TIBCO GridServer®

Administration Guide

Version 7.1.1 | July 2026

Copyright © 2001-2026. Cloud Software Group, Inc. All Rights Reserved.

Contents

Contents **2**

Typographical Conventions **9**

Introduction **10**

Before You Begin 10

The GridServer Administration Tool **11**

Overview 11

Getting Started 12

Navigating the Administration Tool 12

User Accounts and Role-Based Access Control **19**

Managing Users **20**

About Authentication and Authorization 20

Types of Users 20

Role-Based Access Control 21

Using GridServer Built-In Authentication 24

Using GridServer Built-In Authorization 24

Creating User Accounts 25

Resetting User Accounts From the Command Line 26

Using LDAP Authentication and Authorization 26

Configuring GridServer for LDAP Authentication or Authorization 27

Authentication Schemes Supported in Comparison Mode 29

Security Notes 31

LDAP Configuration Example 31

Using Windows Authentication and Authorization 32

Configuring Windows Authentication 33

Configuring Windows Authorization 34
Using Pure Kerberos Authentication 34
Configuring Pure Kerberos Authentication 35
Configuring Pure Kerberos Authorization 38
Managing Multiple Brokers with Grid Single Sign-On (SSO) 39
Grid SSO Configuration 39
Constraints and Limitations 40
Client Routing 40
Routing Clients With Roles 41
Routing Clients On The Broker Routing Page 41
Routing Clients With The Driver API 42

Managing Services 43
Deploying Services 43
About Grid Libraries 44
Using Grid Libraries from a Service 50
Super Grid Libraries 50
Deployment 51
Disabling Resource Deployment 51
Bridges 54
JREs 54
Packaging Grid Libraries 57
Distributing Grid Libraries 57
Grid Library Filters 61
JAR Ordering File 63
Uploading and Deploying with the Admin API 64
Running Services 64
Registering a Service Type 64
Service Run-As 65
compressData 65
encryptionEnabled 66
Using Run-As 67

Scheduling 68
Reschedules and Retries 69
The Scheduler 70
Common Scheduler Features 76

Managing Engines 82
Engine Routing and Balancing 82
Balancing and Service Discriminators 83
Engine Weight-Based Balancer 84
Home/Shared Balancer 84
Engine Balancer Configuration 86
Engine Upper and Lower Bounds 87
Failover Brokers 87
Example Use Cases 88
N+1 Failover with Weighting 88
Engine Localization with Sharing 88
Engine Configuration 89
Editing an Engine Configuration 89
Creating a New Engine Configuration 90
Copying an Engine Configuration 90
Setting the Engine Configuration Used by Engines 90
Setting the Director Used by Engines 91
Configuring Engines With Multiple Network Adapters 92
Configuring Engine Daemons to Use SNAT 93
Using the System Classloader on an Engine 93
Configuring a Global Shared Grid Library Directory 93
Configuring When Engines Run 94
Manual Mode 94
Auto Mode 95
Configuring How Many Engines Run 98
Running Engines in Multiplexed Mode 98
Communication and Task Scheduling 100

Configuration 100
Configuring 64-bit Engine Daemons to run 32-bit Services 101
Configuration 101
Specifying that a Service is win32 102
Routing 32-bit Tasks to 64-bit Engines 102
Configuring a Caching HTTP Proxy Server 102
Configuring an External Engine Daemon Admin Tool 104
Quarantine Brokers 104
Quarantine Broker Concepts 105
Quarantine Status on Engines 105
Requirements 106
Configuring a Quarantine Broker 106
Setting Quarantine Status on Engines 107
Quarantine Broker Constraints 108

Grid Fault-Tolerance and Failover 110
The Fault-tolerant GridServer Deployment 110
Heartbeats and Failure Detection 111
Manager Stability Features 111
Engine Failure 111
Driver Timeout and Failure 112
Director Failure 113
Broker Failure 113
Failover Brokers 114
Task Fault Tolerance 115
Batch Fault-Tolerance 116
GridCache and PDriver Fault-Tolerance 117

Administration and Maintenance 118
Configuration Issues 118
Installation on Machines With Multiple Network Adapters 118
Using UNC Paths in a driver.properties File 118

Renaming a Broker 119
Moving a Manager 119
GridServer Manager Administration Procedures 120
Backup / Restore 120
Importing and Exporting Manager Configuration 120
Setting the SMTP Host 122
Configuring the Timeout Period for the Administration Tool 122
Reconfiguring Managers when Installing a Secondary Director 123
Reconfiguring the Engine Communication Port 123
Promoting a Secondary Director to Primary Director 124
Configuring SNMP 124
LogLogic Integration 125
Configuration 126
Logging Message Format 127
Output 129
Elasticsearch Integration 130
Configuration 130
Message Format 134
Database Maintenance 134
Database Types 135
Internal Database Reset 136
Internal Database Backup 136
Performing Reporting Database Maintenance 137
The Batch Scheduling Facility 137
Terminology 138
Editing Batch Definitions 139
Batch Components 142
Service Runners 145
Scheduling Batch Definitions 146
The Batch Schedule Page 146
Running Batches 147
Deploying Batch Resources 147

Batch Fault-Tolerance 147

Optimizing the Grid 148

Diagnosing Performance Issues 148

Tuning Data Movement 148

Diagnosing GridServer Issues 154

Troubleshooting Overview 154

Reporting an Issue 154

Obtaining Log Files 154

Manager Logs 155

Engine and Daemon Logs 156

Application Server Logs 157

Monitoring the Tomcat Application Server 158

Monitoring Engines Using JMX 158

Diagnosing Network Issues 158

Diagnosing Engine Issues 161

Diagnosing Driver Issues 164

Diagnosing Manager Issues 165

Manager Port Issues 166

Out of Memory Issues 166

Deployment Issues 166

GridCache Issues 168

Database Issues 169

Troubleshooting Tools 171

Task Admin Page 171

Task Queue Dump 172

Enabling Enhanced Task Instrumentation 172

Process Explorer 173

Dependency Walker 173

Event Streaming by Using Apache Kafka 174

Configuring Event Streaming 174
Events Captured by Apache Kafka 175

Reporting Database Tables 178
Data Type Mapping 178
batches 178
brokers 179
broker_stats 179
driver_events 180
engine_events 181
engine_info 181
engine_stats 182
event_codes 183
jobs 183
job_status_codes 185
roles 186
tasks 186
task_status_codes 187
user_events 188
users 188

Scheduler Instrumentation Database Table 190
scheduler_info 190

Engine Instrumentation Database Table 192
engine_ins 192

TIBCO Documentation and Support Services 193

Legal and Third-Party Notices 195

Typographical Conventions

The following table lists the typographical conventions used in this guide:

Convention	Use
TIBCO_HOME	Many TIBCO products must be installed within the same home directory. This directory is referenced in the documentation as *TIBCO_HOME*. The default value of *TIBCO_HOME* depends on the operating system. For example, on Windows systems, the default value is `C:\tibco`.
DS_INSTALL	TIBCO GridServer® installs into a directory within *TIBCO_HOME* named datasynapse. This directory is referenced in the documentation as `DS_INSTALL`. The default value of *DS_INSTALL* depends on the operating system. For example, on Windows systems, the default installation directory is `C:\tibco\datasynapse`.
DS_MANAGER	The *Manager directory* contains the read-only software files; by default, it is a directory within *DS_INSTALL* named `manager`, and is referred to as *DS_MANAGER*. For example, on Windows systems, the default Manager directory is `C:\tibco\datasynapse\manager`.
DS_DATA	The *data directory* is the location of all volatile files used by the application server such as server properties and configuration. By default, it is a directory within *DS_INSTALL* named `manager-data`, and is referred to as *DS_DATA*. For example, on Windows systems, the default data directory is `C:\tibco\datasynapse\manager-data`.

Introduction

The *TIBCO GridServer® Administration* is for administrators who maintain DataSynapse GridServer installations. It describes how GridServer works and how to use the GridServer Administration Tool. Topics include how to schedule and route Services, deploy resources, manage failover Brokers, and perform other frequent tasks. This guide also provides advanced information about security, tuning, database administration, and log files.

Before You Begin

This guide assumes that you know GridServer concepts. If you do not, see the *TIBCO GridServer® Installation Guide* for information about the GridServer component architecture and principles of operation.

Before beginning, you must already have a DataSynapse GridServer Manager running and know the hostname, username, and password. If this isn't true, see the *TIBCO GridServer® InstallationGuide* or contact your administrator.

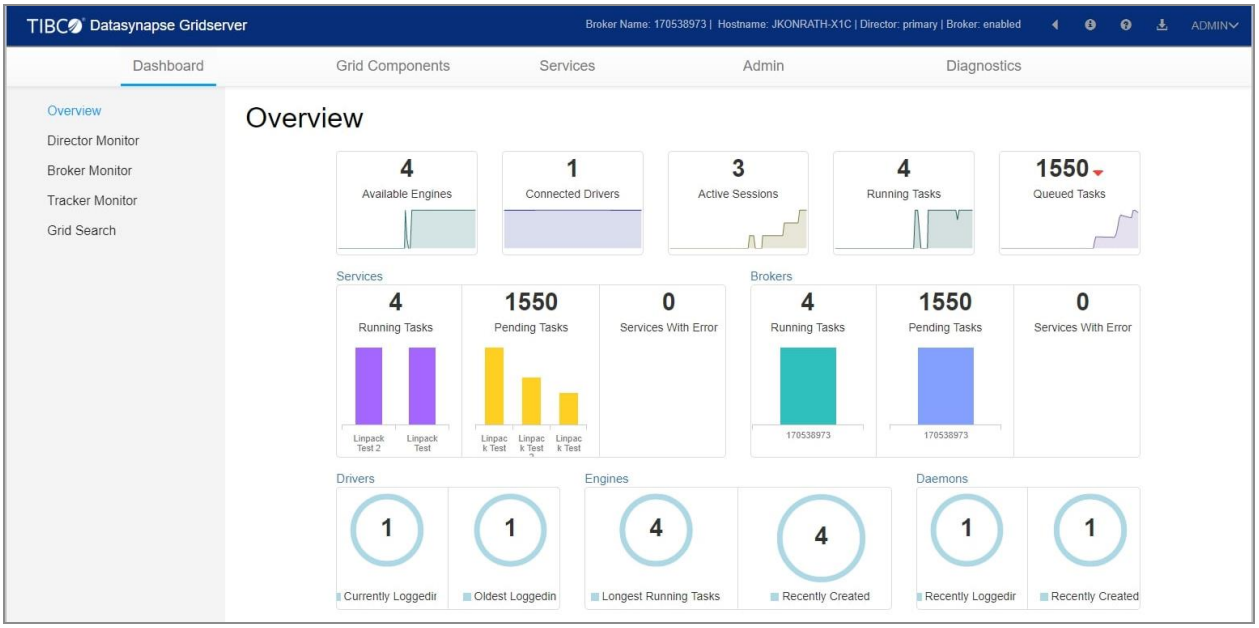
The GridServer Administration Tool

This section provides information about the GridServer Administration Tool, the web-based tool that enables the GridServer administrator to monitor and manage the Manager, its Engines, and Drivers.

Overview

The GridServer Administration Tool is a web-based tool that enables the GridServer administrator to monitor and manage the Manager, its Engines, and Drivers.

Authorized users can access the GridServer Administration Tool from any compatible browser, anywhere on the network. Administrative user accounts provide password-protected, role-based authorization.



The GridServer Administration Tool

In the GridServer Administration Tool, you can:

- Monitor Service and task execution and cancel Services
- Monitor Engine activity and restart Engines

- View and modify Manager and Engine configuration
- Download Engine installation archives
- Manage user accounts
- Subscribe to email notification of events
- Edit Engine properties and change values
- Configure routing of Drivers and Engines to Brokers
- View the GridServer API and download documentation
- Download the files necessary to develop and run GridServer applications
- Diagnose issues and monitor the grid

Getting Started

To use the GridServer Administration Tool, you must have access to the GridServe Manager from any supported browser that has JavaScript enabled.

In the browser, open `http://`*hostname*`:`*port* (where *hostname* is the address of the Manager, and *port* is the port on which it is listening, which is 8080 by default.); the Manager prompts you for a username and password.

Navigating the Administration Tool

The Administration Tool consists of a number of pages, organized in the following ways:

The Global Navigation Bar

The global navigation bar at the top of the Administration Tool contains links and information about your current login session.

The Global Navigation Bar

The following information is shown in the global navigation bar:

- The Broker name, which is a random number generated during installation.
- The Hostname of the Manager.
- If the Manager has a Primary Director, Secondary Director, or no Director.
- If the Manager has an enabled Broker, failover Broker, or no Broker.

The banner includes icons that open the following links:

- The **Information** panel, which displays component versions, built date, and applied updates.
- The **Help** panel, which displays context-sensitive help for this page, and other documentation links.
- The **Downloads** panel, which contains download links for SDKs and Engine installations.
- A **Profile** panel, which shows your current username, security roles, and has links to change your password or log out of the Administration Tool.

The Navigation Bar

The navigation bar contains categories, with each category having a submenu in the left sidebar. Each submenu has links to several pages, which are displayed in the main content area. Some submenus have subcategories at the top, which you can click to show different collections of page links.

The following categories are available:

Administration Tool Categories

Menu	Contents
Dashboard	All of the sub-pages of the Dashboard.
Grid Components	Pages used to manage, view, and configure Drivers, Brokers, and Engines.
Services	Pages used to manage, view, and submit Services and Batches.
Admin	Pages used for configuration, system and user administration, and reinstallation.

Menu	Contents
Diagnostics	Pages providing logs and information useful when diagnosing issues.

When describing navigation to a page, the following format is used: **Category > Submenu > Page**. For example, to view the **Engine Admin** page, click **Grid Components > Engines > Engine Admin**.

Tables and Table Controls

Most pages have items or information shown in tables. For example, the **Grid Components > Engines > Engine Admin** page displays a table with a row for each Engine in the grid. The following controls enable you to perform actions on the listed items, or change the items that are displayed.

Action Controls

Each table item has an action control (...), which opens a list of actions for the selected items in the table. Some of these perform actions on table items, while others open a new page. If no items are selected, this acts as a Global Action control, which opens a list of actions to perform on all items.

Search Control

The **Search** control is displayed on any page containing a table. There are two types of searches: a simple search, and a query search.

The Search Control

For the simple search, enter a string in the **Search** box, select a column to search from the list, and click the search icon. A new table appears, containing the matching rows.

Click the icon next to **Search** and select **Query** for a query search. This enables you to search properties using regular expressions consisting of, at minimum, a property, operator, and value. Terms can also contain wildcards and regular expressions, as defined by `java.util.regex.Pattern`. Click **Search** to run the query.

You can enter your own complex query by selecting **Query Builder**. If you first define a query in the **Query Builder** and then select **Query**, the query is displayed in the **Find** box.

Columns Control

The **Columns** control enables you to change the columns that are displayed in a table. When you click the Columns control, an overlay appears, listing all the columns currently in the table and columns that can be added.

To display a hidden column, click the **+** symbol next to the column in the **Add Columns** list. To hide a column, click the **X** symbol next to the column in the **Columns** list.

*The Columns overlay for the **Grid Components > Engines > Engine Admin** page*

When you add a column, it is added to the bottom of the list. The table displays the columns from left to right in the order of this list. To change the order of columns, click and drag the column name in the **Columns** list.

After you have made changes, click **Save** to apply them. To return to the default visible columns, click **Revert**.

The displayed table rows are always sorted by a column that has an arrow in it, either facing up or down. You can click this arrow to reverse the sort order of entries in the table,

or click another column to change the sort column. Table order is only kept for that page view and is not persisted.

Refresh

Pages in the GridServer Administration Tool are automatically refreshed every ten seconds by default to display the most current information available. Click the **Refresh** control to view the last time a page was refreshed, or to disable automatic refreshes. You can also customize the refresh rate by setting the `AJAX Refresh Interval` property on the **Admin > System Admin > Manager Configuration** page, in the Security section.

Pager Controls

The **Pager** controls, shown above and below tables, enable you to step through multiple pages of information, or specify how many rows appear on a page. Select a page number from the **Page** list, or select a range from the items list to display those items. You can select a greater number of items listed per page in a table or display all of the items; type a number in the **Results Per Page** box and click **Go**.

Exporting Table Data

Most information shown in Administration Tool tables can also be accessed programmatically using the GridServer Admin API. The Admin API can also be accessed with SOAP Web Services; the WSDLs are available from the **Grid Components > Drivers > Web Services** page.

For example, to generate a list of information about all logged in Engines as normally presented on the **Engine Admin** page, you can write a simple client application that connects to the Manager and uses the `getAllEngineInfo` method of the `EngineAdmin` Web Service.

For more information about using the Admin API, see the *TIBCO GridServer® Developer's Guide*.

The Dashboard

The dashboard provides current information about the status of your grid, summarizing statistics into a single at-a-glance overview. The **Dashboard > Overview** page is updated automatically every ten seconds.

Status tiles on the dashboard display an indicator of that components' current state. It also displays an arrow showing the trend of that statistic, and a small graph showing the last ten minutes of the changes to that statistic. Click any status tile to open a dialog that shows more detail.

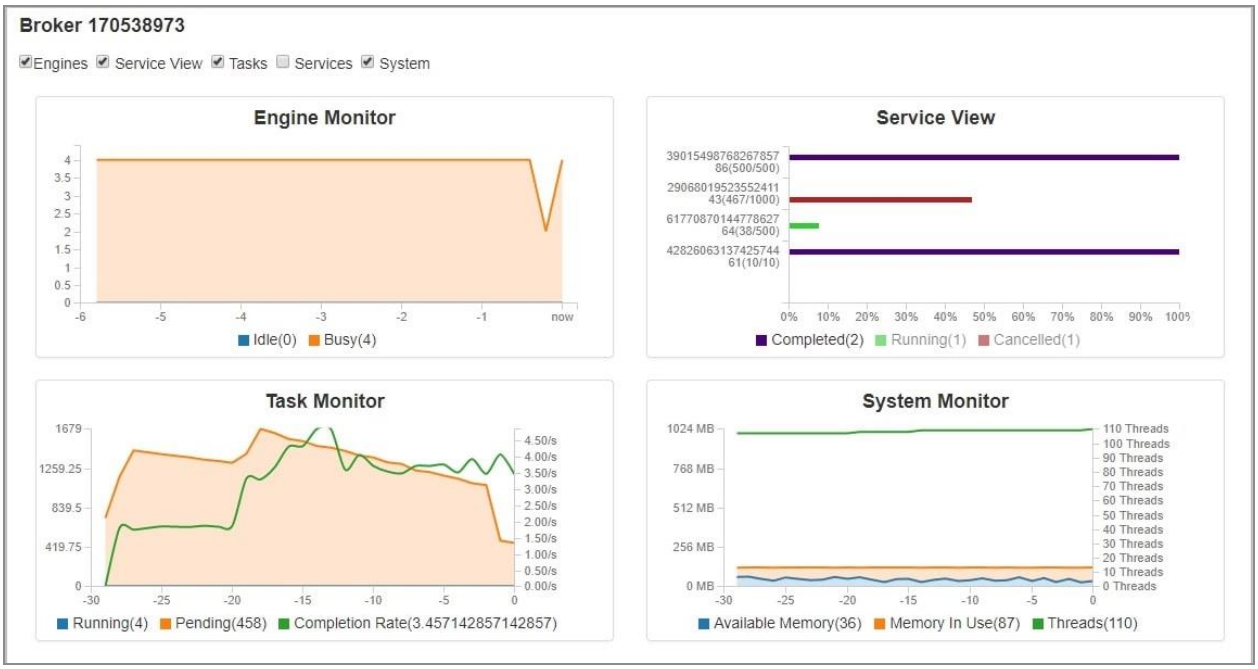
When you click the status tiles in the bottom row for grid components, the dialog also shows a small table with links to recently active components, and a link to that component's admin page. For example, if you click the Recently Created Engines status tile, it shows a table of the recently created Engines, and a link to the Engine Admin page.

The Broker and Director Monitors

The pages such as the Service Session Admin page and Engine Admin page can be used to oversee the running of Services on your grid. However, there are two graphical tools that can be used to provide a simpler overview of status information about your system. Both Directors and Brokers have a graphical monitor available, which can be displayed in its own window.

Display the Broker or Director Monitor by going to **Dashboard > Broker Monitor** or **Dashboard > Director Monitor**. Note that the Director Monitor is not available on standalone Brokers, and the Broker monitor is not available on standalone Directors.

Both monitor pages display up-to-date information about your grid. The Director Monitor contains graphs with statistics on Engines, tasks, Services and machine status, including thread and memory information. The Broker Monitor contains similar information about one specific Broker.



The Broker Monitor

User Accounts and Role-Based Access Control

To log in to the GridServer Administration Tool, you must have an account and password. GridServer supports a customizable system of role-based access control to provide account security and enable different users to access different areas of the interface.

User accounts are assigned one or more Security Roles. Each Security Role defines a set of permissions. A permission is the approval to use, see, or access a GridServer resource. There are four default Security Roles: **Configure**, **Manage**, **Service**, and **View**. The **Configure** role is for administrators and allows access to any part of the Administration Tool. By default, the root account you created at installation is set to the **Administer** role; you can also create accounts with full access for other administrative users. Users accounts assigned other Security Roles have more limited access; some pages and features either function differently, or are not available or visible.

You can create custom roles by going to the **Admin > User Admin > Role Admin** page and selecting **Create New Role** from the global actions menu. This enables you to create a new role, then add or remove what permissions are available to user accounts assigned that role. You can also use the editor to add or remove permissions from the predefined roles.

For more information about configuring user accounts and security roles, see About Authentication and Authorization.

Managing Users

GridServer user accounts enable you to specify which users are allowed to access different features within the Administration Tool, and control what resources can be used by different client software.

About Authentication and Authorization

Authentication is the process of determining if an entity is what it claims to be. GridServer provides a built-in authentication service and user repository, plus the ability to use an external LDAP server, Windows native authentication, or Kerberos.

Authorization is the process of determining what features a user has access to on a system. GridServer provides Role-Based Access Control (RBAC) to provide this, in which Users can be assigned roles manually, by using LDAP groups and by using Windows Domain groups.

Types of Users

GridServer users requiring authentication might be utilizing GridServer with client applications (Drivers), interactively using the web-based Administration Tool, or programmatically accessing the Admin interface. Each type is authenticated through the same mechanism.

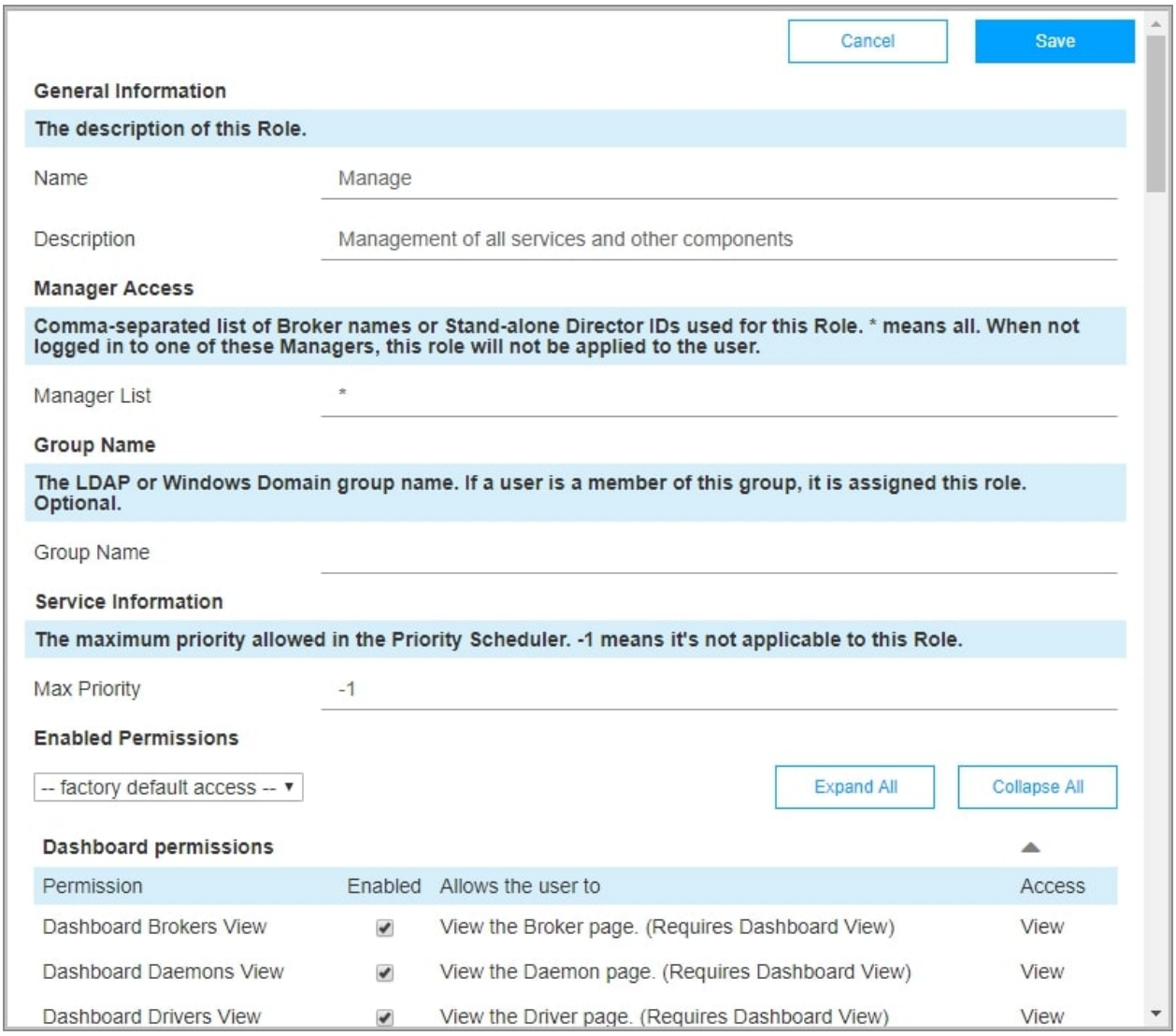
GridServer authenticates users according to the method defined by the administrator. After a grid user is authenticated, the user receives an authentication token to use in further correspondence. In the case of Administration Tool or Web Services users, the authentication token is a standard HTTP session cookie. In the case where computer users connect with the DataSynapse APIs, the authentication token is a DataSynapse object.

When a Driver attempts to log in, but has an authentication failure (incorrect user name or password), it stops trying to log in. If the failure is due to some other issue (such as an LDAP Server being temporarily down), it is not treated as an authentication failure. Any current or future service/cache methods throw an appropriate exception. The Driver process must be restarted at this point to alleviate the situation.

Operating system accounts are used to start GridServer software components, like the Manager, Engine, and Driver. It is generally not required to use a superuser operating system account to start any GridServer component.

Role-Based Access Control

Security Roles can be added and edited in the GridServer Administration Tool on the **Admin> User Admin > Role Admin** page. Each role contains a set of permissions that you can enable or disable. Each permission corresponds to a GridServer page, action, or feature. For example, you might want to disable permissions in a role to prevent a subset of users from editing Engine Daemons or managing Brokers. You can also enable permissions in a role, like enabling Service-role users to view the current Manager log. You can also use the **Role Admin** page to view what features are accessible in a given role.



Editing a Security Role

Security roles are assigned to a User via the **Admin > User Admin > User Admin** page. Also, when using LDAP or Kerberos authentication, they can be auto-assigned via LDAP groups, and when using Windows authentication, via the Windows Domain groups.

Editing Security Roles

To edit a Security Role:

1. Go to **Admin > User Admin > Role Admin**.
2. Select a role, click the **Actions** list, and select **View/Edit**.

3. Optionally, to create a new role, with no role selected, click the **Actions** list and select **Create a new Security Role**.

   The View/Edit Role page appears.

4. Select or clear the checkboxes next to the permissions you want enabled in this role. You can select another role from the **Copy** list, which selects that role's enabled features into the current role.

   You can also edit the following role attributes:

   — The name and description of the role.

   — A list of Managers on which this user can log in, or * for all Managers. If a user with this role attempts to log in to a Manager not listed, the login fails.

   — A corresponding LDAP group for the role. When an LDAP user from that group logs into GridServer, they receive this role.

   — The maximum priority a user with this role can assign to a Service.

5. When you are done editing the role, click **Save** to save changes, or **Cancel** to discard them.

The following actions are also available for each role:

- You can make a copy of a role with **Copy**.
- The **Delete** action completely removes a role. Note that you cannot delete a role if it is currently assigned to any user.

Security Role Notes

Service Session Admin methods or actions require the user to have Service Username Access to the Service in question. For example, the **Service Session Admin** page only shows a user's Services, and that user can only cancel their own Services.

Security Roles also affect the ability to use GridServer Web Services to interact with GridServer. For a list of GridServer Web Service objects and methods enabled by role, see the *GridServer Developer's Guide*.

Security Roles do not filter Services that are submitted before changing the associated Security Roles in an account. For example, a long-running Service is active and you change a user's account's Security Role association from Configure to View. In such a scenario, the user still has Configure-role access to that Service.

The Root Account Role

When a Manager is first installed, an initial account is created. This account, the *root account*, has the Root role assigned to it. The root account contains all permissions, similar to the Configure role. It is internal to the Manager, regardless of authentication mode. For example, when LDAP authentication is enabled and the LDAP server is unavailable, the root account is still available.

There are a number of restrictions related to the Root user role:

- The root account cannot be deleted.
- The root account's role cannot be changed. All other information in the root account (name, email, password, and so on) can be changed.
- You cannot add the Root role to any other account.

Using GridServer Built-In Authentication

GridServer's built-in authentication mechanism uses the embedded Director database (the internal database) to authenticate users. Passwords are stored in the database as secure hashes; when a password is entered, it is hashed and the two are compared.

GridServer built-in authentication includes options for minimum user name length, minimum password length, password complexity, password aging, and application behavior on password failure. To configure password authentication, go to **Admin > System Admin > Manager Configuration > Security** and change the settings under the **Admin User Management** heading.

Each user account is mapped to one or more Security Roles, which dictate what features of the Administration Tool they can use.

Using GridServer Built-In Authorization

GridServer's built-in role-based access control can be used to define what features a user can access on the system. Depending on the authentication type used, there are different methods for assigning roles to users.

If you are using GridServer's built-in authentication, user authorization is mapped in the user account stored within GridServer. When you create or edit an account on the **Admin >**

User Admin > User Admin page, the **Security Roles** section of the Edit User dialog enables you to select one or more roles that are given to that user's account.

You can also use LDAP or Windows authorization, and edit roles so they map to users based on their group name. When you edit a role on the **Admin > User Admin > Role Admin** page, set the LDAP Group Name to the name of the group that receives that role.

If you are using Pure Kerberos or LDAP authentication, you can either use built-in authorization or LDAP authorization. If you use built-in authorization, this means you must create accounts for each of your users using GridServer's built-in authentication, and specify one or more roles for each user. Although you need to set passwords when you create the user, they are ignored and Kerberos is used for authentication.

Creating User Accounts

To create a User Account:

1. Log in to the Administration Tool using the Configure-role account created when you first installed GridServer, or any other account with access to the User Manage feature.
2. Go to **Admin > User Admin > User Admin**.
3. Select **Create New User** from the **Actions** list. The New User Information page appears.
4. Enter the username, a password, and confirm the password. Credentials entered here are case sensitive. If you also use Active Directory which is case-insensitive, matching credential syntax is recommended.
5. If using built-in authorization, in the **Security Roles** list, assign one or more roles by selecting the role name in the left list, then clicking the **>>** button. If multiple roles are selected, the account can access features specified in all roles.
6. You can also optionally enter a first and last name, and an E-mail address for notifications.
7. When finished entering the user information, click **Save**.

GridServer sends a notification E-mail to the address provided in the new account, provided that an SMTP host is set in the Manager's configuration. You can customize the templates used for the subject and body of these messages by selecting **Edit Email Notification Template** from the **Actions** list on the **User Admin** page.

Resetting User Accounts From the Command Line

In addition to editing a user account's password in the Administration Tool, you can also change the password of a user account in the local database from the command line on the Manager machine. This is useful if you are locked out of all accounts in the internal database.

To unlock a user account:

1. Shut down the Manager.
2. From the *DS_MANAGER* directory of the Manager install, run the `unlock.bat` or `unlock.sh` script for Windows or UNIX. The script takes two arguments: the user name and a new password. For example:

   ```
   unlock.sh jsmith NewPass123
   ```
3. Restart the Manager.

Using LDAP Authentication and Authorization

LDAP can be used for authentication, where a user is authenticated against a directory entry; authorization, where groups assigned to that user map to GridServer roles; or both authentication and authorization. For example, you can use Kerberos for authentication, but assign roles with LDAP. Or you might wish to use LDAP for authentication, but assign roles manually.

There are two authentication modes in LDAP. The *Bind Mode* authenticates a user using an LDAP bind operation (login) to the LDAP server. If the operation succeeds, the user is authenticated. In *Comparison Mode*, when a user logs in, the credentials of the user are retrieved from the LDAP server and compared to the credentials submitted in the login request. If the credentials match, the authentication is successful. Otherwise, authentication fails. In general, a hashed password is used for comparison.

Note that you do not set Bind Mode or Comparison Mode explicitly. The mode is set implicitly according to User Password Attribute: if User Password Attribute is set to any value, the authenticator uses Comparison Mode, if User Password Attribute is blank, Bind Mode is used.

If you are using LDAP for authentication, you must configure a user lookup. The User DN Format allows you to specify the user with a single parameter substitution in the DN. This is the preferred method, since it requires no LDAP search. If your directory is not

configured such that you can specify this, then use the User Search Base and parameterized User Search String Format. In this case, the User Search Base specifies where to start the search (to optimize it) and the User Search String Format specifies how to match the entry by attribute.

If you are using LDAP for authorization, you must specify how the user's groups are located. If the groups are set as attributes of the user entry, use the User Group Attribute setting. Otherwise, you must use the Group Search Configuration settings.

When the group is retrieved, any role that has the Group setting set to one of these groups is assigned to the user. You can also assign groups manually, with the **Admin > User Admin> User Admin** page, instead of or in addition to assigning with LDAP.

Parameters are specified to the LDAP search using the standard format `{n}`, where *n* is the *n*th parameter. In this case, the user name is the only parameter ever used, so use `{0}` to indicate user name.

Configuring GridServer for LDAP Authentication or Authorization

After Manager installation, when the GridServer Administration Tool is accessed for the first time, you are prompted to create a root user account. This root user account is usable regardless of whether LDAP or the internal database is used for authentication.

To configure LDAP for authentication or authorization, complete the following task to configure the connection, and then complete the tasks for configuring authentication, authorization, or both.

Configure the Connection

Procedure

1. Log in to the GridServer Administration Tool using the root user account on a Manager containing the primary Director.
2. Go to **Admin > User Admin > Authentication**.
3. Select LDAP from the **Authentication Mode** list.
4. In the **Provider URL(s)** box, enter a pipe-delimited list of URLs of your LDAP servers. The first URL is the primary server and the rest are failover servers. For example, `ldap://host1:389|ldaps://host2:636`.

5. If your server does not allow anonymous search, enter values in the **Connection DN** and **Connection Password** boxes. Example values are `cn=admin,dc=company,dc=com` and `mysecret (encrypted form)`.
6. You can optionally use the **JNDI Environment Variables** box to enter a comma-delimited list of `name=value` environment variables to use when connecting to your LDAP server. For example, `com.sun.jndi.ldap.connect.timeout=500`. A list of environment variables can be found in Oracle's JNDI LDAP documentation.
7. You can also optionally configure if the host's IP address is resolved by using **Resolve IP Address**. This is false by default; if changed to true, if there is a failure connecting to the host, the IP address is logged on failures instead of the hostname. This is useful for diagnosing problem servers when using DNS load balancing.
8. To test that the values you have provided results in a working connection, click the **Test Connection** button. A Test Results popup window displays if the test script was able to connect. If not, go back to step 4 and confirm your values.

Configure Authentication

Procedure

1. Configure a user lookup method:
   - For the User DN Format, specify the parameterized DN to be used to locate the user, for instance, `user={0},ou=usernames,dc=company,dc=com.`
   - For the full user search, specify the User Search Base, such as `ou=usernames,dc=company,dc=com`, and the parameterized User Search String Format, such as `(&(objectclass=user)(sAMAccountName={0}))`. This starts searching from the base for any entries that are of class user, and have an attribute called `sAMAccountName` that matches the user name. **User Search Subtree** optionally enables you to widen the scope of the user search to include subtrees. Set this to false when possible to improve search performance and reduce latency.
2. If you are using comparison mode, enter the name of the password attribute on the user entry in the **User Password Attribute** field, and select the appropriate digest method in **Password Digest**. In most cases Auto Detect is appropriate.
3. You can optionally enter attributes to retrieve other fields from your user search. These include the **User First Name Attribute**, **User Last Name Attribute**, and **User Email Attribute** boxes.

Configure Authorization

Procedure

1. Configure a group lookup method:
   - If the groups are assigned as attributes of the user entry, enter that attribute name in the **User Group Attribute** box, for example, `memberOf`.
   - If a separate search is required to get the group information, you must configure the following items: In the **Group Search Format** box, enter the pattern used to match user names to group entries. For example, `(memberUid={0})` returns all groups that have a `memberUid` attribute that matches the user name. In the **Group Search Attribute** box, enter the attribute that provides the name of the group in the group entry. For example, `cn`. In the **Group Search Base** box, enter the base of a group search. For example, `ou=groups,dc=company,dc=com`.
2. Click **Save**.

To map an LDAP group to a Role, go to the **Admin > User Admin > Role Admin** page, edit the Role, and set the Group entry to the name of the group. Note that oftentimes the group name might be in CN format, especially when using group search with a cn attribute. For example, if the cn attribute for the group you want to assign to the Configure role is `CN=Administrators,CN=Builtin,DC=na,DC=tibco,DC=com`,you can edit the Configure role and specify the group as that entire name, not just `Administrators`.

Authentication Schemes Supported in Comparison Mode

When comparison mode is in use, the following LDAP server password hash/encryption schemes are supported:

Supported LDAP Schemes

Scheme	Format	Description	Algorithm	Notes
{SCHEME}Hash	`{crypt}Q8k7rHl9JtTOI`	UNIX crypt	Calculate hash from the	Supported by OpenLD

Scheme	Format	Description	Algorithm	Notes
			clear text password based on algorithm and compare with password digest	AP
	`{SHA}!J78ElrfcxQlheAG/XBSz76Upy5+t65mE`	SHA hashing algorithm followed by the hash		
	`{CLEARTEXT}mypassword`	Clear text password		
{SCHEME}Hash (passwd+seed)	`{SSHA}!J78ElrfcxQlheAG/XBSz76Upy5+t65mE`	Seeded SHA hashing algorithm with the first 6 chars as the seed	Calculate the seed from the hash and calculate hash based on the clear text + salt	
PAM MD5	`$1$qPU.kEzE$Sydn2HVBATM2moKTITsPk0`	Password hash in `$1$[salt]>$[hash]` format. `$1$` is the magic string for MD5 hashing		
Apache MD5	`$apr1$A7lJPWbr$4VO3DXCAD/1U2bOX/fj6a/`	`$apr1$[salt]$[hash]` format.		

Scheme	Format	Description	Algorithm	Notes
		`$apr1$` is the magic string		
Clear Text	`Mypassword`	clear text password in octet string, specified in rfc2256	String comparison	Must not be used

Security Notes

It's not possible for deleted users to access the Administration Tool because the GridServer Director controls all user add/update/delete operations. When a user is deleted or demoted to a group with lower privileges, all other GridServer Managers get the user table update and refresh their local user cache.

When LDAP is used, this is no longer possible, as GridServer won't get notifications for LDAP user updates. Therefore, a timeout strategy is used to revalidate the user authorization. User authorization has a 15 minute TTL that is independent of the Application Server session. A deleted/demoted user does not have indefinite access to the features that are no longer permitted.

LDAP Configuration Example

For a typical example configuration, consider an OpenLDAP server that can authenticate in both Windows and Linux/UNIX domains. In the test LDAP schema, group information is specified in individual group searches.

First, go to Role Administration and set up group names for the roles. Map the Manage role to the support LDAP group.

Then configure the connection to the LDAP server as follows:

- Authentication Mode = `LDAP`
- Provider URL(s) = `ldap://integrated.datasynapse.com:389`

- Authentication Scheme = `simple`

Leave the **User name** and **Password** blank since this directory allows anonymous search.

Next, after successfully testing the connection, configure the user search by using the user DN, and leaving Search String and Search Base blank:

- User DN Format = `uid={0},ou=users,dc=datasynapse,dc=com`
- User Search String Format = not set
- User Search Base = not set
- User Search Subtree = `False`
- User Search Timeout = `5000`
- User Password Attribute = `userPassword`
- User Password Digest = `Auto Detect`

Finally, set up the group search. Since this LDAP schema keeps the groups as separate entities, leave User Group Attribute blank, and specify a separate group search:

- Group Search Attribute = `cn`
- Group Search Base = `ou=groups,dc=datasynapse,dc=com`
- Group Search Format = `memberUid={0}`
- Group Search Limit = `0`
- Group Search Timeout = `5000`
- Group Search Subtree = `True`

Using Windows Authentication and Authorization

Another method of user authentication is *Windows Authentication*. This takes advantage of the native Windows authentication layer when the Manager is running on a Windows machine on a Windows Domain. It uses the Negotiate protocol, like *Pure Kerberos Authentication*, but requires much less configuration. It uses NTLM or Kerberos as the authentication provider, depending on the client; Windows C++ and .NET use NTLM, while other clients use Kerberos. This enables web browsers and Drivers to connect to Managers by authenticating as the current Domain user, with no passwords needed.

Also, all Domain groups that a user belongs to are available as a group that can be mapped to a GridServer Role.

Configuring Windows Authentication

To use Windows authentication, you must configure your Manager, Drivers, and web browsers. Note that you can only use this method if all Managers run on Windows.

Manager Configuration

To configure your Manager to use Windows authentication:

1. Ensure that the Manager is part of the domain with which you want to authenticate.
2. Make sure that the additional third-party LGPL download has been applied. For more information, see the *Installation Guide*.
3. On the Managers:

   In the Administration Tool, go to **Admin > User Admin> Authentication**, and change **Authentication Mode** to **Windows**.

   On the same page, enter the value for the Windows Domain used to authenticate users in the Windows Domain, and click **Save**.

Driver Configuration

For all Drivers, the `DSNegotiateEnabled` property specifies if Negotiate authentication is used. Set this to true in the `driver.properties` file or by using the `DriverManager` API to enable Negotiate authentication.

Also, the following drivers and browsers must be configured, depending on the platform and Driver:

Windows C++ and .NET Drivers

Windows versions of C++ and .NET Drivers use NTLM and do not require any additional configuration.

UNIX C++ and All Java Drivers

UNIX versions of C++ and all Java Drivers (including Windows) use Kerberos rather than NTLM. Configure them using the instructions in All Java Drivers and UNIX C++ and Java Drivers sections.

Browser Configuration

Users' browsers must be configured to use Negotiate authentication. For example, in Microsoft Edge, you add the URL to the Enterprise Mode site list. In Firefox, you use the `network.negotiate-auth.trusted-uris` config parameter. See your browser's documentation for details.

If a user's browser is not configured and they attempt to log in to the Administration Tool, the browser presents them with a challenge popup screen, and they can log in manually.

Configuring Windows Authorization

When a user logs in, the list of Domain groups that user belongs to is available. All roles that are assigned to those groups are then assigned to the user.

Note that all the users must be assigned to a group that is mapped to at least one role to be able to log in. Additional roles might be mapped with Windows groups or built-in authorization.

Using Pure Kerberos Authentication

If you are not running all Managers on Windows, and you require authentication that does not require a password, you use *Pure Kerberos* authentication. This also enables web browsers and Drivers to connect to Managers by authenticating with a Windows domain, and Drivers no longer have to store passwords.

The section assumes that you already have a Kerberos Realm set up, with a Kerberos Key Distribution Center (KDC). Since the most common Kerberos installation is a Windows Domain on which the KDC is on a Domain Controller (DC), this section proceeds with that as an example.

Configuring Pure Kerberos Authentication

The following is an overview of the process:

On each Manager, there is a user that is a client of the Key Distribution Center (KDC) on the DC. This must be a user on the Windows domain. This can be an existing user, or a new user can be created. Also, this user does not need to be the same user who runs the Manager process, although it can be.

A Service Principal Name (SPN) is created on the KDC. This is the name by which a client uniquely identifies an instance of a service. It consists of the user name and Manager hostname. There might be more than one per Manager; for example, one for the fully-qualified hostname, one for the short name, and additional ones for aliases if any.

A keytab file is created for the user and placed on the Manager's file system. It contains the user credentials, and allows that user to use the KDC.

The Manager is then configured to use the realm, given the keytab location and DC hostname.

Note that it is not necessary for the machine to be on the domain. While it is true for a Windows Manager, there is no requirement for a UNIX Manager to be added to the domain.

To configure your Manager, you need the following information from your IT department:

- The name of your realm
- The fully-qualified hostname of the DC

The following table lists the various values that are used in the following procedure. Substitute your own values for the following:

Setting	Value
Realm Name	*REALM.NAME*
DC Fully Qualified Hostname	*dc.domain.com*
Manager Fully Qualified Hostname	*manager.domain.com*
GridServer Username	*gs_user*

Manager Configuration

To configure your Manager for pure Kerberos authentication:

1. Map one or more Server Principal Names. Log in to *dc.domain.com*. Open a console, and run the following to create SPNs for the short and fully qualified names:

```
setspn -A HTTP/manager gs_user
setspn -A HTTP/manager.domain.com gs_user
```

2. Create the keytab file:

   From that same DC, generate a keytab:

```
ktpass \
/princ HTTP/manager.domain.com@REALM.NAME \
/ptype KRB5_NT_PRINCIPAL \
/crypto all \
/mapuser gs_user@REALM.NAME
/pass {gs_user password}\
/kvno 0\
/out GridServerUser.keytab \
```

   Move this file to your Manager, and place it in a location and set permissions such that only the user that runs the Manager can read it. Because it contains credential information, it must be kept secure.

3. The KDC client requires a random sequence to protect the session. You can generate one using the openssl command line tool by running the following command:

```
openssl rand -hex 12
```

4. Edit the DS_MANAGER`/webapps/livecluster/WEB-INF/web.xml` file:

   Uncomment the `kerberosFilter` section.

   Set the param-value of `kerberos.principal` to `HTTP/manager.domain.com@REALM.NAME`

   Set the param-value of `kerberos.keytab` to the location of the keytab file on your Manager.

   Set the param-value of `signature.secret` to the random sequence that was just generated.

Driver Configuration

For all Drivers, the `DSNegotiateEnabled` property specifies if Negotiate authentication is used. Set this to true in the `driver.properties` file or by using the `DriverManager` API to enable Negotiate authentication. Otherwise username and password is used.

Also, the following platform or drivers must be configured, depending on the platform and Driver:

All Windows

- You must enable TGT on the system running the Driver. For more information, see `https://support.microsoft.com/en-us/topic/updates-to-tgt-delegation-across-incoming-trusts-in-windows-server-1a6632ac-1599-0a7c-550a-a754796c291e`.
- The `allowtgtsessionkey` registry key must be set. Add or change this registry value:

  `HKEY_LOCAL_MACHINE\System\CurrentControlSet\Control\Lsa\Kerberos\Parameters`

  Value Name: `allowtgtsessionkey`

  Value Type: `REG_SZ`

  Value Data: `1`
- The Driver user cannot be a member of the Local Admin group.

All Java Drivers

Java includes support for Kerberos authentication natively. The standard method of doing this is via the `login.conf` and `krb5.conf` files.

Convenience properties have also been added to the `driver.properties` file so that you do not need to set these files up:

- `DSKerberosRealm`: the name of the kerberos realm. For example, `REALM.NAME`
- `DSKerberosKDC`: The hostname of the KDC. For example, `dc.domain.com`

(Note that if your application is already set up to use Kerberos via standard methods, you do not need to set these two values.)

Windows C++ and .NET Drivers

Create a file called `krb5.ini` and put it in the `c:\windows` directory:

This file must contain the following:

```
[libdefaults]
default_tgs_enctypes = AES256-CTS RC4-HMAC DES-CBC-MD5 DES-CBC-CRC
default_tkt_enctypes = AES256-CTS RC4-HMAC DES-CBC-MD5 DES-CBC-CRC
preferred_enctypes = AES256-CTS RC4-HMAC DES-CBC-MD5 DES-CBC-CRC
[domain_realm]
.domain.com = REALM.NAME
dc = REALM.NAME
dc.domain.com = REALM.NAME
```

UNIX C++ and Java Drivers

UNIX Kerberos clients use a ticket cache. This ticket cache must be populated prior to running the Driver.

This is done by running the `kinit` command, which prompts the user for their password, and populates the cache with a ticket.

Browser Configuration

Users' browsers must be configured to use Negotiate authentication. For example, in Microsoft Edge, you add the URL to the Enterprise Mode site list. In Firefox, you use the `network.negotiate-auth.trusted-uris` config parameter. See your browser's documentation for details.

If a user's browser is not configured and they attempt to log in to the Administration Tool, the browser presents them with a challenge popup screen, and they can log in manually.

Configuring Pure Kerberos Authorization

Because Kerberos only provides authentication, authorization is provided with LDAP or built-in authorization.

To be able to log in, all users must belong to at least one role that is assigned to a Windows Domain group, which corresponds to an Active Directory LDAP group. See, Using LDAP Authentication and Authorization. You only need to configure Authorization. Additional Roles might be mapped with LDAP groups or built-in authorization.

Managing Multiple Brokers with Grid Single Sign-On (SSO)

The Grid Single Sign-On (SSO) feature enables a user to be logged in to all Managers on a Grid after having logged in once to any Manager.

> **Note**
> The use of Grid SSO is discouraged; using Kerberos with LDAP or Windows authentication is preferred to achieve single sign-on. Grid SSO must never be used when Pure Kerberos or Windows Authentication are enabled.

The following prerequisites are required before you can enable Broker SSO:

- All Managers must reside in the same network sub-domain
- The Primary Director's hostname that appears in the **Admin > System Admin > Manager Reinstall > Local Configuration > Hostname** field must be a fully-qualified name in this sub-domain.

Grid SSO Configuration

To configure Grid Single Sign-On:

1. Log in to the GridServer Administration Tool on the Manager containing the Primary Director.
2. Go to **Admin > System Admin > Manager Configuration > Security**.
3. Click the **Security** link.
4. Set **Grid Single Sign-On sub-domain** to the network subdomain used by your Managers. At installation, GridServer attempts to determine the subdomain name based on the fully qualified host name of the Director, and the value is entered in the **Grid Single Sign-On sub-domain** field. If this value is incorrect, you must change it before enabling SSO.
5. Set **Grid Single Sign-On** to `true`. Note that you must configure all Directors and Brokers with their fully qualified hostnames.

6. Repeat this configuration for each Manager containing a Director in your grid. You do not need to configure Managers with only Brokers.

Constraints and Limitations

The following constraints apply when using Grid SSO:

- You must enable cookies in your browser.
- When a Primary Director fails, the Grid SSO session is lost. If you have a Secondary Director configured for failover, you can log in to it and start a new SSO session.
- Single sign-on enables you to log in to other Managers that are in the same network subdomain. For example, if your Primary Director is `director.grid.example.com`, logins to any Brokers in `grid.example.com` can be authenticated with Grid SSO. You can't use Grid SSO to connect to Brokers in other subdomains or domains. For example, if your Primary Director is `director.grid.example.com`, you can't automatically log in to Brokers in `prod.example.com` or `grid.example.co.uk`.
- Only the following fields accept multi-byte characters:
  - **Username**, **Password**, and **Confirm Password** on the initial log in page.
  - All fields on the **Admin > User Admin > Run-As Credentials** page.
  - **Choose Username** on the User Event page.
  - **Username**, **Firstname**, **Lastname**, **Password**, and **Confirm Password** on the **Users > Users** page.
  - **RunAsUser** on the Edit Service Type page.
- Grid SSO must not be used if Negotiate Authentication is used.

Client Routing

You can route Clients to Brokers by using Roles, the Broker Routing page, and the Driver API.

Routing Clients With Roles

The easiest and most common method of routing clients is to set the Manager List in a user role, and then set a user name in the `driver.properties` file. This routes clients directly to a set of listed Brokers.

To configure the Manager List:

1. In the GridServer Administration Tool, go to **Admin > User Admin > Role Admin**.
2. Select a role, click the **Actions** list, and select **View/Edit** , or to create a new role, with no role selected, click the **Actions** list and select **Create a new Security Role**.
3. Enter the list of Brokers to which you want the client to be routed.
4. Click **Save**.
5. Go to **Admin > User Admin > User Admin**.
6. Select a user, click the **Actions** list, and select **Edit User**, or to create a new user, with no role selected, click the **Actions** list and select **Create New User**.
7. Assign the role to the user.
8. Edit the `driver.properties` file for the Driver, and set the `DSUsername` and `DSPassword` properties to the user name and password of the user assigned the role you created/edited.

Routing Clients On The Broker Routing Page

You can route clients to Brokers using rules based on Driver properties. This is similar to how you can route Engines to Brokers by creating routing rules based on user-defined properties. This limits Drivers that log in to a Broker based on the value of one or more properties, using comparators you have defined.

To enable Driver Routing:

1. Define one or more properties in the `driver.properties` file used by the Driver. You can view a Driver's properties at the **Grid Components > Drivers > Driver Admin** page.
2. Go to the **Grid Components > Brokers > Broker Routing** page. Select a Broker from the list.
3. Click **Edit**.

4. In the Driver column are comparators for Drivers. Enter one or more comparators by entering a property name, selecting an operator, entering a value, then clicking **Add**. You can also select **Is Missing**, which evaluates the comparator as true if the property is not present on a Driver.

Multiple comparators are ANDed together. If the comparators evaluate to true, the Driver is allowed to log in to the Broker. Note that property names and values are case-sensitive.

You can add a single property in a comparator that holds different values by using a comma-delimited value. For example, Broker A can have a comparator matching a `username` property to be equal to `TestUser1` or `TestUser2` . If a Driver submits with either `TestUser1` or `TestUser2` values, the Tasks are then routed to Broker A. In this use case, the query must be: `username equals TestUser1,TestUser2`.

Routing Clients With The Driver API

You can use the `connect(String broker)` method of the `DriverManager` API on all Driver platforms to force a client to log in to a specified Broker.

Managing Services

This section provides information about deploying, running, and scheduling Services.

Deploying Services

GridServer uses *Grid Libraries* to distribute classes, libraries, and other resources to Engines. Grid Libraries provide a solution to the problem of distinct Services requiring different versions of the same resource. They provide the following features:

- Version control, including optional automatic selection of the most current version of a Grid Library.
- Resource upgrading without interrupting current Sessions.
- Specification of dependencies on other Grid Libraries.
- Specification of which C++ runtime to use and non-default JREs via dependencies.
- All-in-one packaging for JAR files, native libraries for multiple OSes, .NET assemblies, Command Service executables, R scripts, and Engine Hooks.
- Specification of Environment Variables and Java System properties.
- Engines that require different compiler support libraries can participate in the same Service Session.
- Parameterization of package configuration through the use of property substitution files.

The *Resource Deployment* feature replicates Grid Libraries from a Manager to Engines. In the simplest sense, this enables you to copy a set of bundled resources to each Engine to run a Service.

This section details how to use Grid Libraries for Service deployment to your GridServer installation.

About Grid Libraries

A Grid Library is a set of resources and properties necessary to run a Grid Service. It also contains configuration information that describes how these resources are to be used. For example, a Grid Library can contain JAR files, native libraries, configuration files, environment variables, hooks, and other resources.

A Grid Library is deployed as an archive file in ZIP or gzipped TAR format, with a `grid-library.xml` file in the root that describes the Grid Library. It might also contain any number of directories that contain resources.

Grid Libraries are identified by name and version. Versions are optional, but recommended. They are used to detect conflicts between a desired library and a library that has already been loaded. Versions also provide for automatic selection of the latest version of a library. A GridServer Service can specify that it is implemented by a particular Grid Library when its Service Type is registered, or by using a Service Option.

Grid Libraries can specify that they depend on other Grid Libraries. Like the Service Option, such dependencies can be specified by the name, and optionally the version. Also, nearly all aspects of a Grid Library can be specified to be valid only for a specific operating system. This means that the same Grid Library can specify distinct paths and properties for Windows and Linux. However, only the appropriate set of package options is applied at run time.

Variable Substitution

You can use placeholder variables in a `grid-library.xml` file, which are then substituted with their value as defined in a properties file or in an OS environment variable. This enables quick changes in properties in the `grid-library.xml` file without redeploying the Grid Library.

If the `grid-library.xml` file contains a property with a value contained with the `$` character, such as `$mydir$`, it is substituted with the value in one of three places, in this order:

- A default properties file in your Grid Library named `grid-library.properties`. This can provide baseline values for your variables.
- An external properties file, named with the same name as the Grid Library archive, with the extension `.properties`, in the Grid Library deployment directory. Values in an external properties file replace those defined in the default properties file within the Grid Library.

- A defined OS environment variable. This value replaces the value defined in either properties file.

> **Note** If the substitution is not found in the file, the empty string, "", is substituted.

Substitutions are allowed anywhere in a string within the content of property value elements and path elements. Multiple substitutions per string are allowed. `$` characters can be treated as literals by escaping them with another `$` character. Windows paths that are specified in the *library*`.properties` file must escape the `\` character with another `\`.

Versioning

Versioning provides the following functionality:

- It allows for deployment of new versions of libraries and deletion of old versions without interrupting currently running Service Sessions.
- It provides for specifying conflicts, or libraries that cannot coexist with each other.
- It allows for a Service Session or dependency to specify the use of the latest version of a Grid Library.

To use versioning, you must specify the Grid Library version in the configuration file. An Engine can load only one version of the library with the same name at any time. If the version is not specified, it is implied to be 0.

The version is a String and **must** adhere to the following version format. This format is

```
[n1].[n2].[n3]...
```

where `nx` is an integer, and there might be one or more version points.

For instance,

```
4.0.1.1,  4.1,  3
```

are in the proper comparable version format.

The integer at each version point is evaluated starting at the first point, and continues until a version point is greater than the other. If a version point does not exist for one, it is implied as zero.

For instance

```
4.0.0.1 > 4.0
4.0.0.5 < 4.0.1.1
```

To specify that a dependency or Service uses a particular version of a Grid Library, the version field is set to that value. To specify that it uses the latest version, the field is left blank.

> **Note:**
> If a version is specified, it must match exactly. That is, `3.0.0` is not the same as `3`. If the library's version is `3.0.0` and the Service specifies `3`, the Service does not find that library and then it fails.

If a version is specified but not in this format, and there are multiple versions of a library, the "latest version" is undefined. Thus, automatic selection of the latest version is only possible when all Grid Libraries with the specified name provide a version in the proper format.

By default, if a Service has been set to use the latest version of a Grid Library, all Engines work on the latest version at the time the Service is started. This is regardless of whether a newer library has been deployed. This can be changed by setting the `GRID_LIBRARY_STRICT_VERSIONING` option in the `driver.properties` file to false. When false, if you deploy a newer version of the library while the Service is running, Engines that have not yet worked on the Service use the newer version. Engines that worked on it before deployment continue to use the older version.

Dependencies

Grid Libraries might specify dependencies on other Grid Libraries. A dependency specification resolves to a particular Grid Library using two values:

- **grid-library-name** - the name of the Grid Library, as specified in the dependency's XML
- **grid-library-version** - the version of the Grid Library, as specified in the dependency's XML. OS compatibility is determined by checking the `os` and `compiler`

tags for the top-level element in the dependent Grid Library. If not specified, it uses the latest version supported by the OS

Note that if a dependency resolves to more than one Grid Library, the dependency used is undefined.

Two dependent libraries conflict if they have the same library name, but different versions.

It is possible to specify an OS attribute to a `<dependency>` element for ignoring Grid Libraries that do not apply to an Engine's particular operating system. For example, if a Grid Library contains native libraries for multiple platforms, you can specify OS-specific dependencies on the bridge Grid Libraries such that the Engine only loads the bridge corresponding to its operating system.

Note that if a dependency is missing, the Engine logs a warning. Rather than the current task failing, the Engine attempts to continue loading all necessary libraries to run the task.

Conflicts

A conflict between two Grid Libraries means that these libraries cannot be loaded concurrently. When there is a conflict between a loaded Grid Library and a Grid Library that is required by a Service, the Engine must restart to unload the current libraries and load the requested library.

The following circumstances result in a conflict:

- **Version Conflict** - the most common conflict arises with versioning, and typically when upgrading versions or using more than one version of the same library concurrently. This conflict arises when a Grid Library with the same `grid-library-name` as the requested Grid Library, but a different version, is loaded.
- **Explicit Conflict** - there can be situations in which different Grid Libraries can conflict with each other due to conflicting native libraries, different versions of Java classes, and so on. Because the Engine cannot determine these implicitly, the `conflict` element can be used to specify Grid Libraries that are known to conflict with this Grid Library.

  Additionally, the value of the `grid-library-name` can be set to `"*"`. This means that this Grid Library can conflict with all other Grid Libraries (aside from its dependencies), and it is guaranteed that no other Grid Libraries load concurrently with this Grid Library. Note that this is only allowed if the Grid Library is not a dependency. If the `"*"` is used as a conflict in a Grid Library that is a dependency, a verification error occurs.
- **Dynamic Version Conflict** - a Grid Library conflict occurs if:

   a. Dynamic versioning is used.
   b. The latest version of a Grid Library or Grid Library dependency has changed due to an addition or removal of a dependency since the Grid Library has been loaded.

- **Variable Substitution Conflict** - a Grid Library conflict occurs if its variable substitution file has changed since it has been loaded.

JNI Library Loading

When a Service Session is set to use a Grid Library, that library is loaded. Loading is the process of setting up all resources in the Grid Library for use by the Service. A library is loaded only once per Engine session.

First, the library loads itself, and then it loads all dependencies. The library loader uses the depth-first, or preorder traversal algorithm when loading libraries. When there are a number of dependencies in a Grid Library, the order in the XML is considered left-to-right with respect to the algorithm. The library search order for `lib-path` and `jar-path` is determined by their respective lists. Certain aspects of a load might require a restart, and possibly re-initialization of the state.

The following steps are performed by a load of the root library and all dependencies:

1. It checks for conflicts with currently loaded Grid Libraries. If so, it restarts with the requested Grid Library and clears out the current state of any loaded libraries.
2. If new `lib-path`s have been added for its OS, they append to the current list of `lib-path`s. The state of loaded libraries includes all libraries already loaded, plus the requested library. Note that specifying a JRE dependency has this effect.
3. If new `jar-path`s have been added for its OS, the JAR files and classes are added to the classloader.
4. If new `assembly-path`s have been added, it adds them to the .NET search path.
5. If new `command-path`s have been added for its OS, it is added to the search path for Command tasks.
6. If new `hooks-path`s have been added, any hooks in the path are initialized.
7. If the default is current and a Grid Library is requested, the Engine restarts.

State Preservation

Under most cases, when an Engine shuts down, it preserves the current state of which Grid Libraries it has loaded. When it starts back up, it loads all Grid Libraries that were loaded when it shut down. As Grid Libraries are loaded, the path elements they contain are added to a "master" list of paths for that type of path element. For example, if a Grid Library contains a `lib-path` specification, that `lib-path` is appended to the list of `lib-path` values obtained from already-loaded Grid Libraries.

Note that this means that it is up to the creator of the Grid Libraries, which are deployed on the grid, to ensure that the ordering of library paths does not lead to loading the wrong library. For example, consider two different Grid Libraries each provide DLLs in their `lib-paths` that share the name. Because of OS-specific library loading conventions, the one used is the first one in the aggregate `lib-path` from across all loaded Grid Libraries. Likewise, for Java classes, when more than one copy of the same class is in the classloader, it is undefined which class loads. Therefore, it is important to either subdivide Grid Libraries appropriately when there is a possibility that such conflicts can arise. You can also use the `conflict` element to explicitly state conflicts.

Grid Library and RunAs State information persists on normal Engine shutdowns. It includes task failures aside from crashes. If the Engine does not shut down normally, such as if it crashes, or if the Daemon kills the process due to it exceeding the shutdown timeout, the state is reset.

If an Engine shuts down due to a conflict, it clears the current state and sets up for only the requested Grid Library on restart. This is referred to as preloading. If an Engine shuts down due to internal library inconsistencies or a crash, the state is not saved. State is also cleared on all instances when a Daemon is disabled and re-enabled.

Task Reservation

If an Engine requires a restart to load a Grid Library, the task is reserved on the Broker for that Engine. The Engine is instructed to log back into the same Broker, and takes that task on log in. You can condfigure the timeout for this at **Admin > System Admin > Manager Configuration > Services**.

Environment Variables and System Properties

All environment variables and Java System properties for a Grid Library and all dependencies are set each time a task is taken from a particular service that specified that Grid Library. (They are not cleared after the task is finished.) Environment variables are set

via JNI so that they can be used by native libraries or .NET assemblies. They are also passed into Command Services. Note that environment variables such as `PATH` and `LD_LIBRARY_PATH` must not be changed through this mechanism. Rather, `library-path` and `command-path` are reserved for manipulating these variables.

Using Grid Libraries from a Service

Services can specify a Grid Library to use by setting the `GRID_LIBRARY` and optionally the `GRID_LIBRARY_VERSION` Service Options. This is typically set by Service Type in the **Services > Services > Service Types** page, although it can be set programmatically on the Session. Services can specify a Grid Library to use by setting the corresponding Service Option values. If the version is not set, a Service uses the latest version of a Grid Library.

If a Service needs to find resources in a Grid Library, it can use the Grid Library Path. This value is a path value that includes the root directories of all Grid Libraries currently loaded. For Java, .NET, and C++, the path is `EngineProperties.GRID_LIBRARY_DIR`; for command services, it is the environment variable `ds_GridLibraryPath`.

Super Grid Libraries

A Grid Library can be declared as a Super Grid Library. This means that it is always loaded when the Engine starts up. The typical use case for this is to have an `EngineHook` that queries the system for some information, which is used to set `EngineSession` properties prior to the Engine running any tasks.

To specify that a Grid Library is a Super Grid Library, set the `super` attribute in the `grid-library` element. For example, `<grid-library .. super="true" />` . Super Grid Libraries also cannot have conflicts or dependencies. Other libraries cannot depend on or conflict with them.

Super Grid Libraries are loaded upon startup before anything else. They are ignored on conflict checks for `*` (all).

If a new Super Grid Library is deployed while an Engine is running, it is loaded. If a new version of an existing Super Grid Library is deployed while an Engine is running, the Engine restarts.

Deployment

Grid Libraries are deployed using the resource deployment page at **Services > Services > Grid Libraries**, or by using the Admin API. When a Grid Library is uploaded, it is first verified to ensure that the ZIP archive is not corrupt, and that the `grid-library.xml` file validates against the Grid Library DTD. If there is an error, it is displayed next to the file on the **Services > Services > Grid Libraries** page.

The Resource Manager then replicates uploaded, valid libraries to all Engines. Variable Substitution property files also must be placed in this directory. Engines download Grid Libraries based on the attributes in the root level `grid-library` element. Grid Libraries whose attributes match the properties of a particular Engine are downloaded by that Engine and ignored by Engines with non-matching properties. If no attributes are specified in this element for a particular Grid Library, all Engines download it.

Adding or removing Grid Libraries or property files do not trigger an Engine and Daemon restart. It is not necessary to restart until the Engine actually needs to use the Grid Library, and even then only if necessary according to the loading procedure. If a deployed Grid Library is changed, it does cause the Daemon and Engines to restart. Also, it is the responsibility of the user not to delete Grid Libraries loaded by active Services from the Libraries page, as that might lead to library load failures for tasks that are run later.

If you are not using the Resource Manager for replication, you can use an alternate shared Grid Library directory. Then set the **Grid Library Path** in all Engine Configurations to point to this directory, instead of the default replicated location. When changes are made to this library, you must then use the **Update** button on the **Services > Services > Grid Libraries** page on the Primary Director. This sends a message to all Engines to check and update their Grid Libraries using the Grid Library Manager.

The Resource Manager uses secure hashes as file signatures when determining if a file has changed on the Manager. All files are signed when the Manager starts. After that, a file is only signed again if the file's last modified time has changed since the sign.

Disabling Resource Deployment

There are some situations in which you might not want to use the Resource Manager for replication. For example, you might want to use another shared location for Grid Libraries, deploy resources manually, or deploy different sets of resources to different Brokers.

There are four different strategies for disabling resource deployment:

- Disabling Director to Broker synchronization on the Director.
- Disabling Director to Broker synchronization on the Broker.
- Disabling Broker to Engine synchronization on the Broker.
- Disabling Broker to Engine synchronization in an Engine Configuration.

Each of the strategies is described below.

Note that for manual deployment, when you deploy Grid Libraries, you must click the **Update** button on the **Services > Services > Grid Libraries** page to notify all Engines to rescan their directories.

Disabling Director to Broker Synchronization on the Director

If you want to have different sets of resources on different Brokers, you can disable the Director to Broker resource synchronization on the Director. This causes none of the Brokers reporting to that Director to synchronize resources with that Director. The Brokers continue to synchronize resources with their Engines.

To disable Director to Broker synchronization on the Director:

1. Go to **Admin > System Admin > Manager Configuration > Resource Deployment**.
2. Under the **Director Settings** heading, change **Synchronize Resources To All Brokers** to **False**.

When you disable Director to Broker Synchronization, if for any reasons your Engines move to a different Broker, they need to synchronize their resources with the new Broker. This can result in a significant delay in the Engine being able to take tasks and can cause a severe increase in network traffic depending on how many Engines move.

⚠ **Warning**	Grids that rely on Engines being shared are strongly recommended not to use this option. If you must use this option, minimize Engine movement by disabling Engine sharing and balancing. In general, we do not recommend you use this option and consider Grid Library filters instead.

Disabling Director to Broker Synchronization on the Broker

In some situations, it might be desirable to disable Director to Broker resource synchronization on specific Brokers, while still permitting Broker to Engine resource

synchronization. For example, allowing "last week's" tasks to run to completion on a limited subset of the grid, while permitting "this week's" tasks to be launched with new resources.

This is similar to disabling synchronization on the Director, but because the setting is on the Broker, it is used on a case-by-case basis. You can have some Brokers synchronize, and disable synchronization on others.

To disable Director to Broker synchronization on a Broker:

1. Go to **Admin > System Admin > Manager Configuration > Resource Deployment**.
2. Under the **Broker Settings** heading, change the value of **Synchronize Resources From Director** to `False`.

Note

If you disable synchronization on the Director, no Brokers synchronize with the Director regardless of this setting on the Broker.

Disabling Broker to Engine Synchronization on the Broker

If you want to deploy resources manually to Engines, such as when you use a shared resource location on a shared file system, you can disable synchronization between a Broker and all of its Engines.

To manually deploy resources:

1. Go to **Admin > System Admin > Manager Configuration > Resource Deployment**, and under the **Broker Settings** heading, change the value of **Synchronize Resources to Engines** to `False`.
2. Unpack the required Grid Libraries in a shared location, such as a shared network drive that the Engines can access. Manually extract the resource files, as the Engine does not unpack them.
3. In the Engine Configuration, set **Grid Library Path** to the location of the shared resources from step 2.

Note

When Broker to Engine synchronization is disabled, Director to Broker synchronization still occurs. This means that the Engines still auto-upgrade themselves when an update is installed.

Disabling Resource Synchronization in Engine Configuration

To change resource synchronization on a per-Engine basis, in the Engine Configuration, under the **Resource Validation** heading, change the value of **Synchronize Resources** to `False`. When an Engine with this setting disables log in, it does not synchronize resources.

This is useful when you want to isolate a set of Engines and prevent them from doing any resource synchronization from any Broker. This must not be used if you want to deploy your resources manually.

> **Note** When resource synchronization is disabled in the Engine Configuration, Engines do not auto-upgrade themselves when an update is installed or when the grid is upgraded.

Bridges

Bridges are Grid Libraries that enable Engines to run non-Java Services, such as C++, .NET, and R. All Bridges are pre-packaged and deployed in the Grid Library replication directory when you install or upgrade the GridServer Manager.

JREs

In the rare event that a particular service cannot use the default JRE that is deployed to the Engines, a JRE can be packaged as a Grid Library. The Service's top-level Grid Library then declares it as a dependency. When an Engine takes a Task, it then restarts using this JRE. Note that the JRE must be a supported version.

JREs are packaged as `jre_name-version.gz` or `jre_name-version.zip` where, `jre_name` includes `jre-os`. The `version` is the JRE version, for example, `11.x.x`. The os is the platform, such as `linux64`, `win64`, `linux`, or `win32`.

For example, for linux 64: `jre-linux64-11.x.x.tar.gz`.

For a JRE Grid Library, you can optionally specify JVM arguments in the XML. To do so, add an `<arguments>` element to the root element. It can take any number of `<property>` elements, each containing a `<name>` element and an optional `<value>` element.

If the property has a value, the argument `name=value` is added. Otherwise, only the `name` argument is added.

If the same argument is set in the Engine Configuration and the Grid Library, the Grid Library overrides the Engine Configuration.

> **Note** Specifying the JVM debug port inside a Grid Library results in unpredictable behavior and is not supported. Set this functionality with the **Debug Start Port** setting on the **Grid Components > Engines > Engine Configurations** page.

Example: Creating a Multi-Platform JRE Grid Library

Here is an example of how to use the same JDK version Grid Library for Linux and Windows:

> **Note:** GridServer 7.1.1 includes Java Runtime Environment (JRE) gridlibs for Java 11, 17, and 21, but does not include a gridlib for Java 11. Note that Java 17 and 21 do not include separate JREs in the GridServer 7.1.1 package.

1. Download the target JDK version installer for each targeted platform from the Oracle website, https://www.oracle.com/in/java/technologies/javase/jdk11-archive-downloads.html.
   For example, for Linux64, download `jdk-11.x.x_linux-x64_bin.tar.gz` and for Windows-x64, it is `jdk-11.x.x_windows-x64_bin.zip`.
2. Extract the downloaded JDK `tar.gz` file for Linux or `.zip` file for Windows at the desired location.
3. Create a `grid-library.xml` file with the following contents:

   For Linux:

```
<?xml version="1.0" encoding="UTF-8"?>
   <grid-library jre="true" os="linux64">
   <grid-library-name>jre-linux64</grid-library-name>
   <grid-library-version>11.x.x</grid-library-version>
   <lib-path>
      <pathelement>./bin</pathelement>
      <pathelement>./lib</pathelement>
      <pathelement>./lib/server/</pathelement>
```

```
      </lib-path>
      </grid-library>
```

For Windows:

```
   <?xml version="1.0" encoding="UTF-8"?>
   <grid-library jre="true" os="win64">
   <grid-library-name>jre-win64</grid-library-name>
   <grid-library-version>11.x.x</grid-library-version>
   <lib-path>
      <pathelement>jre/bin</pathelement>
      <pathelement>jre/lib</pathelement>
      <pathelement>jre/lib/server/</pathelement>
   </lib-path>
</grid-library>
```

Note that the order of `pathelement` is important. For example, if you want the server VM, then `jre/lib/i386/server` must come before `jre/lib/i386`. This also applies to the `native_threads` directory.

4. Create the Grid Library archive for each platform containing the `grid-library.xml` that you created above and save the `grid-library.xml` in the extracted JDK folder.
5. Rename the JDK folder to JRE folder for the targeted platform and OS with the JRE version number.
   For example, `jre-linux64-11.x.x` for Linux and `jre-win64-11.x.x` for Windows.
6. Archive the JRE folder to create JRE package as `tar.gz` file for Linux and `.zip` file for Windows.
   For example, `jre-linux64-11.x.x.tar.gz` for Linux and `jre-win64-11.x.x.zip` for Windows.
7. Navigate to **Services > Services > Grid Libraries**.
8. Select **Upload Grid Library** at the top-right of the page.
9. Click **Choose File**, browse to choose the `tar.gz` or `.zip` JRE file, and then click **Upload**.
10. Add the following dependencies to your application:

```
   <dependency>
      <grid-library-name>jre-win64</grid-library-name>

      <grid-library-version>11.x.x</grid-library-version>
   </dependency>
```

```
<dependency>
   <grid-library-name>jre-linux64</grid-library-name>

   <grid-library-version>11.x.x</grid-library-version>
</dependency>
```

After performing these steps your application uses the exact JRE version that you have specified for each platform.

Packaging Grid Libraries

For more information about how to create and package Grid Libraries, see the *GridServer Developer's Guide*.

Distributing Grid Libraries

The GridServer system provides a Resource Deployment mechanism for securely distributing Grid Libraries. The Grid Libraries to be deployed are uploaded to the Primary Director. The resources on the Director are synchronized to Brokers, and then Brokers synchronize the files with Engines. The files are secure in that they cannot be accessed by anyone on the network, only the Engines.

Maker/Checker Support

In an enterprise grid environment, there might be multiple groups responsible for creating resources. It might be necessary to minimize contention between the groups. For instance, two groups might use the same core third party library and have packaged it as a Grid Library; but each might have assigned the same version number to a slightly different patch level, causing Engines to restart frequently.

To address this, there are two sets of resources for two different user roles. The "maker" user has access to a staging area, where they can upload resources to a staging area. The "checker" user then validates the resources (ensuring that there is not a version conflict in our example) and deploys the resources. A "maker" user has a Security Role with access to the Resource Deployment Maker feature (such as the default Manage role.) A "checker" has access to the Resource Deployment Checker feature (such as the default Configure role.)

If a Director to Broker sync fails for any reason, or if a file sign check fails for any reason, the Broker to Engine sync is disabled until the next successful sync or sign. When in this state, Engines act as if sync is disabled, and continue being operational with their current set of resources.

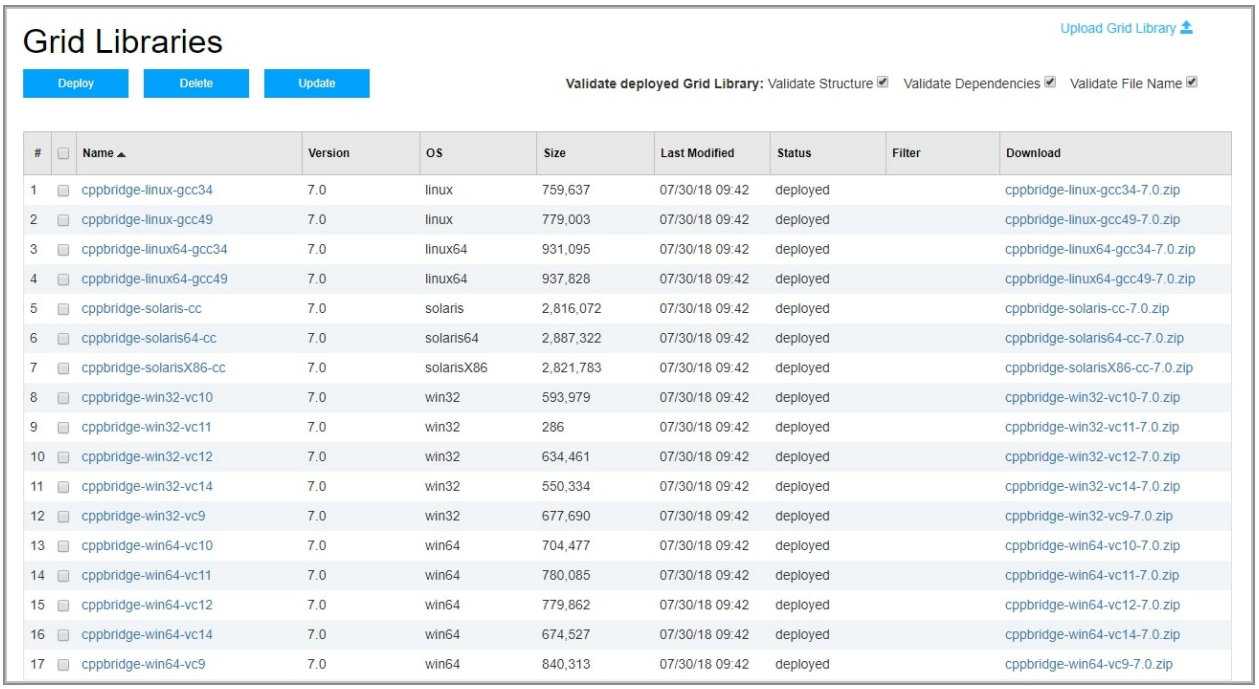
The Resource Deployment Interface

The Administration Tool provides a graphical interface to manage resources synchronized to Engines. To manage resources, on the Primary Director, go to **Services > Services > Grid Libraries**. The Grid Libraries page features a file browser that can be used to manage Grid Libraries.

The **Grid Libraries** page displays a list of Grid Libraries. To upload a Grid Library or Filter:

1. Click **Upload Grid Library** at the top of the list.
2. Click **Choose File**, browse to a file, and then click **Upload**.

   This transfers the file into the staging directory. To download a Grid Library, click the file name in the list.



#	Name	Version	OS	Size	Last Modified	Status	Filter	Download
1	cppbridge-linux-gcc34	7.0	linux	759,637	07/30/18 09:42	deployed		cppbridge-linux-gcc34-7.0.zip
2	cppbridge-linux-gcc49	7.0	linux	779,003	07/30/18 09:42	deployed		cppbridge-linux-gcc49-7.0.zip
3	cppbridge-linux64-gcc34	7.0	linux64	931,095	07/30/18 09:42	deployed		cppbridge-linux64-gcc34-7.0.zip
4	cppbridge-linux64-gcc49	7.0	linux64	937,828	07/30/18 09:42	deployed		cppbridge-linux64-gcc49-7.0.zip
5	cppbridge-solaris-cc	7.0	solaris	2,816,072	07/30/18 09:42	deployed		cppbridge-solaris-cc-7.0.zip
6	cppbridge-solaris64-cc	7.0	solaris64	2,887,322	07/30/18 09:42	deployed		cppbridge-solaris64-cc-7.0.zip
7	cppbridge-solarisX86-cc	7.0	solarisX86	2,821,783	07/30/18 09:42	deployed		cppbridge-solarisX86-cc-7.0.zip
8	cppbridge-win32-vc10	7.0	win32	593,979	07/30/18 09:42	deployed		cppbridge-win32-vc10-7.0.zip
9	cppbridge-win32-vc11	7.0	win32	286	07/30/18 09:42	deployed		cppbridge-win32-vc11-7.0.zip
10	cppbridge-win32-vc12	7.0	win32	634,461	07/30/18 09:42	deployed		cppbridge-win32-vc12-7.0.zip
11	cppbridge-win32-vc14	7.0	win32	550,334	07/30/18 09:42	deployed		cppbridge-win32-vc14-7.0.zip
12	cppbridge-win32-vc9	7.0	win32	677,690	07/30/18 09:42	deployed		cppbridge-win32-vc9-7.0.zip
13	cppbridge-win64-vc10	7.0	win64	704,477	07/30/18 09:42	deployed		cppbridge-win64-vc10-7.0.zip
14	cppbridge-win64-vc11	7.0	win64	780,085	07/30/18 09:42	deployed		cppbridge-win64-vc11-7.0.zip
15	cppbridge-win64-vc12	7.0	win64	779,862	07/30/18 09:42	deployed		cppbridge-win64-vc12-7.0.zip
16	cppbridge-win64-vc14	7.0	win64	674,527	07/30/18 09:42	deployed		cppbridge-win64-vc14-7.0.zip
17	cppbridge-win64-vc9	7.0	win64	840,313	07/30/18 09:42	deployed		cppbridge-win64-vc9-7.0.zip

*The **Services > Services > Grid Libraries** page*

Each Grid Library can be in one of the following states:

- New — An undeployed resource in the staging directory.

- Deployed — A deployed resource in the deployment directory.
- Error — There was an issue with an uploaded file.

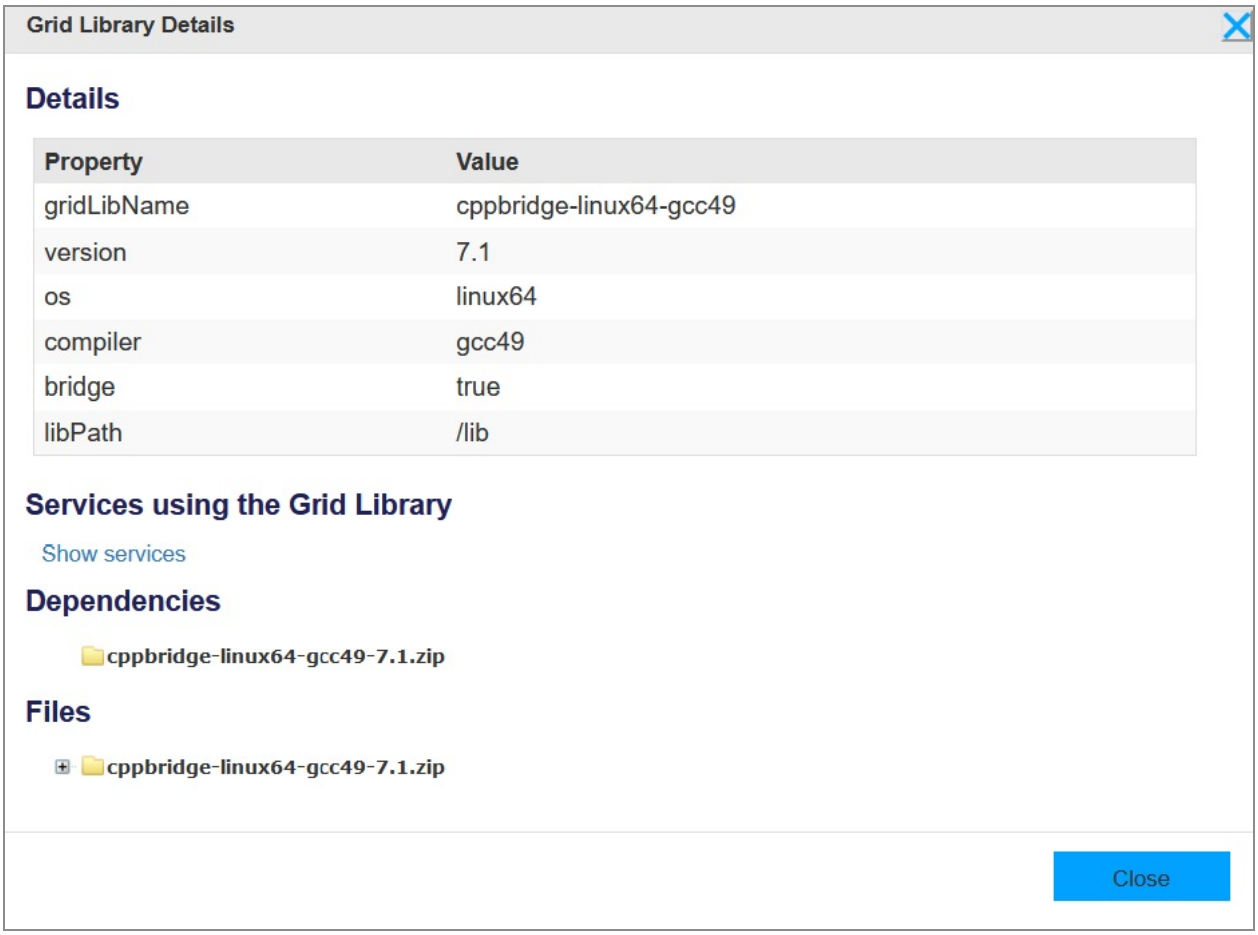
The following conditions cause an error in an uploaded Grid Library:

- A Grid Library in the staging directory has the same name as a Grid Library in the deployment directory. Delete a deployed Grid Library before uploading a replacement. The Administration Tool returns an error message if you attempt to upload a file with the same name as a deployed file. If a resource is in error status, the first line displays attributes of the resource file in the deployment directory, and the second line displays attributes in the staging directory.
- The Grid Library's archive file is corrupt.
- The Grid Library's `grid-library.xml` file does not validate against the Grid Library DTD.

The following buttons can be used on selected Grid Libraries:

- **Deploy** — Deploy selected resources from the staging to the deployment directory.
- **Delete** — Delete selected directories and files from the staging and deployment directory. If the resource is in error status, only its staging copy is deleted.
- **Update** — Force an immediate file signing request and populate changes to the Engines.

You can also show details for a Grid Library by clicking its name in the list. This opens a Grid Library Details window, which displays the following:



The Grid Library Details page

- Properties and values set in the Grid Library's `grid-library.xml` file.
- All Services using the Grid Library, directly or indirectly.
- A list of the Grid Library's dependencies. If there are multiple dependencies, this is a tree that you can browse and can click to view all dependencies.
- A tree of all files and directories in the Grid Library that you can browse.

Grid-Specific Resources for Multiple Grids

If you migrate Engines among multiple grids, you can configure GridServer to use a different resources directory for each grid. This eliminates full Grid Library synchronizations when Engines are migrated among grids, such as when using Federator to manage these grids.

To enable this feature, go to **Admin > System Admin > Manager Configuration > Resource Deployment** and in the Director settings, change the value of **Use Grid Specific**

Resources to true. If set to true, resources are downloaded to `resources/grid-`*gridID* in the Engine installation directory, where *gridID* is the Manager ID of the Primary Director instead of the `resources` directory. (For example, on a Windows Engine, this can be `C:\TIBCO\DataSynapse\Engine\resources\grid-1925105476`.) Note that the Primary Director must be restarted for the change to take effect.

You can also configure how many grids Engines can synchronize with by changing the value of **Number of Grids to Sync**. This value specifies the number of grids that have resources retained on the Engine. If the value is set to N and an Engine already has resources that are downloaded from N+1 grids, the resources from the least recently visited grid are deleted to conserve disk space. This value is a Director-level setting and must be set consistently across multiple grids to reduce resource re-download effectively. Note that all Brokers must be restarted for the change to take effect.

Grid Library Filters

You can limit which Engines synchronize a Grid Library by using Grid Library filters. This enables you to define a filter file in XML that specifies the Engines based on Engine properties. This decreases hard drive usage and network bandwidth in deployments where not all Engines can or are allowed to work on certain types of Services.

A Grid Library filter is an XML file with the same name as its associated Grid Library and ends with the extension `.filter`. When an Engine requests a resources sync, the list of Grid Libraries are filtered based on the matching of the conditions.

Grid Library Filter Document Type Definition (DTD)

A Grid Library filter has the following DTD:

Element	Description	Elements and Attributes
`grid-library-filter`	root element	**Elements** • `property-condition-set*`

Element	Description	Elements and Attributes
`property-condition-set*`	A set of conditions that must be satisfied for the Engine to synchronize	**Elements** • `property-condition*` • `property-condition-set*` **Attributes** • `operator?`
`property-condition*`	A condition that must be satisfied for the Engine to synchronize	**Elements** • `property` • `comparison` **Attributes** • `null-compare?`

A `property-condition` uses the following comparison attributes:

- `contains`
- `notContains`
- `matches`
- `notMatches`
- `equal`
- `notEqual`
- `greaterThan`
- `lessThan`
- `greaterThanEqual`
- `lessThanEqual`

The `operator` attribute can be `and` (the default) or `or`. The `null-compare` attribute can be `false` (the default) or `true`.

Filter Example

The following code is an example of a simple filter which only deploys a given Grid Library to Engines running four or more instances:

```
<grid-library-filter>
  <property-condition-set>
    <property-condition>
      <property>
        <name>numInstances</name>
        <value>4</value>
      </property>
      <comparison>greaterThanEqual</comparison>
    </property-condition>
  </property-condition-set>
</grid-library-filter>
```

Deploying Grid Library Filters

Grid Library filters are uploaded in the GridServer Administration Tool at **Services > Services > Grid Libraries**, using the same control used to upload Grid Libraries. When you upload a filter, it is first validated to ensure it is valid XML and the comparison, property, value, and type elements are valid and `property.name` elements are not empty. Also, there is a 16 KB size limit for filters, which is checked.

After the filter is validated, you are prompted if you want to deploy it. If you click **Cancel**, the filter is not added to the Manager.

Deployed filters are shown in the list of entries on the **Services > Services > Grid Libraries** page, and you can delete them the same way you delete a Grid Library. Unlike Grid Libraries, you can upload a filter with a duplicate name to replace an existing filter with a revised version.

JAR Ordering File

If you are using multiple JAR files and need the classloader to load them in a specific order to prevent conflicts, you can specify the order in which they are loaded. To do this, create a file called `index.libs` in the JAR path root and put the names of the JAR files, one per line, in the order in which they must be loaded. Those not in the list load afterward in no specific order.

Uploading and Deploying with the Admin API

You can programmatically upload, deploy, download, and delete Grid Libraries and filters to a Manager using the JDriver Admin API. The `ServiceAdmin` class has methods you can use to manipulate resources on a Manager. It includes the following related methods:

- `uploadResource` – uploads a resource into the temporary staging directory
- `deployResources` – copies resources from staging to deployed
- `downloadResource` – downloads a copy of a resource
- `deleteResources` – removes resources
- `listResources` – returns a list of resources on the Manager

You can also determine Grid Libraries loaded on Engines by using the `EngineAdmin` class to get the `EngineProperties`. The `GRID_LIBRARIES` property is a list of all downloaded Grid Libraries, and `GRID_LIBRARIES_UPDATE_TIME`, is the time at which they were last updated.

For more information, see the Java API documentation.

Running Services

To run a Service, you must deploy the Service's classes to Engines, and set options that are used when the Service is used. This is done in the Service Type, which specifies the Grid Library containing a Service's implementation, binds classes to Service methods, and sets any Service options. One of the options that can be set is Service Run-As, or the user account that is used to run the Service.

Registering a Service Type

To use a Service, you must first register a Service Type from the GridServer Administration Tool.

To register a Service Type:

1. Go to **Services > Services > Service Types**.
2. A list of existing Service Types appears on that page, along with a line for adding a new Service Type.

3. Enter the Service Type Name on the blank line.
4. Select the Service Implementation, then click **Add**.
5. A window with several options appears after clicking the **Add** button.
   - For Java Service Types, enter the fully qualified class name for the Service.
   - For .NET Service Types, enter the class, assembly, and domain name for the Service.
   - For R Service Types, enter the function interface for the Service.
   - For dynamic libraries, enter the library name for the Service.
   - For commands, enter the command line for the Service.

   The window also allows you to enter options for the Service Type.

> **Note:** After you register a Service Type, you must deploy the implementation to your Engines. If you submit tasks without deploying a Service implementation, the Driver submits the tasks with no error, but no Engines take tasks.

Service Run-As

There are often cases where Services require specific user permissions to access needed resources. By creating the Engine process as a given user, all Service invocations that are run by the Engine can operate with these permissions. **Service Run-As** (or **RA**) allows for specification of authentication domain accounts under which Service invocations run.

By default, all RA credentials authenticate on the Engine Daemon to verify that the credentials are valid for the Engine's authentication domain. You can disable Service RA authentication on the Broker, but do so only when you have a specific reason. For example, if you are using Kerberos or Windows authentication, you must disable this if your Drivers have Negotiate enabled, since there is no password available.

compressData

If you want the initial input and output data to be compressed, you must set the **compressData** field as `True`. To disable the feature, set the value of the **compressData** field as `False`.

Default value: Not set

encryptionEnabled

If you want the initial input and output data to be encrypted, you must set the **encryptionEnabled** field as `True`. To disable the feature, set the value of the **encryptionEnabled** field as `False`.

Default value: Not set

Types of Credentials

You can specify Service Run-As credentials for a given Service in one of two ways: as stored credentials or pass through credentials.

- **Stored Credentials** — Enter Service Run-As credentials on the Director with the GridServer Administration Tool. These credentials are synchronized with all Brokers. These credentials are linked to Services in the Service Type Registry by specifying the username in the **RunAsUser** field. Credentials in the repository consist of a username and a password. The username can be in Windows `DOMAIN/username` format if domain-specific authentication is required. UNIX Engines ignore this domain.
- **"Pass through" Credentials** — The Driver provides the username of the current Principal that is logged in and is running the Driver. The password is provided as a DriverManager property, `CURRENT_USER_PASSWORD`. These are referred to as "pass through" credentials. A password set on the Driver is necessary to prevent user account spoofing between authentication domains. For example, logging in as a local user on the Driver machine to pose as an LDAP user in the credentials DB.

Pass through credentials are indicated for a Service in the Service Type Registry with the `$` token. This token is substituted with the username of the current principal that is running the Driver process. The token might also be prefixed with a Windows domain if domain-specific authentication is required. UNIX Engines ignore this domain.

Note

When you create a Service using pass through credentials and the `Collection.NEVER` Service option, failover does not work under all circumstances.

- If the Broker accepts the Service and the Engine has not yet picked up the tasks, failover works correctly and tasks rerun successfully.
- If the Broker accepts the Service and the Engine has already picked up the tasks, tasks do not rerun successfully, and an Exception is generated. Pass through credentials do not persist through failover.

Using Run-As

To use Run-As, you must do three things: set up Engines, add credentials, and associate credentials with Service Types.

Engine Setup

For information about how to set up Engines for Service run-as, see “Configuring Run-As for Windows Engines” and “Configuring Run-As for UNIX Engines” in the *GridServer Installation Guide*.

Managing Credentials

The Credentials DB is a store of RA credentials on the Director and Brokers used for RA services. It is maintained on the Director and synchronized with Brokers.

The **Run-As Credentials** page in the Administration Tool lets you create, edit, and delete RA credentials.

To add new Credentials to your Manager:

Procedure

1. In the Administration Tool, go to **Admin > User Admin > Run-As Credentials**.
2. Enter the name of a credential, a password, and then enter the same password again.

3. Click **Add**.

Managing Run-As in Service Types

The Service Type Registry entries allow specification of an RA user name for use with that Service. To specify a Run-As user for a Service Type:

Procedure

1. In the GridServer Administration Tool, go to **Services > Services > Service Types**.
2. Select an existing Service Type, click to the **Actions** control, and select **Edit Service Type**. This opens the Service Type Editor window.
3. In the Service Type Editor window, under the **ContainerBinding** header, enter the user name in **RunAsUser**.

Note that in this field, you can use `$` to indicate the Driver's current user. Leaving this value blank (the default) indicates that the process runs as the same user running the Engine Daemon.

If you are adding a user name that contains unicode characters, you must change to the correct code page to match the user name.

The **Specify Additional RunAs User** permission is needed to specify a user other than the Driver or Engine Daemon process.

It is also possible to specify a Windows domain in the **RunAsUser** field. For example, if you are using a UNIX Driver (which is not in a Windows domain) and you want to run Services on Windows Engines using a specific user and domain, you can specify this in the form `domain/username`. The forward slash translates to a backslash. For example, specifying `DATASYNAPSE/BILL` runs Services as the user `BILL` in the `DATASYNAPSE` Windows domain (`DATASYNAPSE\BILL`).

Scheduling

One of the responsibilities of Brokers is *scheduling*, which is the management of Services and tasks on Engines and interactions between Engines and Drivers. This section gives more details on how scheduling works, and the method used to determine what tasks in a Service are sent to what Engines.

Most of the time, the scheduling of Services and tasks on Engines is completely transparent and requires no administration. However, to tune performance, or to diagnose and resolve

problems, it is helpful to have a basic understanding of how the Broker manages scheduling.

Recall that clients create Service Sessions on the Broker. Each Service Session consists of one or more tasks, which might be performed in any order. The scheduler determines the optimal match of Engines to Services. Whenever an Engine reports to the Broker to request work, the Broker assigns a task from that Service to the Engine. When an Engine completes a task, it is queued on the Broker for collection by the client. If an Engine is interrupted during processing, the task is requeued by the Broker.

Reschedules and Retries

Before the discussion of scheduling behavior, we must first define the terms *Retry* and *Reschedule* within the context of scheduling tasks.

Retry

A Retry is when a task is re-queued due to a known failure of the task. Such failures can be due to an error condition in the implementation, an error due to inability to download data, or the failure of an Engine (the monitor has detected that the Engine is no longer connected but it has not logged off.) It is always the result of the Engine returning the task as failed to the Broker. When a task is retried, it is always placed at the front of that session's queue. The scheduler manages a retry count for each task, so that a limit can be placed on the number of allowed retries.

Reschedule

A Reschedule is when a task is re-queued when it might or might not have failed. When a task is rescheduled, it is by default placed at the back of that session's queue, unless the **Reschedule First** configuration option on the Broker is set to true. (Go to **Admin > System Admin > Manager Configuration > Services** to set it.) The scheduler also manages a reschedule count for each task. The following conditions result in a reschedule:

- Engine Logoff: When an Engine logs off gracefully while running a task (such as when UI or CPU idle conditions are met, or there is a forced rebalance), the task is rescheduled, but the reschedule count is not incremented, since there was no task error.
- Redundant Rescheduler: If any of the Redundant Rescheduler strategies are in effect, tasks might be rescheduled to other Engines. By default, those tasks are allowed to

continue to run on the current Engines, in case they finish before the rescheduled tasks. In this case, the reschedule count is increased.

Timeout Behavior

When the `INVOCATION_MAX_TIME` option is set, it specifies that any invocation of a request might not exceed this value. If a task times out on an Engine, it can either retry or be rescheduled, depending on what makes more sense for your application. If retried, the current Engine's invoke process ends, and the task is assigned to another Engine. If rescheduled, the current Engine task continues execution. In either case, the appropriate count is incremented.

The default behavior (retry) is set on the Broker. It can also be set for the Service Type on the Service Type Registry page, or programmatically when the Service Session is created.

The specific timing involved with a retry/reschedule depends mainly upon three properties: The Task Max Time, the scheduler interval and the Engine heartbeat.

The moment the task is picked up by the Engine, the start time is marked. The scheduler wakes up at least once in every scheduler interval seconds to check for any tasks in progress that exceed the max time. If **Reschedule on Timeout** is false (the default), the scheduler logs off Engines that have timed out, causing the tasks to retry immediately. The tasks are placed at the top of its session's queue. Note that the Engine on which the task is running does not restart itself until the next message is sent, typically a heartbeat. If true, those tasks are redundantly rescheduled, and Engines that have timed out are allowed to continue; the task is complete as soon as any Engine completes it.

In general, the maximum time is the Task Max Time plus the Poll Period of Service Rescheduler. For instance, if Task Max Time = 50 sec and Poll Period= 60 sec, best = 50 secs and worst = 110 secs. However, it can take up to the Engine Heartbeat for the Engine on which a task was retried to log back in.

The Scheduler

The scheduler is the component that is used on a GridServer Broker to assign tasks to Engines. It attempts to make optimal matches based on criteria such as the session priority level or SLA group, affinity, and Serial Service and Priority execution modes.

Scheduler Overview

The scheduler aims to schedule tasks to Engines by attempting to have the proper amount of Engines allocated to all active Service Sessions at any given time. A *Scheduling Event* is any event that might result in a task being assigned to an Engine, such as an Engine finishing a task, an Engine logging in, or new tasks added. On any given scheduling event, the scheduler decides the number of Engines each Session must have at the time, based on static and dynamic criteria, and then assigns the appropriate number of Engines to sessions based on how many the Session needs to reach the ideal level.

There are two modes under which the Scheduler can operate: **Priority**, or **Service Level Agreement (SLA)**.

Priority Mode

Every GridServer Service has an associated **priority**. By default, there are ten priority levels, ascending in priority 0-9. When set to 0, the Service is suspended, meaning that no tasks are assigned. Priority can also be set to **Urgent**, which is covered later in this section. The priority is set when a Service is created. It can be changed at run time in the Administration tool and by the Admin API.

A **Priority Weight** is associated with each Priority Level. The weight defines the amount of Engines allocated to a session relative to all other active sessions. For example, if Session A and B have weights of 2.0, and Session C has weight 4.0, and there are eight Engines, Session A and B get allocated two Engines each, and Session C gets four. To set the weights, go to **Admin > System Admin > Manager Configuration > Services** and change the **Priority Weights** property. By default the weights are linear.

The number of priority levels can be changed. However, a large number of priority levels can impact performance when Serial Priority is not enabled, so it is recommended that the Serial Priority is enabled in this case.

There are two algorithms that are used in Priority mode, **Usage**, which allocates Engines to all running Services as fairly as possible, and **Time**, which simply allocates Engines to Sessions in the order in which they were created. Also, when **Serial Priority Mode** is enabled, Sessions of a higher priority are assigned Engines when needed before lower priority Sessions. By default, Usage is used with Serial Priority Mode disabled.

Usage Algorithm

The scheduler takes into account the amount of usage that the Session has received over a given historical window of time. The “usage” refers to the amount of Engine clock time that

the Session has occupied during that window. When a Session is created, it is initialized in such a way that it simulates as if it was running ideally over this window.

This usage provides the ordering in which Engines are allocated to Sessions. This addresses starvation issues, round-off error (the number of ideal Engines is rarely an integer), and under/over-utilization due to discrimination, changes in the number of available Engines, and so on.

On a scheduling event, Sessions are assigned the ideal number of Engines less the amount that are currently allocated, in the order of least to most usage.

This approach can be seen as analogous to a CPU thread scheduling algorithm. Each Session is a “thread”, the Engines are the “CPU”, the window is the sample period, and each task is an uninterruptible unit of CPU time allotted to a thread.

Whenever an Engine or set of Engines is available for scheduling, the scheduler decides how many Engines each session must be allocated. In general, that value is:

Ideal Engines per Session = All Engines * Session Priority Weight / Total Weight,

where “Total Weight” is the sum of all Priority Weights of active sessions. This value is rounded up to the next integer to prevent starvation for an ideal calculation of < 0.5, and assures that the sum of Ideal Engine’s is always at least as large as Total Engines. This algorithm also takes into account if the actual number of Engines that can be allocated is less than the ideal, such as when a Session is towards the end, or when Max Engines is used.

Recall that a Session’s usage is considered to be the total Engine clock time spent on the session over the last configurable amount of time. This includes running and completed tasks. When a Session is created, it must initialize its usage. The simplest, most fair method of doing this is to assume it has been operating in a steady state over the window with the ideal non-rounded number of Engines. The variables that monitor usage are then initialized as such. If no sessions are active, it initializes them such that the session’s ideal is the total number of Engines currently on the Broker.

Whenever there is an event that requires a scheduling episode, the scheduler assigns the proper number of Engines to each session for it to be at its ideal amount. This assignment is performed in order of least to most priority-normalized usage. If there are any unassigned Engines remaining after this initial round based on usage (typically due to disallowed conditions preventing assignment), a second tier round robin assignment is performed.

Time Algorithm

The Time algorithm is used by setting Serial Service Execution to true. This algorithm works as follows:

When a Session is created on the Broker, it is placed in the queue. On each scheduling episode, the scheduler simply iterates through the queue and assigns all idle Engines it can to each Session. Normally, only the first Session is assigned Engines, except when that Session is finishing up, or if Discriminators prevent Engines from running on that Session. A Session keeps its place in queue until it is destroyed regardless of whether or not it has tasks in queue or running.

Serial Priority Mode

When Serial Priority Mode is enabled, the scheduler ensures that Sessions are assigned Engines in order of priority. The scheduler iterates through each priority level in descending order, and assigns as many Engines to Sessions at that level as possible. Either the Time Algorithm or the Usage Algorithm, depending on whether Serial Service Execution is enabled, is used on the subset of Sessions at the same Priority level. Note that this means that Priority takes precedence over creation time.

Intrinsic Affinity

The scheduler uses the fact that an Engine has initialization data and updates from a particular Service to prioritize routing of subsequent requests to that Service. This feature, called *affinity*, reduces data movement, because unneeded Engines are not recruited into the Service. For example, Engine A has worked on Session X and Engine B has not. If both are idle and a task is submitted by X, Engine A is assigned the task. However, if Engine A is busy, Engine B is assigned the task. You can also use the `AFFINITY_WAIT` Service option to control how long a queued request avoids allocation to an available Engine that has no affinity, in the hope of later being matched to an Engine with affinity.

Affinity is not used when using the Time algorithm.

For more information about tuning or customizing how the scheduler uses affinity, see Optimizing the Grid.

Priority Aggregation

Priority Aggregation is a setting that can be enabled for the usage algorithm. When enabled, the amount of Engines to be allocated is now aggregated over the entire group of

Sessions running at a priority level, rather than per Session. That is,

Ideal Engines per Session = All Engines * Session Priority Weight / Total Weight / Sessions at Priority

This mode is used when you want to guarantee a known distribution of Engines amongst priority levels regardless of how many Sessions are running at that level.

Example:

With 100 Engines total, 1 Session at level 6 gets 60, and 1 Session at level 4 gets 40.

Without priority aggregation, if another level 4 Session is added, each level 4 Session now wants 29, and the level 6 wants 43. With it enabled, the level 6 Session still gets 60, and each level 4 Session gets 20.

Urgent Priority Services and Preemption

A Session's priority can be set to **Urgent** when that Session must be serviced immediately, even preempting running tasks if necessary. An urgent Service's weight is hard-coded to be essentially infinite, so that they are assigned all available Engines.

When an Engine is preempted, the task it is currently running is canceled and rescheduled, and the Engine becomes available for new tasks. Engines are preempted on a Service under the following conditions: if after being assigned all free Engines a Service can still make use of more Engines, then it might preempt some busy Engines, subject to two constraints that can be adjusted with configuration properties. First, the urgent Service must have been in the queue for **Preempt Delay** Seconds. Second, the percentage of Engines in the grid running urgent Services cannot exceed **Preemptable Engine Percent**.

For example, if this property is set to 50, and 47 percent of the Engines are currently running urgent Services, then at most three percent are preempted. This value is not a hard limit on the number of Engines that might be running urgent Services, because free Engines are allocated to urgent Services regardless of how many Engines are already running urgent Services.

The scheduler chooses Engines for preemption based on the following rules: Engines running an urgent Service are never preempted. An Engine running a task from a Service with lower priority is generally selected in preference to one running a higher-priority task. However, if the lower-priority task has been running for a long time, a short-running, higher-priority task might be preempted instead. The **Preempt Threshold Minutes** property determines the value at which this crossover happens. For example, if this property is set to 30, then an Engine that has just started running a priority 2 task is chosen for preemption over an Engine that has been running a priority 1 task for more

than 30 minutes. The formula is as follows: `priority + (runningMillis / preemptThresholdMillis)`.

> **Warning** Preemption can have a significant performance impact on your grid and cause scheduling problems with other Services. It must be used with caution.

Other important points concerning priority Services and preemption:

- Tasks canceled by preemption are not subject to a rescheduling limit, since they are not considered failures.
- To prevent preemption from ever occurring, set **Preemptable Engine Percent** to 0.
- The first Service on the queue might not get all free Engines if it doesn't have enough tasks, it is already using its maximum number of Engines, or it discriminates against some Engines. Free Engines that are not taken by the first urgent Service are first offered to the other urgent Services on the queue, and then to all other Services.

SLA Mode

When the SLA mode is used, the scheduler guarantees that a number or percentage of Engines on the Broker is allocated to a group of Services, provided enough Engines are available.

To enable SLA scheduling, go to **Admin > System Admin > Manager Configuration > Services** and set **SLA Scheduler Enabled** to true. Then you must define the Broker's **SLA Groups**, which is a comma-delimited list of groups and values. The values must be either all integers, or all floating points where 0 < x <1. When integers, it indicates that the SLA is the actual number of Engines, otherwise it is the percentage of Engines currently logged in to the Broker.

For example, setting SLA Groups to `a=10,b=12,c=20` specifies that group A's target is 10 Engines, group B's target is 12 Engines, and group C has a 20-Engine target. Alternatively, an SLA Groups setting of `a=.25,b=.5,c=.1` sets the A, B, and C's targets to 25%, 50%, and 10% of Engines, respectively.

Sessions are assigned to groups by setting `Description.SLA_GROUP_NAME` on the Description. All Sessions must have this set; if not set or set to an invalid group name, the Session is rejected by the Broker.

On every scheduling event, the scheduler calculates how many Engines must be allocated to each SLA Group, and then assign Engines to Sessions as follows:

- The entire list of Sessions is ordered by the same **Usage** algorithm used in the Priority mode. This ensures fairness in the steady-state. Scheduling is performed round-robin within an episode, starting at the Session with the least amount of usage.
- When a Broker has enough Engines to meet all group SLA Engine needs, Engines are first assigned so that all SLA Groups have their SLA number met. Then any remaining Engines are divided up among the groups, weighted by their SLA numbers, rounded to the next integer.
- When a Broker does not have enough Engines to meet group SLA Engine needs, the Engines are allocated to groups by weight as in the case of the remainder when there are enough Engines.

For example, you have set **SLA Groups** as `a=20, b=30, c=50`, and there are nine remaining Engines after SLA Engine needs are met. 9 * 20 / 100 = 1.8, so group A gets 2 remainders. Likewise, group B gets 3, and group C gets 5. Note that in this case, one of the groups gets one less than their number due to rounding, but the usage algorithm corrects these inequities in the long run.

The SLA scheduler does not take Affinity into account, nor is there any analog to Urgent Priority.

SLA Task Preemption

In the case of long-running tasks, you can use the task preemption option with the SLA scheduler to better distribute Services to idle Engines. When the **SLA Preemption Enabled** property is set to true, when one SLA group is idle, other groups might be allocated its Engines. If Services are then added to the idle group, tasks are preempted to reallocate Engines needed to meet its SLA.

Common Scheduler Features

The features in this section are common to both scheduler modes.

Grid Library Aware Scheduling

The GridServer scheduler does not schedule a task to an Engine if that Engine does not yet have the root Grid Library for that Service. Additionally, when an Engine logs in, it does not wait until it is synchronized to run tasks. Rather, it works on any Services it can while it is synchronizing any new libraries. This allows for straightforward library deployment on large grids where it might take hours to fully sync. Services can be started at any time instead of waiting until all Engines are synced.

Engine Blacklisting

If a Service sets the option `engineBlacklisting` (`ENGINE_BLACKLISTING`) to true, then Engines that fail on a task from that Service do not receive any other tasks from that Service. The default is false. '"fail" means any action that results in a failed task being sent back to the Manager, regardless of whether that failure was due to Engine hardware, Engine environment, or Service implementation code. It does not include events such as the Engine going offline to user activity, since that does not result in a task failure.

You can also set the option `failuresBeforeBlacklist(FAILURES_BEFORE_BLACKLIST)` to a number of task failures before an Engine is blacklisted.

Blacklisted Engines are excluded for a particular Service Session only. They can freely accept tasks from any other Service, regardless of Service Type, assuming the other Services haven't also blacklisted the Engine or have some Conditions in place that prevent it. Blacklisted Engines can also be shared with other Brokers that need Engines.

To remove an Engine from all blacklists, go to **Grid Components > Engines > Daemon Admin** and select **Clear from Blacklists** from the **Actions** list.

You can get a list of blacklisted Engines using the GridServer API. In Java, the `getBlackListedEngines` method in `com.datasynapse.gridserver.admin.ServiceAdmin` retrieves a list of Engines that have been blacklisted for a given Service. It is also available for C++ and .NET. See the GridServer API for more information.

Engine Greylisting

If your Tasks are more heterogeneous, you might not want to use blacklisting, because a single Task failure might not imply failure of all Tasks. Greylisting enables you to make it less likely that an Engine works on a Service Session than other Engines, without completely excluding that Engine. This is done by lowering affinity for an Engine for the Task retry.

To use Engine Greylisting, set the Service option `engineGreylisting` (`ENGINE_GREYLISTING`) to true. When a Task fails, the Service Affinity for that Engine is reduced by a configurable amount, which by default is 5. To change this amount, go to **Admin > System Admin > Manager Configuration**, and under the **Affinity** heading, change **Greylist Affinity** to a negative value.

Greylisting can be used with blacklisting, if the `failuresBeforeBlacklist` is greater than zero. The Service Affinity is reduced upon each Task failure until `failuresBeforeBlacklist` is reached and the Engine is blacklisted.

The **Clear from Blacklists** action described above also clears greylists.

Engine Properties

Engines have a number of intrinsic properties, such as available memory or disk space, performance (megaflops), operating system, and so forth, that the condition can use to define eligibility. Custom properties can also be defined on the grid, and property values can be assigned on a per-Engine basis. These properties are used in a number of the following features.

Engine Tiers

Often time customers might have distinct sets of resources that they might want to use on a preferred basis. Engine tiers provide a mechanism to specify the order in which groups of Engines must be scheduled.

For example, you might have a set of high-performance dedicated blades, a pool of older servers, and a group of desktop computers used as part-time grid resources when idle. Ideally, you might not want to use older servers unless the high-performance servers are completely in use, and avoid using desktop machines unless both other groups were busy.

Engine tiers are defined on the Broker at **Admin > System Admin > Manager Configuration > Services**, under the **Scheduling** heading, with the **Engine Tiers** property. The property's format is an ordered, comma-delimited list of Engine property name-value pairs. For example, for the above scenario, if you had a `type` Engine property, you might use `type=blade,type=server,type=desktop`. The list is ordered from highest-tier to lowest-tier. Engines matching the first tier are always scheduled before those in the second tier, and so on.

If an Engine matches more than one property, the highest tier is matched. If it matches none of the defined tiers, it is scheduled after all other tiers.

Conditions

Conditions are rules that are applied to Services and tasks that affect how they are scheduled to Engines.

A *Discriminator Condition* limits the execution of tasks to a subset of Engines. If an Engine is ineligible to take the next waiting task, it is assigned the first task it is eligible to take. An *Affinity Condition*, like intrinsic affinity, provides for prioritized routing of tasks to Engines, but does not prevent any Engines from taking tasks. These conditions typically are based on Engine Properties. Users can also implement custom versions of these conditions. (Note that Affinity is only used in Priority Mode with the Usage algorithm.)

Task Affinity provides the ability to run a set of Tasks on the same Engine or set of Engines. For example, if a Task loads a large dataset, and a number of subsequent Tasks use the same dataset, you can add a Task Affinity Condition to those Tasks so they prefer to run on the Engine that ran the first Task.

When a *QueueJump Condition* is added to a task submission, the task is added to the front of the Session's queue so that it is the next task taken.

Dependency Conditions are used to create workflow amongst Sessions and tasks. A task can be set to wait until another task or Session completes; likewise with Sessions. Dependent tasks or Sessions can also optionally fail if the dependency fails.

For more information about using Conditions, see the *GridServer Developer's Guide*.

Redundant Rescheduling

Redundant rescheduling addresses the situation in which a handful of tasks, running on less-capable processors, might significantly delay or prevent Service completion. The basic idea is to launch redundant instances of long-running tasks. The Broker accepts the first result to return. Remaining instances are not immediately canceled; it waits to either finish, or waits until the Service finishes. Redundant rescheduling is also useful when completion of long running tasks is critical.

By default, redundant task rescheduling is not enabled. With pools of more capable or nearly identical Engines, fastest task execution occurs when there is no redundancy from rescheduling. In general, rescheduling is only appropriate when there are widely different capabilities in Engines. To enable redundant rescheduling, you must enable one of the three strategies, and set the `REDUNDANT_RESCHEDULING_ENABLED` Service option to true on each Service you want to redundantly reschedule.

Note In situations where a group of Engines might slow down a running task, using discrimination can be more efficient than redundant rescheduling.

Three separate strategies, running in parallel, govern rescheduling. Tasks are rescheduled whenever one or more of the three corresponding criteria are satisfied. However, none of the rescheduling strategies apply for any Service until a certain percentage of tasks within that Service have completed. The `Strategy Effective Percent` property determines this percentage.

The rescheduler scans the pending task list for each Service at regular intervals, as determined by the `Poll Period` property. Each Service has an associated `taskMaxTime`, after which tasks within that Service are rescheduled. When the strategies are active (based on the `Strategy Effective Percent`), the Broker tracks the `mean` and `standard deviation` of the (clock) times consumed by each completed task within the Service. Each of the three strategies uses one or both of these statistics to define a strategy-specific `time limit` for rescheduling tasks.

Each time the rescheduler scans the pending list, it checks the elapsed computation time for each pending task. Initially, rescheduling is driven solely by the `taskMaxTime` for the Service; after enough tasks complete, and the strategies are active, the rescheduler also compares the elapsed time for each pending task against the three strategy-specific limits. If any of the limits is exceeded, it adds a redundant instance of the task to the waiting list. (The Broker resets the elapsed time for that task when it gives the redundant instance to an Engine.)

The `Reschedule First` flag determines whether the redundant task instance is placed at the front or the back of the waiting list. That is, if `Reschedule First` is `true`, rescheduled tasks are placed at the front of the queue to be distributed before other tasks that are waiting. The default setting is `false`, which results in less aggressive rescheduling.

Each of the three strategies computes its corresponding limit as follows:

- The `Percent Completed Strategy` waits until the Service nears completion (as determined by the `Remaining Task Percent` setting), after which it begins rescheduling pending tasks that are taking longer than the average completion time for tasks within the Service.
- The `Average Strategy` returns the product of the mean completion time and the `Average Limit` property. That is, this strategy reschedules tasks when their elapsed time exceeds some multiple (as determined by the **Average Limit**) of the mean completion time.

- The `Standard Dev Strategy` returns the mean plus the product of the `Standard Dev Limit` property and the standard deviation of the completion times. That is, this strategy reschedules tasks when their elapsed time exceeds the mean by some multiple (as determined by the `Standard Dev Limit`) of the standard deviation.

Managing Engines

This section contains information for managing GridServer Engines. For information about installing Engines, see the *GridServer Installation Guide*. For information about troubleshooting Engine issues, see Diagnosing Engine Issues.

Engine Routing and Balancing

Engines are dynamically allocated resources. They can migrate among Brokers based on criteria such as load and policy. Use the Engine Balancer to manage log ins and re-route Engines to maintain an optimal balance across the grid. The Engine Balancer is a component of the Director. The Primary Director's balancer always runs. The Secondary Director's balancer runs only if the Primary is down.

The Director handles routing and load balancing as follows:

1. The Director regularly polls Brokers for the states of Engines on the Brokers. The Director tests routing mechanisms against each Engine and determines the optimal location for each Engine. Changes in the states of Engines due to load balancing requirements result in changes in the optimal distribution of Engines on Brokers.
2. The Director sends a request to each Broker that has Engines that must be moved, to log those Engines off.
3. Engines that must return to their home Broker log out immediately, regardless of the task timeout setting.
4. Shared Engines that are busy restart immediately without finishing the current task. Engines that are not busy log out immediately.
5. After an Engine logs off or restarts, it then logs in to the optimal Broker.

Two balancer algorithms are available. Choose one according to how you plan to use the grid:

- The `weight-based` balancer algorithm attempts to distribute Engines equally by relative weights, and it also allows rule-based routing using Engine properties.
- The `Home/Shared Balancer` routes Engines based on an Engine's assigned Home Brokers, and the sharing policy of Home Brokers to other Brokers. Both balancers

take into account the number of running and pending tasks on each Broker, and the desired maximum and minimum number of Engines for each Broker.

If you change the Engine Balancer on the Director, you must restart it. Also, all balancer settings must be equal on Primary and Secondary Directors. You can configure routing settings for online or offline Brokers.

Balancing and Service Discriminators

When the Director polls Brokers for Engine information, it also collects information about Service Discriminators and blacklisted Engines. The Director avoids a situation where Engines are at their home Broker and cannot take Services due to Service discrimination or blacklisting, but also it is not shared with another Broker. (Task-level discriminators are not taken into account.)

To report a Service discriminator, all of the following conditions must be true for a Service:

- The Service is not complete.
- The Service has pending tasks.
- The number of busy Engines working on the Service does not exceed the max Engines option of the Service.

To limit CPU and network usage during balancing, the maximum number of discriminators reported by each Broker can be configured on the Director. You can configure it at **Admin > System Admin > Manager Configuration > Engines and Clients > Max Service Discriminators**. This setting specifies the number of Services with discriminators per Broker that are considered.

If a Broker has more outstanding services with Service discriminators than the maximum specified, a message is logged in the Broker logs, similar to the following message:

```
INFO: [EngineSharing] Maximum number of discriminators 10 is reached
when collecting service level discriminators
```

When this occurs, the remaining discriminators over max (ten in the above example) are not considered when allocating Engines to this Broker. This does not prevent the Engines from being reallocated to the Broker even if the Engines do not satisfy the discriminators. If Max Service Discriminators is set to 0, no discriminators are considered.

If the Max Service Discriminators is set to a very high value, reporting of balance data from the Broker to the Director takes more time. This slows down the balancing. If balancing is slowed down to an unacceptable level, lower the value of the parameter.

Engine Weight-Based Balancer

The Engine weight-based balancer allocates Engines to Brokers based on each Broker's Engine weight value. This value is the amount of Engines that are allocated to the Broker relative to the other Brokers' weights, when all Brokers are idle. The algorithm also considers session load and reallocates idle Engines to busy Brokers as they are needed. You can see a Broker's Engine weights value on the **Grid Components > Brokers > Broker Admin** page.

When it is necessary to restrict some Engines to a set of Brokers, the Engine weight-based balancer permits rule-based routing through Engine Properties. You can route Engines by their intrinsic properties, such as `cpuTotal`, and by user-defined properties. Create and assign user-defined properties with the **Grid Components > Engines > Engine Properties** page. Use the **Grid Components > Brokers > Broker Routing** page to set up routing rules based on these properties.

Home/Shared Balancer

The Home/Shared Engine balancer uses an algorithm in which an Engine has a set of Home Brokers that it always works on while it has outstanding tasks, yet the Engine can be shared with other Brokers when there are no outstanding tasks on any home.

The balancer uses Broker needs and Engine preferences for Brokers to allocate Engines to Brokers. Each Engine divides the existing Brokers into tiers (unordered sets of Brokers). The two default tiers are:

- The Engine's home Brokers
- The shared Brokers of those home Brokers

You can introduce a third tier by splitting shared Brokers into two groups: preferred shared brokers and common shared Brokers. The higher the tier, the more the Engine prefers Brokers in that tier.

The balancer uses the following rules:

- An Engine is routed to the highest-tiered Broker that has pending tasks. If multiple Brokers in the same tier have pending tasks, the choice is made at random, as if all weights were 1. The number of Engines moving to a Broker is capped at the number of pending tasks if there are more pending tasks than available Engines in the tier.
- An Engine leaves its current Broker only if there is a needy Broker in a higher tier. An Engine does not move to a lower-tiered Broker unless it is idle.
- Failover Brokers are never allocated Engines unless they have pending tasks.
- If `Options.MAX_ENGINES` is set on a Service and the number of Engines from other Brokers are not shared with this Broker to run the Service. If there are more Engines on a Broker than the sum of all `MAX_ENGINES` values across active Services, the excess Engines are reported as available for sharing.

Use the GridServer Administration Tool to configure Brokers. Configure an Engine's home Broker with the **Grid Components > Engines > Engine Configurations** page. Configure Broker tiers (which Brokers share Engines with other Brokers) with the **Grid Components > Brokers > Broker Configuration** page as follows:

1. To set the first tier of Brokers, fill in the **Preferred Broker Sharing** field. Supply a comma-delimited list of the Brokers with which the current Broker shares Engines.
2. To set the second tier of Brokers, fill in the **Common Broker Sharing** field. Supply a comma-delimited list of other Brokers with which the current Broker shares Engines.
3. You can also define additional tiers of Brokers by delimiting them with semicolons in the **Common Broker Sharing** field. For example, to enter a second and third tier of Brokers, you can enter the list B1, B2, B3; C1, C2, C3.

For example, an Engine configuration's home Brokers are A and B. A's preferred broker is C; its common list is D,E. B's preferred Broker is F; its common list is G. An Engine with this configuration uses the following preferences: first: A, B; second: C, F; third: D, E, G. Within each tier, Brokers are equal, and ordering doesn't matter.

You can also use the Admin API to get or set the tiers. In `com.datasynapse.gridserver.admin.BrokerAdmin`, use the methods `setSharedBrokers` and `getSharedBrokers` to set or get the tier string. You can also create a Batch Definition that uses the Admin API to change the tiers according to a time schedule. For more information about using the Admin API, see the *GridServer Developer's Guide*.

On the **Grid Components > Brokers > Broker Configuration** page, you can also set a minimum number of idle home Engines for a Broker by adding the **Min Idle Home Engines** property column to the page. If the idle home Engine count is below this value, home Engines (idle or busy) are not logged off or shared to other Brokers.

Engine Balancer Configuration

To configure Engine Balancing on the Director, go to **Admin > System Admin > Manager Configuration > Engines and Clients**, and change the following properties:

Setting	Description
Engine Balancer	The Engine balancer to use: Weight-Based or Home/Shared.
Rebalance Interval	The amount of time, in seconds, between balancing episodes. (Previously called the Poll Period.)
Logoff Timeout	The amount of time in seconds that an Engine waits to finish a task before logging off.
Broker Query Timeout	The maximum time to wait for Broker's reply for balancer data queries. The value is in milliseconds.
Soft Logoff	If true, Engine logoffs restart the JVM only when needed by a home Broker. This enables them to retain state and log in faster.
Engine Fraction	The fraction of extra Engines that moves to another Broker on a balance. This can be set to less than 1 to dampen Engine movement. For example, if the fraction is 0.5 and the balancer determines that a Broker has eight extra Engines, it moves four on the first balance. Assuming those Engines move, on the next balance it determines that there are four extra and moves two, and so on.
Engine Balance Maximum	The maximum number of Engines that can move to another Broker on a rebalance. The maximum applies over the entire grid. For example, if this property is set to 100 and the balancer determines to rebalance 200 Engines (after taking Engine Balance Fraction into account), then only 100 Engines are actually rebalanced.
Max Service Discriminators	The maximum number of service discriminators to consider when reporting balance data from each Broker.
Allow Routing	If Broker Routing properties are used when the Sharing balancer is

Setting	Description
When Sharing	enabled. Under most circumstances, when using the Sharing balancer, it is best not to use routing.
Treat Pinned Engines as Busy	If pinned Engines are reported as busy from a sharing point of view. In some use cases, treating a pinned Engine as a busy Engine can reduce Engine fluctuation and improve overall grid performance. Note that this setting affects Engine sharing only. For information about Engine Pinning, see the *GridServer Developer's Guide*.

These settings must be identical on all Directors.

Engine Upper and Lower Bounds

You can configure upper and lower bounds on the number of Engines that can be logged in at a given time. Set these upper and lower bounds on the **Grid Components > Brokers > Broker Admin** page. If the columns for bounds are hidden, add them using the **Column** control. The minimum value specifies that the balancer algorithm always leaves at least this amount of Engines (assuming there are this many) on the Broker unless the Engines are needed by Brokers in higher sharing tiers. The maximum value is the cap on the total number of Engines to allow on the Broker. The balancing algorithms use both values.

Failover Brokers

A Failover Broker temporarily takes over running Service Sessions when the Client has no other Brokers to which it is permitted to connect. From the Client perspective, Failover Brokers become part of the pool of active Brokers when there are no other non-Failover Brokers on which the client is permitted. From the Engine perspective, a Failover Broker becomes part of the active pool when there are active sessions in progress on that Failover Broker. In either case, the algorithm now views this Broker as a non-Failover. It is important to take Failover Brokers into account when setting up the routing configuration. For example, if you are setting up a role to allow a client on only one Broker under normal conditions, you must also include a Failover Broker in its Manager List if you wish this client to have a failover if its main Broker goes down.

See Grid Fault-Tolerance and Failover for more information.

Example Use Cases

This section describes example use cases of client routing and load balancing.

N+1 Failover with Weighting

An organization has four groups using all available Engines in a grid. One group has a guaranteed allocation of at least half of the grid anytime that it needs it, and the other three groups share the remaining Engines.

- Brokers - set up five Brokers. Each group gets a Broker, plus one is used for failover.
- Drivers - create four roles, one for each group. In each role, set the **Manager List** value to the group's Broker and the failover Broker. Assign the roles to the appropriate users.
- Engines - use the weight-based Engine Balancer. Adjust **Engine Weight** on the Broker Admin page so the first group's Broker is weighted at 3.0, and the other three groups' Brokers are weighted as 1.0. Set the failover Broker weight at 1.0, so that a group is not assigned any more resources than normal if their Broker goes down.

Engine Localization with Sharing

A company has two groups, one in New York and one in London. Each has a single middleware application that has a Driver that connects to its own Broker. Each group also has a set of CPUs that it expects to always be working on their own calculations. However, there are times when one group's Broker is idle, so they are allowed to share with each other.

- Brokers - set up four Brokers, a regular and a failover for each group. Each regular Broker shares with the other regular Broker, plus its own failover Broker.
- Drivers - create two roles, one for each group. In each profile, set the **Manager List** value to the group's Broker and its failover Broker. Assign the roles to the middleware application user.
- Engines - use the Home/Shared Engine Balancer. Set up two Engine Configurations, "London" and "New York", that set the Engines' homes to their respective Brokers.

In this scenario, the application always connects to its local Broker, unless it is down, in which case it moves to its failover. Whenever that Broker has pending requests, all of its

Engines are always local. If the other group's Broker is idle, or if it does not need all of its Engines, any of its idle Engines are routed to the Broker that needs it.

Engine Configuration

Engine and Engine Daemon behaviors are controlled by a centralized profile called an *Engine Configuration*. When new Engine Daemons are installed, they use the default Engine Configuration for their platform (Windows, Linux, or other supported platforms.) Individual Engine instances take the configuration of their Engine Daemon. You can edit existing Engine Configurations, or create Engine Configurations for different subsets of Engines on your grid.

Editing an Engine Configuration

To change an Engine's settings, for example, to point it to a new Director, you can edit the Engine Configuration that is used by the Engine. Settings change on all Engines using that Configuration.

⚠ **Warning**	Changing an Engine Configuration causes Engines using that Configuration to restart.

To change values in an Engine Configuration:

1. In the GridServer Administration Tool, go to **Grid Components > Engines > Engine Configurations**.
2. Select a configuration in the Configuration list. The list contains the names of all default Engine Configurations, plus any custom configurations that have been created.
3. Change any of the values that appear in the Engine Configuration.
4. Click **Save** to keep your changes, or **Cancel** to revert changes.

⚠ **Warning**	When adding values to the Environment Variables box, it is possible to set variables, particularly `PATH`, that can cause an Engine to fail to start.

Creating a New Engine Configuration

In some situations, you might need a subset of Engines to behave differently than other Engines on your grid. For example, you might want a large group of Engines to report to a different Director, or use a different compiler runtime. Instead of individually configuring each Engine, you can create a new Engine Configuration with different Director settings, and assign the new Configuration to a subset of Engines.

To create an Engine Configuration:

1. Go to **Grid Components > Engines > Engine Configurations**, and click **Add**.
2. Select a platform for the Engine Configuration from the list.
3. Next to the selected platform, enter a name for the Configuration.
4. Click **Create**.

The initial values of the new Engine Configuration are the same as the default Configuration for the selected platform. You can then change the values of the configuration and click **Save**.

Copying an Engine Configuration

In a situation where you need several similar Engine Configurations, it's often faster to make a copy of an existing Engine Configuration, rather than creating many individual Configurations.

To copy an existing Engine Configuration:

1. Go to **Grid Components > Engines > Engine Configurations**.
2. Select an existing Engine Configuration from the list.
3. Click **Copy**. You are prompted for a name for the new Configuration.

Setting the Engine Configuration Used by Engines

To change the Engine Configuration of an Engine Daemon

Procedure

1. Go to **Grid Components > Engines > Daemon Admin**.
2. In the **Configuration** column, select an Engine Configuration from the list next to an Engine Daemon.

To change a large number of Engine Daemons' Configuration at once

Procedure

1. Go to **Grid Components > Engines > Daemon Admin**.
2. Use the **Results Per Page** and **Search** controls to limit the table to only the Engine Daemons you want to change. For example, enter win32 in the search box and select **OS** as the column to search, then click **Go**. If you have more than twenty win32 Engines, enter 100 in the **Results Per Page** box before you do your search. The goal is to get all of the Engine Daemons you want to change on one page.
3. In the **Actions** list, select **Configure Daemons on Page**.
4. The **Engine Daemon Editor** page opens, displaying all of the Engine Daemons previously in the table.
5. Select an Engine Configuration from the **Configuration** list.
6. Click **Save**.

Setting the Director Used by Engines

The Primary and Secondary Directors for an Engine are set during Engine installation. You can later change the Directors to which an Engine reports, by changing the Engine Configuration used by the Engine.

For Linux64 Engine

Prior to moving Engines from all versions earlier than 7.0.0_hotfix08 to 7.1.0, you must perform the following steps:

1. Stop the server.
2. As a best practice, take a backup of the existing `engine.sh` file.
3. Replace `engine.sh` file - from the `engineScriptUpdate` folder to

`datasynapse/manager/webapps/livecluster/engineUpdate/linux64/` and `datasynapse/manager-data/engineUpdate/linux64/` folder.

4. Start the server.
5. Let the Engines log in and then restart the Engine Daemon.
6. Move the Engines.

> **Note:**
> In case of Grid Architecture, perform the steps from step 1 to step 5 on the Primary Director.

Configuring an Engine's Director

1. Go to **Grid Components > Engines > Engine Configurations**.
2. Select the Engine Configuration used by the Engine. This is typically the operating system of the Engine.
3. In the **Directors and Brokers** section, change **Primary Director URL** and **Secondary Director URL** to the corresponding addresses and ports of the Primary and Secondary Directors, in the format `http(s)://address:port`.

Note that this changes the Directors for all Engines using that Engine Configuration. Also note that when moving Engines from one grid to another, Engines lose any custom Engine properties that are defined on the former Director’s grid, because those are stored in the grid’s administrative repository.

Configuring Engines With Multiple Network Adapters

In some network configurations, the host running an Engine can have more than one physical or logical network interface. When the Engine starts, it checks what IP address is in use and uses it to advertise its file server location. In cases where there is more than one network interface, it's possible for the Engine to pick the wrong one. To remedy this it is possible to force the Engine to choose a specific interface. To configure the Engine to use a different network interface, select the Engine Configuration it uses on the Engine Configuration page, and set the **Net Mask** value under the **File Server** heading to match the network range on which to run the Engine.

Configuring Engine Daemons to Use SNAT

You can configure Engine Daemons for use on the other side of a NAT from the Manager and Drivers. To do this, set the environment variable `DS_USE_SNAT_IP_ADDRESS` on an Engine Daemon to the IP address that you want it to report to the Manager and all Drivers as its address. Additionally, you must set **Self Ping** to `FALSE` in the Engine Configuration.

Using the System Classloader on an Engine

In most situations, the default classloader is used to load client classes. In some instances however, some applications might require the use of the system classloader. This requires deploying extra files to enable the Engine to use the system classloader.

To use the system classloader:

1. Copy the `DS_MANAGER/webapps/livecluster/WEB-INF/etc/DSEngine.jar` file to `DS_DATA/engineUpdate/shared`.
2. Go to **Grid Components > Engine > Engine Configurations**, select an Engine Configuration, and change the value of **Classloader** to **System**.

Note that this requires an Engine restart for any Grid Library that contains a `jar-path`.

Configuring a Global Shared Grid Library Directory

You can configure Engines to use a shared directory for Grid Libraries instead of Engines downloading their own copies.

To configure a global shared Grid Library directory:

1. Disable Engine synchronization on the **Admin > System Admin > Manager Configuration > Resource Deployment** page.
2. On the **Grid Components > Engines > Engine Configurations** page, set the **Grid Library Path** to a shared directory with read-only access that is available from all Engines.
3. Unpack all the necessary Grid Libraries into the shared Grid Library directory, with each library in its own subdirectory matching the archive filename.

4. Deploy the corresponding Grid Library archives to the Broker. The Broker still requires the same access to the Grid Libraries it always had, though Engines can now access them with an alternate method.

Notes:

- Grid Libraries do not automatically update when using this deployment method. Manually unpack and update the shared Grid Libraries, and restart the Engines.
- The shared Grid Library directory must contain both the Grid Library archive files and the extracted archives.
- Permissions are not maintained in `.zip` archives regardless of Broker OS.
- Permissions might be maintained when using `.tar`, `.gz` or `.tgz` Grid Libraries.
- If the configuration does not maintain permissions in any container or distribution Grid Libraries, you must ensure that the executable permissions are set properly for any scripts that run.

Note

Using a global shared Grid Library directory with multiple Engines using the same network share can cause a significant drop in performance.

Configuring When Engines Run

You can configure GridServer to avoid conflicts between work and regular use of the machine. This is called adaptive scheduling, which you configure to adapt the Engine's activity level to your computing environment. The following section describes the methods used to configure when an Engine Daemon runs Engines. Unless otherwise indicated, all of the configuration options are available in Engine Configurations, which are modified on the **Grid Components > Engines > Engine Configurations** page.

Manual Mode

When Manual Mode is enabled in an Engine Configuration, the Engine Daemon runs at all times. This is the default. You can change an individual Engine Daemon to auto mode, or

clear the **Enabled** option under **Manual Mode** on the Engine Configuration page to switch all Engines using that configuration to Auto Mode.

Auto Mode

Auto Mode lets you specify if Engines run on a computer, based on its utilization. For example, you might not want to run Engines when someone is logged in to the computer, or if its CPU utilization is above a certain percentage.

For Windows Engines, there are two methods for determining utilization: **User Idle** and **Processor Utilization**. When you select Auto Mode, select one of the two options. The default is User Idle. Auto Mode for UNIX Engines always uses Processor Utilization mode. In addition, you can also limit the hours an Engine runs by using the **Restricted Hours** settings. Each setting is described below.

User Idle

The User Idle feature is available for Windows Engines only. User Idle starts Engines if mouse and keyboard input have been idle for a given time on the computer hosting the Engine Daemon. This time is entered in seconds, and the default is 600 (10 minutes). Engines halt when there is mouse or keyboard input.

User Idle also has a percentage setting, which is used to determine when Engines stop running, in addition to UI activity. If processor utilization (in percent) on the host computer exceeds this value, Engines stop. To disable this feature, set it to 100%, the default.

Processor Utilization (Windows Engines)

The Processor Utilization option enables Engine Daemons to monitor system CPU usage, and start or stop Engines based on this statistic.

To use this option, enter two values: a percentage and a number of seconds. When CPU utilization drops below the given percentage for the given number of seconds, Engines start. For example, by default, Engines start when CPU utilization is below 50% for more than 30 seconds. When utilization goes above the same CPU threshold value for the specified number of seconds, Engines stop.

CPU usage on single CPU systems is calculated as the total CPU usage, minus the CPU overhead of the Engine. Note that this does not include CPU usage for the Engine, invoke,

or CPU idle processes. In multi-CPU systems where an Engine Daemon launches multiple Engine instances, CPU utilization is calculated independently for each CPU.

In multi-CPU systems, all Engine instances start and stop incrementally. This is called **Incremental Scheduling**, and is the default. When a CPU threshold value is reached, Engines start or stop one at a time. After an Engine starts or stops, there is a delay for a configurable interval—by default, the interval is 10 seconds. Utilization is checked again on the next CPU after the interval delay, and the process repeats.

If you do not select Incremental Scheduling, Engine Daemons use **Non-Incremental Scheduling**. In this case, all Engines on an Engine Daemon start or stop at the same time.

Processor Utilization (UNIX Engines)

On UNIX Engines, the Processor Utilization option works similar to its Windows counterpart, but there are some specific changes:

- You can configure both a starting and a stopping threshold and time. This lets you, for example, start Engines at 40% CPU utilization but stop them at 50%. (The Engines must also be above or below the CPU utilization percentage for the specified period of time.)
- The CPU usage sampling is averaged over a configurable period of time, with the default at 10 seconds.

When using incremental scheduling, Engines are started only when all other Engines are busy. This results in fewer Engine restarts when starting a new Service that requires a restart to load Grid Libraries, and restarts due to download of new libraries when idle.

Also, in non-incremental scheduling, CPU utilization is the average CPU utilization across all CPUs, and not individual CPU utilization. This total CPU utilization percentage is calculated by adding the CPU utilization for each CPU and dividing by the number of CPUs.

For example, if a four-CPU computer has one CPU running at 50% utilization and the other three CPUs are idle, the total utilization for the computer is 12.5%. Likewise, if the maximum CPU threshold is set at 25% on a four-CPU machine and four Engines are running, and a non-Engine program pushes the utilization of one CPU to 100%, all four Engines exit. Even if the other three CPUs are idle, all Engines still exit. In this example, if the minimum CPU threshold is set at 5%, all four Engines restart when total utilization is below 5%.

This only applies to UNIX Engines, when non-incremental scheduling is used. With incremental scheduling on UNIX Engines, and in all scheduling on Windows Engines, each Engine Daemon only looks at the CPU utilization for the CPU on which it is running.

Similar to Windows, CPU utilization calculation does not include CPU usage for the Engine, invoke, or CPU idle processes.

Excluding Processes From Utilization Calculations

Windows and Linux Engines can also be configured to exclude a list of processes from utilization calculations.

The **Exclude the following list of processes from the processor utilization calculation** property in Windows and Linux Engine configurations can be used to specify processes that are ignored in utilization calculations. The value of this parameter must be a comma or semicolon-delimited list of case insensitive names, and are typically the executable names without extension.

Any process spawned by a Service must be in this list to prevent the Service from shutting down the Engine.

Note that Engine, invoke, and CPU idle processes are already excluded from utilization calculations by default.

Restricted Hours

When using Auto Mode, you can also specify a range of hours when Engines run. For example, if you want Engines to only run from 9:00 AM to 5:00 PM daily, configure this using Restricted Hours. You can also have GridServer ignore restricted hours settings on weekends.

To configure Restricted Hours:

1. Go to **Grid Components > Engines > Engine Configurations**.
2. Either select an existing Engine Configuration from the list to modify, or create a new profile.
3. Under the **Restricted Hours** heading, select the first checkbox.
4. Enter a time range when the Engines are allowed to run. For example, if you want Engines to run from 9:00 AM to 5:00 PM, enter 9:00 and 17:00.
5. Select the second option to ignore Restricted Hours settings on Saturdays and Sundays.

Restricted Hours settings do not apply to Manual Mode.

Configuring How Many Engines Run

There are three settings in the Engine configuration that determine how many Engine instances an Engine Daemon runs.

The `SMP Enabled` property specifies if the Engine configuration starts an Engine instance per processor core on the machine. This is set to True by default. Specifically, when SMP is enabled, the Engine Daemon runs a number of Engine Instances equal to the value of the `Minimum Engine Instances` property or the number of processor cores on the machine, whichever is higher. By default, this is set to 1, meaning the number of Engine instances always are the number of cores.

If SMP is disabled, the Engine Daemon always runs the number of Engine instances set in the `Minimum Engine Instances` property in the Engine configuration.

You can also adjust the number of Engine instances used, to reserve a certain number of cores for non-Engine activity. The value of the `Core Relative Engine Instances` parameter is added to the value specified in `Minimum Engine Instances`. The relative value can be a positive or negative number.

For example, if you want to ensure that a single core is always available with no Engine running on it, set the value of `Core Relative Engine Instances` to -1. If a 4-core machine uses this Engine configuration, its number of Engine instances is the higher of its number of cores and the minimum instances (4) plus the relative instances value (-1), or 3. If a 2-core machine uses this Engine configuration, it starts one Engine instance, leaving the other core free.

You can also manually specify the number of instances that run for an Engine Daemon. If you go to the **Grid Components > Engines > Daemon Admin** page, there is a list in the **Instances** column that enables you to select Default or auto SMP. You can type a number into this control to override the number of instances that are run.

Running Engines in Multiplexed Mode

On machines with multi-core processors, GridServer, by default, runs an Engine instance in its own process per logical CPU. While this enables each process to work on different Services, it also means each process runs in its own JVM, and requires its own set of application data. This can cause unwanted overhead when running the same Service across many CPU cores.

To address this issue, you can run Engines in multiplexed mode, meaning that all Engine instances run in a single process. This enables all Engine instances on that machine to share the same set of application data, and multiplex their communication with the Broker.

Multiplexed Engines appear as individual Engines in the GridServer Administration Tool and by the Admin API. Each multiplexed Engine has its own log, work, temp, and data directories. They produce the same output to the reporting database.

You can't mix single-process Engine instances and multiplexed Engines on the same machine. For example, on an eight-core machine, you can't have two processes with four multiplexed Engines each; you can only have eight multiplexed Engines in one process.

Multiplexed Engines have the following differences with Engines running in their own processes:

- All Engines are logged in to the same Broker at any given time. This is enforced by the balancer.
- All Engines work on the same Service session at any given time. This is enforced by the Scheduler
- All Engines reside in the same classloader, which means they can all share the same set of Service Session data.
- They share the same initialization data and Service object. The Service init method is only called once, and all updates are synchronized across all Engines.
- If one Engine has to log off, all Engines in the process are also logged off with the same reason.
- Incremental scheduling is not supported.
- Autopack is not supported.
- The Service implementation must be threadsafe. See the *GridServer Developer's Guide* for more information about multiplexed mode development considerations.
- Running more than one task from a recursive Service is not supported.
- All multiplexed Engines share one set of properties. For example, getting the instance property always reports instance 0, regardless of the Engine instance.

When using multiplexed Engines, Engine resources are shared. The first Engine does all library loading, and this Engine is the first to start work. This also means that Engine Hooks are only loaded once, as hooks are loaded when Grid Libraries are loaded.

When using GridCache, there is one cache per process, and all communication is handled by Engine instance 0.

Communication and Task Scheduling

When Engines are configured as multiplexed, communication with the Broker is multiplexed over a single HTTP keep-alive session to reduce overhead. Also, only one heartbeat is communicated per process. Heartbeats are received by Engine instance zero and simulated to the other Engine Proxies.

To prevent starvation of Engines to other Service sessions, a lease duration is defined. This is the amount of time a set of multiplexed Engines work on one Service before they stop working on that Service and are allowed to work on others if necessary.

Given a lease duration of 5 minutes, the lifecycle for this is as follows:

- When work is started on the Grid, Engine instance 0 picks up a task.
- Immediately following that, all other instances pick up tasks from the same Service session.
- Assuming there is enough work, the instances continue to take tasks from that session.
- After 5 minutes have passed, when any instance finishes a task, it does not pick up more tasks.
- As soon as the last instance has finished, immediately after the response has been written, Engine instance 0 requests another task from any appropriate session. It can be the same session or another session.

Configuration

To enable multiplexed mode, set the Multiplexed Mode property to true in the Engine configuration. The default value is false.

To set the lease time, go to **Admin > System Admin > Manager Configuration > Services**, and set the value of the `Multiplexed Engine Lease` property. The default time is one minute.

The `MULTIPLEXED_MASTER_ID` Engine property is set on any multiplexed Engine to contain the `sessionID` of Engine instance 0 within its process.

There are two buffer timeouts:

- `MULTIPLEXED_ENGINE_MAX_BUFFER_WAIT`: This is an absolute time set per Service.

- `MULTIPLEXED_ENGINE_AVG_BUFFER_WAIT`: This is a factor that is multiplied by the average task duration or the tasks that run during the lease. The default is 0.5. So if the average duration is 10 seconds, this timeout is 5 seconds.

The timeout is measured starting from the first send request.

Configuring 64-bit Engine Daemons to run 32-bit Services

The 64-bit Windows Engine Daemon can be configured to allow running 32-bit Services. If allowed, when an Engine takes such a task, it clears all existing loaded libraries and restarts as a 32-bit process.

Configuration

On the Windows Engine Configuration page, there is a setting called **Additional Platforms** under **Classes, Libraries,** and **Paths**. It is a list with two values, `None` and `win32`. By default it is `None`. Setting it to `win32` on a Windows 64-bit configuration enables the behavior. Setting it to a Windows 32-bit configuration has no effect. This sets an Engine property named `additionalPlatforms` to `win32` or an empty string.

When this property is enabled, Engines download any Grid Libraries with the OS set to **win32** in the root element, in addition to all others it normally syncs.

Note that because Engines take tasks only if they have the root Grid Library of a Service downloaded, this inherently allows these Engines to take 32-bit Windows Services.

When this property is enabled, the following behavior occurs:

- When a 32-bit task is taken by a 64-bit Engine, it restarts as a 32-bit Engine, clearing all Grid Libraries except for the one from the Service and its dependencies
- When a 32-bit task is taken by a 32-bit Engine, it loads normally.
- When a task that is anything but 32-bit is taken by a 32-bit Engine, it restarts as a 64-bit Engine, clearing all Grid Libraries except for the one from the Service and its dependencies.
- When a task that is anything but 32-bit is taken by a 64-bit Engine, it loads normally.

Specifying that a Service is win32

Any Service for which the root Grid Library is marked as `os="win32"` is considered to be a win32 Service.

If a Service's root Grid Library is not marked as such, the Engine remains 64-bit and fails.

Routing 32-bit Tasks to 64-bit Engines

By virtue of the Engine having the 32-bit Grid Library, they by default are allowed to run such Services by the scheduler.

Note that existing applications might not use the OS attribute on the root Grid Library; rather they might use discrimination instead to route Services to win32 Engines. To use 64-bit Engine Daemons on these Services, you must set the OS attribute on that Grid Library.

Configuring a Caching HTTP Proxy Server

Consider a scenario where a GridServer deployment where a Broker and its Engines are separated by a WAN. In this case, it can be inefficient to transfer the same data over the WAN to multiple Engines from the Broker or the Clients. One solution is to use an HTTP proxy server (such as Squid Web Cache) to cache the session’s init data. Any Engine that works on the session must transfer this init data. You can specify a proxy server in an Engine configuration, and the proxy server caches the Service data for other Engines also using the same proxy server.

To use a proxy server, such as Squid, for resource synchronization or data transfer, configure `Proxy Host` and `Proxy Port` parameters to the proxy hostname and port. In case you want to use Proxy with authentication along with `Proxy Host` and `Proxy Port`, you must use `Proxy Username` and `Proxy Password`. HTTPS proxy server uses the `Proxy Scheme` field.

The HTTP Proxy Scheme uses the `FORWARD` Proxy concept whereas the HTTPS uses the `REVERSE` Proxy Concept.

The HTTP Proxy Scheme supports `Basic` and `Digest` authentication and HTTPS supports `Basic` authentication.

For NON-SSL servers, Proxy uses the HTTP scheme whereas for SSL server Proxy uses the HTTPS scheme.

The `Use Proxy for Data Transfer` causes Engines to use the HTTP or HTTPS proxy server for the download of any session's init data. The `Use Proxy for Resource Synchronization` property causes Engines to use HTTP or HTTPS proxy for resource synchronization download.

If Engines are configured to use a proxy server and the proxy is not available, the Engine does not attempt to download the Service data using alternate connection parameters. It is the administrator's responsibility to ensure that the proxy server is up and properly configured. The administrator must consider implementing DNS or IP failover if high availability is required.

Token security for all resource downloads is not supported when using a proxy server. It is assumed that the proxy server is in the same LAN and the LAN is secure. Note that the proxy server cannot download a resource until one of the Engines provides a valid download token.

Note that the following potential problems might occur when using caching proxy servers for resource synchronization:

- Engines can download stale copies of an updated resource from the proxy server. This occurs when the cache timeout is too long or a recently downloaded resource is updated shortly afterward.
- The proxy server might not be able to serve a cached copy of an updated resource if multiple Engines start downloading the resource in a very short time span. This can occur when using Squid even if the `collapsed_forwarding` parameter is enabled.

To address these issues, adjust the cache size to ensure that the proxy server can cache large files. Additionally, analyzing the resource download pattern and the proxy server configuration is recommended to achieve optimal results. The important factors to consider are:

- Size of the total resources that require downloading
- Size of the total proxy cache, the maximum size per object
- Maximum age of a cached object
- Proxy server behavior for concurrent requests
- Configure the HTTP proxy server to ignore the no-cache header. For example, the following `refresh_pattern` option in the `squid.conf` file causes the squid cache to ignore no-cache headers for URLs matching the regular expression of "`^http://.*/livecluster/resourcesproxy`":

  ```
  refresh_pattern ^http://.*/livecluster/resourcesproxy 0 20% 4320 ignore-no-cache
  ```

Configuring an External Engine Daemon Admin Tool

It is possible to configure an external administration or management tool or system that can be opened from GridServer's Administration Tool. When configured, you can open the tool from the Engine Daemon Admin page with an item in the Actions control. Selecting the new item opens a URL to the external tool. A set of runtime macros enable you to customize the URL further.

To configure an external Engine Daemon Admin Tool:

1. In the GridServer Administration Tool, go to **Admin > System Admin > Manager Configuration > Engines and Clients**.
2. In the **Daemon Admin External Tool URL** box, enter the URL to your external tool.
3. The following runtime macros can be used: `${DaemonID}`, `${UserName}`, `${IP}`, `${Status}`, `${PrimaryDirector}`, `${SecondaryDirector}`, and `${OS}`. For example, `http://example.com/grid/reports.py?${DaemonID}` might be used to look up reports for an Engine Daemon with an external query tool.
4. In the **Daemon Admin External Tool Name** box, enter a description of the configured tool.

To use an external tool, select it from the Actions control next to an Engine Daemon on the Engine Daemon Admin page. It is named "Link to" plus the name you defined in the **Daemon Admin External Tool Name** box.

Quarantine Brokers

In a security-oriented environment, it can be necessary to prevent new or untrusted Engines from joining a grid and downloading potentially sensitive application data or resources until the GridServer administrator explicitly grants permission for them to join. You can exercise this control by using a *Quarantine Broker*, a dedicated Broker in a grid, used only for Engine staging and verification. When Engines do not have permission to log in to other production Brokers on your grid, they can only log on to the Quarantine Broker and await permissioning by an administrator.

The quarantine status is set on any Engine Daemon with the Administration Tool or Admin API. If an Engine Daemon's quarantine status is set to "Verified", the Engines managed by

the Daemon might log in to the production Brokers after an Engine Daemon restart. The Engines managed by a quarantined Engine Daemon might only log in to the Quarantine Broker.

Quarantine Broker Concepts

To use a Quarantine Broker, there are two components: the Quarantine Broker, and a method to *Set Quarantine* (send Engine Daemons to the Quarantine Broker) and *Clear Quarantine* (change Engine Daemon status to allow Engine Daemons to log onto production Brokers.)

The Quarantine Broker is specified on the Director. When the Quarantine Broker is specified, the Director allows individual Engine instances to log in to either the production Brokers or the Quarantine Broker based on the value of quarantine status of the managing Engine Daemon. If the quarantine status of an Engine Daemon is “Verified”, all Engines managed by the Engine Daemon might only log in to the production Brokers. Otherwise, the Engines might only log in to the Quarantine Broker.

When using Engine balancing and a Quarantine Broker, the balancing only occurs on the production Brokers. The Quarantine Broker is ignored for the purposes of Engine balancing, and only verified Engines are balanced between the production Brokers.

If the Quarantine Broker is not set in a grid, Engines might log in to any Brokers in the grid allowed by routing and balancing configurations.

Quarantine Status on Engines

The Director determines if an Engine is quarantined by looking for an Engine property called `QuarantineStatus`. The `QuarantineStatus` property value is set to “New Engine” on all newly installed Engines. This ensures that all new Engines are quarantined upon installation when there is a quarantine Broker on the Grid.

When an Engine Daemon has its `QuarantineStatus` property set to any other string than “Verified”, including the null string or having the property missing, it is considered quarantined, and its Engines are only allowed to log in to the Quarantine Broker. When an Engine Daemon has its `QuarantineStatus` property set to the string “Verified”, it is verified, and its Engines are now allowed to log in to other Brokers as per its routing rules.

An unverified Engine Daemon might be cleared from the Administration Tool by changing the `QuarantineStatus` property. This can be automated by using the Admin API. A verified

Engine Daemon can be quarantined similarly. There is also an API method that can be called in a Service Session to quarantine an Engine Daemon.

After quarantine status change, the Engine Daemon needs to be restarted for the managed Engines to log in to the intended Broker set. This restriction minimizes the risk of setting the quarantine status by mistake.

Requirements

All Directors require the same Quarantine Broker setting. When failover is configured, this means the Secondary Director has the same Quarantine Broker configuration if the Primary Director fails.

Although a grid might have multiple Brokers, either for redundancy or volume, you can only define one Quarantine Broker in a grid.

When using a Quarantine Broker, you must have an account with **Manager Configure Edit** access in its Security Role (only available in the Configure role by default) to clear quarantine status of an Engine Daemon or set or change the Quarantine Broker definition. An account must have **Engine Properties Edit** access in its Security Role (available in Configure and Manage roles by default) to set Engine Daemon quarantine status. Any role with **Engine Daemon View** access (all roles by default) can see Quarantine Broker and Engine Daemon quarantine status.

Configuring a Quarantine Broker

To add a Quarantine Broker:

1. Install an additional Broker, if you don’t already have an extra one installed.
2. Determine the name of the Broker you wish to use. Broker names are automatically given at installation, and are typically numeric. In the GridServer Administration Tool, **Grid Components > Brokers > Broker Admin** shows a list of your Brokers, with their names in the **Broker Name** column. Find the name of the Broker you wish to use it for a Quarantine Broker. Note that you can also use this page to change the names of Brokers, if you want to give the Broker a more logical name.
3. Go to **Admin > System Admin > Manager Configuration > Engines and Clients**. Under the heading **Quarantine Broker**, enter the name of the Broker you want to be the Quarantine Broker, then click **Save**. You do not need to restart the Manager.

4. Repeat this configuration (with the same Broker name) for both Primary and Secondary Directory on your grid.

The Quarantine Broker settings persist after Manager restart or future Manager upgrade. If you must change or remove a Quarantine Broker, repeat step 3, but enter a new or blank value for the name.

Setting Quarantine Status on Engines

There are two ways to set and clear quarantine status: interactively with the GridServer Administration Tool, or programmatically with the GridServer Admin API. You can also use the API to self-quarantine Engines. Each method is described below. Also, the following constraints apply when setting quarantine status:

- Due to XML limitations for Engine properties, special XML chars are not allowed in the `QuarantineStatus` property.
- If there is a problem setting the `QuarantineStatus` property, it throws an `AcccessException` (a subclass of `AdminException`) based on the new value.

Using the GridServer Administration Tool

Use the Administration Tool to set or modify the `QuarantineStatus` property in one of the following ways:

- From **Grid Components > Engines > Daemon Admin**, select **Edit/View Properties** from the **Actions** list. Select **QuarantineStatus** from the **Properties** column, and enter a new value.
- From **Grid Components > Engines > Daemon Admin**, select **Set Property for Daemons on Page**, then select **QuarantineStatus** from the Engine Property List, and change the value of the Property.

Using the Admin API

The following method in the `EngineDaemonAdmin` Admin API interface can be used to set and clear Engine Daemon quarantine status:

```
public String setProperty(long id, String key, String value) throws
Exception
```

The value of `key` must be `QuarantineStatus` to set the Engine Daemon quarantine status. The value `id` is the identifier of the interested Engine Daemon. The value might be retrieved by calling `EngineDaemonAdmin.getEngineDaemonIds()`.

The corresponding API call for `setProperty` are also available for C++ (`dsdriver::EngineDaemonAdmin.setProperty`) and .NET (`EngineDaemonAdmin.SetProperty`)

Engine Self-Quarantine Using an API Call

You can also quarantine an Engine using the following method in the Java API:

```
com.datasynapse.gridserver.engine.EngineSession.quarantine (String
reason) throws GridServerException;
```

This method might be called in a Service Session. When this method is called, a synchronous message is sent to the Broker, the Broker then forwards the message plus the Engine session ID to the active Director synchronously. When the Director receives the message, it finds the Daemon ID based on the Engine session ID and sets the Engine property. The Director asynchronously restarts the Engine Daemon identified by the Daemon ID. If any exception occurs, it propagates back to the Engine. Due to the fact that the Engine ID is not passed from the Engine to the Broker, this API call can not quarantine any other Engine Daemons besides the managing Daemon of the current Engine.

The affected Engine Daemon restarts for the quarantine status change to take effect. When the affected Engines receive the restart command, any running Services are canceled, as usual.

The corresponding API call for `quarantine` are also available for C++ (`dsdriver::EngineSession.quarantine`) and .NET (`EngineSession.quarantine`)

Quarantine Broker Constraints

There is no finer configuration to direct Engine instances to specific Brokers using this feature.

Due to the fact that there is only one quarantine Broker defined in a grid, several issues might arise as a result:

- Quarantined Engines that don’t collocate with the quarantine Broker need to send traffic over the WAN.

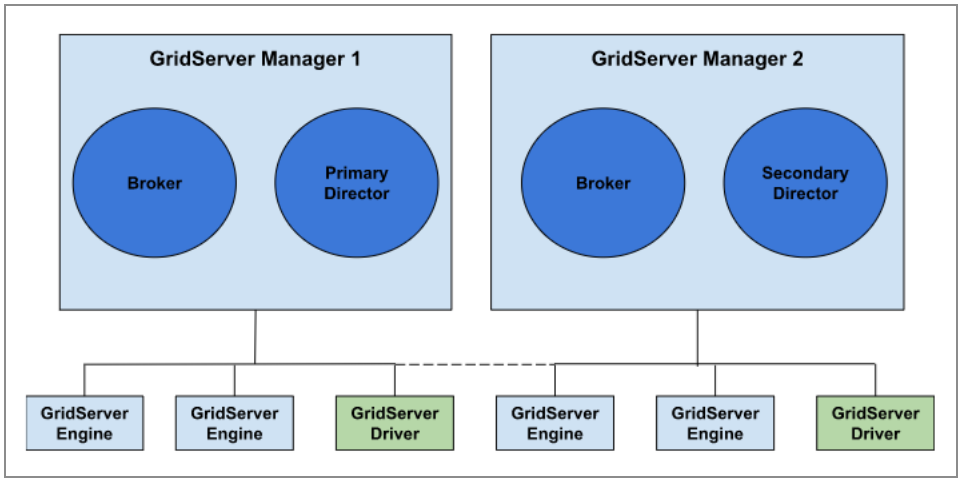
- If the Quarantine Broker is not up or accessible, quarantined Engines cannot log in to any Brokers in the grid. This might generate heavy login retry load when the network is partitioned or unstable.
- Drivers are not treated in any special manner with regard to routing to Quarantine Brokers. This is because some might wish to run tests that require a Driver on Quarantined Engines. Routing rules are required to prevent production Drivers from logging in to Quarantine Brokers.

Grid Fault-Tolerance and Failover

GridServer is a fault-tolerant and resilient distributed computing platform. The GridServer platform recovers from a component failure, and guarantees the running of Services over a distributed computing grid with diverse, intermittent compute resources. This section describes what GridServer does in the event of Engine, Driver, and Manager failure. Failures of components within the grid can happen for a number of reasons, such as power outage, network failure, or interruptions by end users. For the purposes of this discussion, failure means any event that causes the grid components to be unable to communicate with each other.

The Fault-tolerant GridServer Deployment

A DataSynapse GridServer deployment consists of a Primary Director, an optional Secondary Director, and one or more Brokers. Drivers and Engines log in to the Director, which routes them to one of the Brokers. Directors balance the load among their Brokers by routing Drivers and Engines to currently running Brokers.



A minimal fault-tolerant GridServer deployment contains two Directors, a Primary and a Secondary, and at least two Brokers. The Brokers, Engines, and Drivers in the grid have the network locations of both the Primary and the Secondary Directors. During normal operation, the Engines and Drivers log in to their Primary Director; the Secondary Director is completely idle.

Other GridServer topographies, such as having multiple Managers to handle volume or to segregate different types of Services to different Managers, are discussed in the *GridServer Installation*.

Heartbeats and Failure Detection

Lightweight network communications sent at regular intervals, called *heartbeats*, are sent between GridServer components, such as from Drivers to Brokers, from Engine Instances to Brokers, and from Engine Daemons to Directors. A Manager detects Driver and Engine failure when it does not receive a heartbeat within the configurable heartbeat interval time. Drivers detect Broker failure by failing to connect when they submit tasks or poll for results. Engines detect Broker failure when they attempt to report for work or return results. To minimize unnecessary messaging, a heartbeat is sent only if no other message has been sent within the heartbeat interval.

Heartbeat period for clients can be configured in the GridServer Administration Tool at **Admin > System Admin > Manager Configuration > Communication**.

Manager Stability Features

Several precautions prevent Manager failure due to excessive traffic. For instance, the number of threads used for file update is limited. This prevents a large number of file updates from Brokers to Engines from preventing other HTTP activity due to use of all of the HTTP threads on the application server. Instead, Engines retry the download later when this maximum is reached. By default, this is set at 50 threads, but can be changed in the GridServer Administration Tool; go to **Admin > System Admin > Manager Configuration > Communication**, and change the `Maximum Resource Download Connections` property.

The number of Broker/Director messaging threads is also limited. If this limit is reached, clients retry rather than immediately fail.

Engine Failure

Network connection loss, hardware failure, or errant application code can cause Engine failure. When an Engine goes offline, the work assigned to it is queued again and assigned to another Engine. Although work done on the failed Engine is lost, the task is assigned to

a new Engine. Engines that have built up a considerable state or cache or that are running particularly long tasks can cause a larger loss if Engine failure occurs. This can be avoided by shortening task duration in your application or by using the Engine Checkpointing mechanism. For more information about task duration, see Optimizing the Grid.

Each Engine has a checkpoint directory where a task can save intermediate results. If an Engine fails and the Manager retains access to the Engine machine's file system, a new Engine copies the checkpoint directory from the failed Engine. It is the responsibility of the client application to handle the correct resumption of work given the contents of the checkpoint directory.

Note that if an Engine Daemon logs off the Director or otherwise fails, it does not log off its Engines. Provided the failure has not caused the Engines to also fail, they continue working and return results when completed.

Driver Timeout and Failure

When a client application fails, the Broker detects the failure when the Client does not send a heartbeat and does not log back in within a specified time.

This time is defined as the `Client Timeout Minutes` plus the Driver Heartbeat Timeout. The Driver Heartbeat Timeout is the `Max Millis Per Heartbeat` property times the `Timeout Factor` property. Note that for Max Millis Per Heartbeat, this setting is the maximum value. The actual value is randomly between half this value and the value. (For example, the default value is set at 15000 milliseconds. This means the value is between 7500 and 15000 milliseconds.)

For example, by default, `Client Timeout Minute` is 5 seconds (300,000 ms), `Max Millis Per Heartbeat` is 16,000 ms and **Timeout Factor** is 15. This means that the client timeout is between 300,000 + 8,000 * 15 and 300,000 + 16,000 * 15 ms.

The Client Timeout Minutes is set on the Manager at **Admin > System Admin > Engines and Clients > Client Management > Client Timeout Minutes**. The Driver Heartbeat Timeout the `Max Millis Per Heartbeat` property times the `Timeout Factor` property. Both are set at **Admin > System Admin > Communication > HTTP Connections > Driver Heartbeat**.

When client timeout happens, any currently running Services are canceled. If this happens, application failure recovery or restart is the responsibility of your application.

The exception to cancellation is fully submitted Services of type `Collection.LATER` or `Collection.NEVER`. Also, if a Client is collecting results from a `Collection.LATER` type of

Service, none of the outputs are removed until all are collected and the Client destroys the Service, so that if a Client fails during collection it can restart and recollect the outputs.

It is also possible to collect tasks from a failed client application for Services of type `Collection.AFTER_SUBMIT` and `Collection.IMMEDIATELY` if cancellation has not yet occurred. To do this, you must collect and pass in the Driver session ID, in the same fashion as the collection of a `Collection.LATER` Service. For more information about the collection of `Collection.LATER` Services, see "Deferred Collection (Collect Later)" in the *GridServer Developer's Guide*.

All Driver file servers return a Server Unavailable code with instructions to retry if they are processing too many concurrent requests. This significantly reduces the chance of a Service invocation failing due to a temporarily overloaded Driver.

Director Failure

If the Primary Director fails, the Secondary Director takes over balancing and routing Drivers and Engines to Brokers. The Directors do not maintain any state, so no work is lost if a Director fails and is restarted. Also, both Directors follow the same rules for routing to Brokers, so it makes no difference which Director is used to log in.

The Primary Director is also responsible for an internal database, which contains data needed by the Manager for operation, such as the User list, routing properties, and so on. You can modify these values only on the Primary Director. This database synchronizes with the Secondary Director while both are running. The Secondary Director backs up this database on every database backup so that the grid can remain in operation when the Primary Director is down. Features that modify the internal database are not available from the Administration Tool when a Secondary Director is active.

Broker Failure

Like the Director, the Broker is a robust application that runs indefinitely. The Broker typically only fails when there is a hardware failure, power outage, or network failure. However, the fault tolerance built into the Drivers guarantees that all Services complete even in the event of failure.

The most likely reason that a Driver disconnects from its Broker is a temporary network outage. Therefore, the Driver does not immediately attempt to log in to another Broker. Instead, the Driver waits a configurable amount of time to reconnect to the Broker to

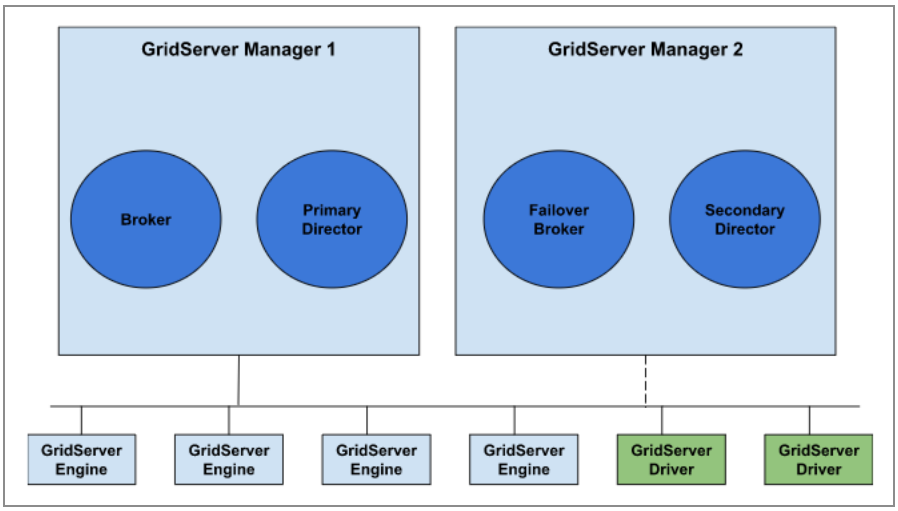
which it was connected. After the configured wait time expires, the Driver attempts to log in to any available Broker. Specify the configured amount of time as `DSBrokerTimeout` in the `driver.properties` file. The property is `BROKER_TIMEOUT` in the API.

After the Driver times out and reconnects to another Broker, all Service instances resubmit any outstanding tasks and continue. Tasks that are already complete are not resubmitted. The Service instances also resubmit all state updates in the order in which they were originally made. From the Service instance point of view, there is no indication of error, such as an exception or failure. There is just the absence of any activity during the time in which the Driver is disconnected. That is, all Services run successfully to completion as long as eventually a suitable Broker is brought online.

If an Engine is disconnected from its Broker and there are no Failover Brokers, the process shuts down, restarts, and logs in to any suitable Broker. Any work is discarded. Failover brokers are described in the next section.

Failover Brokers

In the fault-tolerant configuration, you can set up some Brokers as Failover Brokers. When a Driver logs in to a Director, the Director first attempts to route it to a non-Failover Broker. If no non-Failover Brokers are available, the Director considers all Brokers, and typically routes the Driver to a Failover Broker.



GridServer configuration with Failover capability

A Failover Broker without active Services is not considered for Engine routing. If it has active Services, it is considered like any other Broker, and follows Engine routing like any

other Broker. Thus, if a Failover Broker becomes idle, its Engines are routed back to other Brokers.

The Primary Director monitors the state of all Brokers on the grid. If a Driver that is logged in to a Failover Broker is able to log in to a non-Failover Broker, it is logged off so it can return to the non-Failover Broker. All running Services continue on the new Broker by auto-resubmission.

By default, all Brokers are non-Failover Brokers. Designate one or more Brokers within the grid as Failover Brokers when you want those Brokers to remain idle during normal operation.

Task Fault Tolerance

Task fault tolerance enables an Engine to continue executing a task even if it logs off of a Broker, so that it does not lose work due to a Broker failure. This means that if an Engine is working on a task, and it logs off the Broker, it does not immediately exit. Rather, it continues to work on that task, while continuing attempts to log in to a Broker with the Service on which it is working. If it does not log back in within a defined time period, it exits. If it does log back in, it notifies the Broker that it is working on the task. If the task is completed, it immediately sends the result; otherwise, it does so upon completion.

Using this feature is only recommended when you have individual tasks that take many hours to finish. For example, if a report runs during the night and some tasks take eight hours to process, task fault tolerance ensures that the eight-hour tasks don’t have to start from the beginning if the Broker fails at 7 AM. Enabling task fault tolerance can diminish the efficiency of the grid, since it redundantly schedules all outstanding tasks. For short tasks, it’s usually more efficient to simply recalculate tasks in the event of a Broker failure.

Consider the following example of task fault tolerance:

1. An Engine and Driver are connected to Broker A. The Driver submitted a Service, and an Engine is working on that Service.
2. Broker A goes down.
3. The Driver tries to reconnect with Broker A. The Engine continues working and tries to reconnect to Broker A.
4. After five minutes, the Driver gives up attempts to connect to Broker A. It connects to Broker B and resubmits outstanding work.

5. The Service is now on Broker B. The Engine logs in to Broker B and indicates that it is taking that task. If the Engine already finished its work, it immediately writes the task. Otherwise, after it completes its work, it writes its task.

If another Engine takes the task by the time the original Engine logs in, no attempt is made to cancel the task on the Broker. It is the same as a redundantly rescheduled task.

The situation is similar when an Engine logs into a Failover Broker and works on a task. When the Driver switches back to the Primary Broker, the Engine logs off the Failover Broker and reconnects to the Primary Broker. The task is not canceled.

To enable task fault tolerance, go to **Admin > System Admin > Manager Configuration > Engines and Clients**, and change the value of **Engine Timeout Minutes**. Make the Engine timeout longer than the Driver's timeout, which is the value of `DSBrokerTimeout` set in the `driver.properties` file (five minutes by default.) Note that changes to this value take effect at the next Engine Login.

To use task fault tolerance, another Broker must be available for failover, and the Client running the session must fail over to the Broker and resubmit its session.

No attempt is made upon login of the Engine running a fault-tolerant task to cancel that same task if it has already been taken by another Engine.

Batch Fault-Tolerance

Batch Schedules that exist on a Manager are persistent, provided the **Next Run** field is not `never`. This persistence provides failover capability in the event of a Manager failure, because the Batch Schedules still exist when the Manager is restarted.

The following Batch Schedules are persistent:

- Absolute schedules
- Relative schedules with repeat
- Cron schedules

All persistent Batches restart when the Manager restarts, as if they are scheduled for the first time. Batch runs scheduled for the time when the Manager is down are ignored.

GridCache and PDriver Fault-Tolerance

GridCache supports fault tolerance as described below. Note that Primary and failover Brokers must have their clocks synchronized for GridCache failover.

Because PDriver uses GridCache for its implementation, the same fault-tolerance capability described below applies to PDriver jobs.

Client

If any client puts data in the cache and subsequently fails or logs out, that data is still available to all other clients. This is because the Broker maintains the master index and a complete view of the cached data. This availability does not apply to the local caching mode where a region has a local loader that does not synchronize with the other local caches.

Broker Restart

You can configure GridCache to survive Manager restart and failure. GridCache's cache index is rebuilt on system startup; objects persisted on the Broker's file system are recovered. If some or all of the cache is stored in memory, that information is lost.

Failover

A failover Broker can manage a GridServer cache when a regular Broker goes down, if the persistent cache directory is on a shared filesystem. Configure the location of this filesystem from the Manager Configuration page in the GridServer Administration Tool. When the regular Broker goes down and the failover Broker takes over, the failover Broker builds its cache index and begins managing the cache from the shared filesystem. All clients that then fail over to the failover Broker can get references to the existing cache regions on the shared file system.

Note that a failover Broker can be configured to fail over only to one shared cache directory. Therefore, a failover Broker cannot serve as a failover for multiple Brokers with different cache directories; a different failover Broker has to be used for each Broker.

Note that a failover Broker can fail over to *only one* shared cache directory. One failover Broker cannot serve as a failover for multiple Brokers that have different cache directories. Brokers with different cache directories require separate failover Brokers.

Administration and Maintenance

This section contains several procedures that are commonly used when administering a GridServer Manager. Most of the tasks outlined below use the Administration Tool. The Administration Tool also provides online help, which further describes each page's features.

Configuration Issues

Installation on Machines With Multiple Network Adapters

In some network configurations, on a machine with more than one network interface, a GridServer component can default to using the incorrect interface. You can configure the component to use the correct interface as follows:

- Drivers: To configure a Driver to use a different network interface, edit the `driver.properties` file and set the `DSLocalIPAddress` property to the IP number of the correct interface. For example:

  ```
  DSLocalIPAddress=192.168.12.1
  ```

- Engines: To configure an Engine to use a different network interface, select the Engine Configuration that is used by the Engine on the Engine Configuration page, and set the **Net Mask** value under the **File Server** heading to match the network range on which to run the Engine.

Using UNC Paths in a driver.properties File

To use UNC paths to specify a hostname or directory within a `driver.properties` file, you must change all backslashes (\) to forward slashes (/) in the path.

For example, to change the log directory to the UNC path `\\homer\service1-dir`, change the following line:

```
DSLogDir=./ds-data
```

to this:

```
DSLogDir=//homer/service1-dir
```

Renaming a Broker

Renaming a Broker can cause issues with Engine migration. Specifically, its Engines log in to a new Broker and start taking tasks, but save the old Broker name as a Broker to log in to after restart. After restart, they cannot log in to the requested Broker and restart again, repeating this cycle.

To avoid this issue, restart the Director after renaming a Broker.

Moving a Manager

If you must move a Manager from one machine to another, you must take a few things into account:

- The new Manager needs to have the same fully-qualified domain name as the old one.
- The directory structure of the new Manager needs to exactly match the old one. In other words, it needs to be at the exact same location in the file system.

Using a different IP address for the new Manager works, but other components on the grid might have cached the old IP address of the Manager. This means you might need to restart your Primary Director. If it is the Primary Director you are updating, you might need to restart all Brokers attached to it. If you are unsure whether you must restart, check if your components are logging into your Manager as expected.

GridServer Manager Administration Procedures

The following procedures are commonly used when maintaining a GridServer installation. Most of the procedures have to do with the GridServer Manager. See Managing Engines for more information about managing Engines.

Backup / Restore

Backup of GridServer Managers requires an OS-level file copy of the DS_DATA directory.

Backup Procedure

To back up a GridServer installation:

- Archive (with GNU tar or ZIP) or simply copy the DS_DATA directory. (This is the directory that is specified by `DS_DATA_DIR` variable.)

Restore Procedure

To restore a GridServer installation:

1. Unpack the original GridServer Manager installation using WinZip or a similar tool for Windows; on a UNIX system, unpack using `tar`.
2. If you previously set `DS_DATA_DIR` in the `server.bat` or `server.sh` file, edit that file in the new installation and change the definition again.
3. Copy the backup DS_DATA directory to its previous location.

Importing and Exporting Manager Configuration

GridServer Managers support the ability to export the configuration and deployment of Directors and Brokers into a signed JAR file format and later import this same format to another Manager. You can migrate Manager configurations from a UAT or prototype grid to a production grid, simplify administration of multiple Manager systems, or disseminate an organization's default Manager configuration among all clusters in the organization.

The Import/Export feature imports and exports the following component configurations:

- Manager configurations

- Service Types
- Batches/Batch schedule
- Service Runners
- Contents of the internal repository
- GridCache Schemas
- Resource files, including Grid Library files
- Credentials repository
- Server Hooks

You can only Import/Export the internal repository and Manger Configurations on Managers of the same version and update. For example, if you export a GridServer 5.1 Manager configuration, you can't import it on a GridServer 6.1 installation.

⚠ **Warning**	Not all Manager configuration items are exportable. Server environment-specific items, such as reporting database configuration, are not exported.

To export a configuration

Procedure

1. In the GridServer Administration Tool, go to **Admin > System Admin > Manager Import/Export**.
2. Select the configurations to include in the JAR. In each configuration group, you can select individual items, or choose **Select all** to include all items in that group.
3. Click **Export**.
4. A File Download dialog appears. Click **Save** to save the JAR file.

To import a configuration

Procedure

1. Go to **Admin > System Admin > Manager Import/Export**.
2. Click the **Browse** button next to the **Provide File for import** box.

3. Browse to the location of the JAR file containing the GridServer Manager configuration export.
4. Click **Upload** to begin the import.
5. A list of configurations found in the JAR file are displayed, with configurations highlighted in red if they install over existing configurations. Select the configurations you want to import, then click **Import**.

If the Manager requires a restart for changes to take effect, a message is displayed and the Manager automatically shuts down.

Setting the SMTP Host

You can configure the GridServer Administration Tool to send email notifications, configured on the **Admin > System Admin > Email Notification** page. It also sends new account messages when a new user account is created. To send the email, you must configure a SMTP host for the Manager. This is typically configured during Manager installation, but you can add or change the value afterward.

To set the SMTP host:

1. Go to **Admin > System Admin > Manager Configuration > Admin**.
2. Under the **Mail** heading, in **SMTP Host**, enter the name of your SMTP server.
3. In **Contact Address**, enter the email address of an administrative contact. A notification is sent from this address to the new user when their account is created on the Administration Tool.
4. Click **Save**.

Configuring the Timeout Period for the Administration Tool

For security purposes, the GridServer Administration Tool times out after a certain period and requires users to log in again. By default, the timeout period is 60 minutes.

To change the timeout period:

1. Go to **Admin > System Admin > Manager Configuration > Security**.

2. Under the **Admin User Management** heading, change the value of **Admin Browser Timeout**.

Reconfiguring Managers when Installing a Secondary Director

When you install a Manager that includes a Secondary Director, you must also configure the Manager containing the Primary Director. This registers the Secondary Director's address with the Primary Director, as well as reconfigures the Engine and Driver configurations.

To reconfigure the Manager containing the Primary Director, select **Admin > System Admin > Manager Reinstall**, and enter the Secondary Director's address and port on the corresponding page. This configures the Primary Director to recognize the Secondary Director, and reconfigures Engine and Driver configurations accordingly.

Reconfiguring the Engine Communication Port

By default, the Manager uses port 8000 for communication with Engines. This port can be changed, but you must also change Engine configurations to log in at the new port.

⚠ **Warning**	If you reconfigure the Manager's messaging port prior to updating an Engine's configuration, the Engine is no longer able to reach the Manager. Consequently, Engines can not log in after the Manager Reconfigure process, and you must re-install them. To avoid this issue, change the port used for Engine communication first.

To change the port used for Engine communication:

1. Go to **Grid Components > Engines > Daemon Admin**.
2. Select **Configure All Daemons** from the Global Actions list.
3. Change the Primary Director property to the URL with the new port.
4. All Engine Daemons then log off and no longer appear in the Daemons list.

5. Edit the `server.xml` file in the base directory location defined in DS_DATA. (If DS_DATA is not defined, the file is in `TIBCO_HOME\manager-data\conf` for Windows, or `TIBCO_HOME/manager-data/conf` for UNIX.)
6. To change the messaging port, replace the value of 8000 in the following line:

   ```
   <Connector port="8000"
   ```

7. Restart the Manager. After restart, all Daemons reappear in the Daemons list.
8. Go to **Grid Components > Engines > Daemon Admin**, select **Configure All Daemons**, and change all Primary Director URLs back to “default”.

Promoting a Secondary Director to Primary Director

When a Primary Director fails and a Secondary Director takes over, it has a backup copy of the internal database that is used to store user accounts and other properties. However, because it can’t resynchronize this information to the Primary Director when it returns, it is read-only and cannot be modified.

In a situation where the failed Primary Director is unavailable for some time, you might need to promote the Secondary Director to the role of Primary Director, and possibly assign another Secondary Director. Note that this requires a failover setup with two Director machines that are both capable of handling a volume necessary for your grid’s demands, plus an optional Secondary Director.

To promote a Secondary Director to Primary Director:

1. Log in to the Secondary Director in the GridServer Administration Tool.
2. Go to **Admin > System Admin > Manager Reinstall**.
3. Select **Manager Configuration** and click **Next**.
4. In the **Director** list, select **Primary**.
5. Finish the configuration.

Configuring SNMP

GridServer supports Simple Network Management Protocol (or SNMP), which can generate alerts (called traps) on a per-event basis. For example, you can send events such as 'Server Started' and 'Engine Died' as traps to an SNMP monitoring station. The SNMP interface is

administered through an administrative plug-in on the GridServer Manager. The traps themselves are defined in a Management Interface Base (MIB), which is designed for applications. An MIB specific to DataSynapse is defined and included with the Manager.

To configure and enable SNMP support for your Manager:

1. Go to **Admin > System Admin > SNMP Event Traps**.
2. Enter the hostname and port of your SNMP server in the **Host** and **Port** fields, then click **Add**.
3. If you have multiple SNMP servers, repeat step 2 for each server.
4. In **SNMP Version**, select the version of the SNMP protocol your servers use.
5. Select each event in the event list for which you want to generate a trap.
6. Go to **Admin > System Admin > Manager Configuration > Admin**.
7. Under the **SNMP** heading, set **enabled** to `True` for the Broker, Director, or both.

The DataSynapse MIBs are at `DS_MANAGER/webapps/livecluster/WEB-INF/etc/snmp`.

Some SNMP events generate traps from the Broker, while others generate traps from the Director. The following table lists events that generate traps, sorted by Broker or Director:

SNMP Trap Events

Broker Trap Events	Director Trap Events
DriverAddedEvent, DriverRemovedEvent, EngineAddedEvent, EngineDiedEvent, EngineBlackListedEvent, EngineGreyListedEvent, EngineRemovedEvent, JobCancelledEvent, JobFinishedEvent, JobRunning, MemoryWarningEvent, TaskFailed	BrokerAddedEvent, BrokerRemovedEvent, EngineDaemonAddedEvent, EngineDaemonRemovedEvent, LocalDatabaseBackupFailureEvent, RemoteDatabaseBackupFailureEvent, ServerStartedEvent

LogLogic Integration

GridServer can seamlessly integrate with TIBCO LogLogic® for analysis of Manager and Engine logs. You can optionally configure GridServer to send log messages to a syslog daemon, where they can be captured by LogLogic.

Configuration

GridServer log items can be sent to a centralized syslog daemon, which is then used for capture by LogLogic. This is configured in the Manager Configuration or the Engine Configuration.

Manager Configuration

To configure syslog output for a Manager, go to **Admin > System Admin > Manager Configuration > Logging > Syslog**.

Engine Configuration

To configure Syslog output for a group of Engines, go to **Grid Components > Engines > Engine Configurations** and edit the properties in the **Logging to a Syslog Server** section. This affects all Engines using that Engine Configuration.

Configuration Properties

Configuration properties are as follows:

Syslog Configuration Properties

Property	Description	Default
Enabled	If syslog output is enabled.	false
Syslog Server Address	The hostname or IP address of the syslog server.	
Syslog Server Port	The port of the syslog server	514
Components	The comma-separated list of components that are logged. The component is the value that is in brackets on a log message. If the logger is set to use the Fully Qualified	

Property	Description	Default
	Class Name, the component name is just the unqualified class name. Use the * character to log all components. (The * character can only be used to indicate all, it cannot be used in component names.)	
Max Log Level	The highest level of messages that are sent to syslog. Note that log messages higher than the default log level are not sent; this setting only limits what is sent to syslog.	ALL
Tag	The syslog tag, which is typically the name of the program or process that generated the message.	GridServerEngine or GridServerManager
Time Zone	The time zone of the time stamp. Currently only GMT and default time zones are supported.	
Facility	The syslog facility level.	1

Logging Message Format

Facility

The syslog facility value is an integer from 0-23. See the Syslog protocol definition in RFC 5424 for more information about facility values.

Severity

The severity is an integer from 0-7. The following table describes the severity levels, and how Java log levels are mapped to them:

Syslog Severity Levels

Code	Severity	Java Level	Description	General Description
0	Emergency		System is unusable.	A "panic" condition usually affects multiple apps/servers/sites. At this level it usually notifies all tech staff on call.
1	Alert		Action must be taken immediately.	Must be corrected immediately, therefore notify staff who can fix the problem. For example, the loss of a primary ISP connection.
2	Critical		Critical conditions.	Must be corrected immediately, but indicates failure in a secondary system. For example, the loss of a backup ISP connection.
3	Error	`SEVERE`	Error conditions.	Non-urgent failures, these must be relayed to developers or admins; each item must be resolved within a given time.
4	Warning	`WARNING`	Warning conditions.	Warning messages, not an error, but indication that an error occurs if action is not taken. For example, file system 85% full - each item must be resolved within a given time.
5	Notice		Normal but	Events that are unusual

Code	Severity	Java Level	Description	General Description
			significant condition.	but not error conditions - might be summarized in an email to developers or admins to spot potential problems - no immediate action required.
6	Informational	`INFO, CONFIG`	Informational messages.	Normal operational messages - might be harvested for reporting, measuring throughput, and so on - no action required.
7	Debug	`FINE,FINER, FINEST`	Debug-level messages.	Info useful to developers for debugging the application, not useful during operations.

Output

The output is in standard syslog format:

```
<PRI> HEADER MSG
PRI: Integer value: (Facility * 8) + Severity
HEADER: Timestamp Address
```

- The timestamp format is *MMM dd HH:mm:ss*
- The address can be hostname or IP. The fully qualified domain name as reported by the OS is used.

  ```
  MSG: TAG: CONTENT
  ```

- The separator between `TAG` and `CONTENT` can be a number of things, but a colon is most common.
- The `CONTENT` is the GridServer log message, without the initial timestamp.

Elasticsearch Integration

To support integration with Elasticsearch, Manager log messages and Server events can be sent to a REST endpoint as JSON. This enables you to use Elasticsearch for Engine and Broker usage analysis, and to perform quicker and easier issue diagnosis with log parsing. This feature can also be used for any tool like Elasticsearch that accepts messages on an HTTP(S) REST endpoint as JSON.

Configuration

To configure REST output for logging or server events, go to **Admin > System Admin > Manager Configuration > Logging > Log to REST** or **Server Events to REST**. This feature is supported only for Managers, and not Drivers or Engines.

Log Message to REST Configuration

The following properties are related to sending log messages to a REST endpoint:

Log Message to REST Configuration

Name	Description	Default
Enabled	If sending log messages to REST is enabled.	false
Server URL	The complete URL path of the location that receives the log messages. The following substitution variables are supported: • `${hostname}` the short (not fully qualified) hostname • `${managerId}` the Manager ID as found on the **Admin** section of the Manager Configuration	

Name	Description	Default
	page • `${dateFormat:`*format*`}`, where *format* specifies a date according to the Java `SimpleDateFormat` class. For example, `http://logdb.mycompany.com/datasynapse-${hostname}-${managerId}-${dateFormat:yyyy-MM-dd}/log`	
Username	Username, if using Basic Auth.	
Password	Password, if using Basic Auth.	
Max Log Level	The highest level of messages that is logged. This can only be used to restrict what is logged to the server from the default settings. Typically this is set to `ALL`, unless you want to send higher level messages to another logging device but restrict what is logged to this server.	`ALL`
Backlog	The number of messages that can be queued for send.	1000
Discard when Backlogged	If true, subsequent messages are discarded when the backlog is reached. Otherwise, the caller is blocked.	`false`
Timesta	The format of the 'timestamp'	`yyyy-MM-dd'T'HH:mm:ss.SSSZZ`

Name	Description	Default
mp format	property	
Time Zone	The time zone of the time stamp, currently only GMT and default time zone is supported.	
Excluded LogRecord Properties	A comma-delimited list of properties of the Java `LogRecord` class that is excluded. By default, the following are excluded: `sourceClassName`, `sourceMethodName` because they can impact performance; `millis` because a formatted timestamp is often preferred; `sequenceNumber` because it might not be useful.	`millis,sequenceNumber,sourceClassName,sourceMethodName`
HTTP Timeout	The connect and read HTTP timeouts, in milliseconds	5000

Server Event to REST Configuration

The following table lists properties related to sending server events to a REST endpoint:

Server Event to REST Configuration

Property	Description	Default
Enabled	If sending server events to REST is enabled.	false
Server URL	The complete URL path of the location that receives the log messages. The following substitution variables are supported: • `${hostname}` the short (not fully qualified) hostname.	

Property	Description	Default
	• `${managerId}` the Manager ID as found on the **Admin** section of the Manager Configuration page. • `${dateFormat:`*format*`}`, where *format* specifies a date according to the Java `SimpleDateFormat` class. For example, `http://logdb.mycompany.com/datasynapse-${hostname}-${managerId}-${dateFormat:yyyy-MM-dd}/log`	
Username	Username, if using Basic Auth.	
Password	Password, if using Basic Auth.	
Backlog	The number of messages that can be queued for sending.	1000
Discard when Backlogged	If true, subsequent messages are discarded when the backlog is hit. Otherwise, the caller blocks.	false
Timestamp format	The format of the `timestamp` property	`yyyy-MM-dd'T'HH:mm:ss.SSSZZ`
Time Zone	The time zone of the time stamp. Currently only GMT and default time zone are supported.	
Excluded Events	A comma-delimited list of integer event types that are excluded.	
Include Type	If true, the integer event type is included in the message.	true
Include Name	If true, the event name is included. For	false

Property	Description	Default
	example, `TASK_SUBMITTED`.	
HTTP Timeout	The connect and read HTTP timeouts in milliseconds	5000

Message Format

Each log message is JSON formatted, and includes the following information:

- `timestamp`—when the event occurred (**not** when it was sent.)
- type—The integer type, from the `ServerEvent` class in the API.
- name—The name, from the `ServerEvent` class.
- properties—The properties map for that event.

For example:

```
{
   "timestamp":"2014-04-29T17:45:18.512-0400",
   "type":28,
   "name":"TASK_SUBMITTED",
   "properties":{
      "ServiceSessionID":"21837468246",
      "ServiceInvocationID":"3".
      ...
   }
}
```

Database Maintenance

Each GridServer Manager contains an embedded database running on each Director. This internal, or admin database stores administrative data, such as User, Engine, Driver, and Broker information. You can also configure GridServer to use an external reporting database to log events and statistics. GridServer does not include a reporting database; you must use your own enterprise-grade database.

Database Types

There are two databases used by the GridServer Manager, each of which are described below.

The Reporting Database

The external reporting database is optionally used to store events and statistics. Depending on configuration settings, this database can grow quickly. Using a robust external database is recommended if you plan to make use of the reporting capabilities. The specific types of data that are stored in the reporting database are configurable in the Administration Tool at **Admin > System Admin > Manager Configuration > Database.** Install the external database on any machine reachable from the Broker over a standard network connection.

If a grid is deployed across a large WAN on which latencies can be large, it's best not to have all Brokers write to the same reporting database, especially if the database is in a remote location. You can override a Director's reporting database settings with a Broker-specific database setting. To do so, go to **Admin > System Admin > Manager Configuration > Database**, and set **Override Primary Director** setting to True. When True, it uses the given values, otherwise it gets the values from the Director.

For information about installing an external database for the reporting database, the *GridServer Installation Guide*.

The Internal Database

GridServer's internal database stores admin data such as User, Engine, Driver, and Broker information. In typical cases, the internal database is read at Manager startup, and only written to thereafter if user-driven admin events occur, such as adding a user, Engine, or Manager. The internal database is required to start the Manager. This database is an embedded component of the GridServer software.

Authenticating to the Reporting Database with Active Directory (AD)

The GridServer Manager can authenticate its connection to a Microsoft SQL Server reporting database using on-premise Active Directory (AD) credentials through NTLM. This method serves as an alternative to traditional SQL login authentication.

To configure this, navigate to **Admin > System Admin > Manager Configuration > Database**. On this page, enter the details for a dedicated AD service account:

- **User Name:** Provide the bare `sAMAccountName` of the AD account (do not include the domain prefix). The domain is specified within the connection URL.
- **Password:** Enter the password associated with the AD account.

For more information about the `connection-string` parameters and SQL Server login setup, refer to "Configuring a ReportingDatabase" section in the *TIBCO GridServer® Installation* guide.

This setting authenticates only GridServer Manager's database connection. It operates independent of user authentication to the Manager as explained in Using Windows Authentication and Authorization and other grid component authentication to the Manager as explained in Using Pure Kerberos Authentication.

Internal Database Reset

If, for some reason, you need to reset the internal database to its original state as initially installed, you can use the following procedure.

Warning This procedure removes all administrative data!

To reset the internal database:

1. Shut down the Director.
2. Delete the contents of the `DS_DATA/db` directory.
3. Open the `DS_DATA/conf/installation.properties` file, and set `DSConfigureOnStartup=true`
4. Start the Director.
5. Go to the Administration Tool. You are presented with the installation page. Complete the installation, and restart.

Internal Database Backup

The internal database used by GridServer is automatically backed up at a regular interval. A copy of `Internal.properties` and `Internal.script` is made and stored in `DS_`

`DATA/db/backup`. If a Secondary Director is installed, the internal database is replicated to the Secondary Director at the same time, and when the Secondary Director logs in.

Internal database backup is configured at **Admin > System Admin > Manager Configuration > Database**, under the **Admin Database Configuration** heading. The **Backup Cron** property dictates when backups are made by using the cron format described above. By default, it is set to back up the database at 3:00 AM daily. The `Backup` property is set to enable or disable back up. It is set to `enabled` by default.

> **Note** Database backups can be very resource-intensive. Scheduling them during off-peak hours is recommended.

Performing Reporting Database Maintenance

GridServer Managers do not perform any cleanup of data in the reporting database. This is considered a database maintenance activity. As such, it is not managed by GridServer. Consult your database's documentation for more information about how to perform database cleanup.

To temporarily disconnect the external reporting database for periodic database maintenance, you can suspend the database connection. This backlogs all write events posted to the reporting database until reactivation.

To suspend the database connection, go to **Admin > System Admin > Manager Configuration > Database**, and change the **Connection Suspended** property from False to True.

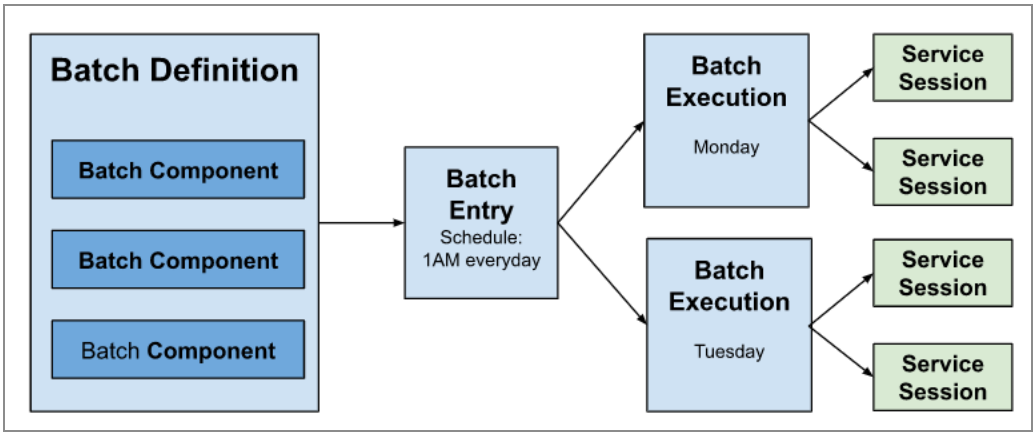
While suspended, all write events posted to the reporting database are backlogged until set back to False. You can also safely stop the database while it is suspended. Set the backlogs sufficiently high so that events are not discarded while suspended. All events are backlogged to memory.

The Batch Scheduling Facility

Commands and Services can be scheduled to run on a regular basis by using the Batch Scheduling Facility. A *Batch Definition* contains instructions in the form of components that define scheduling and which commands and services the Batch runs. When the Batch

Definition is scheduled on the Manager, it creates a *Batch Entry*, which typically waits until its scheduled time, then runs, creating a *Batch Execution*. Services are run by using an embedded Driver on the Manager.

By using the Administration Tool, you can write a Batch Definition with specific scheduling instructions. You can specify a Batch Definition to immediately run when scheduled, or it can wait until a given date and time. A Batch Definition can be submitted to run at a specific absolute time, or a relative time, such as every hour. They can also be written to wait for an event, such as a new, modified, or a deleted file.



A Batch Definition consists of Batch Components. When a Batch Definition is scheduled, it creates a Batch Entry and runs as defined by the Batch Components. When it runs, it creates a Batch Execution file, which then runs the components according to the definition.

Batch Definitions contain one or more components contained within a `batch` component. A `Command` component contains a program that is run by the Batch Definition. A `schedule` or `event` component specifies when subsequent `Command` components run.

Terminology

The following terms are used to describe components related to the Batch Scheduling Facility:

Batch Terminology

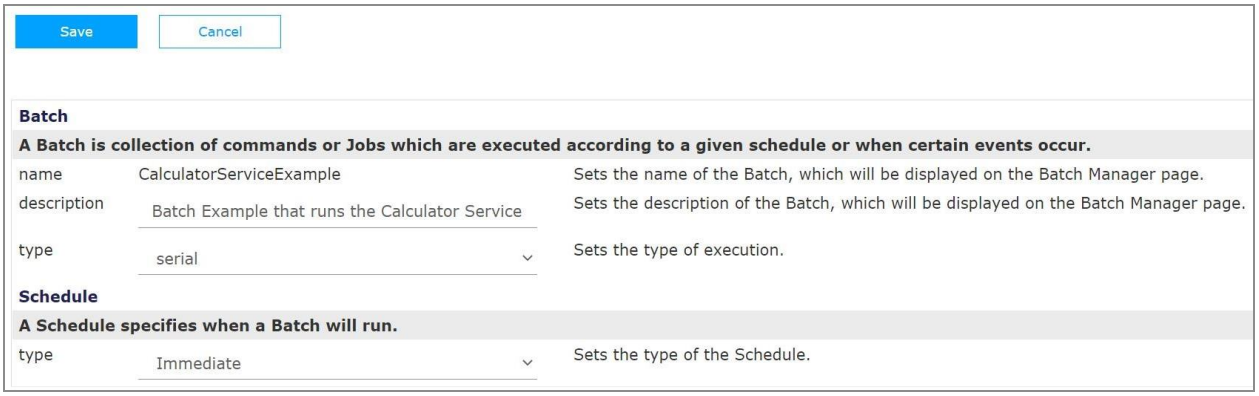
Name	Page	Description
Batch Definition	Batch Definitions	How a Batch is written. The Batch Definition is edited from the Batch Definitions editor which contains

Name	Page	Description
		components that define the Batch. Once created, you can manage the Batch from the **Batch Definitions** page.
Batch Component	Batch Registry editor	When a Batch Definition is created, it consists of a Batch component, which can contain other components, such as ServiceCommand components, Conditional components, and other Batch Components. The Batch Definitions editor lets you add, remove, and edit Batch components and other components it contains.
Batch Entry	Batch Schedule	When a Batch Definition has been instantiated by being scheduled on the **Batch Schedule** page, a Batch Entry is created. The Batch Entry either runs immediately, or waits to run, depending on what scheduling components are added to the Batch Definition.
Batch Execution	Batch Admin	When a Batch Entry runs, it creates a Batch Execution, which does whatever was defined in the Batch Definition. For example, if a Batch Definition uses the ServiceCommand to start ten Service Sessions, the Batch Execution does that. The Batch Execution is managed on the Batch Admin page. Any actual Service Sessions created can be managed on the **Grid Components > Services > Service Session Admin** page.
Service Runner	Service Runners	Service Runners enable you to define a registered Service Type with options and init data that can be used in a Batch Definition.

Editing Batch Definitions

To create a Batch Definition, go to **Services > Batch > Batch Definitions**. This page contains a list of Batch Definitions on the Manager, plus a blank box for entering the name

of a new Batch Definition. When you select a Batch Definition, click the **Action** list, and you can select **View/Edit Batch Definition** to edit a Batch Definition, **Rename Batch Definition** to rename a Batch Definition, **Copy Batch Definition** to copy a Batch Definition, **Delete Batch Definition** to remove a Batch Definition, **Export Batch Definition** to save an XML file of the Batch Definition, or **Schedule Batch Definition** to place a Batch Definition in the Manager's Batch queue. You can also select **Batch View** to display a graphical representation of the Batch Definition in a new window.



The Batch Definition window

To edit a Batch Definition, either select **View/Edit Batch Definition** from an existing Batch Definition's **Action** list, or type the name of a new Batch Definition in the empty box at the end of the list and click **Add**. This opens the Batch Definition window containing parameters for your new Batch Definition. You can then change the values of parameters, and click **Save** to save the values as a Batch Definition on the Manager, or click **Cancel** to exit and discard any changes you have made.

The Batch Definition parameters are as follows:

Batch Definition Parameters

Parameter	Description
Batch Component	
Name	The name of the Batch Definition. If this is a new Batch Definition, this is the name you initially typed in the blank box prior to selecting **Add**, and is not editable. (You can rename a Batch Definition by selecting the **Rename** action from the Batch Definitions page.) If an additional Batch component is added to a Batch

Parameter	Description
	Definition, you can set its name.
Description	Sets the description of the Batch, that appears on the Batch Manager page.
Type	Determines how a Batch Definition is run, either in serial or parallel. You can set the following types: • `parallel` - all Batch components are run when the Batch Definition is scheduled. • If set to `serial` - Batch components are run in the order in which they were added. If any of the components fail, it prevents the Batch from continuing, and the Batch fails. The default is serial.
Schedule Component	
Type	Sets the type of the Schedule. You can set the following types: • `Immediate` - the Batch Definition runs when scheduled. • `Absolute` - the Batch Definition runs once according to the date set in startTime. • `Relative` - the Batch Definition runs after the specified number of minutes in minuteDelay as well as repeating or running immediately with respect to repeat and runNow. • `Cron` - the Batch Definition runs according to the values set in the cron entry. (The same cron format is used as the one in the database configuration. See "Cron Format" for more information. • `Manager Startup` - the Batch Definition runs when the Manager first initializes.
Add component	Adds a component to the Batch Definition. A Batch Definition can contain one or more components.

Batch Components

The parameters in the Batch Definition editor correspond to components contained in the Batch Definition. Each Batch Definition can contain one or more Batch components. These components can be commands, events, or other Batch Definitions. For example, a LogCommand Component is shown below. To add a component to a Batch Definition, select a component from the **add component** list.

A Batch Component

Batch components are processed in a Batch Definition in order when Batch Type, described above, is set to serial. To move a component's order in the Batch Definition, click the **Move Up** or **Move Down** buttons in the upper-right corner of each Batch component. You can also remove a Batch component by clicking the **Remove** button in the upper-right corner.

Each of the types of Batch components that can be added to a Batch Definition are described below. By default, **Extended Help** is displayed. To display only the first sentence of help, select **Help** from the Help control in the upper right corner. To suppress the help display, click **No Help**.

Batch Components

Name	Description
Batch	Contains another Batch Definition. This can be used to create a complex or multi-leveled Batch Definition. For example, a parent Batch Definition can start each day, starting with two child Batch Definitions, each with different schedules or conditions. For each new Batch component, you must set the same parameters for a Batch Definition as described above. You can then add additional components to the Batch.
Conditional	Provides conditional processing when running Batches. The component specified by the **test** is run. If it runs successfully,

Name	Description
	the component specified by **success** runs. If it fails, the component specified by **failed** runs. The component specified in the **test**, returns success in the following conditions: • Command returns `Command.SUCCESS` • `ServiceCommand` creates the Service and submits the invocation without exception • `ServiceRunnerCommand` creates the Service and submits all invocations without exception
BatchReference	Contains a reference to a registered Batch Definition that gets loaded when scheduled from the Batch Registry.
Command	Runs an implemented method in a deployed class. Properties in Batch commands are based on the Java Beans specification. In a Java class, you define methods that match a pattern to get and set property values. For example, if you have a property called `MyName`, you can define two methods: `Public class MyCommand implements Command { public void setMyName(String value) { MyName = value; } public String getMyName() { Return MyName; } public int run() { ; //do stuff return Command.SUCCESS; } }` On the Batch Registry page, adding this command to a Batch, you can add a property with the name `MyName` and value `Bob` by entering these in the two text fields and clicking **Add property**.
ServiceCommand	Starts a Service. You can specify a Service type registered on the Manager and method name to run. You can also specify a

Name	Description
	Service reference ID, Service action, and input and init data for the Service. The Service reference ID enables you to reference the Service from another Service Command. Data is comma-delimited. You can add ServiceDescription, ServiceOptions, and Discriminator components to a Service by using a Service Runner.
ServiceRunnerReference	Loads the specified registered Service Runner. See below for information about registering a Service Runner.
AdminCommand	Runs a command via the GridServer Admin API. For more information about using the Admin API, see the *GridServer Developer's Guide*.
EmailCommand	Sends an email message from a Batch Definition, for notification or alerts. You can enter a comma-delimited list of email addresses for recipients, and a message string, used as a subject and a body. Note that for the batch processor to send an email, you must define an SMTP server and Contact Address in your Manager Configuration. To do this, go to **Admin > Manager Configuration > Admin**, and enter the values for **SMTP Host** and **Contact Address** under the **Mail** heading. Email sent by the batch processor is sent from the Contact Address.
EmailFilesCommand	Sends an email message from a Batch Definition that includes files as attachments. It is typically used to send the output of a previous command by saving that output to a file. You can enter a subject, a message body string, a comma-delimited list of email addresses, and a semicolon-delimited list of files, which are sent as attachments in the message. The setup rules given above in the description of the EmailCommand component also apply to the EmailFileCommand component.

Name	Description
ExecCommand	Runs a command from a Batch. This runs a command from the application server's root directory. You can set an input, output, and error file, plus a log file for the command to be run.
LogCommand	Writes a string to the Manager log. This is useful for testing Batches or indicating when a Batch is starting or stopping.
WaitCommand	Halts for a moment before proceeding. The amount of wait time is specified in seconds. Note that this component is only useful for generating a wait time when the Batch type is serial.
EngineWeightCommand	Sets the Engine distribution weighting relative to other Brokers. Only Brokers that are part of a Broker/Director Manager installation are listed.
Event	Makes a Batch File wait for an implemented event to take place. You can use this to pause until a specific condition in a class you deployed has occurred.
FileEvent	Makes a Batch wait for a file event to occur to the specified file before completing the remaining items in the Batch Definition. Specifically, it enables you to watch a file and wait until it is created, deleted, or modified before proceeding. The file specified in FileEvent supports wildcards in the base filename. Any file matching the wild-card rule can trigger the file event. Wildcards supported include `*`, to match the rest of the base filename, and `?`, to match one character in the base filename. The wildcard naming role is the same as what is implemented by `org.apache.commons.io.filefilter.WildcardFilter` (see `http://jakarta.apache.org/commons/io/api-release/index.html` for more information.)

Service Runners

Service Runners enable you to define a registered Service Type with options and init data that can be used in a Batch Definition. It can also be used to chain together Service Types

and Discriminators into a single unit that can be used in a Batch Definition.

To create a Service Runner, go to **Services > Batch > Service Runners**. Type the name of a Service Runner in the box and click **Add**. This opens a Service Runner Editor page, where you can choose a Service Type and enter init data, a description, and method names and input data for invocations. You can also use the list at the bottom of the page to add Discriminators, Service input description data, and Service options.

The **Service Runners** page also lists all Service Runners existing on a Manager. Using the **Actions** controls, you can edit, rename, copy, delete, export, or launch each Service Runner.

Scheduling Batch Definitions

After creating a Batch Definition with the Batch Definition editor page, the Batch is listed among the other Batch Definitions on the **Batch Definition** page. However, these Batch Definitions are not actually running on the Manager yet. To create a Batch from a Batch Definition, you must first schedule it. This actually instantiates a Batch and inserts it into the Manager's batch queue.

To schedule a Batch Definition, go to **Services > Batch > Batch Definitions**, and find the Batch Definition in the list. Select **Schedule Batch Definition** from the **Actions** control. This schedules the Batch Definition and opens the **Batch Schedule** page, displaying the new Batch Entry.

Note that when you schedule a Batch to run at some point in the future, and then you edit or change Batch Definitions, the Batch uses the Batch Definitions as they were defined when the Batch was scheduled, and not the current ones. If you want to change a Batch Definition for a Batch you have scheduled in the future, you must remove the scheduled Batch, change the Batch Definition, and reschedule it.

The Batch Schedule Page

Batch Entries on a Manager can be listed and administered on the **Batch Schedule** page. To do this, go to **Services > Batch > Batch Schedule**. All Batch Entries resident on the Manager are listed. To remove or edit an existing Batch Entry or view logs or Batch executions, select a command from the **Actions** control next to the relevant Batch.

Running Batches

Batch Entries automatically run when they reach the scheduled time or conditions defined in their Batch Definition. When this happens, Batch Executions are created and displayed on the **Services > Batch > Batch Admin** page. PDriver Batches (which are also Batch Executions) are also displayed on this page. From this page, you can monitor Batch Executions, search for logs, and display the Batch Monitor to view what parts of a Batch have completed.

Any Services that are run by the Batch Execution are displayed on the **Services > Services > Service Session Admin** page. From there, you can cancel Service Sessions, view tasks, or do any other actions you normally do with a Service. Note that it is possible to have a Batch Execution run a Service that continues to run, even after the Batch Execution reports that it is finished.

Deploying Batch Resources

Java Services, Commands, and other resources must be placed in `DS_DATA/batch/jar` to be properly loaded by the embedded Driver.

Batch Fault-Tolerance

Batch Schedules that exist on a Manager are persistent, provided the **Next Run** field is not `never`. This provides failover capability in the event of a Manager failure, as the Batch Schedules still exist when the Manager is restarted.

The following Batch Schedules are persistent:

- Absolute schedules
- Relative schedules with repeat
- Cron schedules

All persistent Batches are restarted when the Manager is restarted, as if they are scheduled for the first time. Batch runs that were to occur during the time when the Manager is down are ignored.

Optimizing the Grid

This section provides information about tuning your Grid's settings for optimal performance.

Diagnosing Performance Issues

To find bottlenecks in application performance, use GridServer's Instrumentation feature. With instrumentation enabled, you can get detailed timings of each request submitted to the Broker. These timings highlight scheduling overhead, data marshaling time and network delays.

Note that Instrumentation measures only GridServer-related times. It does not show other application delays due to, for example, excessive database load.

For information about turning on Instrumentation, see Enabling Enhanced Task Instrumentation. Also see the *GridServer Developer's Guide*.

Tuning Data Movement

Efficient handling of data can often make or break achieving performance gains in a Grid-enabled application. Instrumentation reveals problems with having too much data per request: serialization, deserialization and network transport times are high compared to the actual Engine-side compute time. There are a number of remedies for inefficient data movement. We survey them here in order from simplest to most complex.

Set Invocations Per Message > 1

By default, the Driver sends one message to the Broker per task submission. For Services that use the model of 'Submit Many, Collect Results", this is not efficient. Any Service that submits a reasonably large number of tasks at one time must always have the `InvocationsPerMessage` option set greater than one. (The optimal number depends on the environment; 25-50 is a good starting point). After submission, `flush` must be called on the

Service to send any remaining buffered tasks. (`waitUntilInactive` and `destroyWhenInactive` also implicitly flush.)

Collect After Submit

The default mode of results collection is immediate. As in the previous section, if your model submits many tasks, then waits for results, this can cause inefficiencies. Specifically, the collection of results can slow down submission of tasks, resulting in idle Engines that can be working on your Service. Setting the collection type to `AFTER_SUBMIT` ensures that the tasks are worked on as soon as possible. After submission, call `waitUntilInactive` or `destroyWhenInactive` to begin collection.

Stateful Processing

GridServer supports two related mechanisms that link client-side service instances to Engine-side state, thereby reducing the need to transmit the same data many times. The two mechanisms are initialization/update data, and Service affinity.

Initialization/Update Data

Making data that is constant across an entire set of task requests into Service initialization data is recommended. Initialization data is transmitted once per Engine, rather than once per request. Designing long-lived volume-based applications that typically process thousands of requests, and compute-intensive applications so that they create many small requests, rather than few large ones is also recommended, for a variety of reasons. See the *GridServer Developer's Guide* for more information.

If a piece of data is not constant throughout the life of the application, but changes rarely (relative to the frequency of requests), it can be passed as initialization data and then changed by using an update method.

The **Service Session Size** parameter, located on the **Grid Components > Engines > Engine Configurations** page under the **Service Caches** heading, controls how much initialization data can be stored on an Engine in aggregate. In other words, if the total size of init data across all loaded service instances exceeds the set value of the parameter, then the least-recently used Service instances are purged from the cache. If Instrumentation shows a non-zero time for Engine Download Instance the second or subsequent time an Engine receives a request from a service, that indicates that the service instance was purged from the cache. Increasing **Service Session Size** might then result in improved performance.

Affinity

The GridServer scheduler uses the fact that an Engine has initialization data and updates from a particular Service to route subsequent requests to that Service. This feature, called affinity, further reduces data movement, because unneeded Engines are not recruited into the Service. (However, if the Service has pending requests, available but uninitialized Engines are allocated to it.) Affinity can be further exploited by dividing the state of an application across multiple client-side Service instances, called Service Sessions. The application then routes requests to the instance with the appropriate data. For example, in an application dealing with bonds, each Service instance can be initialized with the data from one or several bonds. When a request comes in for the value of a particular bond, it is routed to the service instance responsible for that bond. In this way, a request is likely to arrive on an Engine that already has the bond data loaded, yet no Engine is burdened with the entire universe of bonds.

The `STATE_AFFINITY` Service option is a number that controls how strongly the scheduler uses affinity for this service. The default is 1, so set it to a higher value to give your service preference when Engines are being allocated by affinity.

The `AFFINITY_WAIT` Service option controls how long a queued request avoids allocation to an available Engine that has no affinity, in the hope of later being matched to an Engine with affinity. Use this option when the initialization time for a service instance is large. For instance, say it takes five minutes to load a bond. If `AFFINITY_WAIT` is set to two minutes, then a queued request is not assigned to an available Engine that lacks affinity for two minutes from the time the first Engine becomes available. If an Engine that already has loaded the bond becomes available in those two minutes, then the request is assigned to that Engine, saving five minutes of startup time.

The `AFFINITY_DEPTH` Service option is used for invocation-level affinity to determine how deep into the request queue the affinity score must be calculated between all available Engines. It must be greater than zero (the default) if adding affinity to tasks. Larger values can result in longer scheduling episodes, so this number must be chosen wisely.

Affinity can also be set based on Engine Properties instead of state, by using a *Property Affinity Condition*. The scheduler then calculates the affinity score based on state, and then adds a defined number for each satisfied Property Affinity Conditions you have added to the Service.

Compression

Setting the `COMPRESS_DATA` Service option to true (in the Service client or on the **Services > Services > Service Type** page) causes all transmitted data to be compressed. For large

amounts of data, the transmission time saved more than makes up for the time to do the compression.

Packing

Packing multiple requests into a single one can improve performance by amortizing the fixed per-request overhead of GridServer and the application over multiple units of work. The fixed overhead includes TCP/IP connection setups for multiple transits, GridServer scheduling, and other possible application initialization steps.

GridServer's `AUTO_PACK_NUM` Service option is an easy way to achieve request packing. If its value is greater than zero, then that many requests are packed into a single request, and responses are unpacked, transparently to the application. (If the application makes fewer than `AUTO_PACK_NUM` requests, then the accumulated requests are transmitted after one second.) Auto-packing amortizes per-request overhead, but does not factor out common data.

Direct Data Transfer

By default, GridServer uses Direct Data Transfer (DDT) to transfer inputs and outputs between Drivers and Engines. When Driver-Engine DDT is enabled, the Driver saves each request as a file and sends a URL to the Broker. The Engine assigned to the request gets the URL from the Broker and reads the data directly from the Driver. Engine-Driver DDT works the same way in the opposite direction. Without DDT, all data must needlessly go through the Broker.

DDT is efficient for medium to large amounts of data, and prevents the Broker from becoming a bottleneck. However, if the amount of data read and written is small, disabling DDT might boost performance.

Disable Driver-Engine DDT in the `driver.properties` file on the client. Disable Engine-Driver DDT from the **Grid Components > Engines > Engine Configurations** page.

Shared Directories and DDT

In some network configurations, it might be more optimal to use a shared directory for DDT rather than the internal file servers included in the Drivers and Engines. In this case, the Driver and Engines are configured to read and write requests and results to the same shared network directory, rather than transferring data over HTTP. All Engines and the Driver must have read and write permissions on this directory. Shared directories are configured at the Service level with the `SHARED_UNIX_DIR` and `SHARED_WIN_DIR` options. If

using both Windows and UNIX Engines and Drivers, you must configure both options to be directories that resolve to the same directory location for the respective operating systems.

Caching

Service initialization data is effectively a caching mechanism for data whose lifetime corresponds to the Service Session. Other caching mechanisms can be used for data with other lifetimes.

If the data is constant or rarely changing, use GridServer's resource deployment mechanism to distribute it to Engine disks before the computation begins. This is the most efficient form of data transfer, because the transfer occurs before the application starts.

GridCache can also be used to cache data. GridCache data is stored on the Manager and cached by Engines and other clients. See the *GridServer Developer's Guide* for more information.

Data References

GridServer supports Data References: remote pointers to data. A Data Reference is small, but can refer to an arbitrary amount of data on another machine. Data References are helpful in reducing the number of network hops a piece of data needs to make. For instance, imagine that an Engine has computed a result that another Engine might want to use. It can write this result to GridCache. But if the result is large, it travels from the writing Engine to the GridCache repository on the Broker, and then to the reading Engine. If the first Engine writes a Data Reference instead, the second Engine can read the data directly from the first Engine. Data References hide this implementation from the programmer, making network programming much simpler.

The data referenced in a data reference is periodically deleted. By default, this happens every 168 hours, or 7 days. You can configure this time, either to retain data for a longer period, or to delete data more frequently and free space on the client and Engine filesystems. To change this period, go to **Admin > System Admin > Manager Configuration > Services**. Under the **Data Transfer** heading, change the value of **File Time To Live**, and click **Save**.

HTTP Proxy for Engine Data Transfer

In a GridServer deployment where a Broker and its Engines are separated by a WAN, it can be inefficient to transfer the same data over the WAN to multiple Engines from the Broker or the Clients. One solution is to use an HTTP proxy server (such as Squid Web Cache) to

cache the session's init data, which any Engine that works on the session must transfer. You can specify a proxy server in an Engine configuration, and the proxy server caches the Service data for other Engines also using the same proxy server.

For more information about using an HTTP proxy for Engine data transfers, see Configuring a Caching HTTP Proxy Server

Diagnosing GridServer Issues

This section contains information about how to find information to diagnose GridServer issues. It contains information about troubleshooting your installation and gathering information that is helpful if you contact TIBCO for support.

Troubleshooting Overview

When troubleshooting a GridServer installation, first try the following:

1. Ensure that there are no service outages in your environment, such as network or database changes.
2. Read the log files, as described below.
3. Go through the relevant diagnostic sections below.

Reporting an Issue

To report a GridServer issue, go to the TIBCO Customer Support site at support.tibco.com.

Obtaining Log Files

There are several logs generated by GridServer. Depending on what kind of issue you are troubleshooting, you might need to examine one or more logs. These include Manager, Driver, Engine, and Engine Daemon logs.

The current log is always named `gridserver.log`, `driver.log`, or `engine-Instance#.TIMESTAMP`. When logs reach a configured size, they are “rolled over”, and the old log is time stamped. Logs are also moved when a non-Driver component first starts, and are timestamped with the last modification date of the file. Optionally, the process ID of the component doing the logging can be added to the filenames of the current log file.

Manager Logs

Manager Logs are generated on the console window on Windows machines if the Manager is not run as a service. They are also generated on UNIX machines if the Manager is run in the foreground on the console. Because GridServer is usually run as a service or in the background, there are several other ways to view the Manager log:

- In the GridServer Administration Tool, go to **Diagnostics** > **Real Time Log**. This displays new lines of the log as they happen, in a new window. However, it does not display any historical information. Click the **Snapshot** button to open a frozen duplicate of the current log window.
- Go to **Diagnostics > Manager Diagnostics**. This page enables you to display the Manager log, and other information in a specified time period. You can display or create a .ZIP file of the results.

  For example, to view Manager log results, select **Manager Log**, then select a time range. You can then display the log on-screen by clicking **Display**, or save it in a compressed file by clicking **Download**.
- The Manager log is available directly at `DS_DATA/logs/server/*` or the location specified at **Admin > System Admin > Manager Configuration > Logging**.

The Manager log can be set to different levels of granularity, ranging from **Severe**, which provides the least amount of logging information, to **Finest**, which logs the most information. By default, this level is set at **Info**. For debugging purposes, it might be necessary to set the level higher, to **Finer** or **Finest**.

To change the log level:

1. Go to **Admin > System Admin > Manager Configuration > Logging**.
2. In **Default Debug Level**, select a new level.

Configuration options for Manager logs, such as maximum file length and timestamp format, can also be set on the **Manager Configuration** page, in the **Logging** section.

Warning

If you set the log level to Finer or Finest, a large volume of logs are generated, which can adversely affect performance and overwhelm your Manager's file system over time. Remember to change back the log level after you are done troubleshooting.

Engine and Daemon Logs

Each Engine and Engine Daemon generate their own logs. These logs can be accessed directly on Engines. However, because Engines are typically installed in several different machines, there are ways to view logs remotely from other computers. The following procedures describe how to read Engine logs.

To view log files directly, look in the following directories in each Engine install directory:

- Engine instance logs: *install-dir*/`work`/*machine name–instance*/`log`/`*`
- Engine Daemon logs: *install-dir*/`profiles`/*machine name*/`logs`/`engined.log`
- Also examine other `.log` files in the Engine tree.

To read a remote Engine log in a scrolling window

Procedure

1. Go to **Grid Components > Engines > Engine Admin**.
2. Select an Engine.
3. From the **Actions** list, select **Remote Log**.

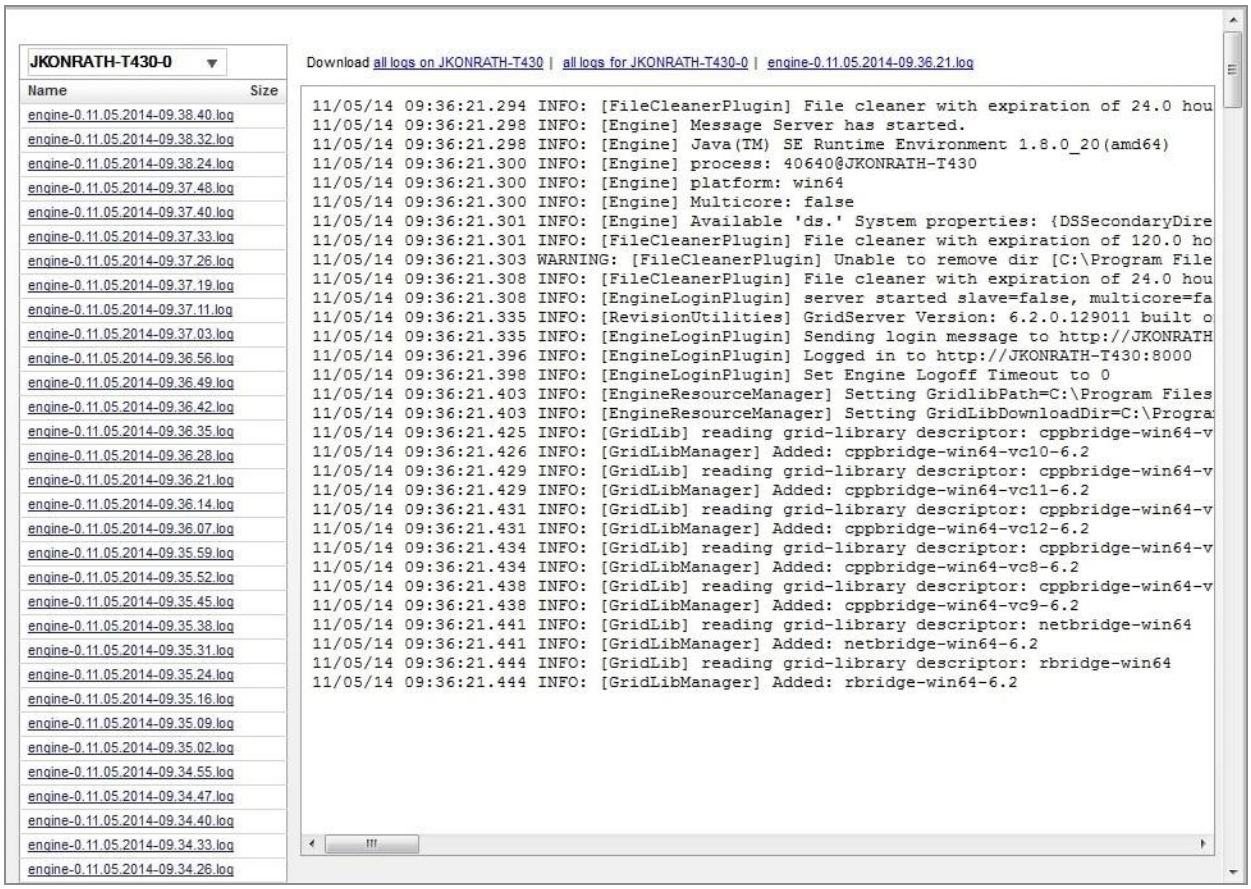
   This opens a window that displays the log for the Engine. Logging information is displayed as it is generated. This does not, however, display any prior logging history.

To access previous logs remotely

Procedure

1. Go to **Engines > Engine Admin**.
2. Select an Engine.
3. From the **Actions** list, select **Log Files**.
4. A window opens with a list of links for each of the logs residing on that Engine. The links are displayed by date and time. You can do any one of the following actions:
   - Select an Engine Daemon or a particular Engine from the list in the upper left. This shows all of the log files on that Engine Daemon or Engine and their sizes. You can also type in the list box to filter the list to partial matches quickly.
   - Click a log file name and its content is displayed to the right in the window.

— Click the links in the upper right to download a ZIP archive of all log files on the host, a ZIP archive of all log files of an Engine Daemon or Engine instance, or a particular log file.



The Log Files window

You change the log level for Engines in Engine configurations, in the **Log** section. You can also set other logging options in Engine configurations, such as the maximum log size, and when log files are automatically cleaned.

Application Server Logs

The Tomcat application server used to run the GridServer Manager also generates logs that can be helpful in diagnosing issues. Logs are maintained in the base directory in the `logs` directory.

Monitoring the Tomcat Application Server

When troubleshooting the Tomcat Application Server, you can use JMX to monitor Tomcat.

For more information about using JMX and Tomcat, see http://tomcat.apache.org/tomcat-6.0-doc/monitoring.html.

Monitoring Engines Using JMX

JMX can also be used to monitor Engines' JVMs. You can enable this in an Engine Configuration, in the **Engine JVM** section, by setting a command-line argument to start the JMX server.

A parametrized incremental port is available for passing into Engine JVM arguments. For example, setting the command-line argument `-Dcom.sun.management.jmxremote.port=${startport:9977}` sets the JMX port to 9977 plus the Engine instance number.

Diagnosing Network Issues

A common issue in a GridServer grid is that a misconfiguration in component communication on a network causes Service failure. This section gives a few common issues to troubleshoot in this situation. For more information about configuring GridServer with regard to networks, see the *GridServer Installation Guide*.

Direct Data Transfer (DDT)

By default, DDT is enabled. When a Service creation or request is initiated on a Driver, the init data or request argument is kept on the Driver, and only a URL is sent to the Manager. When an Engine receives the request, it is sent to the URL and downloads the data directly from the Driver. DDT is set in the `driver.properties`, by setting `DSDirectDataTransfer` to true. Also, by default, the Driver downloads the output data directly from the Engine. This is set in the Engine Configuration by setting the **Direct Data Transfer Enabled** option to `true`. The Driver uses an internal file server by default, at the next available port from the value of `DSWebserverPort`, set in the `driver.properties` file. The Engine listens, by default, on port 27159. This is set in the Engine Configuration.

DDT transfers input and output data over HTTP using the Apache HttpClient library. This transfer activity is logged at the `Finer` level in both the Engine and Driver logs. It does not display at the `Fine` level. To verify that the data is being transferred directly, set the Engine and Driver log level to `Finer`.

When the Engine reads the input data, the download of the input (`.in`) file displays in the Engine log. For example:

```
Finer: [MainClientExec] Executing request GET /ds-
7466344146886677638/5.in HTTP/1.1
```

When the task retrieves the output data, the download of the output (`.out`) file displays in the Driver log. For example:

```
Finer: [MainClientExec] Executing request GET /data/bapa101-0/ddt/ds-
2491378560399007707/0.out HTTP/1.1
```

The remote host and port for each request, (for example, `192.168.32.137:1420` for the Driver's file server, or `engine-host:27159` for the Engine) display in the adjacent Opening connection and Host: header lines.

The full DDT URLs (of class `com.livecluster.util.UrlByteSource`) are also visible in the task and result data in the Driver log. For example:

```
Finer: [wire] http-outgoing-0 >> "<dataSource
c="com.livecluster.util.UrlByteSource" ...><property n="url"
v="<http://192.168.32.137:1420/ds-7466344146886677638/5.in>"/>"

Finer: [wire] http-outgoing-0 << "<resultSource
c="com.livecluster.util.UrlByteSource" ...><property n="url"
v="<http://10.126.209.12:27159/data/bapa101-0/ddt/ds-
2491378560399007707/0.out>"/>"
```

The most common problems with DDT are firewall issues. Use `telnet machine:port` to test connections between components. If you are having problems with DDT, try disabling it and temporarily running in data transfer mode.

Note: NET 4.8 and .NET 8 Driver require `FINEST` (not `FINER`) on the driver side — `HttpSupport.Get()` gets the URL log on `Level.FINEST`.

Data Transfer

Setting the `DSDirectDataTransfer` option in `driver.properties` to false causes the Driver to upload all input data to the Broker. Also set **Direct Data Transfer Enabled** to `false` in your Engine configurations, and the Driver downloads the output data from the Broker. The data transfer settings for the Broker are at **Admin > System Admin > Manager Configuration > Services**, under the **Data Transfer** heading. They are **Store Input to Disk** and **Store Output to Disk**.

Connection Issues

When using data transfer mode, messages like the following message mean that the Engine has lost connection to the Director:

```
Warning: [WebServerBridgePlugin] Failed ping attempt on
http://161.2.27.160:27159/data/ping.html, java.lang.RuntimeException:
java.net.ConnectException: Connection refused: connect
```

Ensure that the Director is running and that the IP address in the log is the IP address of your Director. From your Engine machine, telnet to port 27159 of the Director. If the connection is refused, you might have a network problem. When using DDT, telnet between Driver and Engine to rule out network problems.

Timeout Issues

If you get timeout messages such as the following message, you might need to adjust the configuration for this client:

```
04/15/05 09:18:57.964 Warning: [global] Error reading from
http://10.47.117.158:27159/data//2500-dklptt3z-0/ddt/4937299722762820807
/0.out.z: java.io.IOException: Timed out reading
http://10.47.117.158:27159/data//2500-dklptt3z-
0/ddt/4937299722762820807/0.out.z
```

With Direct Data Transfer (DDT), the settings to adjust are at **Admin > System Admin > Manager Configuration > Communication**, under the **Data Transfer** heading. If the Driver is timing out attempting to read the output file from the Engine, increase the values under **Driver Data Transfer**. Also check that you can access the Engine's file server port 27159 using telnet.

If you are not using DDT for Engines, or if you are using the .NET Driver, the relevant settings are under the **HTTP Connections** heading of the page above. For example, you might want to increase the **Read Timeout** setting.

Diagnosing Engine Issues

The following section gives some information about reading logs related to Engines, and solutions to some common issues.

If problems occur with one particular Engine on the grid and the cause is not immediately obvious, it might be easier to reinstall the Engine rather than go on a long troubleshooting exercise. If the problem persists after reinstallation of the Engine, then investigate for issues with the network, application, or machine setup.

For information about managing Engines, see Managing Engines

Engine Logins, Restarts, and Failures

After GridServer starts, Engine Daemons and Engines log in to the Manager. In the Manager log, messages similar to the following message appear when this happens:

```
Info: [EngineEvent] EngineDaemon:S08048-10.103.8.48:Added
Info: [EngineEvent] Engine:Joe-0:Added
```

The Broker sends periodic heartbeats to the Engine. If these heartbeats fail, you see messages similar to the following message:

```
Info: [ProxyMonitorPlugin] Killing proxy S08049-10.103.8.49 on
EngineDaemonServicePlugin
Info: [EngineEvent] Engine:S08048-0:Logoff:Killed by the proxy monitor
Warning: [EmploymentOfficePlugin] Engine:S08048-0:Died
```

If the Engine cannot perform a heartbeat with the Broker then after 3 retries, you see the following message:

```
Warning: HeartbeatPlugin Couldn't send a heart beat to the Manager
failure to process HTTP request in POST: Connect failed, so the client
logs off.
```

If the Engine Daemon fails, you see the following message:

```
Warning: [EngineDaemonServicePlugin] Engine Daemon:S08049-
10.103.8.49:Died
```

You can lengthen the period between heartbeats at **Admin > System Admin > Manager Configuration > Communication**.

If the Engine fails, by default it restarts and tries to log in again. Failure messages in the Manager log look similar to the following message:

```
Info: [Scheduler] Engine:NCSILS9027B1GRD-0:Logoff:Ping failed on local
webserver, restarting instance in one minute
Fine: [EngineProxy] Logging off: NCSILS9027B1GRD-0
Fine: [EngineLoginManagerPlugin] Logging off proxy + NCSILS9027B1GRD-0
code=3
reason=Ping failed on local webserver, restarting instance in one minute
Info: [EngineEvent] Engine:NCSILS9027B1GRD-0:Removed
```

The following information is included in the Engine Daemon log reports:

```
Info: [Scheduler] Engine:nldn8347dww-
0:NotifyKillTask:1208119245372388313-1208119245372388313-0
Info: [Scheduler] Engine:nldn8347dww-0:TaskDied:1208119245372388313-
1208119245372388313-0
Info: [Scheduler] Engine:nldn8347dww-0:Logoff:Killed by the proxy
monitor
Warning: [EngineEvent] Engine:nldn8347dww-0:Died
Info: [EngineEvent] Engine:nldn8347dww-0:Removed
```

JVM Issues

If there are messages similar to the following message in the Manager log, the JVM might be running out of memory:

```
Severe: [HeartbeatPlugin] while sending heartbeat
java.lang.OutOfMemoryError: unable to create new native thread
```

You can increase the Engine JVM maximum heap size in the Engine configurations.

If an Engine fails, or the logs on an Engine end abruptly, the cause might be a Java failure. Check for Java HotSpot compiler error logs in the Engine root directory. They have names like `hs_err_pidXXXX.log`, and contain information about problems in native code. The

information can be used for a web search to see if it is a known problem. Also check if any native C code is being called by the application that fails.

Connection and Firewall Problems

A common problem is that Drivers or Engines are not connecting to the Director or Broker. This is typically due to Firewall or DNS issues. Correct DNS configuration is essential in GridServer installations. Use the `telnet` command to test connections from the Manager to the Engine and conversely.

All supported Windows versions enable the Windows Firewall by default. This automatically blocks any incoming traffic. To make sure that your Engine can properly communicate, the inbound port for the Engine's File Server must be open to traffic. By default, this port is set to 27159. It can be changed in the Engine Configuration. Configure your Windows Firewall to enable use of this port by your Driver machines.

Another possibility is that one of the components is assigning ephemeral ports outside of the range that can be opened. Sometimes systems assign ports outside the range of 49152-65535. You can check this by using `netstat -a`.

Engine Daemon Cannot Log On to Manager

If an Engine Daemon does not connect to a Manager, check the URL in the `intranet.dat` file in the root of the Engine installation directory and see if you can make a connection to it from the Engine machine.

Thread Dumps on Engines

To get thread dumps on Engines, use the java Visual VM tool. It is available at https://visualvm.github.io/.

Using Fusion to Debug .NET Assembly Load Failures

With C# code, a runtime library load failure can take a number of forms such as a `FileLoadException` and might be difficult to debug. The notification only mentions that the assembly load failed, but not why it failed.

To obtain more detailed debugging information about assembly load/bind failures, use the Microsoft Fusion logging system. It is included in Visual Studio .NET:

1. Start `FUSLOGVW.EXE` before opening your application.

2. Open your application.
3. After the failure has occurred, click the **Refresh** button in the Fusion logging window. An entry related to the process you just ran must appear.
4. Highlight this entry and click **View Log** to get a detailed report of the .NET Framework's attempts to load your assemblies.

Diagnosing Driver Issues

The following section gives some information about reading logs related to Drivers, and solutions to some common issues.

Driver Cannot Log In to Manager

When a Driver cannot log in to a Manager, messages similar to the following message appear in the Manager log:

```
LoginPlugin Can't log in to PrivateBrokerConnection
LoginPlugin Failed login on
http://162.60.27.152:8000/livecluster/director/PublicDriverChannel so
trying next. Error:
File not found:
http://162.60.27.152:8000/livecluster/director/PublicDriverChannel
```

When this happens, try the following:

- Ping and telnet to the Manager and port number to test network connectivity.
- Check the username, password, and Director values in the `driver.properties` file or values coded in application.
- If using SSL, make sure the certificate is valid.

Client Timeout Issues

You might see timeout error messages in the Driver log similar to the following message:

```
java.io.IOException: Unexpected exception while reading data from
http://172.24.68.49:1667/ds-6353115010724789381/job.tasklet:
com.livecluster.util.threadpool.TimeoutException: Thread DefaultPool-6:
http://172.24.68.49:1667/ds-6353115010724 789381/job.tasklet timed out
```

The **Client Timeout** setting allows for the Driver to log back in after logging off due to temporary network issues, without interrupting Services.

Go to **Admin > System Admin > Manager Configuration > Communication**. Under the **HTTP Connections** heading, you can increase read and write timeout values for Drivers.

Also, the Driver must not fail over to another Broker too quickly. This is set in `DSBrokerTimeout` in the `driver.properties` file. The default is 300 seconds, or 5 minutes.

To ensure your tasks are not lost in the event of Broker failure, perform the following actiond:

1. GNavigate to **Admin > System Admin > Manager Configuration > Engines and Clients**.
2. Under the **Engine Login** heading for a period of time longer than the **Client Timeout Minutes** on the same page, under the **Client Management** heading, set **Engine Timeout Minutes** at.
   For example, if **Client Timeout Minutes** is set to 5 minutes, a good number for **Engine Timeout Minutes** is 8 minutes. Note that changes to this value are only applied at the next Engine log in.

Manager Turning Away Clients

By default, Drivers are turned away if there is a version mismatch between Client and Manager. A message similar to the following message appears in the Manager log:

```
[BrokerLoginManager] Turning away client: Client module version
mismatch: ClientJavaVersion version: 7.0.0: should be 7.1.0.
```

To override this, go to **Admin > System Admin > Manager Configuration > Admin**, and under the **Version Management** heading, set **Allow Driver Version Mismatch** to `true` until the Client can be upgraded.

Diagnosing Manager Issues

The following section gives some information about reading logs related to Managers, and solutions to some common issues.

Manager Port Issues

If your Broker does not log in to the Director, check the Manager logs for a message similar to the following message:

```
BindException while attempting to create a ServerSocket at
myaddress:5635
```

This means another process is using port 5635, which is used for communication between the Broker and Director. If it is another Director, or another process that cannot be stopped, that port number must be changed on the Director and Brokers to another port. Use the `netstat -a` command to check port usage.

Out of Memory Issues

If your Manager appears to run out of memory, and an out of memory error is printed to the Manager log, you might need to increase the Java heap size. This typically only happens if the Manager is running many Services with many inputs.

The Java maximum heap size is set in the `server.sh` or `server.bat` file, and is 4096 MB by default. It can be increased by changing the environment variable `MAX_HEAP`.

Deployment Issues

The GridServer Manager is responsible for deploying application resources to Engines. The Manager periodically generates a checksum list of every file in its resources area. If any checksums have changed, or have been deleted or added since the last time the list was generated, it notifies the Engines.

Do not deploy resource files that only differ by case in UNIX. Resource files names must be different. If you deploy different files named `hello.zip` and `Hello.zip`, they collide on Windows machines. Also, you cannot deploy files that contain `#` or end in `.tmp`.

For more information about resource deployment and Grid Libraries, see Deploying Services.

Grid Library Issues

If you have a `GridLibraryException` failure when deploying a Grid Library, check your `grid-library.xml` for incorrect file names. Also ensure that files and file reference in `grid-library.xml` are of the correct case. Also check that the `jar-path`, `lib-path`, `assembly-path`, and `command-path` are correctly specified.

.Net GridLib - DLL Missing

When deploying a .NET assembly, all DLL libraries in `win32/lib` and all other directories specified in Engine Configuration are not used. Put all related DLL files into the same Grid Library, or another Grid Library that this Grid Library depends on. A missing DLL causes a log message similar to the following:

```
INFO  DSForward Message DD7C5440-D3B9-42D6-9201-F7E7CE3AE847 submitted
DSHandler Service invocation 0 failed for DD7C5440-D3B9-42D6-9201-
F7E7CE3AE847
Exception: DataSynapse.GridServer.Client.ServiceInvocationException
Message: spri12d10052
Source: GridServerNETClient
```

Service Failures

If a Service is not defined in the **Services > Services > Service Types** page, a message similar to the following is produced in the Driver log:

```
Service Type not found in the Service Type Registry: JavaCalculator
```

If the JAR file containing the class of the Service has not been loaded into the correct library, a message in the Driver log is produced:

```
Info: [TaskDispatcher$DispatcherJob] Canceling due to task failure and
AutoCancel.LIBRARY_LOAD, Exception type:
java.lang.ClassNotFoundException
```

If a Service fails, an error is returned to the Callback method. Check your code for errors.

```
Info: [ServiceEvent] SubmittedTask:3393405147689298243-
3393405147689298243-1
CalculatorCallback::handleResponse : Result for task [B@c713d2
Info: [ServiceEvent] CompletedTask:3393405147689298243-
```

```
3393405147689298243-0:Total:2
CalculatorCallback ::handleError : Id = 1, Exception :
java.lang.Exception: service failed [java.lang.Exception: service failed
```

If a Service fails to run even if there are available Engines, check that the Service or Engine discrimination is not preventing the Service task from running. Also check for usage of dependencies in the application. The Service or task might be waiting for a Service or task it is dependent on.

GridCache Issues

GridCache is a repository on the Broker that is cached by Drivers and Engines (This is the GridCache Global mode). The Driver or Engine writes to the GridCache and then the Broker caches them automatically to the Server and propagates the changes so the file appears local to Drivers and Engines. When the cache is changed, the Manager sends invalidation messages in the heartbeat messages to the Engines and Drivers.

For more information about GridCache, see the *GridServer Developer's Guide*.

Timeout Issues

Messages similar to the following message are caused by timeouts accessing GridCache:

```
com.livecluster.util.threadpool.TimeoutException: Thread DefaultPool-91:
sun.net.www.protocol.http.HttpURLConnection:http://chialseg45:8000/livec
luster/gridcache/?query=1/
```

You can increase GridCache client read and write timeout at **Admin > System Admin > Manager Configuration > Cache**.

Eviction of Cache Entries

There is an eviction process, which removes entries from the cache when they expire. A message similar to the following message appears in the logs:

```
Finer: [GridCache] Evicting stale entries from the cache
```

A `Time To Live` value is configured in the cache schema. The default is -1, meaning entries never expire. In the case of a Global Cache, if an entry has expired, it is removed, and

hence removed from all of the locally cached copies. Note that when a schema changes, the changes only apply to regions created after that schema change. The changes do not affect existing regions using the schema.

In addition, there is also a Keep Alive time associated with the cache, set in the cache schema. This specifies how long the locally cached copies of the region remain on the client once there are no longer any local references to the cache. References are decremented when you call `close()` on the cache or the Cache reference is collected by garbage. The default is 60 seconds.

Client Runs Out of GridCache Space

If the in-memory cache size is exceeded, information is pushed to disk (except for .NET Drivers or CPP Drivers, which do not support disk-based cache.) If the disk cache size is exceeded, older entries from the disk cache are removed to make way for new entries.

For example, if the Engine has run out of space for the GridCache, messages similar to the following message appear:

```
Fine: [GridCache] Pushing GridCacheRegion::trade_10985 out of the cache
```

To prevent this, you can increase the cache sizes. For Engines, the **Disk Cache Size** and **Memory Cache Size** settings are in the Engine Configuration. For Drivers, they are the `DSCacheMaxMemInMB` and `DSCacheMaxDiskInMB` properties in the `driver.properties` file.

Database Issues

GridServer has two databases: an internal database, only used internally by the Manager; and an optional external database for reporting. For information about installing the reporting database, see the *GridServer Installation Guide*. For a complete description of the reporting database schema, see Reporting Database Tables.

Connection Problems

If GridServer loses connection to a database, you see a message in the Manager log similar to this:

```
Warning: [DirectorReportingPlugin] Database is unavailable:
java.sql.SQLException: while creating new connection to
:jdbc:oracle://rcfdsp01.com:9158/, Connection refused
```

Use a tool like DBVisualizer to access the SQL database using the JDBC URL and Driver JAR to check if you can connect and access the database.

JDBC Problems

If GridServer cannot find the JDBC driver, a message similar to this appears:

```
error for postgres:
SQL block processing is ON:
        "begin" begins block
        "end" ends block
        "/" terminates block statement
        Overridden newline character separates statements in blocks
Performing Class.forName() on driver org.postgresql.Driver
Driver jar needs to be placed in the classpath
java.lang.ClassNotFoundException: org.postgresql.Driver
        at java.net.URLClassLoader.findClass(URLClassLoader.java:382)
        at java.lang.ClassLoader.loadClass(ClassLoader.java:418)
        at sun.misc.Launcher$AppClassLoader.loadClass(Launcher.java:355)
        at java.lang.ClassLoader.loadClass(ClassLoader.java:351)
        at java.lang.Class.forName0(Native Method)
        at java.lang.Class.forName(Class.java:264)
        at com.datasynapse.commons.sql.CreateDB.validateProperties
(CreateDB.java:112)
        at com.datasynapse.commons.sql.CreateDB.<init>(CreateDB.java:52)
        at com.datasynapse.commons.sql.CreateDB.main(CreateDB.java:43)
```

The JDBC JAR needs to be in `DS_MANAGER/webapps/livecluster/WEB-INF/lib`.

Reporting Database Write Failures

If GridServer cannot write to the reporting database, it starts discarding events, and this message appears:

```
[BrokerReportingPlugin] Client Backlog full, event discarded
```

Also, the Manager might fail to commit a piece of reporting data, and this message appears:

```
Severe: [DirectorReportingPlugin] engine stat commit failed
```

This must not affect operation of the grid, as the reporting features are separate from the core functionality. If this message occurs once, it is most likely caused by a temporary connection problem with the reporting databas,e and can safely be ignored. If several messages like this occur, it indicates that the reporting database parameters need to be adjusted, especially if reporting data is relied on for statistical data.

Reducing the Transaction Isolation to `TRANSACTION_READ_UNCOMMITTED` improves the performance of inserts into the reporting database. Increasing the number of connections to the database does not help because GridServer has a set number of threads running to the database from the backlogs. Increasing the size of the backlogs helps deal with peaks of activity in the Manager. Increasing the commit interval to 30 or 60 seconds also improves performance. Use of auto commit is not recommended for a production grid.

Troubleshooting Tools

The following tools and utilities can be helpful when troubleshooting a GridServer issue.

Task Admin Page

You can view Service invocation status data on the **Task Admin** page, which can be helpful when troubleshooting Service execution. Normally, Service invocation data is purged after a Service has completed. This can be changed by setting the `PURGE_INVOCATION_DATA` Service option. This data is stored in memory on the Broker, so this setting might need to be modified depending on how the Service is used. The following options are available:

- `INVOCATION_COMPLETED` — data for each request is purged as soon as the request has completed. In this case, the data is only available when the request is queued or being processed. This must be used when the Service might be a long-running open Service, as each entry requires memory.
- `INVOCATION_COMPLETED_SUCCESSFULLY` — data for each request is purged as soon as it has completed, but only if the request was successfully completed. This is useful when diagnosing failure of long-running Services.
- `SERVICE_COMPLETED` — data is purged when the Service instance has completed. As long as the instance is running, the data is available. This is the default.

- `SERVICE_REMOVED` — data is removed when the completed Service instance is removed from the Admin interface. This enables you to view data after an instance completes, at the cost of memory overhead on the Broker. The instance is removed either automatically according to the Service Cleanup settings on the Broker, or when it is manually removed.

Task Queue Dump

You can view a textual representation of the Task Queue, which can be helpful in debugging. The Task Queue Dump contains information about Service sessions and tasks including task counts, retries, and conditions.It can be accessed from the **Diagnostics > Manager Diagnostics** page, by selecting Task Queues, and requires a Broker on the Manager.

Enabling Enhanced Task Instrumentation

Normally, a submitted task or remote Service Invocation's execution time is measured only from start to finish. But often it is useful to be able to track the time that is spent in the various stages of this process, including input serialization, disk writing, task message submission, task queuing, task fetching, data transport, input deserialization, task processing, output serialization, output transport, queuing, and so on. This helps you understand the timing characteristics of distributed computing, optimize the process, and diagnose problems with greater ease.

To enable enhanced task instrumentation

Procedure

1. Go to **Admin > System Admin > Manager Configuration > Services**.
2. Under the **Instrumentation heading,** set **Enable** to `True`.
3. Click **Save**.

When enabled, task instrumentation applies to all Services on the Manager.

Warning

Task instrumentation slows down the Manager, and also requires additional disk space, so it is important to disable it after you finish using it. It is NOT recommended for production systems.

To view data generated by enhanced task instrumentation

Procedure

1. Go to **Services > Service Session Admin**.
2. Find the Service that you wish to view, and select **View Instrumentation** from the **Actions** menu. Note that this choice only appears after the Service finishes running.

A table appears showing data collected by enhanced task instrumentation for the Service. For more information about instrumentation, see the *GridServer Developer's Guide*.

Process Explorer

Process Explorer is a free Windows utility from sysinternals.com. It provides detailed information about a process. For a particular process, you can view the DLLs it has loaded and the resource handles it has open. This is useful when Engines stop responding, wait on files indefinitely, or DLL issues of applications running on Engines. A Find capability enables you to track down a process that has a resource opened, such as a file, directory or Registry key, or to view the list of processes that have a DLL loaded.

Dependency Walker

Dependency Walker is also very useful for troubleshooting system errors related to loading and executing modules. It detects many common application problems such as missing modules, invalid modules, import/export mismatches, circular dependency errors, mismatched machine types of modules, and module initialization failures. From Visual Studio or from `dependencywalker.com`, install `depends.exe`.

Event Streaming by Using Apache Kafka

Event streaming exposes the internal events that are generated in GridServer so that users can use the information for purposes such as monitoring. You can stream component-wise events by using Apache Kafka Message framework.

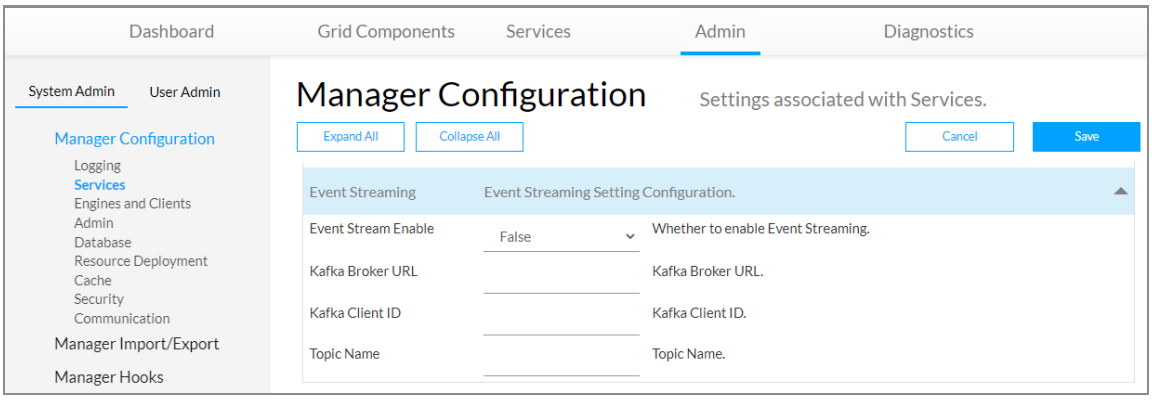
Configuring Event Streaming

Prerequisites

- You must have Apache Kafka running.
- GridServer Manager must be up and running.

Procedure

1. Log in to the GridServer Administration Tool with an account that has a Security Role with the Manager Configuration Edit feature enabled.
2. Go to **Admin > System Admin > Manager Configuration > Services > Event Streaming**.
3. On the Director, configure the values in the Event Streaming section. You can configure Kafka settings on the Primary and the Secondary Director.



Manager Configuration

Refer the following table to configure the Manager:

Property	Value
Event Stream Enable	Whether to enable event streaming. If it is set to `True`, you must also set the `Kafka Broker URL`, `Kafka Client ID`, and `Topic Name` properties.
Kafka Broker URL	Kafka Broker URL. The URL format must be: `Kafka installed Machine HostName:Port Number`.
Kafka Client ID	ID of the Kafka Client If you use event streaming, the Kafka Client ID value is appended with `_Broker` for a Broker and `_Director` for a Director. For example, if you enter the Kafka Client ID as `Grid_Client`, it is appended as follows: For Broker: `Grid_Client_Broker` For Director: `Grid_Client_Director`
Topic Name	Name of the topic

4. Click **Save**.

Events Captured by Apache Kafka

The following events are captured by using Apache Kafka for GridServer:

Events	Description
Broker Added	Broker is added to the Director
Broker Removed	Broker is removed from the Director

Events	Description
Driver Added	Driver is added to the Broker
Driver Removed	Driver is removed from the Broker
Engine Added	Engine is added to the Broker
Engine Blacklisted	Engine is blacklisted
Engine Daemon Added	Engine Daemon is added to the Director
Engine Daemon Removed	Engine Daemon is removed from the Director
Engine Died	Engine dies (did not log off properly)
Engine Greylisted	Engine is greylisted
Engine Removed	Engine is removed from the Broker
Memory Warning	Server exceeds the configured threshold of memory usage
Service Session Cancelled	Service session is cancelled
Service Session Completed	Service session is completed
Service Session Started	Service session starts
Task Assigned	Task is assigned to an Engine
Task Completed	Task is completed
Task Error	Task fails
Task Submitted	Task is submitted to the Broker

Note:
These events are recorded in the log file generated over Apache Kafka. You can also specify a custom log directory in the `server.properties` file of Apache Kafka. To capture streamed events, write your own consumer code.

Reporting Database Tables

GridServer uses a simple relational database to report grid processing events for historical analysis. This appendix describes the tables in the reporting database.

Data Type Mapping

The following table lists mappings for data types that vary depending on the database type.

Data Type	MS SQL	Oracle	PostgreSQL	HSQL
bigint	bigint	int	bigint	bigint
timestamp	datetime	timestamp	timestamp	timestamp
text	text	clob	text	longvarchar
varchar	varchar	varchar2	varchar	varchar
identity	int not null identity	int (incl. sequence+trigger)	serial or bigserial not null	int generated by default as identity (start with 1) not null
bigidentity	bigint not null identity	int (incl. sequence+trigger)	bigint not null identity	int generated by default as identity (start with 1) not null

batches

Batches that have been scheduled or run.

Column name	Data type	Description
server	varchar(255)	The Manager where the Batch resided or ran
batch_id	bigint not null	The Unique ID number of the Batch Entry
time_stamp	timestamp not null	Timestamp of the event
event	int not null	Event code
class	varchar(255)	Class in the Batch
execution_id	bigint	Unique ID number of the Batch Execution, if applicable
description	text	Description of the Batch Event

brokers

Information about all Brokers that have participated in this grid.

Primary key: pk_brokers(broker_id)

Column name	Data type	Description
broker_id	int not null	Broker ID number
broker_url	varchar(255)	The Broker's configured base URL
broker_name	varchar(255)	The name of the Broker

broker_stats

All statistical reports from Brokers are stored in this table.

Primary key: pk_broker_stats(broker_id, time_stamp)

Column name	Data type	Description
broker_id	int not null	The unique ID of the Broker
time_stamp	timestamp not null	Timestamp of the report
num_busy_engines	int	Number of Engines busy at report time
num_total_engines	int	Number of Engines logged in at report time
num_drivers	int	Number of Drivers logged in at report time
uptime_minutes	float	Time since Broker start in minutes
num_jobs_running	int	Number of Services running at report time
num_tasks_pending	int	Number of tasks pending (not yet assigned to Engines) at report time

driver_events

Reports from Brokers generated when a Driver logs in or out.

Primary key: pk_driver_events(driver_event_id))

Column name	Data type	Description
driver_event_id	identity	ID of the Driver event
username	varchar(255)	Driver username
hostname	varchar(255)	The name of the host running the Driver
time_stamp	timestamp not null	Timestamp of the report
broker_id	int	The ID of the Broker where the event

Column name	Data type	Description
		occurred
event	int	0 for an add, or the reason code for a remove – reason codes are in the event_codes table

engine_events

Reports from Brokers that are generated when an Engine is added or removed. For example, when an Engine logs in or logs out.

Primary key: pk_engine_events(engine_event_id))

Column name	Data type	Description
engine_event_id	identity	The ID of the Engine event
engine_id	bigint not null	The unique ID of the Engine
instance_id	int	The number of the Engine instance
time_stamp	timestamp not null	Timestamp of the report
broker_id	int	The ID of the Broker where the event occurred
event	int	0 for an add, or the reason code for a remove – reason codes are in the event_codes table

engine_info

Administrative information for all Engines that have ever logged in to this Director.

Primary key: pk_engine_info(engine_id)

Column name	Data type	Description
engine_id	bigint not null	The unique ID of the Engine
username	varchar(255)	The username used by the Engine
guid	varchar(255)	Another unique such as a MAC address
IP	varchar(255)	The IP address used by the Engine
install_date	timestamp	When the Engine was installed

engine_stats

Statistic reports from Engine Daemons.

Primary key: pk_engine_stats(engine_id, time_stamp)

Column name	Data type	Description
engine_id	bigint not null	The unique ID of the Engine
time_stamp	timestamp not null	Timestamp of the report
cpu_utilization	float	%CPU total utilization
ds_cpu_utilization	float	%CPU used by DataSynapse processes
total_ram_kb	bigint	Installed RAM reported by the OS in kilobytes
free_ram_kb	bigint	Free RAM reported by the OS in kilobytes
disk_mb	bigint	Free disk reported by the OS in megabytes
num_invokes	int	The number of Engine processes currently running

event_codes

Mappings of event codes to descriptive text.

Primary key: pk_event_codes(code)

Column name	Data type	Description
code	int not null	Number of the code
name	varchar(255)	Description of the code

jobs

Historical information about all Services that have been run by GridServer.

Primary key: pk_jobs(job_id, start_time)

Column name	Data type	Description
	bigint not null	The Service ID
service_type_name	varchar (255)	The Service Type used for the Service
job_class	varchar (255)	The Service class being run. For example: Java: `examples.calculator.service.JavaCalculator` Dynamic Library: `Calculator` .NET: `NETCalculatorService.dll:DataSynapse.Examples.Services.NETCalculator` Command: `ls`
start_time	timestamp not null	When the Service was started

Column name	Data type	Description
end_time	timestamp	When the Service finished
job_status	int	Service status (see job_status_codes table)
num_tasks	int	Number of tasks in the Service
task_time_std	float	Standard deviation of task completion time, in seconds
task_time_avg	float	Mean task completion time, in seconds
priority	int	Service priority when submitted
end_priority	int	Service priority when complete
driver_username	varchar (255)	Submitting Driver user name
driver_hostname	varchar (255)	Submitting Driver hostname
job_name	varchar (255)	Optional descriptive Service name from Description
app_name	varchar (255)	Optional descriptive application name from Description
description	varchar (255)	Optional description from Description
dept_name	varchar (255)	Optional descriptive department name from Description
group_	varchar	Optional descriptive group name from Description

Column name	Data type	Description
name	(255)	
indiv_name	varchar (255)	Optional descriptive individual name from Description
broker_id	int	ID of Broker that ran the Service
task_time_min	bigint	Minimum task time, in milliseconds
task_time_max	bigint	Maximum task time, in milliseconds
session_size	bigint	Service Session size in kilobytes
input_size_total	bigint	Sum of task input size in kilobytes
output_size_total	bigint	Sum of task output size in kilobytes
gridlibrary	varchar (255)	The Grid Library used for this Service instance
gridlibrary_version	varchar (255)	The Grid Library version

job_status_codes

Mappings of event codes to descriptive text.

Primary key: pk_job_scodes(code)

Column name	Data type	Description
code	int not null	Number of the code
name	varchar(255)	Description of the code

roles

Contains user roles.

Primary key: pk_roles(name)

Column name	Data type	Description
name	varchar(255)	Name of the role
description	text	Description of the role
features	text	Features within the role
managers	text	Managers assigned to the role
ldapgroup	varchar(255)	LDAP group of the role
driver_info	text	Driver info for the role

tasks

Historical information about all tasks that have been run by GridServer.

Primary key: pk_tasks(task_rec_id)

Column name	Data type	Description
task_rec_id	bigidentity	
job_id	bigint not null	Service ID
task_id	int not null	Task ID
engine_id	bigint	Engine that eventually ran the task
start_time	timestamp	When the task was started, or 1970-01-01 00:00:00.0 (UNIX epoch) if the task was never started
end_time	timestamp	When the task finished
task_status	int	Task status (see task_status_codes table).
num_reschedules	int	Number of times the task was retried
engine_instance	int	Number of Engine instance that ran the task
task_info	varchar(255)	Task information
broker_id	int	The Broker that handled the task
description	varchar(255)	Description of the task
input_size	bigint	Size of the Task's input in kilobytes
output_size	bigint	Size of the task's output in kilobytes
submit_time	timestamp	When the task was submitted

task_status_codes

Contains mapping of event codes to descriptive text.

Primary key: pk_task_scodes(code)

Column name	Data type	Description
code	int not null	Number of the code
name	varchar(255)	Description of the code

user_events

Historical user events.

Primary key: pk_user_events(user_event_id)

Column name	Data type	Description
user_event_id	identity	The ID of the event
server	varchar(255)	The Server where the event occurred
username	varchar(255)	The user recording the event
IP	varchar(255)	IP address of the user that caused the event
time_stamp	timestamp not null	When the event occurred
handler	varchar(255)	The Internal handler class that recorded the event
event	text	A description of the event

users

GridServer user accounts.

Primary key: pk_users(user_id)

Column name	Data type	Description
user_id	identity	The ID of the user
username	varchar(255)	The user name
user_access	int	The account’s user access
user_info	text	The account’s user info
personalization	text	Stores information about table and UI personalization
roles	text	Roles assigned to the user

Scheduler Instrumentation Database Table

GridServer uses a simple relational database to report operations of GridServer Scheduler. The `scheduler_info` table is described here.

Scheduler Instrumentation Database Schema Data Type Mapping

The following table lists mappings for the data types that vary depending on the database type:

Data Type	MS SQL	Oracle	PostgreSQL
varchar	varchar	varchar	varying
bigint	bigint	number	bigint

scheduler_info

Column name	Data Type	Description
type	varchar(20)	It has various types such as - Start-Episode, End-Episode, matchItems, Checkpoints, Waiting Job List, and Waiting Engine List
time	bigint	Timestamp

Column name	Data Type	Description
value	varchar(2048)	It contains value in JSON format

Engine Instrumentation Database Table

GridServer uses a simple relational database to report the Engine Balancing process. The `engine_ins` table is described here.

Engine Instrumentation Database Schema Data Type Mapping

The following table lists mappings for the data types that vary depending on the database type:

Data Type	MS SQL	Oracle	PostgreSQL
varchar	varchar	varchar	varying
bigint	bigint	number	bigint

engine_ins

Column name	Data Type	Description
type	varchar(20)	It has various types such as - Engine Disallowed, Engine Selected, Engine to Broker, Engine Logoff, and Director Balancing
time	bigint	Time stamp
value	varchar(2048)	It has a different value stored as per the type

TIBCO Documentation and Support Services

For information about this product, you can read the documentation, contact TIBCO Support, and join TIBCO Community.

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

Documentation for TIBCO GridServer® is available on the TIBCO GridServer® Product 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.
- To submit ideas, suggestions, or feedback for TIBCO products, visit our product Support 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. 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, GridServer, FabricServer, GridClient, FabricBroker, LiveCluster, and SpeedLink 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 © 2001-2026. Cloud Software Group, Inc. All Rights Reserved.