify the data in the table.

Examples of read operations are GET operations, running SQL SELECT statements, and creating a Listener on the table.

Examples of write operations are PUT and DELETE operations and SQL INSERT statements.

To set permissions on a table, see Enabling Permission Checking on Data Grids and Tables.

How Do You Know If You Have Permissions on a Table?

When permission checking is enabled, only tables that the user has some permission on are shown in the grid metadata returned by `tibdgConnection_GetGridMetadata()` .

Enabling Permission Checking on Data Grids and Tables

This topic helps you set up a grid with permissions enabled so that you can subsequently enable permission checking on the tables in the data grid.

Before you begin

1. Shut down the ActiveSpaces data grid and TIBCO FTL servers.
2. Ensure that Transport Layer Security (TLS) has been configured for the TIBCO FTL

servers. For more information, see Enabling Transport Encryption on a Data Grid.

3. Configure the appropriate users and roles that are accessed by the TIBCO FTL server. For more information, see Authentication and Authorization. Remember that the users and roles can be one of the following types:
    - Users and roles for client applications that are granted table permissions.
    - Users who can use SQL to create or modify table definitions with the `tibdg-ddl` role. For more information, see ActiveSpaces Custom Roles.
    - The user with the `tibdg-internal` role that is needed to start the ActiveSpaces grid processes. For more information, see ActiveSpaces Custom Roles.
4. After creating the necessary users and roles for authentication and authorization purposes, start the secure TIBCO FTL servers.
5. Ensure that transport encryption has been enabled for the ActiveSpaces data grid (`encrypted_connections=all`). For more information, see Enabling Transport Encryption on a Data Grid.

Procedure

1. Enable permission checking on the ActiveSpaces data grid. For more information, see Enabling Permission Checking when Creating or Modifying a Data Grid.

    Permission checking is now enforced when the grid processes are started. They are also enforced when you access tables to perform read and write operations.
2. Start the ActiveSpaces data grid processes with a user account that has the `tibdg-internal` role. For more information, see Starting Data Grid Processes With Authentication.
3. Create or modify the table definitions in the ActiveSpaces data grid to grant users or roles permission to access the table. For more information, see The tibdg Commands to Set Permissions on a Table.

ActiveSpaces Custom Roles

Permission checking introduces two new roles that are specific to ActiveSpaces. These roles must be configured in the FTL Server.

The tibdg-internal Role

When permission checking is enabled to run a node, proxy, or state keeper, you must have the `tibdg-internal` role in addition to any other roles required to start up the process with authentication and authorization. If you do not have this role, the process exits during startup. For more information, see Authorization Groups.

The tibdg-ddl Role

When permission checking is enabled, to create or modify a table by using SQL, you must have the `tibdg-ddl` role in addition to any other roles required to run as a client user with authentication and authorization. For more information, see Authorization Groups.

Impact of Permissions on SQL DDL Statements

To create a table by using SQL, you must have the `tibdg-ddl` role. The table is created with the user having read and write permissions on it. To modify or delete a table by using SQL, you must have the `tibdg-ddl` role and write permission on the table. Ensure that you grant permissions to the users or roles that are expected to use the table. If not, they cannot use the table. For more information about creating a table by using SQL DDL commands, see Defining a Table by Using SQL DDL Commands.

Enabling Permission Checking when Creating or Modifying a Data Grid

To enable permission checking when creating or modifying a data grid, you must set the following properties:

- Set `encrypted_connections=all`. For more information about encrypting connections, see Enabling Transport Encryption on a Data Grid.
- Set `permissions=enabled`.

Note: When all connections are encrypted, ensure that all ActiveSpaces processes, both clients and servers, must be started by using usernames and passwords. For more details, see Security.

Procedure

1. To enable permission checking when creating or modifying a data grid, use the following commands:

Option	Command To Set Permission
Creating a data grid	`tibdg grid create encrypted_connections=all permissions=enabled ...`
Modifying a data grid	`tibdg grid modify encrypted_connections=all permissions=enabled ...`

Note: If you modify these options on a running grid, ensure that you restart the grid for the changes to take effect.

The tibdg Commands to Set Permissions on a Table

When data grid permission checking has been enabled, by default, tables created by using the `tibdg create table` command do not have permissions set on them, meaning no users can have access to the data in the table. Only after users or roles are granted permissions on the table, are users able to read or write data in the table.

The `tibdg` tool offers two options, `grant` and `revoke` to control the permissions given to a user or a role. For convenience, the `grant` and `revoke` commands support granting all permissions to a user or role, which is the equivalent to granting or revoking both read and write permissions.

Permissions can be granted to or revoked from a specific user or a role. If a permission has been granted to a role, then any user with that role has that permission.

Granting Permission to a User or a Role

Before you begin

- Ensure that you have transport encryption in the grid by setting `encrypted_connections=all` in the grid configuration. See Enabling Transport Encryption on a Data Grid.
- The user or role must exist in the TIBCO FTL realm server. For more details, see "Configuring Authentication and Authorization" in TIBCO FTL® *Security*.

Procedure

1. On a specific table, to grant permissions for a user, use the `tibdg user grant` command. Use the following syntax:

   ```
   tibdg user grant table-name user-name [READ|WRITE|ALL]
   ```

2. On a specific table, to grant permissions for a role, use the `tibdg role grant` command. Use the following syntax

   ```
   tibdg role grant table-name role-name [READ|WRITE|ALL]
   ```

Revoking Permission from a User or a Role

After a permission has been granted on a table, it can be subsequently revoked by using the `tibdg user revoke` or `tibdg role revoke` command.

Before you begin

- Ensure that you have transport encryption in the grid by setting `encrypted_connections=all` in the grid configuration. See Enabling Transport Encryption on a Data Grid.

Procedure

1. On a specific table, to revoke permissions from a user, use the `tibdg user revoke` command. Use the following syntax:

```
tibdg user revoke table-name user-name [READ|WRITE|ALL]
```

2. On a specific table, to revoke permissions from a role, use the `tibdg role revoke` command. Use the following syntax:

```
tibdg role revoke table-name role-name [READ|WRITE|ALL]
```

ActiveSpaces Monitoring Service

ActiveSpaces Monitoring Service is a web-based tool to monitor your data grid and its component processes.

The monitoring information includes the user operations on the ActiveSpaces grid and the basic health of the data grid. The user operations include PUT, GET, and DELETE operations. Statistics such as the number of concurrent queries run and the number of active listeners help you gauge the overall health of the data grid.

ActiveSpaces Monitoring Service includes the following dashboards:

- ActiveSpaces Grid Activity
- ActiveSpaces Nodes Activity
- ActiveSpaces Proxies Activity

The following is an example of the ActiveSpaces Grid Activity dashboard when the data grid is actively handling PUT, GET, UPDATE, and DELETE activities.

If you run the samples provided, you get a simple data grid with one node, one state keeper, and one proxy.

Note: The metrics `tib_as_node_iterget_op_count` and `tib_as_node_queryget_op_count` represent the number of batches getting operations (dependent on the prefetch) and not the number of individual rows getting operations.

Using ActiveSpaces Monitoring Service

The Grafana dashboards used until ActiveSpaces 4.0 are now deprecated. ActiveSpaces 4.1 and later depends on the TIBCO FTL dashboards, which use InfluxDB dashboards.

Before you begin

Ensure that the realm service is up and running.

Procedure

1. Open a command window. Navigate to one of the following paths:

Option	Description
ActiveSpaces 4.0 or earlier dashboards (Deprecated Grafana dashboards)	*TIBCO_HOME*/as/<*version*>/legacy_monitor/scripts
ActiveSpaces 4.1 dashboards	*TIBCO_HOME*/as/<*version*>/monitor/readme.md

2. To use the ActiveSpaces Monitoring service, follow the steps listed in the `readme.md` file.
3. Open a browser and in the address field enter the URL for the ActiveSpaces Monitoring Service.

   The default value is `http://<hostname>:3000`.
4. On the dashboard's landing page, provide the login credentials.

   The default login credentials are username `admin` and password `admin`.

Result

Monitoring data is displayed in the dashboards.

Important: If the dashboard does not display any data, ensure that the realm service is running.

Installing or Uninstalling ActiveSpaces Processes as Windows Services

Using an ActiveSpaces process as a Windows service is beneficial to start the process automatically when the computer starts up and when you want the process to continue running even if the associated user is not logged into the system. In addition, you can define a policy on the action to take on the failure of the Windows service. You can also define dependencies on other Windows services.

To arrange ActiveSpaces processes as Windows services, use the `prunsrv` tool, which is part of the Apache Procrun package. The ActiveSpaces installer includes this tool on Windows platforms. For documentation, see Apache Commons. To start the TIBCO FTL servers as a Windows service, see "TIBCO FTL Processes as Windows Services" in *TIBCO FTL® Administration*.

Installing ActiveSpaces Processes as Windows Services

A single command is used to install each process as a Windows service. The command is a call to `prunsrv.exe` with the `install service` parameter (`//IS`). The following generic references are made to the installation environment:

- *TIBCO_HOME* is the top-level installation directory for TIBCO products.
- *version* is the current version of the product. For example, the current version is ActiveSpaces.

Before you begin

Ensure that both Java and the TIBCO FTL Windows package and the TIBCO ActiveSpaces Windows package are installed on a local disk of the host computer (not on a mapped network drive).

Procedure

1. Model your command based on the following template:

```
TIBCO_HOME\as\<as_version>\bin\prunsrv.exe //IS/tibdgservicename --
DisplayName="TIBCO Service Name" --Install=TIBCO_HOME\as\<as_
version>\bin\prunsrv.exe
--StartMode=exe
--StartImage=TIBCO_HOME\as\<as_version>\bin\tibdg***.exe
--LibraryPath=TIBCO_HOME\as\<as_version>\bin;TIBCO_HOME\ftl\<ftl_version>\bin
--StartParams=-n;process_name;otherparams
--StopMode=exe
--StopImage=TIBCO_HOME\as\<as_version>\bin\tibdg.exe
--StopParams=processtype;stop
```

Tip: If you plan to copy the code snippet, remember to remove line breaks for all the sample commands that are associated with `prunsrv.exe`.

Notice these aspects of the command-line template:

- `--Install` is the file path of the `prunsrv` executable.
- `--LibraryPath` is the directory containing TIBCO FTL and ActiveSpaces DLL files.
- `--StartParams` contains the command-line parameters needed to start the process. Semicolon (;) is the separator character, as prunsrv does not allow spaces.
- `--StopParams` contains the command-line parameters needed for the `tibdg` administration utility. These include the URL to one or more TIBCO FTL servers and the name of the ActiveSpaces process to stop. Semicolon (;) is the separator character, as `prunsrv` does not allow spaces.

Uninstalling ActiveSpaces Processes as Windows Services

One command uninstalls any Windows service that you installed by using `prunsrv`.

Procedure

1. Model your command based on the following template:

```
TIBCO_HOME\as\<as_version>\bin\prunsrv.exe //DS/service_name
```

Deployment Scenario for Running ActiveSpaces Processes as Windows Services

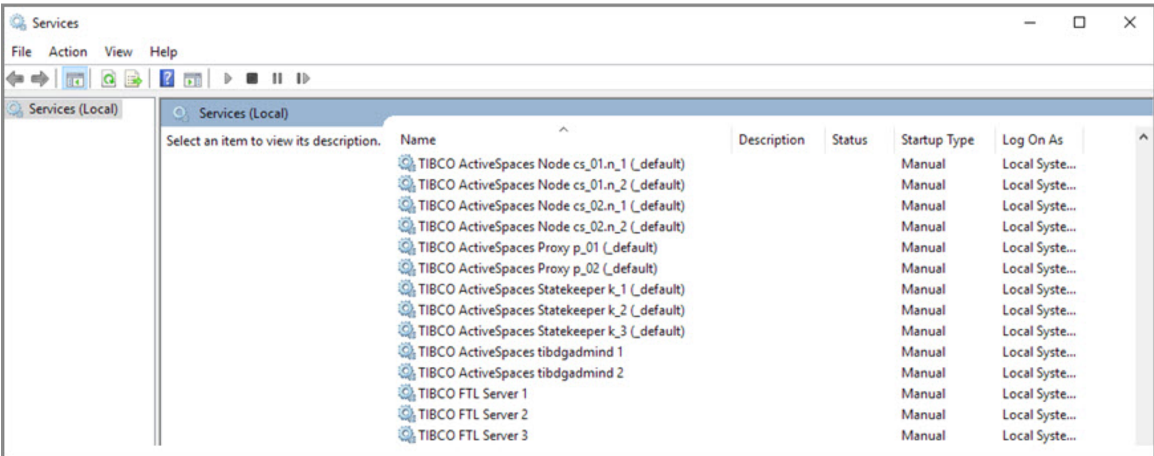
For a system running ActiveSpaces as Windows Services, the same recommendations apply as they would with any other deployment as to how the processes must be spread out across multiple computers for fault tolerance and scaling. For more information, see the "Best Practices for a Production Environment" section in *TIBCO ActiveSpaces® - Enterprise Edition Concepts*.

The deployment scenario shows how to run more than one ActiveSpaces processes as a Windows service on the same computer effectively. The name of the data grid is `default`. The following processes are running in the data grid:

Processes	Numbers
Realm Service	Three
State Keeper	Three
Node	Two nodes per copyset
Proxy	Two
Copyset	Two

The processes in the data grid run as Windows Services.

ActiveSpaces Processes as Windows Services



Preparing for Installation

Before installing ActiveSpaces processes as Windows services, create folders to save the realm data and the logs. Create a TIBCO FTL configuration file and `tibdg` configuration scripts that can be later used to create the data grid.

Procedure

1. Create the following directories:

```
mkdir C:\activespacesdata
mkdir C:\activespacesdata\_default
mkdir C:\activespacesdata\logs
mkdir C:\activespacesdata\realm_data
```

   The logs from the TIBCO FTL servers and ActiveSpaces processes are stored in the `logs` directory. The realm data from the TIBCO FTL servers are stored in the `realm_data` directory. The ActiveSpaces grid data for a data grid named `_default` is stored in the `_default` directory.

2. Create a TIBCO FTL configuration file with the following content and save it to `C:\activespacesdata\ftl.yaml`.

Note: TIBCO recommends that you run a separate TIBCO FTL server on each computer.

```
globals:
  core.servers:
    ftl1: localhost:8085
    ftl2: localhost:8185
    ftl3: localhost:8285
servers:
  ftl1:
  - realm:
      data: C:\activespacesdata\realm_data
      logfile: C:\activespacesdata\logs\ftl1-rs-log.txt
  ftl2:
  - realm:
      data: C:\activespacesdata\realm_data
      logfile: C:\activespacesdata\logs\ftl2-rs-log.txt
  ftl3:
  - realm:
      data: C:\activespacesdata\realm_data
      logfile: C:\activespacesdata\logs\ftl3-rs-log.txt
services:
  realm: {}
```

3. Create a `tibdg` configuration script with the following content and save it to `C:\activespacesdata_default.tibdg`.

```
grid create copyset_size=2
copyset create cs_01
copyset create cs_02
node create --copyset cs_01 --dir C:/activespacesdata/_default/cs_
01.n_1_data cs_01.n_1
node create --copyset cs_01 --dir C:/activespacesdata/_default/cs_
01.n_2_data cs_01.n_2
node create --copyset cs_02 --dir C:/activespacesdata/_default/cs_
02.n_1_data cs_02.n_1
node create --copyset cs_02 --dir C:/activespacesdata/_default/cs_
02.n_2_data cs_02.n_2
keeper create --dir C:/activespacesdata/_default/k_1_data k_1
keeper create --dir C:/activespacesdata/_default/k_2_data k_2
keeper create --dir C:/activespacesdata/_default/k_3_data k_3
proxy create p_01
```

```
proxy create p_02
table create t1 key long
column create t1 value string
```

This configuration file is used later to create the data grid.

Installing TIBCO FTL Server as a Windows Service

Before you begin

Complete the steps mentioned in Preparing for Installation.

Procedure

1. Based on the YAML file created in Preparing for Installation, install three FTL servers, each pointing to `ftl.yaml` configuration file and each having a unique name.

```
TIBCO_HOME\ftl\<ftl_version>\bin\prunsrv.exe //IS/tibftlserver1 --
DisplayName="TIBCO FTL Server 1"
--Install=TIBCO_HOME\ftl\<ftl_version>\bin\prunsrv.exe
--StartMode=exe --StartImage=TIBCO_HOME\ftl\<ftl_
version>\bin\tibftlserver.exe
--LibraryPath=TIBCO_HOME\ftl\<ftl_version>\bin --StartParams=-n;ftl1;-c;
C:\activespacesdata\ftl.yaml --StopMode=exe --StopTimeout=30
--StopImage=TIBCO_HOME\ftl\<ftl_version>\bin\tibftladmin.exe --
StopParams=--ftlserver;
http://localhost:8085;-x
```

```
TIBCO_HOME\ftl\<ftl_version>\bin\prunsrv.exe //IS/tibftlserver2 --
DisplayName="TIBCO FTL Server 2"
--Install=TIBCO_HOME\ftl\<ftl_version>\bin\prunsrv.exe
--StartMode=exe --StartImage=TIBCO_HOME\ftl\<ftl_
version>\bin\tibftlserver.exe
--LibraryPath=TIBCO_HOME\ftl\<ftl_version>\bin --StartParams=-n;ftl2;-c;
C:\activespacesdata\ftl.yaml --StopMode=exe --StopTimeout=30
--StopImage=TIBCO_HOME\ftl\<ftl_version>\bin\tibftladmin.exe --
StopParams=--ftlserver;
http://localhost:8185;-x
```

```
TIBCO_HOME\ftl\<ftl_version>\bin\prunsrv.exe //IS/tibftlserver3 --
DisplayName="TIBCO FTL Server 3"
--Install=TIBCO_HOME\ftl\<ftl_version>\bin\prunsrv.exe
--StartMode=exe --StartImage=TIBCO_HOME\ftl\<ftl_
version>\bin\tibftlserver.exe
--LibraryPath=TIBCO_HOME\ftl\<ftl_version>\bin --StartParams=-n;ftl3;-c;
C:\activespacesdata\ftl.yaml --StopMode=exe --StopTimeout=30
--StopImage=TIBCO_HOME\ftl\<ftl_version>\bin\tibftladmin.exe --
StopParams=--ftlserver;
http://localhost:8285;-x
```

Tip: If you plan to copy the code snippet, remember to remove line breaks for all the sample commands that are associated with `prunsrv.exe`.

In this example, a separate port was specified for each of the servers (8085, 8185, 8285) to allow all three TIBCO FTL servers to run on the same computer. In most cases, the same port can be used if the TIBCO FTL servers were being run on three different computers. The ActiveSpaces start parameters for each process must use a pipe (“|”) separated list of URLs so that it can communicate with any of the TIBCO FTL servers. In this case, the pipe-separated list would be `http://localhost:8085|http://localhost:8185|http://localhost:8285`. If different ports are chosen in the `ftl.yaml` config file, those start parameters must also be updated for the ActiveSpaces processes.

Creating the ActiveSpaces Data Grid

Before you begin

1. Complete the steps listed in Preparing for Installation.
2. Complete the steps listed in Installing TIBCO FTL Server as a Windows Service.
3. On Microsoft Windows, open the Services window and manually start each of the TIBCO FTL servers. Ensure that the TIBCO FTL servers are running. Verify by looking at the logs that are written to `C:\activespacesdata\logs` and the realm data that is written to `C:\activespacesdata\realm_data`.

Procedure

1. Create the example data grid by using the `tibdg.exe` administration tool, which

creates the data grid definition in the realm. The data grid definition saved from Preparing for Installation is found in `C:\activespacesdata_default.tibdg`.

```
TIBCO_HOME\as\<as_version>\bin\tibdg -r
"http://localhost:8085|http://localhost:8185|http://localhost:8285"
-s C:\activespacesdata\_default.tibdg
```

2. After creating the data grid, check the status by using the `tibdg.exe` administration tool.

```
TIBCO_HOME\as\<as_version>\bin\tibdg -r
"http://localhost:8085|http://localhost:8185|http://localhost:8285"
status
```

The output shows the data grid and the processes that are not yet running.

Data Grid Status

```
Grid is not functioning.
FTL healthy.  Up for 74 seconds.
Admin server is available at http://localhost:7171

PROCESSES

     TYPE       NAME         HOST  PID  REV  TXNS  REQS  COPYSET  ROLE  EST SIZE  FS USED  FS CAP  DATA DIR  MAX WRITE
     node   cs_01.n_1  NOT RUNNING

     node   cs_01.n_2  NOT RUNNING

     node   cs_02.n_1  NOT RUNNING

     node   cs_02.n_2  NOT RUNNING


       TYPE  NAME         HOST  PID  REV  ROLE  STATE DIR
     keeper   k_1  NOT RUNNING
     keeper   k_2  NOT RUNNING
     keeper   k_3  NOT RUNNING


      TYPE  NAME         HOST  PID  REV  CLIENTS  REQ  TXN  ITER  STMT  QRY  LSNR  MODE
     proxy  p_01  NOT RUNNING
     proxy  p_02  NOT RUNNING
```

Installing the ActiveSpaces State Keeper as a Windows Service

Before you begin

Perform the following tasks:

1. Before using ActiveSpaces processes as Windows services, ensure that both Java and the TIBCO FTL Windows package and the ActiveSpaces Windows package are installed on a local disk of the host computer (not on a mapped network drive).
2. Complete the steps listed in Preparing for Installation.
3. Complete the steps listed in Installing TIBCO FTL Server as a Windows Service.
4. Complete the steps listed in Creating the ActiveSpaces Data Grid.

Procedure

1. Use the following command to install three state keeper processes as Windows Services:

```
TIBCO_HOME\as\<as_version>\bin\prunsrv.exe //IS/tibdgkeeper_default_k_1
--DisplayName="TIBCO ActiveSpaces Statekeeper k_1 (_default)" --
Install=TIBCO_HOME\as\<as_version>\bin\prunsrv.exe
--StartMode=exe --StartImage=TIBCO_HOME\as\<as_
version>\bin\tibdgkeeper.exe
--LibraryPath=TIBCO_HOME\as\<as_version>\bin;TIBCO_HOME\ftl\<ftl_version>\bin
--StartParams=-
r;"http://localhost:8085|http://localhost:8185|http://localhost:828
5";-n;k_1;--logfile;C:\activespacesdata\logs\_default-k_1-log.txt
--StopMode=exe --StopTimeout=30 --StopImage=TIBCO_HOME\as\<as_
version>\bin\tibdg.exe
--StopParams=-
r;"http://localhost:8085|http://localhost:8185|http://localhost:828
5";keeper;stop;
k_1
```

```
TIBCO_HOME\as\<as_version>\bin\prunsrv.exe //IS/tibdgkeeper_default_k_2
--DisplayName="TIBCO ActiveSpaces Statekeeper k_2 (_default)" --
Install=TIBCO_HOME\as\<as_version>\bin\prunsrv.exe
```

```
--StartMode=exe --StartImage=TIBCO_HOME\as\<as_
version>\bin\tibdgkeeper.exe
--LibraryPath=TIBCO_HOME\as\<as_version>\bin;TIBCO_HOME\ftl\<ftl_version>\bin
--StartParams=-
r;"http://localhost:8085|http://localhost:8185|http://localhost:828
5";-n;k_2;--logfile;C:\activespacesdata\logs\_default-k_2-log.txt
--StopMode=exe --StopTimeout=30 --StopImage=TIBCO_HOME\as\<as_
version>\bin\tibdg.exe
--StopParams=-
r;"http://localhost:8085|http://localhost:8185|http://localhost:828
5";keeper;
stop;k_2
```

```
TIBCO_HOME\as\<as_version>\bin\prunsrv.exe //IS/tibdgkeeper_default_k_3
--DisplayName="TIBCO ActiveSpaces Statekeeper k_3 (_default)" --
Install=TIBCO_HOME\as\<as_version>\bin\prunsrv.exe
--StartMode=exe --StartImage=TIBCO_HOME\as\<as_
version>\bin\tibdgkeeper.exe
--LibraryPath=TIBCO_HOME\as\<as_version>\bin;TIBCO_HOME\ftl\<ftl_version>\bin
--StartParams=-
r;"http://localhost:8085|http://localhost:8185|http://localhost:828
5";-n;k_3;--logfile;C:\activespacesdata\logs\_default-k_3-log.txt
--StopMode=exe --StopTimeout=30 --StopImage=TIBCO_HOME\as\<as_
version>\bin\tibdg.exe
--StopParams=-
r;"http://localhost:8085|http://localhost:8185|http://localhost:828
5";keeper;
stop;k_3
```

Tip: If you plan to copy the code snippet, remember to remove line breaks for all the sample commands that are associated with `prunsrv.exe`.

2. On Microsoft Windows, open the Windows Services panel and manually start each of the state keepers.

   The logs are written to `C:\activespacesdata\logs` and the process data is written to `C:\activespacesdata_default\process_name`.

Installing the ActiveSpaces Node as a Windows Service

Before you begin

Perform the following tasks:

1. Before using ActiveSpaces processes as Windows services, ensure that both Java and the TIBCO FTL Windows package and the ActiveSpaces Windows package are installed on a local disk of the host computer (not on a mapped network drive).
2. Complete the steps listed in Preparing for Installation.
3. Complete the steps listed in Installing TIBCO FTL Server as a Windows Service.
4. Complete the steps listed in Creating the ActiveSpaces Data Grid.

Procedure

1. Use the following commands to install the four nodes (two per copyset):

```
TIBCO_HOME\as\<as_version>\bin\prunsrv.exe //IS/tibdgnode_default_cs01_1
--DisplayName="TIBCO ActiveSpaces Node cs_01.n_1 (_default)" --
Install=TIBCO_HOME\as\<as_version>\bin\prunsrv.exe
--StartMode=exe --StartImage=TIBCO_HOME\as\<as_version>\bin\tibdgnode.exe
--LibraryPath=TIBCO_HOME\as\<as_version>\bin;TIBCO_HOME\ftl\<ftl_version>\bin
--StartParams=-
r;"http://localhost:8085|http://localhost:8185|http://localhost:828
5";
-n;cs_01.n_1;
--logfile;C:\activespacesdata\logs\_default-cs_01.n_1-log.txt --
StopMode=exe --StopTimeout=30
--StopImage=TIBCO_HOME\as\<as_version>\bin\tibdg.exe
--StopParams=-
r;"http://localhost:8085|http://localhost:8185|http://localhost:828
5";node;
stop;cs_01.n_1
```

```
TIBCO_HOME\as\<as_version>\bin\prunsrv.exe //IS/tibdgnode_default_cs01_2
--DisplayName="TIBCO ActiveSpaces Node cs_01.n_2 (_default)" --
Install=TIBCO_HOME\as\<as_version>\bin\prunsrv.exe
```

```
--StartMode=exe --StartImage=TIBCO_HOME\as\<as_version>\bin\tibdgnode.exe
--LibraryPath=TIBCO_HOME\as\<as_version>\bin;TIBCO_HOME\ftl\<ftl_version>\bin
--StartParams=-
r;"http://localhost:8085|http://localhost:8185|http://localhost:828
5";
-n;cs_01.n_2;
--logfile;C:\activespacesdata\logs\_default-cs_01.n_2-log.txt --
StopMode=exe --StopTimeout=30
--StopImage=TIBCO_HOME\as\<as_version>\bin\tibdg.exe
--StopParams=-
r;"http://localhost:8085|http://localhost:8185|http://localhost:828
5";node;stop;
cs_01.n_2
```

```
TIBCO_HOME\as\<as_version>\bin\prunsrv.exe //IS/tibdgnode_default_cs02_1
--DisplayName="TIBCO ActiveSpaces Node cs_02.n_1 (_default)" --
Install=TIBCO_HOME\as\<as_version>\bin\prunsrv.exe
--StartMode=exe --StartImage=TIBCO_HOME\as\<as_version>\bin\tibdgnode.exe
--LibraryPath=TIBCO_HOME\as\<as_version>\bin;TIBCO_HOME\ftl\<ftl_version>\bin
--StartParams=-
r;"http://localhost:8085|http://localhost:8185|http://localhost:828
5";
-n;cs_02.n_1;
--logfile;C:\activespacesdata\logs\_default-cs_02.n_1-log.txt --
StopMode=exe --StopTimeout=30
--StopImage=TIBCO_HOME\as\<as_version>\bin\tibdg.exe
--StopParams=-
r;"http://localhost:8085|http://localhost:8185|http://localhost:828
5";node;stop;
cs_02.n_1
```

```
TIBCO_HOME\as\<as_version>\bin\prunsrv.exe //IS/tibdgnode_default_cs02_2
--DisplayName="TIBCO ActiveSpaces Node cs_02.n_2 (_default)" --
Install=TIBCO_HOME\as\<as_version>\bin\prunsrv.exe
--StartMode=exe --StartImage=TIBCO_HOME\as\<as_version>\bin\tibdgnode.exe
--LibraryPath=TIBCO_HOME\as\<as_version>\bin;TIBCO_HOME\ftl\<ftl_version>\bin
--StartParams=-
r;"http://localhost:8085|http://localhost:8185|http://localhost:828
5";
-n;cs_02.n_2;
--logfile;C:\activespacesdata\logs\_default-cs_02.n_2-log.txt --
```

```
StopMode=exe --StopTimeout=30
--StopImage=TIBCO_HOME\as\<as_version>\bin\tibdg.exe
--StopParams=-
r;"http://localhost:8085|http://localhost:8185|http://localhost:828
5";node;stop;
cs_02.n_2
```

Tip: If you plan to copy the code snippet, remember to remove line breaks for all the sample commands that are associated with `prunsrv.exe`.

2. On Windows, open the Windows Services panel and manually start each of the nodes.

   The logs are written to `C:\activespacesdata\logs` and the process data is written to `C:\activespacesdata_default\process_name`.

Installing the ActiveSpaces Proxy as a Windows Service

Before you begin

Perform the following tasks:

1. Before using ActiveSpaces processes as Windows services, ensure that both Java and the TIBCO FTL Windows package and the ActiveSpaces Windows package are installed on a local disk of the host computer (not on a mapped network drive).
2. Complete the steps listed in Preparing for Installation.
3. Complete the steps listed in Installing TIBCO FTL Server as a Windows Service.
4. Complete the steps listed in Creating the ActiveSpaces Data Grid.

Procedure

1. To install both the proxies, use the following commands:

```
TIBCO_HOME\as\<as_version>\bin\prunsrv.exe //IS/tibdgproxy_default_p_01
```

```
--DisplayName="TIBCO ActiveSpaces Proxy p_01 (_default)" --
Install=TIBCO_HOME\as\<as_version>\bin\prunsrv.exe
--StartMode=exe --StartImage=TIBCO_HOME\as\<as_
version>\bin\tibdgproxy.exe
--LibraryPath=TIBCO_HOME\as\<as_version>\bin;TIBCO_HOME\ftl\<ftl_version>\bin
--StartParams=-
r;"http://localhost:8085|http://localhost:8185|http://localhost:828
5";-n;p_01;--logfile;C:\activespacesdata\logs\_default-p_01-log.txt
--StopMode=exe --StopTimeout=30 --StopImage=TIBCO_HOME\as\<as_
version>\bin\tibdg.exe
--StopParams=-
r;"http://localhost:8085|http://localhost:8185|http://localhost:828
5";proxy;stop;
p_01
```

```
TIBCO_HOME\as\<as_version>\bin\prunsrv.exe //IS/tibdgproxy_default_p_02
--DisplayName="TIBCO ActiveSpaces Proxy p_02 (_default)" --
Install=TIBCO_HOME\as\<as_version>\bin\prunsrv.exe
--StartMode=exe --StartImage=TIBCO_HOME\as\<as_
version>\bin\tibdgproxy.exe
--LibraryPath=TIBCO_HOME\as\<as_version>\bin;TIBCO_HOME\ftl\<ftl_version>\bin
--StartParams=-
r;"http://localhost:8085|http://localhost:8185|http://localhost:828
5";-n;p_02;--logfile;C:\activespacesdata\logs\_default-p_02-log.txt
--StopMode=exe --StopTimeout=30 --StopImage=TIBCO_HOME\as\<as_
version>\bin\tibdg.exe
--StopParams=-
r;"http://localhost:8085|http://localhost:8185|http://localhost:828
5";proxy;stop;
p_02
```

Tip: If you plan to copy the code snippet, remember to remove line breaks for all the sample commands that are associated with `prunsrv.exe`.

2. On Windows, open the Windows Services panel in Windows and manually start each of the proxies.

   The logs are written to `C:\activespacesdata\logs`.

Installing the ActiveSpaces tibdgadmind as a Windows Service

Before you begin

Perform the following tasks:

1. Before using ActiveSpaces processes as Windows services, ensure that both Java and the TIBCO FTL Windows package and the ActiveSpaces Windows package are installed on a local disk of the host computer (not on a mapped network drive).
2. Complete the steps listed in Preparing for Installation.
3. Complete the steps listed in Installing TIBCO FTL Server as a Windows Service.
4. Complete the steps listed in Creating the ActiveSpaces Data Grid.

Procedure

1. Use the following command to install two `tibdgadmind` processes:

```
TIBCO_HOME\as\<as_version>\bin\prunsrv.exe //IS/tibdgadmind1
--DisplayName="TIBCO ActiveSpaces tibdgadmind 1" --Install=TIBCO_
HOME\as\<as_version>\bin\prunsrv.exe
--StartMode=exe --StartImage=TIBCO_HOME\as\<as_
version>\bin\tibdgadmind.exe
--LibraryPath=TIBCO_HOME\as\<as_version>\bin;TIBCO_HOME\ftl\<ftl_version>\bin
--StartParams=-
r;"http://localhost:8085|http://localhost:8185|http://localhost:828
5";-l;localhost:7171 --StopMode=exe --StopTimeout=30
--StopImage=TIBCO_HOME\as\<as_version>\bin\tibdg.exe --StopParams=-
t;http://localhost:7171;admind;stop
```

```
TIBCO_HOME\as\<as_version>\bin\prunsrv.exe //IS/tibdgadmind2
--DisplayName="TIBCO ActiveSpaces tibdgadmind 2" --Install=TIBCO_
HOME\as\<as_version>\bin\prunsrv.exe
--StartMode=exe --StartImage=TIBCO_HOME\as\<as_
version>\bin\tibdgadmind.exe
--LibraryPath=TIBCO_HOME\as\<as_version>\bin;TIBCO_HOME\ftl\<ftl_version>\bin
--StartParams=-
```

```
r;"http://localhost:8085|http://localhost:8185|http://localhost:828
5";-l;localhost:7271 --StopMode=exe --StopTimeout=30
--StopImage=TIBCO_HOME\as\<as_version>\bin\tibdg.exe --StopParams=-
t;http://localhost:7271;admind;stop
```

Tip: If you plan to copy the code snippet, remember to remove line breaks for all the sample commands that are associated with `prunsrv.exe`.

2. On Windows, open the Windows Services panel to start each of the `tibdgadmind` processes manually.

Running an ActiveSpaces Sample

Procedure

1. To verify that the data grid is functioning and all the processes are running, run the following command:

```
tibdg status
```

2. Run the ActiveSpaces sample installed in *TIBCO_HOME*`\as\`*<as_version>*`\samples\bin` by passing the values from the URLs of the TIBCO FTL servers (`-r http://localhost:8085|http://localhost:8185|http://localhost:8285`) and the data grid name (`-g _default`).

Uninstalling the Sample Windows Services

Procedure

1. To uninstall the sample ActiveSpaces processes as Windows services, use the following command:

```
TIBCO_HOME\as\<as_version>\bin\prunsrv.exe //DS/tibftlserver1
TIBCO_HOME\as\<as_version>\bin\prunsrv.exe //DS/tibftlserver2
```

```
TIBCO_HOME\as\<as_version>\bin\prunsrv.exe //DS/tibftlserver3
TIBCO_HOME\as\<as_version>\bin\prunsrv.exe //DS/tibdgkeeper_default_k_1
TIBCO_HOME\as\<as_version>\bin\prunsrv.exe //DS/tibdgkeeper_default_k_2
TIBCO_HOME\as\<as_version>\bin\prunsrv.exe //DS/tibdgkeeper_default_k_3
TIBCO_HOME\as\<as_version>\bin\prunsrv.exe //DS/tibdgproxy_default_p_01
TIBCO_HOME\as\<as_version>\bin\prunsrv.exe //DS/tibdgproxy_default_p_02
TIBCO_HOME\as\<as_version>\bin\prunsrv.exe //DS/tibdgnode_default_cs01_1
TIBCO_HOME\as\<as_version>\bin\prunsrv.exe //DS/tibdgnode_default_cs01_2
TIBCO_HOME\as\<as_version>\bin\prunsrv.exe //DS/tibdgnode_default_cs02_1
TIBCO_HOME\as\<as_version>\bin\prunsrv.exe //DS/tibdgnode_default_cs02_2
TIBCO_HOME\as\<as_version>\bin\prunsrv.exe //DS/tibdgadmind1
TIBCO_HOME\as\<as_version>\bin\prunsrv.exe //DS/tibdgadmind2
```

Tip: If you plan to copy the code snippet, remember to remove line breaks for all the sample commands that are associated with `prunsrv.exe`.

Stopping a Data Grid Gracefully

To stop a data grid, stop all its component processes in this order.

See also, Sample Scripts.

Procedure

1. Optional. Back up the data grid definition to a file.
2. Stop all proxies.

   Stopping all proxies prevents clients from accessing the data grid. Open objects in client programs become invalid, and their methods generate exceptions.
3. Stop all nodes.
4. Stop all state keepers.
5. Optional. Stop the realm service.

Selecting a Secondary Node to be Promoted as the Primary Node

When administratively stopping a node, you can promote a specific secondary node as the primary node for the copyset. As a result, instead of waiting for a secondary node to time out the existing primary node and take over as the new primary node, you can select the new primary node from the existing set of secondary nodes. The `tibdg` tool provides an additional `-promote <node_to_promote>` option to let you select the secondary node.

Procedure

1. To promote a specific secondary node as a primary, use the following command:

```
tibdg -r <realm_url> node stop -promote <node_to_promote> <node_to_
stop>
```

Best Practices for Node Synchronization

In an ActiveSpaces grid with more than one node per copyset, each node has a full copy of the data for that copyset. When a primary node detects that a secondary node is not running, it updates the copyset information in the state keeper to indicate the secondary node is dead or out of sync and no longer expects to receive responses from that node while replicating write operations for that copyset.

When the secondary node is restarted, it goes through a background synchronization process. After synchronization, the secondary node is updated in the state keeper as an alive secondary, meaning it is eligible to take over if it detects that the primary node in that copyset is no longer running.

To avoid data loss, a secondary node that is not synchronized, never attempts to become the primary node in the copyset, even if the primary node is no longer running.

Based on that expected behavior, best practices when stopping and starting node processes such as during an upgrade or other maintenance are:

- After stopping a secondary node and then restarting it, an administrator must wait until the secondary node has completed its background synchronization process before stopping the primary node. In ActiveSpaces 4.6.0 and later, the `tibdg status` and `tibdg node status` commands include information to know if a secondary node is synchronized or not. Log files also include this information.
- An administrator must use the `tibdg node stop` command with the optional `-promote` argument to stop the existing primary node and promote a secondary node in its place. This option minimizes downtime for the secondary to detect that the primary is gone and performs extra validation to ensure the secondary node being promoted is in the synced state. The `tibdg node stop` command when used with the optional `-promote` argument fails when the secondary node to promote is dead or not synchronized.

For more information about the `-promote` option, see Selecting a Secondary Node to be Promoted as the Primary Node.

Timeouts During Maintenance

During grid maintenance, especially when stopping tibdgproxy or tibdgnode processes, an ActiveSpaces client application can experience timeouts for requests it has made to the

grid. The client application should be prepared to handle these timeout errors being generated in the application such as by logging or retrying the request.

For example, stopping a primary node causes requests to time out until the secondary node detects that the primary node is gone and takes over as the new primary node for that copyset. In addition, stopping a synchronized secondary node can cause timeouts until the primary node can successfully update the state keeper to indicate that the secondary node is out of sync.

When you restart the secondary node that is out of sync, a background synchronization process takes place. During the synchronization process, the ongoing live operations that are coming to the grid do not time out. Once the background synchronization of the secondary node is complete, the secondary node performs a small internal final step with a primary node.

If at all there are any operations that are timed out during this final step, the client application must handle such operations.

The `-promote` argument to the `tibdg node stop` command minimizes the amount of time that it takes to stop a primary node and promote a synchronized secondary node in its place.

For more information about the `-promote` option, see Selecting a Secondary Node to be Promoted as the Primary Node.

Clearing a Data Grid Definition

To delete a data grid definition, complete this task.

Procedure

1. Stop all proxies, nodes, and state keepers.
2. Run the `tibdg grid delete` command for the specific data grid that you want to delete.
3. Delete the data directories of the nodes and state keepers.
4. Optional. Create the data grid definition anew.

Checkpoints

ActiveSpaces provides the ability to create checkpoints to save the state and data in a data grid at a specific point in time. Checkpoint files can then be used to restore the data grid.

For details about the types of checkpoints, see "Checkpoints Types" in *TIBCO ActiveSpaces® - Enterprise Edition Concepts*.

Creating a checkpoint fails in the following scenarios:

- A realm service is not reachable.
- A quorum of state keepers is not running.

For more details, see "Checkpoints" in *TIBCO ActiveSpaces® - Enterprise Edition Concepts*.

Checkpoints on disk use the hard link feature of the file system to save space when multiple checkpoints refer to the same file on disk. Copying a checkpoint to a different file system causes the files in that checkpoint to occupy their full amount of space on the new file system.

Creating a Checkpoint

A checkpoint can be created manually by using the ActiveSpaces administration tool `tibdg`. You can also create periodic checkpoints by using ActiveSpaces.

Creating a Manual Checkpoint

Before you begin

Before creating a manual checkpoint, ensure that the following prerequisites are met:

- Ensure that the backup and satellite realm services are connected and are in sync with their regular servers. For example, satellite is in sync with primary, backup is in sync with satellite, and backup is in sync with primary. For more information about

the types of realm services, see the "Server Roles and Relationships" section in the *TIBCO FTL® Administration*.

- Ensure that a quorum of state keepers is running.
- Optionally, switch the data grid to maintenance mode. This ensures that no writes to the data grid occur when the checkpoint is taken. After the checkpoint is created, remember to take the data grid out of maintenance mode.

Procedure

1. Use the following command to create a manual checkpoint:

```
tibdg [-g <grid_name>] -r <realm_service_URL> checkpoint create
<checkpoint_name>
```

Creating a Periodic Checkpoint

The creation of periodic checkpoints and the number of checkpoints to retain is specified when you configure your data grid.

Before you begin

Before creating a checkpoint, ensure that the following prerequisites are met:

- Ensure that a quorum of state keepers is running.

The following data grid configuration options apply to periodic checkpoints:

checkpoint_interval

Use to specify the time interval (in seconds) for creating checkpoints. The default is 0, which means periodic checkpoints are not created.

checkpoint_retention_limit

Use to specify the number of checkpoints to retain. Older checkpoints are automatically removed as long as no active queries use the checkpoint.

Procedure

1. To enable the creation of a periodic checkpoint, use the `grid create` command as

is shown in the following example:

```
grid create copyset_size=2 statekeeper_count=3 checkpoint_
interval=600 checkpoint_retention_limit=5 mygrid
```

Listing Checkpoints

The `tibdg` administration tool can be used to see a list of the checkpoints that are taken for a data grid.

Procedure

1. Use the following command to view the list of checkpoints taken:

```
tibdg [-g <grid_name>] -r <realm_service_URL> checkpoint list
```

Result

The following information is displayed when you view a list of checkpoints:

Field	Description
ID	A unique identifier assigned to each checkpoint.
NAME	Checkpoint name. • *_periodic* is used as the name for all periodic checkpoints. • Manual checkpoints can be assigned a user-friendly name.
DIRECTORY	The name of the checkpoint subdirectory in each node's checkpoints directory.
TIMESTAMP	Timestamp of when the checkpoint was taken.
STATUS	The STATUS field has one of the following values: • **in progress** - When you are currently in the process of taking a checkpoint

- **success** - When the checkpoint was completed successfully.
- **failure** - When the errors occurred while taking the checkpoint
- **mirroring** - When a checkpoint is being received by the mirror grid. The primary grid displays the status as `success`.

Listing Tables in a Checkpoint

The `tibdg` administration tool can be used to see a list of the user tables that a checkpoint contains.

Procedure

1. Use the following command to see the list of tables in a checkpoint:

```
tibdg [-g <grid_name>] -r <realm_service_URL> checkpoint tables
[<checkpoint_name>]
```

Note: If a checkpoint name or ID is not specified, the tables in the last checkpoint taken are displayed.

The list of tables includes information about how the table is configured, including its primary key and secondary indexes.

Deleting Checkpoints

The ActiveSpaces administration tool `tibdg` provides the `checkpoint delete` command to delete a checkpoint by its name or to delete any checkpoint by its ID.

Procedure

1. Use the following command to delete a checkpoint:

```
tibdg [-g <grid_name>] -r <realm_service_URL> checkpoint delete
[<checkpoint_name> or <checkpoint_ID>]
```

Result

This removes the checkpoint from the list of checkpoints, and then periodically, checkpoint directories that are no longer on the list are removed from disk.

Automatically Deleting Old Checkpoints

The `checkpoint_retention_limit` configuration option can be used to ensure that the number of checkpoints does not keep growing and consuming disk space.

For more information about retention limits, see Retention Limits. As mentioned previously, the number of checkpoints to keep for your data grid can be controlled by using the grid configuration option:

- `checkpoint_retention_limit (Default: 0)`

The default `checkpoint_retention_limit` is 0, which means that older checkpoints are not removed.

Validating Checkpoints

There are some scenarios with Disaster Recovery where specifying the data grid to use as the primary grid in a gridset might fail due to checkpoints already taken for the data grid.

Procedure

1. The `tibdg` command can be used to validate the checkpoint a data grid uses before calling the command to specify the primary grid of a gridset:

   ```
   tibdg [-g <grid_name>] -r <realm_service_URL> checkpoint validate
   [<checkpoint_name_or_id>]
   ```

Note: If a checkpoint name or ID is not specified, the latest checkpoint taken is validated.

Checkpoint Properties

You cannot write data into a checkpoint after it has been created, but ActiveSpaces provides the ability to query or retrieve data contained in a checkpoint.

ActiveSpaces supports the following properties for checkpoints:

TIBDG_SESSION_PROPERTY_STRING_CHECKPOINT_NAME

The property is set so that a named checkpoint is used as the data source for the session's read operations (GET, Iterator, Queries, or Statements). Write operations (PUT, DELETE) do not support this property.

TIBDG_TABLE_PROPERTY_STRING_CHECKPOINT_NAME

Set this property so that a named checkpoint is used as the data source for the read operations (GET or Iterator) of a table. Write operations (PUT and DELETE) do not support this property. If this property is not set, the table uses the checkpoint name specified on the session, if any. If this property is set, it overrides any checkpoint name set on the session.

TIBDG_GRIDMETADATA_PROPERTY_STRING_CHECKPOINT_NAME

The property is set so that a named checkpoint is used when retrieving metadata for the data grid or tables.

TIBDG_STATEMENT_PROPERTY_STRING_CONSISTENCY

Set this property to `TIBDG_STATEMENT_CONSISTENCY_SNAPSHOT` when using checkpoints for reading data. Since a checkpoint is already globally consistent, you must not use global consistency when reading or querying with checkpoints.

See the API documentation on the following methods:

- `tibdgConnection_CreateSession`
- `tibdgConnection_GetGridMetadata`
- `tibdgSession_CreateStatement`

- `tibdgTable_CreateIterator`
- `tibdgTable_Get`

TIBDG_SESSION_CHECKPOINT_NAME_LATEST

Use this special checkpoint name with the `TIBDG_SESSION_PROPERTY_STRING_CHECKPOINT_NAME` property to refer to the latest successful checkpoint known by the proxy. This name is useful for clients that want to read from the latest checkpoint without knowing the checkpoint name, or for grids that are configured to create periodic checkpoints automatically.

TIBDG_TABLE_CHECKPOINT_NAME_LATEST

Use this special checkpoint name with the `TIBDG_TABLE_PROPERTY_STRING_CHECKPOINT_NAME` property to refer to the latest successful checkpoint known by the proxy. This property is useful for clients that want to read from the latest checkpoint without knowing the checkpoint name, or for grids that are configured to create periodic checkpoints automatically.

Checkpoint Best Practices

Creating a checkpoint is recommended after the following events:

- When a data grid is first brought up
- After any initial data is loaded into the data grid
- Before and after data grid configuration changes
- Before and after table configuration changes. For example, adding or deleting a table, index, or a column
- Before and after grid data redistribution
- When an abnormal event occurs in the data grid, such as a node goes down.

Additionally, you must determine the time interval to take periodic checkpoints such that the amount of data that can be lost between checkpoints satisfies your data recovery requirements.

Caching Rows in a Proxy

The proxy can cache rows from a checkpoint in memory to improve read performance. It is an optional process. If a proxy is enabled for checkpoint row caching, the output of the checkpoint `get` operation is obtained from the cache of the proxy. If the row is not present in the cache, the proxy obtains the row from the nodes and caches the result. When the proxy restarts, the cached information is lost. Query and iterator result rows are not cached.

For more information about how to specify a checkpoint name in the `get` operation, see Checkpoint Properties.

Note: For clients connecting to mirror grids, all the `get` operations operate on a checkpoint even if no checkpoint is specified in the properties of a client.

Configuration

By default, the cache size is 0 and the caching of checkpoint rows is disabled. To enable caching of checkpoint rows for all proxies in a grid or individual proxies, set the maximum size of the cache, in bytes, in the grid property `proxy_checkpoint_cache_size`. For example, `tibdg proxy modify p1 proxy_checkpoint_cache_size=100000000`. If the proxy property is set, it overrides the grid property, including disabling of caching by setting the property to 0 (`tibdg proxy modify p1 proxy_checkpoint_cache_size=0`). If the `proxy_checkpoint_cache_size` property is modified (either for the grid or proxy) while the proxy is running, the proxy must be restarted for the new value to take effect.

The `proxy_checkpoint_cache_size` property is a soft limit. The cache may exceed the limit if the cache is full and a new row is fetched from a checkpoint. When the cache is full, the cache uses the least recently used (LRU) algorithm to evict rows.

Client Usage

To refer to the latest successful checkpoint known by a proxy, clients can use the special checkpoint names:

- On a session object: `TIBDG_SESSION_CHECKPOINT_NAME_LATEST`

- On a table object: `TIBDG_TABLE_CHECKPOINT_NAME_LATEST`

This checkpoint name is useful for clients that want to read from the latest checkpoint without knowing the checkpoint name, or for grids that are configured to create periodic checkpoints automatically.

If a checkpoint is deleted, the cache of a proxy is not cleared immediately. However, rows from a deleted checkpoint are not obtained from the cache. The memory is reclaimed as the rows are removed from the cache.

At all times, the cache is populated with the rows that are explicitly requested by a client.

Client API Example

When the scope of the checkpoint name property is a table, a client can read from a checkpoint or live data by using the same session object. It is useful for reference tables that are updated infrequently and can be cached in the proxy for faster read performance, whereas other read operations are performed by using live data.

Here is an example API using C language:

Caution: Code snippets in the PDF can have undesired line breaks because of space constraints. Before directly copying and running them in your program, they must be verified.

```
tibdgSession session = tibdgConnection_CreateSession(ex, connection,
NULL);
// Session is created with empty properties - no checkpoint.

tibdgTable t1 = tibdgSession_OpenTable(ex, session, “t1”, NULL);
// tibdgTable t1 opened on live data

tibProperties props = tibProperties_Create(ex);

tibProperties_SetString(ex, props, TIBDG_TABLE_PROPERTY_STRING_
CHECKPOINT_NAME, “c1”);
tibdgTable c1_t1 = tibdgSession_OpenTable(ex, session, “t1”, props);
// tibdgTable c1_t1 opened on checkpoint c1

tibProperties_SetString(ex, props, TIBDG_TABLE_PROPERTY_STRING_
CHECKPOINT_NAME, TIBDG_TABLE_CHECKPOINT_NAME_LATEST);
tibdgTable cLatest_t1 = tibdgSession_OpenTable(ex, session, “t1”,
props);
// tibdgTable cLatest_t1 opened on the latest successful checkpoint
```

```
tibdgRow t1_key = tibdgRow_Create(ex, t1);
tibdgRow_SetLong(ex, t1_key, "key", 0);
tibdgRow t1_live_row = tibdgTable_Get(ex, t1, t1_key);
// row t1_live_row contains row from live data
// live data is not cached in the proxy

tibdgRow c1_t1_key = tibdgRow_Create(ex, c1_t1);
tibdgRow_SetLong(ex, c1_t1_key, "key", 0);
tibdgRow c1_t1_row = tibdgTable_Get(ex, c1_t1, c1_t1_key);
// row c1_t1_row contains row from checkpoint c1

tibdgRow cLatest_t1_key = tibdgRow_Create(ex, cLatest_t1);
tibdgRow_SetLong(ex, cLatest_t1_key, "key", 0);
tibdgRow cLatest_t1_row = tibdgTable_Get(ex, cLatest_t1, cLatest_t1_
key);
// row cLatest_t1_row contains row from the latest checkpoint known by
the proxy
```

Live Backup and Restore

ActiveSpaces live backup and restore is a feature that uses the concept of checkpoints to provide the ability to create a grid-wide consistent backup of a running data grid. A checkpoint is a set of persistent files containing the state and data from a single data grid at a specific point in time. A checkpoint can then be used to restore a complete data grid on the same computer, or to move the entire data grid to different computers.

The backup and restore procedure described in this document can only be used when a single data grid is running in a realm. No other processes can be configured in the realm or the restoring can be corrupted. The backup of the data grid is taken while the processes of the data grid are running. To restore the data grid processes from a backup, all processes of the data grid are first stopped and a full restore of the entire data grid is done from the backup.

To take a backup of an ActiveSpaces data grid, you must take a backup of the following processes:

- Realm Service
- ActiveSpaces State Keepers
- ActiveSpaces Nodes of each copyset

When a data grid has to be restored, you must ensure that the following components are restored:

- The realm service database
- The data grid configuration in the realm service
- The state keepers
- The nodes of each copyset

The data used to restore the data grid configuration, state keepers and nodes must be from the same backup. Before creating a backup of the data grid, you can optionally switch the data grid to maintenance mode to prevent writes from occurring when backing up the data grid. For more information, see Preventing Data Loss by Using the Maintenance Mode.

Backup Data Locations

Realm service

An ActiveSpaces data grid is run inside of a TIBCO FTL realm. A realm embraces all the administrative definitions and configurations that enable communication among the processes of the data grid and its clients. A realm service contains the complete realm definition. For more details, see "Processes in ActiveSpaces" section in the *TIBCO ActiveSpaces® Concepts* guide.

Each realm service has a set of working data files, which contain the configuration information about the FTL realm and ActiveSpaces data grid. These data files are stored in separate locations for each realm service. By default, when a realm service is started, it uses the current directory to store the data files. You can also specify the directory a realm service must use for its data files by passing the `--data` command-line option when starting the `tibrealmserver` executable. If you stop a realm service and then restart it, the realm service reads its configuration from previously existing data files.

Remember that in a realm, only the primary realm service can accept realm configuration updates. The primary realm service deploys its current realm definition to its satellite realm services. Satellite realm services cannot directly accept realm configuration updates from administrators or ActiveSpaces. For more information about the types of realm services, see the "Server Roles and Relationships" section in *TIBCO FTL® Administration*.

Note: Any primary server in the cluster can accept realm configuration updates.

Checkpoints and Realm Services

When an ActiveSpaces checkpoint is created, the primary realm service's database is backed up and the configuration of the realm service is also saved as part of the checkpoint. The copy is named with a timestamp reflecting the time at which the backup was created. The back-up file is created in the following directory of the primary realm service:

```
<realm_service_data_dir>/backups
```

The backup of the database and realm configuration can then be used to restore the

primary realm service. Creating a checkpoint fails if a realm service is not reachable. For example, a checkpoint is not created if primary and back-up realm services are down.

Note: The checkpoints taken as part of the data recovery or mirror grid feature cannot be used to restore as the interaction between the two features is not allowed and can lead to data loss.

State Keepers

ActiveSpaces state keepers store internal governing state information about your data grid. Each state keeper maintains a copy of this internal state information in a file on disk. When defining the data grid configuration, you specify the location of the state keeper files by using the `--dir` configuration option. By default your current directory is used to store the state keeper disk files. For example,

```
keeper create --dir ./k_0_data k_0
```

When a state keeper is first started, it receives the initial data grid configuration from the realm service. While the data grid is running, the state keepers record the current running state of the data grid. If you stop and restart a state keeper, the state keeper process uses the data files from its data directory to recover the data grid's running state.

In a fault-tolerant set of state keepers, one of the state keepers are designated the lead state keeper. If the leading state keeper goes down, one of the remaining state keepers takes over as the lead. A quorum of two state keepers, in a fault tolerant set of state keepers, must be running to ensure data consistency in split brain scenarios. If a state keeper is restarted while a quorum is running, one of the running state keepers updates the restarted state keeper's state.

Checkpoints and State Keepers

When an ActiveSpaces checkpoint is created, the state keeper's internal governing state information is also saved as part of the checkpoint. This checkpoint data file can then be used to restore a state keeper's state when the state keeper is restarted. Creating a checkpoint fails if a quorum of state keepers is not running.

Nodes

Each ActiveSpaces node stores rows of data for the tables that are defined for the data grid. The rows of data are stored in memory and on disk. When defining the data grid

configuration, you specify the location of the node files by using the `--dir` configuration option. By default your current directory is used to store the node's disk files. For example, the following statement indicates that the node stores its disk files by using a top-level directory, `./cs1_n1_data`.

```
node create --copyset cs1 --dir ./cs1_n1_data cs1_n1
```

The location where the node's disk files are stored is referred to as the node's data directory. Under the node's data directory, there are the following subdirectories:

- live - holds the disk files that contain the data stored on the node
- checkpoints - holds the checkpoint-related subdirectories and files

Checkpoints and Nodes

When an ActiveSpaces checkpoint is created, the relevant files needed to restore each node of a data grid are created and stored in the checkpoints subdirectory of each node's data directory. When a checkpoint is created, each running node saves its current state to the following directory:

```
<node_data_dir>/checkpoints/<timestamp>_<epoch>_<counter>_<checkpoint_
name>/d
ata
```

Additionally, the data grid's configuration from the primary realm service and the data grid's internal state from the state keepers are saved by each node of the first copyset defined in your data grid's configuration to the following directory:

```
<node_data_dir>/checkpoints/<timestamp>_<epoch>_<counter>_<checkpoint_
name>/
metadata
```

The checkpoint epoch is always zero unless there has been a disaster recovery failover to another data grid. The checkpoint counter is incremented with each checkpoint that is created. If your data grid is configured with a `copyset_size` greater than 1, the nodes of the first copyset defined for your data grid and identical copies of the metadata files that include `statekeeper-recovery` files.

Copysets

A copyset defines a relationship between multiple nodes for the purposes of data replication. If more than one node is defined for a copyset, one node acts as the primary node and data updates from client applications first occur on that node. The primary node then ensures that the data update is replicated on the other nodes in the copyset. If the primary node goes down for some reason, one of the other nodes in the copyset takes over as the primary node. Updates from client applications continue as usual without any loss of data because all of the data has been replicated from the original primary node to all of the other nodes in the copyset. The nodes of a copyset must reside on different computers to ensure that one computer failure does not cause data loss.

Checkpoints and Copysets

When an ActiveSpaces checkpoint is created, restoring a copyset is done by restoring the realm service configuration, state keeper configuration, and the data for each node of the copyset. There is nothing specific to restore for a copyset itself.

Restoring a Data Grid

To restore a data grid, the following entities must be restored:

- The primary realm service’s database
- The data grid configuration in the realm service
- The state keepers
- The nodes of each copyset

Procedure

1. Determine the ActiveSpaces checkpoint to use for restoring the data grid.
2. Determine the realm service database backup associated with the checkpoint.
3. Stop all data grid processes. For example, clients, proxies, nodes, state keepers, `tibdgadmind`.
4. Stop all realm services. For example, primary, backups, and satellite servers.
5. Copy each node's checkpoints directory to a safe place.

6. Restore the primary realm service's database from the backup associated with the checkpoint.
7. Restart any other realm services.
8. Restore the following processes:
   a. Restore the data grid configuration in the primary realm service from the ActiveSpaces checkpoint. This is to ensure that the data grid configuration is consistent as realm service database is backed up outside of our checkpoint process. Remember that the operations such as adding or deleting a table are not synchronized.
   b. Restore the state keepers from the checkpoint. Ensure that all state keepers are running.
   c. Restore each node from its respective checkpoint data directory. Ensure that all nodes are running.
9. Restart the following items:
   a. Restart any remaining data grid processes such as `tibdgadmind`, or proxies.
   b. Restart ActiveSpaces clients.

   If a restore to a checkpoint, which is not the latest is performed, the files are removed from each node's `checkpoints` subdirectory for the checkpoints taken after the checkpoint is being restored. Therefore, it is important to save each node's `checkpoints` subdirectory before the restore in case you decide you needed to restore from a later checkpoint.
10. After all grid processes have been restarted and verified to be operational, it is recommended but not required to delete the rollback record from the grid configuration to prevent complications during future maintenance operations.

Realm Service Database Restore

After a backup of the primary realm service is created, you can restore the realm service from the backup by using this procedure.

Procedure

1. Stop all the realm services that are running by stopping all TIBCO FTL servers.

2. For the TIBCO FTL Server being restored, ensure its data directory (as defined in the YAML configuration file) is created and empty.

3. From the backup that was previously created, find the appropriate config back-up file or files.

   The naming convention for a back-up file is `<filename>_<timestamp>.<extension>.backup`, where `<extension>` can be an appropriate extension such as '`dat`' or '`persist`'. Note that there may be more than one `.backup` file so be sure to find all such files with a matching <timestamp>.

4. Copy the `<filename>_<timestamp>.<extension>.backup` files into the empty data directory to restore from the backup. Rename the files to `<filename>.<extension>`. As an example, `config_SRV1_<timestamp>.persist.backup` would be renamed to `config_SRV1.persist`.

5. Restart the TIBCO FTL server, which loads the restored `<filename>.<extension>` file and uses the information in that file as the realm definition.

Realm Service Checkpoint Restore

When an ActiveSpaces checkpoint is created for a data grid, the data grid's configuration from the realm service is saved as part of that checkpoint.
The data grid's configuration from the realm service can be found in the following file on any node of the copyset that was first configured for your data grid:

```
<node_data_dir>/checkpoints/<timestamp>_<epoch>_<counter>_<checkpoint_
name>/metadata
/realmserver-grid.json
```

The ActiveSpaces administrative command-line tool `tibdg` is used to restore a data grid's configuration into a realm.

Before you begin

To restore your data grid configuration into a realm, you must ensure that you first complete the following steps:

1. Ensure that your realm services have been restored by using a back-up database associated with the checkpoint.

2. Copy the `realmserver-grid.json` file to a location that is easily accessible when

you run `tibdg`.

3. Load the `realmserver-grid.json` data grid configuration into the realm service.

Procedure

1. Use the following `tibdg` command is used to restore the data grid configuration from a checkpoint into the primary realm service:

```
tibdg [-g <grid_name>] -r <realm_service_URL> grid load -rollback
\<grid_config_json_file>
```

For example, `tibdg -r http://10.0.1.25:8080 grid load -rollback ./realmserver-grid.json`. Remember that in a realm only the primary realm service can accept realm configuration updates. The `<realm_service_URL>` must be the URL of the primary realm service.

Note: This method must not be used if more than one data grid is configured in the realm service or other FTL applications are also configured in the realm.

Restoring State Keepers

For more information about where to locate the nodes that contain a checkpoint's metadata subdirectory and the state keeper recovery file, see the section, Checkpoints. The state keeper recovery file is named `statekeeper-recovery`.

Before you begin

When restoring your state keepers, ensure that the same checkpoint recovery file is used when restarting each state keeper. To restore your state keepers from a checkpoint recovery file, ensure that you complete the following steps.

1. Ensure that the realm services have been restored as described in the section Realm Service Checkpoint Restore.

2. Ensure that all state keepers are stopped.

Procedure

1. Locate the `statekeeper-recovery` file associated with the checkpoint that you want to restore the state keeper from. Note its path.

2. Restart all state keepers by using the same version of the recovery file. Use the --recovery-file or -R command-line options.

   ```
   tibdgkeeper -r <realm service URL> -n <state keeper name> --
   recovery-file <path to statekeeper-recovery file>
   ```

   The same command can be used to restore a single state keeper or each state keeper of a fault tolerant set of state keepers.

Restoring a tibdg Node

When restoring the nodes of a data grid, all nodes must be restored by using the same checkpoint to ensure a consistent state between the primary and secondary nodes of each copyset and between copysets.
When a checkpoint is created, each running node saves the files needed to restore the node to the following directory:

```
<node_data_dir>/checkpoints/<timestamp>_<epoch>_<counter>_<checkp
oint_name>/data
```

Procedure

1. Stop the node.

2. Move the node's current data directory to a back-up location.

3. Re-create the node's data directory by copying the appropriate checkpoint directories back to their original location under the `checkpoints` directory.

   ```
   <node_data_dir>/checkpoints/<timestamp>_<epoch>_<counter>_
   <checkpoint_name>
   ```

   The nodes read the rollback record in the realm and restore their live directory from the checkpoint directory specified in the rollback record. This ensures that all nodes are restoring the same checkpoint because they fail to start up if they cannot find the checkpoint directory specified by the rollback record.

4. Restart the node.

For example, suppose node cs1_n1 is started with the data directory ./cs1_n1_data. Then on UNIX you would do the following:

```
tibdg -r http://10.0.1.25:8080 node stop cs1_n1
mv ./cs1_n1_data cs1_n1_backup
mkdir -p cs1_n1_data/checkpoints
cp -R cs1_n1_backup/checkpoints/<timestamp>_00000000_00000001_chkpt
cs1_n1_data/checkpoints/.
tibdgnode -r http://10.0.1.25:8080 -n cs1_n1
```

Removing a Rollback Record

After all grid processes have been restarted and verified to be operational, it is recommended but not required to delete the rollback record from the grid configuration to prevent complications during future maintenance operations.

Leaving the rollback record in the grid indicates that a node completes recovery to that checkpoint on starting. This happens once. However, if a secondary node with a blank data directory is started without access to any checkpoints, an error such as the following may be observed:

```
[timestamp] seve node: Error during start up: File I/O error
[timestamp] seve node: Exception details:
TIBCO Exception:
Error Code = File I/O error
Description = rollback checkpoint ID <checkpoint_ID> not found
on this node
Thread Name = tibdgnode
Stack Trace:
       _performNodeRecoveryCheckpointRestore, 1834
       _performNodeRecovery, 1901
       _tibdgNode_Open, 3798
       main, 694
```

Before you begin

Ensure that the restore process is complete. That is, all grid keepers, nodes, and proxies are restarted following the restore operation.

Procedure

1. Run the 'tibdg rollback delete' command.

   ```
   tibdg -r <realm service URL> rollback delete
   ```

2. Ensure that future invocations of state keepers do not supply the `-R / --recovery-file` command-line argument. Any scripts that are used to start keepers automatically must be updated to remove the `-R / --recovery-file` argument if present.

Result

The `rollback delete` command must indicate that the rollback record is deleted.

What to do next

Whenever you start the state keeper again, ensure that you start it without the -R argument.

Disaster Recovery

Disaster Recovery is a situation where a set of running systems must be replaced by another set of running systems due to failure, damage, loss of connectivity, or other traumatic event. To set up disaster recovery, ActiveSpaces uses the concept of gridsets. A gridset is a group of data grids that share the same set of consistent data. In a disaster recovery setup, a gridset comprises a primary grid and at least one mirror grid.

Primary Grid

A data grid that is listed as the primary grid of a gridset is a primary grid. All operations included in the ActiveSpaces API are permitted on primary grids.

Mirror Grid

A data grid that is included in a gridset but is not currently the primary grid of that gridset is a mirror grid. The mirror grid is also referred to as a disaster recovery (DR) grid. Data received at a DR grid is a logically consistent checkpoint of the data from the primary grid (no partially committed transactions). For more information about checkpoints, see Checkpoints. Only read operations are allowed on mirror grids (for example, GET, queries, iterators). Read operations are run against the most recent checkpoint that has been mirrored from the primary grid.

For more information about gridsets and types of grids, see "Disaster Recovery Concepts" in *TIBCO ActiveSpaces® - Enterprise Edition Concepts*

Note: When a grid is added to a gridset, it cannot execute the Live Backup and Restore steps. With DR/mirroring, the grids in the gridset stay in sync with each other so the checkpoints taken by primary and mirror grids in a gridset do not include the required restore information needed for the Live Backup and Restore process.

Suggested Deployment Model for Disaster Recovery

To set up disaster recovery, a suggested model is to have a primary grid in one location and a mirror or disaster recovery (DR) grid in another location. This provides redundancy when the entire location hosting the primary grid experiences a disaster and requires failover to another location.

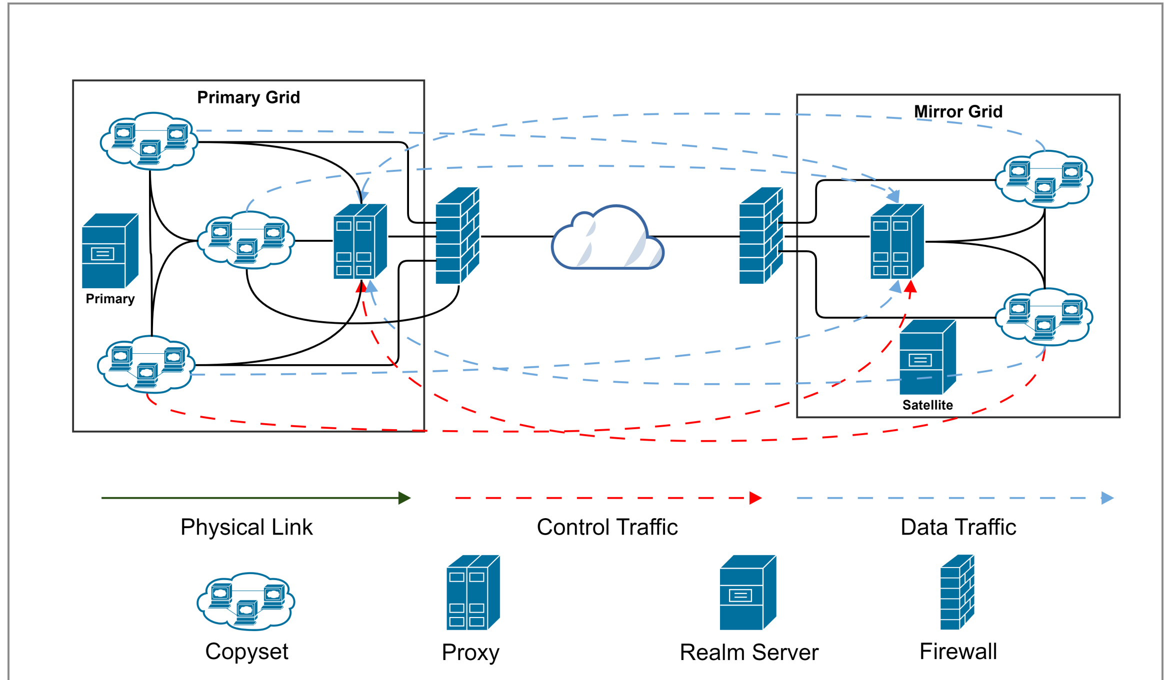
Grids in a gridset do not need to have the same number of copysets. They should be sized so that their capacity is sufficient to take over in the event of a disaster.

Both control and data traffic flow between each of the copyset nodes and the proxies configured in remote grids. All nodes in all copysets must be able to contact the proxies configured for DR in all other grids in the gridset.

The diagram shown later in this topic illustrates the deployment model. In this case, the primary data grid comprises a primary realm service and three copysets (each containing three replicas) behind a firewall. Data is then being mirrored across a WAN link (the cloud shape) to another location where there is another firewall and then a proxy. The mirror grid on the other location comprises a satellite realm service and a group of two copysets (each containing three replicas).

This demonstrates the ability of a mirror grid that has a different number of copysets from the primary grid. The dashed lines in the diagram also show the data flow where the control traffic and data traffic are sent to the proxy at the mirror grid, which is what you must configure with specific IP addresses and ports so that the proxy at the mirror grid is accessible to the primary grid. In addition, the proxy at the primary grid must be accessible to the mirror grid for traffic to flow in that direction as well. In the event of a disaster when the primary grid location becomes inaccessible, you can manually set the mirror grid to be the new primary grid as of the last consistent checkpoint that was mirrored to that location.

Figure 1: Deployment Model



A Quick Look at Setting Up Disaster Recovery

Consider the most common use case, where you create two data grids, create a gridset, and add them to a gridset in the right order.

For more information about gridsets, see "Gridsets" in *TIBCO ActiveSpaces® - Enterprise Edition Concepts*. The following sequence helps you set up a disaster recovery model.

#	Steps	References
1	Create a data grid with a name that is planned to be the primary grid (grid1).	Defining a Data Grid.
2	Create a data grid with a name that is planned to be a mirror/DR grid (grid2).	Defining a Data Grid.
3	Create at least one proxy in each data grid with the appropriate `proxy_mirroring_ static_listen_host` and	Create a proxy by following the steps in

#	Steps	References
	`proxy_mirroring_static_listen_port` values to set the IP address and port for how other data grids communicate with this data grid.	Defining a Data Grid. To configure static mirroring host and port, see Configuring a Proxy with Static Mirroring Host and Port.
4	Create a gridset by using the `tibdg` tool with the following command: `tibdg gridset create gridset1`	Creating a Gridset.
5	Add the first data grid to the gridset (the first data grid added becomes the primary grid) by using the `tibdg` command: `tibdg gridset add gridset1 grid1 proxy_static_ ip:proxy_static_port`	Adding Data Grids to a Gridset.
6	Add the second data grid to the gridset (which becomes a mirror grid) by using the `tibdg` command: . `tibdg gridset add gridset1 grid2 proxy_static_ ip:proxy_static_port`	Adding Data Grids to a Gridset.
7	Start the data grids and create a checkpoint in the primary grid so that it is mirrored to the mirror grid. Perform the following steps: • The primary grid must run a primary realm service • A mirror grid must run a satellite realm service, which is achieved by specifying the following configuration file option: `satelliteof <primary realm URL list>` For details about realm services, see "Realm Service" in *TIBCO FTL® Administration*. For details about satellite realm services, see "Server Roles and Relationships in *TIBCO FTL® Administration*.	To start the data grid processes, see Starting the Data Grid Processes. To create a checkpoint, see Creating a Checkpoint. To understand mirroring, see the following: • Automatic Mirroring. • Setting Up a

#	Steps	References
		Planned Cutover to a Mirror Grid. • Disaster Recovery at a Mirror Grid.
8	In the event of a disaster, make the mirror grid the new primary grid by using the `tibdg` command: `tibdg gridset setPrimary gridset1 grid2`.	See Activating the Mirror Grid as the Primary Grid.

Gridset Configuration

Configuring a gridset involves defining, creating a gridset that is followed by adding data grids to the gridset.

Getting Help on the gridset Command

Gridsets are managed by using the `tibdg` tool.

Procedure

1. For specific options and commands, run `tibdg gridset help`.

```
tibdg gridset help
```

Some of the available options are as follows:

```
    add                 Add a member grid to a gridset
    create              Create a gridset
    delete              Delete a gridset
    list                List all gridsets.
    modify              Modify an existing member grid in a gridset
    remove              Remove a member grid from a gridset
    setPrimary          Set the primary grid in a gridset
```

Creating a Gridset

Procedure

1. When creating a gridset, specify the name of the gridset as the parameter.

```
tibdg gridset create gridset1
```

Result

The gridset is created but it does not have any member grids at this point.

Adding Data Grids to a Gridset

The first data grid to be added to a gridset is made the primary grid. Data grids added after the first grid are mirror grids.

For more information about gridsets, see the "Gridsets" section of *TIBCO ActiveSpaces® - Enterprise Edition Concepts*. For more information about defining a data grid, see Defining a Data Grid.

Procedure

1. To add a data grid to a gridset, specify the gridset name, the data grid name, and a list of proxies that other data grids can use to contact the data grid. This is required for the primary as well as mirror grids.

```
tibdg gridset add gridset1 grid1 10.0.0.1:9001 10.0.0.2:9001
10.0.0.3:9001
```

Warning: Do not create tables on data grids that are intended to be mirror grids. With the `tibdg` tool, you cannot add mirror grids with configured tables in a gridset.

What to do next

After adding a data grid to the gridset, configure at least one proxy that listens on a static mirroring host and port. Some examples of a static mirroring host and port are `10.0.0.1:9001`, `10.0.0.2:9001`, or `10.0.0.3:9001`. The code snippet shown earlier in this topic on adding a data grid to the gridset also has examples of the list of IP addresses and ports. Ensure that other data grids in the gridset can communicate with the newly added data grid at these IP addresses and ports.

For more information about configuring a proxy with a static host and port, see Configuring a Proxy with Static Mirroring Host and Port.

Modifying a Gridset

You can change the list of static IP addresses to use for a specific data grid that is already in the gridset by using the `tibdg gridset modify` command.

Procedure

1. Use the following modify command to change the list of static IP addresses in a gridset.

   ```
   gridset modify gridset1 grid1 ip1:port,ip2:port,new_ip3:port
   ```

Permission Checking in Disaster Recovery Gridsets

To enable permissions in a mirror grid, ensure that the following criteria are met:

1. Enable transport encryption and permission checking for the grids in the gridset.
2. Define the same users and roles in all the grids in the gridset.

When the data is mirrored from the primary grid to the mirror grids, the permissions are also mirrored. Thus, all the grids in the set must have the same users and roles defined. In a production environment, such user and role issues are unlikely to be an issue because all the users and roles typically come from the same LDAP server. However, in a testing or development environment where you rely on file-based authentication, you must ensure that all the FTL servers have a consistent view of the users and roles that exist in the realm.

In addition, since the permissions for the table can only be set in the primary grid, users or roles that access the data only through a mirror grid must be granted access in the primary so that when the data is mirrored, the users can access it.

For more information about enabling permissions on a data grid, see Grid and Table Permissions.

Configuring a Proxy with Static Mirroring Host and Port

To set up disaster recovery, configure one proxy on the primary grid and one proxy on each mirror grid to use a static mirroring host and port. A static mirroring host and port uses an FTL static TCP transport to listen for mirroring operations.
To set static mirroring, configure the following properties:

- `proxy_mirroring_static_listen_port`
- `proxy_mirroring_static_listen_host`

Procedure

1. The following commands help you set up a static mirroring host and port on a primary and mirror grid.

```
tibdg -g grid1 proxy create dr_p1 proxy_mirroring_static_listen_
port=9001 proxy_mirroring_static_listen_host=10.0.0.1

tibdg -g grid2 proxy create dr_p2 proxy_mirroring_static_listen_
port=9001 proxy_mirroring_static_listen_host=10.0.0.2
```

Activating the Mirror Grid as the Primary Grid

Consider that there are two data grids:

grid1

Primary realm service running at 192.0.2.1:8080.

Backup for primary realm service running at 192.0.2.3:8080.

grid2

Satellite realm service running at 192.0.2.2:8080.

Backup for satellite realm service running at 192.0.2.4:8080.
Currently, the primary grid is grid1. The objective is to make grid2 the primary grid.

Note: grid1 and grid2 consist of a cluster of at least three realm services.

Settings to be Included in the YAML File

In grids configured for disaster recovery, the setting `disable.default.routing: true` must be included in the TIBCO FTL YAML file that is used for `tibftlserver` processes. In the YAML file, place the `disable.default.routing` field at the same level as `core.servers` as shown below:

```
globals:
    core.servers:
           SRV1: host:port
           SRV2: host:port
           SRV3: host:port
    disable.default.routing: true
```

Warning: Exercise caution when making the mirror grid as the primary grid because this might result in a data loss if the mirror grid is not in sync with the current primary.

Procedure

1. If you are taking periodic checkpoints on the primary grid, use the following commands to disable periodic checkpoint creation:

```
tibdg -r http://192.0.2.1:8080 -g grid1 status
tibdg -r http://192.0.2.1:8080 -g grid1 grid modify checkpoint_
interval=0.0
```

2. To switch the primary grid to maintenance mode and prevent new writes to the primary grid, use the following command:

```
tibdg -r http://192.0.2.1:8080 -g grid1 grid mode maintenance
```

3. To create a manual checkpoint on the primary grid, use the following command:

```
tibdg -r http://192.0.2.1:8080 -g grid1 checkpoint create
changePrimaryCheckpoint
```

4. On the mirror grid, to verify that the checkpoint has been successfully mirrored, use the following command:

```
tibdg -r http://192.0.2.2:8080 -g grid2 checkpoint list
```

   Repeat the command until you see the checkpoint listed in the command's output.

5. On the primary grid site, stop all grid processes and the tibftlserver processes (these are the primary realm services). On the mirror grid site, stop and restart the satellite realm service as the new primary realm service, which requires removing the `satellite-of` configuration option before restarting:

```
tibftladmin --ftlserver http://192.0.2.2:8080 --shutdown
tibftlserver -c <config file> -n <name>
```

   As a result of these commands, the ActiveSpaces mirror grid processes become the new primary grid and clients reconnect to the new primary realm service.

6. On the mirror grid site, use the following command to change the primary grid of the gridset:

```
tibdg -r http://192.0.2.2:8080 gridset setPrimary gridset1 grid2
```

   When prompted, confirm the change.
   As a result of these commands, the ActiveSpaces mirror grid processes become the new primary grid and clients that connect to this realm service are now connected to the new primary realm service.

7. On the previous primary grid site (where all processes must already have been stopped), back up and then remove all files in the ftlserver's `data` directory.

8. Restart the primary realm service as a satellite realm service, which requires adding the `satellite-of` configuration option to the configuration before restarting:

```
tibftladmin --ftlserver http://192.0.2.1:8080 --shutdown
```

```
tibftlserver -c <config file> -n <name>
```

If running multiple tibftlservers, all must have their `data` directory wiped and configured with the `satellite-of` configuration option prior to restarting.

9. After the satellite TIBCO FTL servers are running, start the grid processes, which must come up as the new mirror grid. To set the new mirror grid back to normal mode, use the following command:

```
tibdg -r http://192.0.2.2:8080 -g grid1 grid mode normal
```

Notice that `tibdg` configuration commands are being sent to the primary realm service for the new primary grid site at http://192.0.2.2:8080.

   a. If you must mirror periodic checkpoints from the new primary grid to the mirror grids, use the following commands:

   ```
   tibdg -r http://192.0.2.2:8080 -g grid2 grid modify
   checkpoint_interval=120
   ```

   b. On the new primary grid site, start a `tibdgadmind` process.

   ```
   tibdgadmind -r http://192.0.2.2:8080 -l 192.0.2.2:7171
   ```

10. Stop any existing clients that write to the data grid. Restart the clients so that they use one or more URLs of the new primary realm service.

Preventing Data Loss by Using the Maintenance Mode

The `tibdg grid mode` command can be used to put a data grid into maintenance mode, which prevents data from being written into your data grid.

See Using tibdg grid mode to Put a Data Grid into Maintenance Mode.

Retention Limits

The metadata required to identify the rows that have changed between checkpoints is stored in journals. These journals are cleaned up as checkpoints are copied to mirror grids.

Limiting the Size of Journals

A limit on the size of the journals can be specified when creating the data grid by using the `checkpoint_journal_max_bytes` setting. When the journal exceeds this size, ActiveSpaces begins deleting journal rows, starting with the oldest available. The journals may be disabled entirely by setting the `checkpoint_journals` option to `disabled`. If journals are disabled or if the maximum size is set too low, ActiveSpaces is forced to rely on bulk mirroring, which results in more network traffic between data grids. For more information about bulk mirroring, see "Bulk Mirroring" in *TIBCO ActiveSpaces® - Enterprise Edition Concepts*.

Limiting the Size of Checkpoints

You can also delete both manual and periodic checkpoints by setting the `checkpoint_retention_limit` property. By default, it is `0`, which means that the older checkpoints are not removed. This setting ensures that the number of checkpoints do not keep growing and consuming disk space.

Automatic Mirroring

Disaster recovery sites check for newly available checkpoints based on the `mirroring_interval` configuration option, set when the data grid is created.

The mirroring interval has no effect on a primary data grid. Setting a mirroring interval is only necessary at the mirror grid.

By default, the system checks for new checkpoints after every 30 seconds. This option can be changed after creating a data grid by using the `tibdg grid modify` command.

Note: Note: Mirroring can be disabled entirely by setting `mirroring_interval` to 0. This can later be set to a value greater than zero to restart the mirroring.

Recovery Objectives

To plan for failure scenarios, you must define how much data loss you can tolerate, and how long you can afford for the data grid to be down. Based on these data points, ActiveSpaces can be configured to provide various levels of service to satisfy your objectives.

Recovery Point Objective

To define a recovery point objective, determine how much data loss your system can sustain in the event of a disaster. This value can be used to determine how frequently checkpoints must be taken, and how frequently a mirror grid must request updates.

The frequency of checkpoints determines how often updates are available, and the frequency of mirror requests determines how quickly a mirror grid begins fetching new checkpoints once they are available. The available bandwidth between sites and the expected update rate must also be considered to determine how long it takes to mirror a checkpoint completely.

Consider the following example: User A can tolerate 1 hour of data loss, expects to write 400 MB/hour, and can transfer the data in about 6 minutes over a 10 Mbps WAN link. If the user takes a periodic checkpoint every hour, the mirror data is at least 66 minutes older than the primary, their recovery point objective is never satisfied. Therefore, they must take checkpoints more frequently to guarantee that they can meet their objective.

Recovery Time Objective

After an event occurs that requires changing to the disaster recovery site, the recovery time objective defines how much downtime a user can tolerate.

This objective guides the disaster recovery cutover process definition. ActiveSpaces disaster recovery cutover involves re-establishing communications with a primary TIBCO FTL realm service to establish communications for ActiveSpaces processes, and then changing one of the available and running DR grids to the new primary grid in the gridset. You must evaluate how this process, which includes the time to switch a satellite TIBCO FTL server manually to a primary TIBCO FTL server, which fits into their existing DR policy to determine how to meet their recovery time objective.

Capacity and Sizing

You must gauge the disk space used by the checkpoint metadata and have an estimate of the query capacity.

Disk Space Used by the Checkpoint Metadata

The amount of disk space used by the checkpoint metadata depends on the rows that are changed, and how often checkpoints are recorded.

For each checkpoint not yet transferred to the mirror grid, the checkpoint metadata has one row for each row changed. For example, if your application repeatedly overwrites the same 100 rows, the checkpoint metadata contains 100 rows for each checkpoint. The size of these rows is around 100 bytes plus the length of the row key. For information about calculating the size of the row key, see the "Sizing Guide" in *TIBCO ActiveSpaces® - Enterprise Edition Concepts*.

Query Capacity

Mirror grids must be provisioned appropriately for the load expected of them. A data grid that serves as both a read replica and a DR grid, must handle the total of both the normal load when being used as a read replica and the normal load of the primary grid, in case it is changed to the primary grid.

Security in a Disaster Recovery Setup

The existing security-related features in ActiveSpaces and TIBCO FTL are applied to the DR feature as well.

This is primarily transport level encryption by using the `encrypted_connections=all` property of a data grid, which must be set to the same value for all data grids in a gridset and which requires the use of a secure primary realm service and a secure satellite realm service.

For more information about enabling transport level encryption, see Enabling Transport Encryption on a Data Grid.

For more information about securing TIBCO FTL servers, see "Securing FTL Servers" in *TIBCO FTL® Administration*.

Disaster Recovery Playbook

All examples in the playbook use the following conventions:

- Realm Services are located at `http:://rs0:8080` for the primary realm service, and `http://rsn:8080` for any satellite realm services.
- The GridSet is named "Gridset".
- The Primary Grid is named "Primary".
- Mirror grids are named "DRn" where n is the number of the mirror grid and can take the value of 1, 2, 3, and so on.

Setting Up a Planned Cutover to a Mirror Grid

A cutover is a point in time when you transition from the primary grid to a mirror grid. A planned cutover is when the transition is planned and when the original primary grid does not fail completely.

For detailed steps on accomplishing a planned cutover, see Activating the Mirror Grid as the Primary Grid.

Procedure

1. Stop writes to the primary grid (maintenance mode can be used to achieve this).
2. Ensure that a final checkpoint is taken (this can be manual or periodic) after the writes have been completed at the primary grid.
3. Ensure that the final checkpoint has been successfully mirrored to the mirror grid with the following command:

   ```
   tibdg -r http://rs1:8080 -g DR1 checkpoint list
   ```

   This command lists the checkpoint ID, the name of the checkpoint, the location of the checkpoint, and the timestamp on the console window.

Tip: You must periodically check the output of the command to determine whether the checkpoint you took has been mirrored.

4. Stop the primary grid including the primary grid's realm service.
5. Select a new data grid that is going to be the primary grid.
6. Restart the realm service of the new primary grid as the "primary" realm service by omitting the `satelliteof` configuration option.
7. Restart the processes of the new data grid that becomes the primary grid.
8. Use the following command to activate that selected mirror grid as the new primary grid:

```
tibdg -r http://rs1:8080 gridset setPrimary Gridset DR1
```

9. Restart all mirror grids (including the old primary grid) by restarting their realm services as "satellite" realm services (specifying the `satelliteof` configuration option and referencing the new "primary" realm service).

Note: The old primary grid must have all processes stopped and the `tibftlserver` data directory must be backed up and then cleared prior to being restarted as a new satellite tibftlserver.

10. Restart the remaining mirror grid processes.

Disaster Recovery at a Mirror Grid

This procedure is used when the primary grid is unavailable and you want to run the cutover.

Procedure

1. If the realm service of the primary grid is down, stop the remaining satellite realm services.
2. Select a new data grid that is going to be the primary grid. Restart the realm service of the new primary grid as the "primary" realm service by omitting the `satelliteof` configuration option. If there are additional mirror grids, restart their realm services

as "satellite" realm services, by specifying the `satelliteof` configuration option and referencing the new "primary" realm service.

3. Use the following command to activate the mirror grid to be the new primary grid.

```
tibdg -r http://rs1:8080 gridset setPrimary Gridset DR1
```

Multiple Mirror Sites

When using multiple mirror sites, each mirror progresses independent of the others. When deciding which site to use as a new primary, determine which checkpoints are available at each site.

Procedure

1. List the checkpoints at each site.

   The checkpoint ID, checkpoint Name, location, and timestamp are displayed as follows:

```
tibdg -r http://rs1:8080 -g DR1 checkpoint list
ID     NAME      DIRECTORY          TIMESTAMP
….
<checkpoint ID>   (periodic)  <checkpoint location> 12:35:00
<checkpoint ID>   (periodic)  <checkpoint location> 12:40:00
tibdg -r http://rs2:8080 -g DR2 checkpoint list
```

```
ID     NAME      DIRECTORY          TIMESTAMP
….
<checkpoint ID>   (periodic)  <checkpoint location> 12:35:00
```

2. To minimize data loss, use the mirror with the most up-to-date checkpoint. When changing primary grids, checkpoints subsequent to the latest checkpoint at the new primary are not valid. Use the following command to set the primary grid:

```
tibdg -r http://rs1:8080 gridset setPrimary DR1
```

Read Replicas

A read replica is configured in the same way as a standard mirror. The distinction is primarily in how a read replica is used. Since a read replica is never used as a primary, it does not store any checkpoint metadata.

Procedure

1. Create your primary grid, and add it to the gridset.
2. Create your read replica, disabling checkpoint journals.

```
tibdg -r http://rs0:8080 grid create checkpoint_journals=disabled
readReplicaGrid
```

3. Add copysets, nodes, and state keepers to the new data grid.
4. Wait for checkpoints to be mirrored.
5. Run read-only operations against the new data grid.

TIBCO Documentation and Support Services

How to Access TIBCO Documentation

Documentation for TIBCO products is available on the Product Documentation website, mainly in HTML and PDF formats.

The Product Documentation website is updated frequently and is more current than any other documentation included with the product.

Product-Specific Documentation

The documentation for this product is available on the TIBCO ActiveSpaces® - Enterprise Edition Documentation page.

How to Contact Support for TIBCO Products

You can contact the Support team in the following ways:

- To access the Support Knowledge Base and getting personalized content about products you are interested in, visit our product Support website.
- To create a Support case, you must have a valid maintenance or support contract with a Cloud Software Group entity. You also need a username and password to log in to the product Support website. If you do not have a username, you can request one by clicking **Register** on the website.

How to Join TIBCO Community

TIBCO Community is the official channel for TIBCO customers, partners, and employee subject matter experts to share and access their collective experience. TIBCO Community offers access to Q&A forums, product wikis, and best practices. It also offers access to extensions, adapters, solution accelerators, and tools that extend and enable customers to gain full value from TIBCO products. In addition, users can submit and vote on feature requests from within the TIBCO Ideas Portal. For a free registration, go to TIBCO Community.

Legal and Third-Party Notices

SOME CLOUD SOFTWARE GROUP, INC. (“CLOUD SG”) SOFTWARE AND CLOUD SERVICES EMBED, BUNDLE, OR OTHERWISE INCLUDE OTHER SOFTWARE, INCLUDING OTHER CLOUD SG SOFTWARE (COLLECTIVELY, “INCLUDED SOFTWARE”). USE OF INCLUDED SOFTWARE IS SOLELY TO ENABLE THE FUNCTIONALITY (OR PROVIDE LIMITED ADD-ON FUNCTIONALITY) OF THE LICENSED CLOUD SG SOFTWARE AND/OR CLOUD SERVICES. THE INCLUDED SOFTWARE IS NOT LICENSED TO BE USED OR ACCESSED BY ANY OTHER CLOUD SG SOFTWARE AND/OR CLOUD SERVICES OR FOR ANY OTHER PURPOSE.

USE OF CLOUD SG SOFTWARE AND CLOUD SERVICES IS SUBJECT TO THE TERMS AND CONDITIONS OF AN AGREEMENT FOUND IN EITHER A SEPARATELY EXECUTED AGREEMENT, OR, IF THERE IS NO SUCH SEPARATE AGREEMENT, THE CLICKWRAP END USER AGREEMENT WHICH IS DISPLAYED WHEN ACCESSING, DOWNLOADING, OR INSTALLING THE SOFTWARE OR CLOUD SERVICES (AND WHICH IS DUPLICATED IN THE LICENSE FILE) OR IF THERE IS NO SUCH LICENSE AGREEMENT OR CLICKWRAP END USER AGREEMENT, THE LICENSE(S) LOCATED IN THE “LICENSE” FILE(S) OF THE SOFTWARE. USE OF THIS DOCUMENT IS SUBJECT TO THOSE SAME TERMS AND CONDITIONS, AND YOUR USE HEREOF SHALL CONSTITUTE ACCEPTANCE OF AND AN AGREEMENT TO BE BOUND BY THE SAME.

This document is subject to U.S. and international copyright laws and treaties. No part of this document may be reproduced in any form without the written authorization of Cloud Software Group, Inc.

TIBCO, the TIBCO logo, the TIBCO O logo, FTL, eFTL, and Rendezvous are either registered trademarks or trademarks of Cloud Software Group, Inc. in the United States and/or other countries.

All other product and company names and marks mentioned in this document are the property of their respective owners and are mentioned for identification purposes only. You acknowledge that all rights to these third party marks are the exclusive property of their respective owners. Please refer to Cloud SG’s Third Party Trademark Notices (https://www.cloud.com/legal) for more information.

This document includes fonts that are licensed under the SIL Open Font License, Version 1.1, which is available at: https://scripts.sil.org/OFL

Copyright (c) Paul D. Hunt, with Reserved Font Name Source Sans Pro and Source Code Pro.

Cloud SG software may be available on multiple operating systems. However, not all operating system platforms for a specific software version are released at the same time. See the “readme” file for the availability of a specific version of Cloud SG software on a specific operating system platform.

THIS DOCUMENT IS PROVIDED “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT.

THIS DOCUMENT COULD INCLUDE TECHNICAL INACCURACIES OR TYPOGRAPHICAL ERRORS. CHANGES ARE PERIODICALLY ADDED TO THE INFORMATION HEREIN; THESE CHANGES WILL BE INCORPORATED IN NEW EDITIONS OF THIS DOCUMENT. CLOUD SG MAY MAKE IMPROVEMENTS AND/OR CHANGES IN THE PRODUCT(S), THE PROGRAM(S), AND/OR THE SERVICES DESCRIBED IN THIS DOCUMENT AT ANY TIME WITHOUT NOTICE.

THE CONTENTS OF THIS DOCUMENT MAY BE MODIFIED AND/OR QUALIFIED, DIRECTLY OR INDIRECTLY, BY OTHER DOCUMENTATION WHICH ACCOMPANIES THIS SOFTWARE, INCLUDING BUT NOT LIMITED TO ANY RELEASE NOTES AND "README" FILES.

This and other products of Cloud SG may be covered by registered patents. For details, please refer to the Virtual Patent Marking document located at https://www.cloud.com/legal.

Copyright © 2009-2025. Cloud Software Group, Inc. All Rights Reserved.