TIBCO DataSynapse GridServer® Manager

Developer's Guide

Version 7.1.0
July 2022

Document Updated: September 2022

Contents

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`.

TIBCO GridServer® Application Development

This section is your starting point for developing applications that use your GridServer installation. The document is divided into several sections to help you understand the principles of the GridServer system, and how to program applications utilizing GridServer.

TIBCO GridServer® Programming Options

There are several options available to you when you adapt your applications to use GridServer. The following sections describe how to use each of them.

Services

Services provide for remote execution of code in a way that is scalable, fault-tolerant, dynamic and language-independent. Services can be written in a variety of languages and do not need to be compiled or linked with DataSynapse code. There are client-side APIs to create Service Sessions using Java, C++, COM, R, and .NET. A Service object on a client can create and use a Service implemented in the same or another language. In the Service model, requests on the client are routed over the network, ultimately resulting in invocations on a remote machine, and response values make the reverse trip.

With GridServer, Services are *virtualized*; rather than send a request directly to the remote machine hosting the Service Session, a client request is sent to the GridServer Manager, which enqueues it until an appropriate Engine is available. The Manager selects which Engine services a request. The first Engine to dequeue the request hosts the Service Session. Subsequent requests can be routed to the same Engine or can result in a second Engine running the Service concurrently. For information about how this decision is made see the *TIBCO GridServer® Administration*. If an Engine hosting a Service Session becomes unavailable, another takes its place. This mechanism, in which a single *virtual* Service Session is implemented by one or more *physical* Sessions (Engine processes) provides for fault tolerance and essentially unlimited scalability.

See Creating Services for details how to implement Services; Accessing Services explains how to use Services in your application.

PDriver

The Parametric Job Driver, or PDriver, is a Driver that can execute command-line programs as a parallel processing service using the GridServer environment. This enables you to write a simple script to run a program on several Engines, and return the results to a central location.

PDriver scripts, which are written in the PDS scripting language, enable you to run the same program on Engines several times with different parameters. A script is used to define how these parameters change.

One way PDriver scripts can achieve parallelism is to iteratively change the value of variables that are passed to successive tasks as parameters. A script can step through a range of numbers and use each value as a parameter for each task that is created. Or, a variable can be defined containing a list of parameters.

For more information see PDriver.

Python

The Python Driver, or PyDriver, is a Driver that can submit Python scripts for execution in the GridServer environment. This enables you to create Python scripts to run on multiple Engines and return the results to a central location. You can achieve parallelism by iteratively changing the script that is passed to successive tasks.

Resource Deployment

Service resource files that are used by Engines are centrally managed, starting at the Director. The centrally located resources on the Director are then synchronized to Managers, which then synchronize them with Engines.

Grid Libraries are the method of deploying resources to Engines. They are an archive containing a set of resources and properties necessary to run a Service, along with configuration information that describes how those resources are to be used. Grid Libraries can contain Java classes and JARs, native libraries, .NET assemblies, configuration files, Java system properties, Engine hooks, and alternate JREs needed to run a Service. They can also contain references to other Grid Libraries as dependencies. A Service Session can use a Grid Library by setting the appropriate options for the Service Type used by the session.

The `build` directory of the SDK includes an example ANT build script that can be used to build Grid Libraries. The examples in the SDK can be automatically packaged as Grid Libraries by using this script and included configuration files. Each Service example contains `grid-library.xml` and `build.properties` files. The `build` directory contains `build.xml`, `deploy.bat`, and `deploy.sh`, which parse the `grid-library-build-properties` files to create Grid Libraries.

For more information about packaging Grid Libraries, see Creating Grid Libraries. For information about deploying Grid Libraries, see the *TIBCO GridServer® Administration*.

Logging and Debugging

GridServer contains comprehensive logging facilities on Engines. This can be used to diagnose problems with Services running on Engines, and your application can write information to these logs. This section contains an overview of GridServer's log facility, plus information about using it from your application, and how to attach a debugger to an Engine if needed.

Logging Overview

GridServer uses the `java.util.logging` package for its internal logging, to provide diagnostic messages to the console and to file. This section covers how to access these logs, and how to interface with the loggers.

The logger uses the following log levels, in order:

Level	Description
Severe	Indicates serious failures
Warning	Indicates potential problems
Info	Displays informational messages
Config	Displays static configuration messages

Level	Description
Fine	Provides tracing information
Finer	Indicates a fairly detailed tracing message
Finest	Indicates a highly detailed tracing message

Typically, the Info level is sufficient for most purposes and is best to use on a busy grid. In some cases, you might need to log at Fine level to diagnose certain issues. For example, you must set the log level to at least Fine to log JMX statistics.

Finer or Finest levels must not be used unless you are debugging a detailed issue and you are directed to do so by TIBCO support, as they might degrade performance and introduce unnecessary logging that can make it more difficult for diagnosing problems. If you must use Finest or Finer on the Manager, consider only increasing the component level.

The log format is: `{timestamp} {level}: [{component}] {message}`

Only messages that are at or above the current log level are logged.

Note It's considered unsafe to set all logging at a level below Info.

An example of a log message:

```
09/20/13 19:19:10.423 Info: [BrokerServicePlugin] Broker:Total:1
```

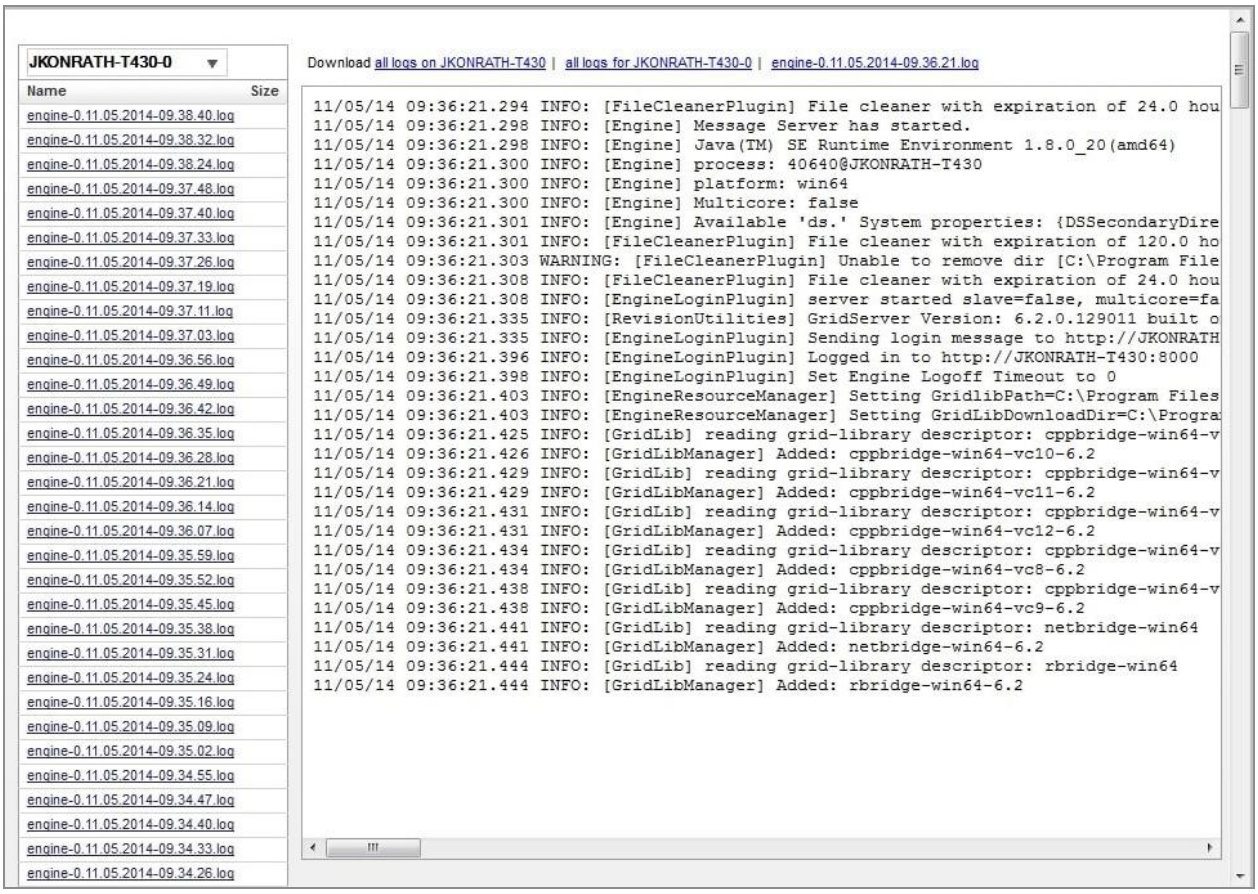
Viewing Engine Logs

There are several ways of viewing the logs. The most straightforward is to view the actual log files using the Log Files feature in the GridServer Administration Tool.

To view an Engine log:

1. In the Administration Tool, go to **Grid Components > Engines > Engine Admin**.
2. Select the Engine for which you want to view a log.

3. Click the **Actions** list, and select **Log Files**.
4. A window opens with a list of links for each of the logs residing on that Engine, listed by date and time. You can do any of the following:
   - 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 quickly filter the list to partial matches.
   - Click on 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 might also wish to view the logs in real time. You can do this with the remote log feature.

To view the remote log

Procedure

1. Go to **Grid Components > Engines > Engine Admin**.
2. Select the Engine for which you want to view a log.
3. Click the **Actions** list, click **Remote Log**.
4. A window opens, displaying the log on the Engine as events occur. You can click **Clear** to clear the log, or **Snapshot** to capture a screen of the log in a new window.

You can also run the Engine in console mode; typically, this is only done during development.

Windows

You can run the Engine from a command line with the command `engine.exe -console`. This starts the Engine in console mode and logging information scrolls in the command window in which you start it.

UNIX

You can run the Engine from a command line with the command `engine.sh startfg`. This starts the Engine in the foreground and the logging information scrolls in the terminal in which you start it. Note that this is only suitable for debugging purposes.

The Engine Log Search page enables you to search for all Severe-level Engine logs for a Service ID across all Engines, and optionally search those results for a keyword. Results are shown with a summary of each matching log for each Engine, with links to corresponding URLs to logs with excerpts. First, logs are searched for the given Service ID; then they are searched for the regular expression "`.*Severe.*`", then they are optionally searched for a given keyword.

To search Engine logs

Procedure

1. Go to **Diagnostics > Service Diagnostics**.
2. Enter a Service Session ID in the **Service Session ID** box, or click a name in the **Service Name** list. The list of Service Names is provided from the Service Admin list, which is the in-memory list of all Services recently run.

3. Enter a keyword in the **Keyword** box, or leave it blank to return all entries.
4. Click **Search**.

Results are shown with a summary of each matching log for each Engine, with links to corresponding URLs to logs.

Writing to Logs

Your Service also logs messages, and you can direct the messages to the DataSynapse logger.

Java

GridServer uses the Java logger, so any messages logged by using a Java Logger object are written to the Engine or Driver log file. For example:

```
Logger mylog = Logger.getLogger("com.mycompany.myproduct.MyClass");
```

The level of logs can be increased on a per-class and package level, to help in isolating issues. For example, you could do the following:

```
LogManager.getLogManager().getLogger
("com.livecluster.admin.servlet.AdminControllerServlet").setLevel
(Level.INFO);
```

Both Drivers and Engines capture `stdout` and `stderr`, so typically no changes are required to existing implementations to capture logs.

Additionally, the DataSynapse logger is registered as the Apache Commons Logging default handler. If your implementation uses this interface, your messages are logged automatically. The following is a map of levels to DataSynapse levels:

Commons	DataSynapse
fatal	Severe
error	Severe

Commons	DataSynapse
warn	Warning
info	Info
debug	Fine
trace	Finer

The Java Driver log manager can be completely disabled so that all Driver log messages are logged according to your configuration of the Java Logging Framework, rather than according to the Driver properties. For example, if the Driver is a part of a large client application that uses a number of Java libraries whose logs are all managed by the Java Logging Framework configuration, you would disable this so that the Driver logs are managed the same way. There is a Driver property, `DSLogUseJavaConfig`, that enables this behavior when set to `true`.

.NET

The .NET `System.Diagnostics.Trace` facility is used for logging; the DataSynapse logger is simply a Trace listener. The DataSynapse logger captures any messages written to the Trace facility. This includes .NET Services; any trace message written by the Service is logged to the Engine log.

C++

The `UtilFactory::log` function is the preferred method of logging to the DataSynapse log. You can use it on both the Engine and Driver.

Note

For logging to become effective, it is necessary to instantiate the Driver message server first — for instance, by creating a Service object, or by calling `DriverManager::connect`.

R

The included `rlogger` package contains a `log` method for logging. For example, the following logging is used in the Pi tutorial included in the SDK:

```
library("rdriver")
library("rlogger")
...
RDPI <- function(N, K) {
rlogger.log(list("RDPI(", N, ", ", K, ") = ", pi), rlogger.FINE)
```

PDriver

The PDS script language provides redirection of `stdout` and `stderr` to a file, via the `stdout` and `stderr` clauses in the execute statement. For example:

```
    execute
        stdout="$DSWORKDIR/pijob.$DSTASKID.out"
        stderr="$DSWORKDIR/error.$DSTASKID"
        ".\resources\win64\lib\PdriverPiCalc.exe $seed $iterations"
```

Writing to the Log directory

The Engine's log directory is `[work directory]/log`. You can view files written to this directory with the Log Files feature. You can write log messages to your own files, and view them with the Administration Tool.

The work directory is available as follows:

- Java: The system property `ds.WorkDir`
- .NET: The `System.AppDomain.CurrentDomain` data value `ds.WorkDir`
- C++, Command Service: The environment variable `ds_WorkDir`
- PDriver: The variable `$DSWORKDIR`

C++/.NET Native stdout/stderr

Aside from the logging methods described above, it is possible to write native `stdout` and `stderr` to files that are managed by the Engine. Enable this with the **Logging for Native Streams** settings in the Engine Configuration. The log files are created in the Engine instance log directory, typically *install-dir*`/work/`*machine name–instance*`/log/*`. The file names

are `engine-stdout-PID.log`, `engine-stderr-PID.log`. The files roll when they exceed the maximum size configured in the Engine Configuration.

This affects only native standard output / standard error for C++ and .NET. Java writes to `System.out/err` go to the regular invoke log file, as do writes from C++ or .NET that use the documented logging facilities.

Debugging Engines

This section covers the basics on how to attach a debugger to an Engine.

Java

The Java Platform Debugger Architecture (JPDA) enables the connection of a debugger to the Engine via a socket. You can set the socket used for debugging in the Engine Configuration. Create a new Engine Configuration or modify an existing one, and change the value of **Debug Start Port**. If you have a single Engine per Engine Daemon, set this value to the port you wish to use on each Engine. If you have more than one Engine per Engine Daemon, the value given in **Debug Start Port** is the port used for the first Engine instance, and the port is incremented for each additional Engine Instance. You can also set **Debug Suspend** to true, which keeps the Engine suspended until a debugger is attached.

.NET, Windows DLL

Microsoft Visual Studio comes with a remote debugging facility. To debug, you must first make sure that you build with debug symbols, and deploy the symbols (PDB) file with the DLL. Once the Engine has logged in, you attach the debugger to the `invoke.exe` process via the **Processes** dialog on the **Debug** menu.

CPPDriver and Linux

GDB can be used to debug native code in CPPDriver or JNI in Linux. Also, GDB can be useful in identifying unusual problems with the Linux JVM. However, there are some subtle issues when trying to use GDB on a JVM, as is the case with the GridServer Engine.

When attaching GDB to the Engine, you must specify the `LD_LIBRARY_PATH` to both the Engine components and the JVM components.

```
LD_LIBRARY_PATH=lib:jre/lib/i386:jre/lib/i386/native_
threads:jre/lib/i386/server:resources/lib/linux
```

You must also obtain the process ID of a running `invoke` (or `invokeGCC34`) process from the `ps` command. It's also easier if you run GDB from the base directory of the Engine install (typically `DSEngine`). The GDB command used is similar to this:

```
gdb bin/invoke $INVOKEPID
```

Replace `bin/invoke` with `bin/invokeGCC34` when using GCC34.

One difficulty with debugging C++ code is that your application shared objects are loaded only when the Service is instantiated, so it becomes difficult to set a breakpoint in the application shared object. (However, more recent versions of GDB feature deferred symbol resolution, which makes this possible.) A technique that works in this instance is to have your application Service method include some conditional code to enter a loop checking some variable value that is never changed by the application code, effectively creating an infinite loop. When you need to attach GDB, trigger the conditional that causes the loop to be entered on the next invocation. Then attach GDB as above. You'll see that the invoke process is stopped while running in the loop. At that point, you can change the loop evaluation value so that the infinite loop is exited, and the code continues to your breakpoint where you can continue debugging.

Creating a Native Stack Trace in Linux

Sometimes when you are troubleshooting native C/C++ code on Linux, you want to generate a stack trace, for example when a SIGSEGV is thrown. Since the JVM on the Engine already traps SIGSEGV and prints out a Java (not native) stack trace, you must override the actions of the JVM and install your own SIGSEGV handler for debugging. The `backtrace_fd()` and `backtrace_symbols_fd()` methods from glibc can be used for this purpose.

To install your own SIGSEGV handler for debugging, add code to your Service initialization method similar to this:

```
#include <execinfo.h>
#include <stdio.h>
#include <signal.h>
#define TRACE_DEPTH 50
void MyService::segv_handler(int signum) {
```

```
    void *trace[TRACE_DEPTH];
    int depth;
    FILE *fp;
    depth = backtrace(trace, TRACE_DEPTH);
    fp = fopen("trace.log", "w");
    backtrace_symbols_fd(trace, depth, fileno(fp));
    fclose(fp);
    abort();
}
void MyService::init() {
    signal(SIGSEGV, segv_handler);
    signal(SIGBUS, segv_handler);
}
```

Notes for Java Developers

Here are some notes for Java developers.

Stop Java Client by using a property

Unlike C++ and .NET clients, a Java client does not stop automatically. It remains pending after submitting Services and collecting results. Instead of calling `System.exit()` to stop the client, you can set a property to change this behavior. Add this line to your code:

```
DriverManager.setProperty(DriverManager.IS_DAEMON, Boolean.TRUE.toString
());
```

You can also enable this by setting `DSIsDaemon=true` in the `driver.properties` file.

When this property is set to true, all client threads are daemon threads, meaning that they allow the process to shut down when all threads have shut down.

HTTP Connection Parameters

The following parameters are specific for Java Drivers:

Parameter	Default value	Description
`DSHttpMaxConnectionTotal`	500	It controls how many maximum concurrent connections the Driver application can have at any given time. To change the default value, you can specify `-DDSHttpMaxConnectionTotal=800` as JAVA OPTION (as one of the System properties) or specify the following parameter: `DSHttpMaxConnectionTotal=`
`DSHttpMaxPerRoute`	80	It controls how many maximum concurrent connections per Route the Driver application can have at any given time. To change the default value, you can specify `-DDSHttpMaxPerRoute=100` as JAVA OPTION or specify the following parameter: `DSHttpMaxPerRoute=`
`DSHttpDDTKeepalive`	`True`	It controls whether to keep the HTTP connection alive for DDT requests. To change the default value, you can specify `-DDSHttpDDTKeepalive=false` as JAVA OPTION or specify the following parameter: `DSHttpDDTKeepalive=`
`DSHttpSubmissionKeepalive`	`True`	It controls whether to keep the HTTP connection alive for tasks submission requests. To change the default value, you can specify `-DDSHttpSubmissionKeepalive=false` as JAVA OPTION or specify the following parameter: `DSHttpSubmissionKeepalive=`

Parameter	Default value	Description
DSHttpImplementation	com.livecluster.util.http.ApacheHttpClientSupportFactory	It controls which HTTP communication implementation is applied. The default is Apache HttpClient. To change the default value to JRE HTTP Connection implementation, you can specify -Dds.DSHttpImplementation=com.livecluster.util.http.JavaHttpSupportFactory as JAVA OPTION (System property) or specify the following parameter: DSHttpImplementation=

Notes for C++ Developers

Changing the C++ Compiler Used with CPPDriver

The CPPDriver and Service bridge libraries are built for nearly all standard compilers used on Windows and Linux. You must link your client application and/or Service implementation with the appropriate libraries for the compiler.

You must also run any C++ Services against the proper C++ bridge libraries. This is done using Grid Libraries, in that any C++ Grid Library must include the proper bridge Grid Library as a dependency These libraries come already deployed in the DS_DATA/deploy/resources/gridlib directory.

Also, because different Linux releases support different compilers which use incompatible versions of the STL, the **GCC Version** property in the Engine Configuration dictates which compiler version of the bridge is supported by the Engine.

You can run Linux C++ Services built against unsupported STL implementations, using an Engine built with no STL conflicts. To use this:

1. Download the Engine installation, and locate the invokeGCC file.

2. Make a copy of `invokeGCC`, called `invokeGCC34`.
3. Replace this file in the appropriate `engineUpdate` subdirectory on your Managers.

C++ Multithreading Requirement

Note that you must compile all UNIX C++ code multithreaded. This includes both Service code and Engine or Driver code.

Using Global Statics in C++ Service Code

By default, the Linux C++ Driver loads application libraries with `RTLD_LOCAL`, which only makes statics available within an object. This is done primarily to ensure that statics defined in a Service Session is unique.

However, this can cause issues in your application, such as when using dynamic cast with RTTI. Instead, you can use `RTLD_GLOBAL`, which makes statics available by any object in a process. There is an Engine Configuration option to set `RTLD_GLOBAL` to true to handle such cases.

Running both 32-bit and 64-bit Services on 64-bit Windows Daemons

The 64-bit Windows Engine Daemon can be configured to allow the execution of 32-bit Services. For more information, see "Configuring 64-bit Engine Daemons to run 32-bit Services" in the *TIBCO GridServer® Administration*.

Running Examples on Visual Studio

To run the Calculator client in windows using any Visual Studio version other than Visual Studio 2013:

1. Open the CPPCalculator solution file in Visual Studio.
2. Set the configuration to "Release" and Platform as "x64".

3. Right-click "Calculator", select **Properties > Configuration Properties > General > Platform Toolset**, and select your version of Visual Studio. Do the same for "CalculatorClient".
4. Right-click "Calculator", select **Properties > Linker > General**, and replace "Output File" with the value for your version of Visual Studio. (For example `..\..\target\o\win64\vc12\Calculator.dll`)
5. Right-click "CalculatorClient" select **Properties > Linker > General**, and replace "Additional Library Directories" with the value for your version of Visual Studio (For example `..\..\..\..\..\cppdriver\lib\vc12\x64;%(AdditionalLibraryDirectories)`)
6. Right-click "CalculatorClient", select **Properties > Linker > Input**, and replace "Additional Library Directories" with the value for your version of Visual Studio (For example `DSDriverVC12.lib;DSUtilVC12.lib;odbc32.lib;odbccp32.lib;%(AdditionalDependencies)`)
7. Modify `env.bat` for your version of Visual Studio. For example:

   ```
   set dsos=win64
   set dscompiler=vc12
   set dsbridgename=win64-vc12
   set devenvpath="%VS120COMNTOOLS%\..\IDE\devenv.exe"
   ```

8. Edit `driver.properties` as needed.
9. Run the `build-and-deploy.bat` script. The Grid Library is deployed with the correct `calculator.dll` file packaged in it.

Other C++ Notes

Microsoft Visual Studio 2013, 2015, 2017, and 2019 redistributables are installed when the Windows Engine is installed with the installer program. Engine updates from the Manager to pre-existing Engines do not install the runtimes.

When using existing Grid Libraries that use JNI on Windows systems, you must enable the `Restart on lib-path change` property in the Engine Configuration.

Notes for .NET Developers

The `GridServerNetClient.dll` references `GridServerNetBridge`, which isn't needed for clients. This causes a build warning about `GridServerNetBridge`, but can safely be ignored.

To throw a custom exception from a .NET Service and retain all data when the exception reaches the .NET client, the exception must implement `ISerializable`. For more information, consult MSDN.

Parts of the `BatchInfo`, `ServiceInfo`, `Options`, and `Discriminator` classes use variable / constant names that are not CLS compliant, and might not be usable from non-C# Framework languages. If this is an issue, contact DataSynapse technical support for a possible workaround.

.NET cache loaders for GridCache must be packaged as a Super Grid Library. For more information about using Super Grid Libraries, see the *TIBCO GridServer® Administration*.

For .NET5 users

For .NET5 users, .NET5 version 5.0.404 for Windows and Linux is supported. The example of .NET5 is under `examples\service\net5` directory and the client assemblies are in the `NET5Driver` directory. For the NET5 service, you must configure Service Type as NET5.

If you are running .NET5 service on Linux, you must change the `GCC build version` to `gcc49` in the Engine Configuration.

If your system does not support globalization and you get the following error:

`Couldn't find a valid ICU package installed on the system`,

add the following environment variable before running NET5: `DOTNET_SYSTEM_GLOBALIZATION_INVARIANT=true`

.NET Driver Upgrades

The .NET Driver (`GridServerNETClient.dll`) is strongly named. This means that when a new major version of the .NET Driver is released in an update, steps might need to be taken for existing clients to allow the assembly to be loaded. For example, that version changed from 5.1.x to 6.0; however, it has remained the same for all 6.x versions.

There are two ways of doing this:

- Rebuild the .NET application with the new `GridServerNETClient.dll`.

or

- Configure the application to allow the new version.

You can do this in various ways, depending on your .NET policy; that is, whether the assembly is deployed into the GAC or used locally.

An example of how to do this when the `GridServerNETClient.dll` is used locally is as follows:

Method 1: Using the Microsoft .NET Framework Configuration tool:

1. Select **Start > Control Panel > Administrative Tools > Microsoft .NET Framework *x.y* Configuration**, where *x.y* is the version of .NET.

   Note that the tool is no longer included with Windows or in the .NET runtime; you must install the .NET Framework SDK to obtain the file.

2. Select **Applications > Add an Application To Configure**.
3. If your application is in the list, click it; otherwise, find it using the **Other...** button.
4. Your application is now in the Applications list. Expand your application, and select **Assembly Dependencies**.
5. Drag the **GridServerNETClient**, noting the version number, to the **Configured Assemblies** icon.
6. Click the **Configured Assemblies** icon. Double-click **GridServerNETClient**, and select **Binding Policy**.
7. Under **Requested Version**, enter the version you noted in step 5. This is the version with which you built your application. Under **New Version**, enter the new version of the `GridServerNETClient.dll` that you just installed. This enables your application to bind with the new version even though you built it with a previous version.

Method 2: Directly creating the file

In Method 1, the .NET tool creates an Application Configuration file in the directory of the application. However, you can create this file yourself.

Procedure

1. Create a file next to your application executable called `my.exe.config`, where `my.exe` is the name of your executable.
2. Add the following as the file's content:

```
<?xml version="1.0"?>
<configuration>
  <runtime>
    <assemblyBinding xmlns="urn:schemas-microsoft-com:asm.v1">
      <dependentAssembly>
        <assemblyIdentity name="GridServerNETClient"
                          publicKeyToken="42129437978483df" />
        <bindingRedirect oldVersion="5.0.0.1-5.1.2.30"
                         newVersion="6.0.0.1" />
      </dependentAssembly>
    </assemblyBinding>
  </runtime>
</configuration>
```

Note that the `oldVersion` is the version you built your application with, and the `newVersion` is the version of the new assembly. In this example, `oldVersion` is a range of versions. If your applications already have a configuration file, edit the configuration file appropriately.

If you have a .NET Service implementation that links to the `GridServerNETClient.dll`, you do not need to perform either of these steps. An `invoke.exe.config` file is included in any .NET upgrade that manages this for you. However, you can rebuild your implementation if you wish.

Notes for Python Developers

The Python client wraps the 64-bit VC14 version of the C++ Driver on Windows, so the 64-bit C++ Driver and Python (3.7.0) need to be in the path and the VC14 Runtime needs to be installed.

Driver Installation

This section describes how to install the GridServer SDK and Drivers and configure the Drivers.

The *Driver* is the component that maintains a connection between the GridServer Manager and the client application. The GridServer SDK, available for Windows and Linux, provides the Drivers.

GridServer SDK Installation

This section describes the GridServer SDK installation. To understand your complete installation procedure, read the SDK instructions and the instructions for each Driver you plan to install.

To install the GridServer SDK:

1. In the GridServer Administration Tool, go to the top navigation bar, and click the **Downloads** icon.
2. Click the SDK for your platform to download it.
3. Unzip or unarchive the SDK.
4. Read the following sections for directions on installing Drivers.

The Java Driver (JDriver)

The Java Driver, also known as JDriver, consists of a JAR file used with your Java application code.

To use JDriver:

1. Ensure that you installed the Java SE SDK (also commonly referred to as the JDK). You can download it from Oracle.
2. Define an environment variable `JAVA_HOME` that contains the location of the JDK.

Each of the Java examples in the `examples` directory of the GridServer SDK includes `env`, `build`, and `run` scripts. The examples demonstrate how to properly set classpaths and environment variables to run a Java application using JDriver.

To use the Driver, add the `DSDriver.jar` file and the `config` directory of the GridServer SDK to your classpath when running your application. Additionally, a `DSDriverNoDeps.jar` is included, which contains only DataSynapse classes and no third-party dependencies, allowing you to upgrade these dependencies if and when necessary.

The C++ Driver (CPPDriver)

The C++ Driver, also known as CPPDriver, consists of libraries and include files that are linked with your GridServer application.

To use the C++ Driver:

1. Set the environment variable `DSDRIVER_DIR` to the path of the `config` directory in the SDK.
2. For Windows, add the applicable directory for your compiler to the `PATH` environment variable.
3. For UNIX, set the `LD_LIBRARY_PATH` environment variable to include the applicable directory in the `lib` directory.

The Parametric Job Driver (PDriver)

PDriver, or the Parametric Job Driver, is a Driver that can execute command-line programs as a parallel processing job using the GridServer environment. This enables you to take a single program, run it on several Engines, and return the results to a central location, without writing any new code.

The Python Driver (PyDriver)

To use PyDriver:

1. Add the 64-bit C++ Driver directory to the `PATH` environment variable.
2. Make certain the Python 3.7.0 directory is in the `PATH` environment variable.

3. For Windows, make certain the VC14 runtime is installed.

Windows Installation

To install PDriver on Windows systems, run the `PDriverInstaller.msi` installer in the `pdriver` directory of the SDK download. By default, this installs PDriver to the `c:\TIBCO\datasynapse\PDriver` directory.

This also installs the VC runtime if it is needed, and associates `.PDS` files with PDriver.

The installer sets the `DS_PDRIVER_CONF` environment variable, which specifies the location of the `driver.properties` file, to the installation directory. Verify that the `driver.properties` file contains the correct Manager and authentication settings so that the Driver points to your local Manager.

UNIX Installation

To install PDriver on UNIX systems

Procedure

1. Copy the files in the SDK directory to your machine and set your path to include `pdriver/bin`.
2. Set the environment variable `DSDRIVER_DIR` to the path of the `config` directory in the SDK.
3. Set the `LD_LIBRARY_PATH` environment variable to include the `cppdriver/lib` and `pdriver/lib` directories.

Note:
Note that only Pdriver `linux64-gcc34` is supported from GridServer version 7.1.0 onwards.

Using PDriver

To run a PDriver script file, invoke the PDriver binary and pass the PDriver script file as the first argument. For example, on UNIX systems:

```
pdriver examples/example.pds
```

Running `pdriver` or `PDriver.exe` without a PDS argument provides a usage message.

The `examples` directory contains several example PDriver script files.

The .NET Driver

The .NET Driver consists of an assembly that includes classes for creating and managing Services from .NET. The .NET Driver is available only for Windows.

Perform the following steps to use the .NET Driver:

1. Find the `GridServerNETClient.dll` in the `NETDriver` directory and link this assembly to your application.
2. Set the environment variable `DSDRIVER_DIR` to the path of the `config` directory in the SDK.

.NET5 is also supported and is available for Windows and Linux. To use .NET5:

- Locate the `GridServerNET5Client.dll` file that is bundled in the SDK's `NET5Driver` directory and link this assembly to your application.

The COM Driver

The COM Driver enables an application using the Component Object Model architecture to access GridServer, enabling distributed parallel execution of the application on a Grid of Engines. The Driver includes an example, an Option Evaluation spreadsheet in Excel that uses GridServer for its calculations. The COM Driver is available only for Windows.

Perform the following steps to install the COM Driver:

1. In the `COMDriver` directory of the SDK, double-click the `setup.exe` to start the installer.
2. After the installer welcome screen appears, click **Next**.
3. Click **Browse** to select a location for the COM Driver. The default location is `C:\TIBCO\DataSynapse\DSCOMDriver`. Click **Next**.
4. Enter the address of the primary and secondary Directors, in the form `hostname:port`. For example, `http://server1.example.com:8000`. You can proceed without filling in any values, but you must later specify your Directors by editing the `driver.properties` file. Click **Next**.
5. Click **Finish**.

The R Driver

The R Driver enables you to write Services in R, and access them from R or from applications in other languages such as Java, C++, or .NET. Applications written in R can also access Services deployed in GridServer.

Prerequisites

- Install and start the TIBCO GridServer® Manager.
- Install and start an Engine (Windows or UNIX).
- Download and extract the TIBCO GridServer® SDK for Windows or UNIX.
- Set the environment variable `DSDRIVER_DIR` to the path of the config directory in the SDK.
  - For Windows:

    ```
    set DSDRIVER_DIR=GRIDSERVERSDK-Windows\config
    ```

  - For UNIX:

    ```
    export DSDRIVER_DIR=GridServerSDK-Linux/config/
    ```

- Install TERR 6.0.
- Set `TERR_HOME` and `R_HOME` to the `TERR` install folder.
  - For Windows:

    ```
    set TERR_HOME=C:\TIBCO\terr60
    set R_HOME=C:\TIBCO\terr60
    ```

  - For UNIX:

    ```
    export TERR_HOME=/opt/qa/TIBCO/terr60
    export R_HOME=/opt/qa/TIBCO/terr60
    ```

- Install Ant 1.9 or later and set `ANT_HOME` to the Ant install folder.
- Add `%TERR_HOME%\bin` and `%ANT_HOME%\bin` to the system path.
  - For Windows:

    ```
    set PATH=%PATH%;%TERR_HOME%\bin
    set PATH=%PATH%;%ANT_HOME%\bin
    ```

— For UNIX:

```
export PATH=$TERR_HOME/bin:$PATH
export PATH=$ANT_HOME/bin:$PATH
```

- For Windows, add the directory `cppdriver/bin/vc15` to the `PATH` environment variable:

```
set PATH=%PATH%;GridServerSDK-Windows\cppdriver\bin\vc15\x64
```

- For UNIX, set the `LD_LIBRARY_PATH` environment variable to include the applicable directory in the lib directory.

  For example, to use GCC3.4 libraries, set the variable to `cppdriver/lib/gcc34`:

```
export LD_LIBRARY_PATH=GridServerSDK-
Linux/cppdriver/lib/linux64/gcc34/
```

Installation

1. Run the following command to build and upload Grid libraries:

   — For Windows, go to `GridServerSDK-Windows/tools/bin` and run `createTERRGL.bat`:

```
createTERRGL.bat --terrHome=%TERR HOME%--64 --complete
```

   — For UNIX, go to `GridServerSDK-Linux/tools/bin` and run `./createTERRGL.sh`:

```
./createTERRGL.sh --terrHome=$TERR_HOME --64 --complete
```

2. Install `rdriver` and `rlogger`:
   - For Windows:
     ```
     TERR.exe CMD INSTALL ".\GridServerSDK-Windows\rdriver\rdriver"
     TERR.exe CMD INSTALL ".\GridServerSDK-Windows\rdriver\rlogger"
     ```
   - For UNIX:
     ```
     $TERR_HOME/bin/TERR CMD INSTALL $SDK/rdriver/rdriver
     $TERR_HOME/bin/TERR CMD INSTALL $SDK/rdriver/rlogger
     ```
3. Change directories to
   - For Windows:
     ```
     GridServerSDK-Windows/examples/service/r/pscl
     ```
   - For UNIX:
     ```
     GridServerSDK-Linux/examples/service/r/pscl
     ```
4. Modify the `build.xml` to change `/bin/Rscript` to `/bin/TERRscript`.
5. Edit the `build.properties` in the `pscl` folder. Set the `dsos` property to `Win64`, save the file. Run `ant`. This builds the Win64 version of the `Rextras` and `Rpscl` Grid Libraries.
6. Repeat the step 5 and set the `dsos` value to `linux64`. This builds the Linux64 version of the `Rextras` and `Rpscl` Grid Libraries.
7. Upload the four Grid Libraries, created earlier, (two `Rextras` and two `Rpscl`) to the Manager and deploy them. Deploy the Rextras first, then Rpscl.
8. Change directories to
   - For Windows:
     ```
     GridServerSDK-Windows/examples/service/r/picalculator
     ```
   - For UNIX:
     ```
     GridServerSDK-Linux/examples/service/r/picalculator
     ```
9. Run `ant` to build and deploy the `Rcalculator` Grid Library.
10. Run `runTERR.bat`. (This is only for Windows.)
11. Edit Service Type registry `RPICalculatorExample` on the UI and select `functionInterface` as `UNWRAP_RETURN`.
12. For Windows, run `runJava.bat` and for UNIX, run `runJava.sh`.

Driver Configuration

Driver configurations reside in the `driver.properties` file, which is included in the GridServer SDK. When you install the Manager, the `driver.properties` file contains hostnames for the primary and secondary Director. To change the default values, edit the `driver.properties` file. For example, edit this file to add a username and password for authentication, change the primary and secondary Director, change the Broker Timeout, and so on.

You can move the `driver.properties` file to another directory. For Java, add the new directory to your classpath. For C++, .NET, R, and the UNIX PDriver, the `DSDRIVER_DIR` environment variable must be set to the location of this directory. For Windows PDriver, the `DS_PDRIVER_CONF` environment variable must be set to the location of this directory.

If you use the following characters in properties in the `driver.properties` file, precede each with a backslash: `#`, `!`, `# =`, `\`, and `:`

Backslash characters in hostname or directory properties receive special handling on Windows Drivers. The first backslash, indicating the root directory, translates as the current Windows drive. Other backslashes are ignored. Forward slashes translate as backslashes.

For example, to set a directory to `c:\sdk\log`, use `/sdk/log`in the `driver.properties` file. To use a UNC path such as `\\homer\job1-dir`, use `//homer/job1-dir` in the `driver.properties` file.

You can also set Driver properties programmatically with the API. When you do this, the `driver.properties` file is unnecessary. You must at least set `DSPrimaryDirector`, `DSSecondaryDirector` (set them both to the same hostname when using a single Director), `DSUsername`, and `DSPassword` (unless using Windows or Kerberos authentication).

If your username or password is non-ASCII, you must express it in Unicode converted format rather than as the Unicode characters. For example:

```
DSUsername=\u6B21\u306E\u30E6\u30FC
```

Configuring Multi-Interfaced Drivers

In some network configurations, on a PC with more than one network interface, a Driver might default to using the incorrect interface, resulting in the Engines using the incorrect IP address for the Driver's file server. To configure the Driver to use a different network

interface, set the `DSLocalIPAddress` property to the IP number of the correct interface. For example:

```
DSLocalIPAddress=192.168.12.1
```

Driver Cleaner Configuration

When Direct Data Transfer (DDT) is enabled and data is transferred directly to the Engines from this Driver, DDT files are persisted for a configurable amount of time, and then deleted by a file cleaner process. The file cleaner deletes files and directories when they are no longer needed, such as files left over by a Service that did not complete normally. Files used by a currently executing Service are not deleted regardless of the files' age or the Driver's configuration.

There are four properties in the `driver.properties` that control the cleaner:

- `DSDataTransferFileExpirationHours`: The amount of time DDT files persist before being deleted. By default, this is set to 120 hours.
- `DSDataTransferFileExpirationFrequency`: The number of times per day that the Driver checks for and cleans expired DDT files. By default, this is set to 6 times. To disable the file cleaner, set this to 0.
- `DSDataReferenceFileExpirationHours`: The amount of time Data Reference files persist before being deleted. By default, this is set to 120 hours.
- `DSDataReferenceFileExpirationFrequency`: The number of times per day that the Driver checks for and cleans expired Data Reference files. By default, this is set to 24 times. To disable the file cleaner, set this to 0.

Multiple Driver Instances

Services can fail if you enable Direct Data Transfer and write a script that instantiates multiple Drivers from the same `driver.properties` file with the same port number. The first Driver opens a web server listening to the defined socket. Subsequent Drivers do not open another web server as long as the first Service is running, but can continue running by using the first Service's server for direct data. However, when the first Service completes, its server ends, causing subsequent Services to fail.

You can avoid this problem by writing a shell script to create Services, each with its own Driver running from its own Java VM. Your script must provide a different port number for the `DSWebserverPort` property normally set in the `driver.properties` file. To write a shell script for this situation, you can remove the `DSWebserverPort` property from the `driver.properties` file and assign a unique port number for each iteration.

More than one Driver can share the same directory. However, if you do so you must set a unique `DSCacheRootDirectory` for each Driver if using GridCache. Also, if using long-running Services, you must set a unique `DSWebServerDir` property so that one Driver's file cleaner does not clean another session's initialization data.

Creating Services

This section provides information about how to create Services.

Overview

Services enable remote, parallel execution of code in a way that is scalable, fault-tolerant, dynamic, and language-independent. You can write Services in a variety of languages and do not need to compile or link them with DataSynapse libraries. There are client-side APIs in Java, C++, COM, R, Python, and .NET. A client written in one language can invoke a Service written in another.

The Service execution model is the same as that of other distributed programming solutions: method calls on the client are routed over the network, resulting in method calls on a remote machine, and return values make the reverse trip. We prefer the term *request* to call or invocation, partly because the operation can be either synchronous or asynchronous.

Use Services to implement parallel processing solutions in which a single computation is split into multiple, independent pieces whose results are combined. To split the computation, divide the problem, submit the individual requests asynchronously, then combine the results as they arrive. Services also work well for executing multiple, unrelated serial computations in parallel.

Steps in Using a Service

Using a Service involves the following steps:

1. **Write the Service, or adapt existing implementations.** A Service can be virtually any type of implementation: a library (DLL or .so), a .NET assembly, a Java class, an R function, a command, script or executable, or even an Excel spreadsheet. You do not need to link a Service with any DataSynapse libraries, but make sure that the remotely callable methods of the Service follow conventions that enable cross-language execution and support stateful Services. For details, see Client Calling Conventions.

For examples of code using Services, see the GridServer SDK and the *GridServer Service-Oriented Integration Tutorial*.

2. **Deploy the Service.** Make the implementation and other resources required for the Service accessible from all Engines. Do this through GridServer's mechanism of resource deployment.
3. **Register the Service Type.** Make the Service visible to clients by registering it as a Service Type in the GridServer Administration Tool.
4. **Create a Service Session from a Client.** Develop a Client Application that accesses the registered Service Type and creates a Service Session. Each Service Session has its own state that is client-specific.
5. **Make requests.** The Client Application calls the methods of a Service Implementation either synchronously or asynchronously.
6. **Destroying the Service Session.** Client Applications destroy Service Sessions when they are done with them.

This section describes how to develop a Service Implementation that runs on an Engine. Accessing Services describes how to use this Service Implementation from a Client Application.

Service Method Compliance

Although Service methods do not link to DataSynapse libraries, they must comply with the rules in the following sections.

Java/.NET Services

Java/.NET Services must comply with these rules:

- Make the Service class public, and make all methods called by the client public.
- A method can take any number of arguments and can have a return type of void. The return values of state methods are ignored, and a Service method with a void return type returns a null.
- If you plan to use the Service cross-language, the arguments and return values must conform to the rules of interoperability, as described in the section Interoperable Types for XML Serialization.

- If only a client of the same language uses the Service, you can use any serializable object for arguments and return values.
- Overloaded methods are prohibited.
- Methods can throw exceptions within a Service, which captures and includes stack trace data and nested exception data when available.

C++ Services

C++ Services must comply with these rules:

- Export all service methods in the shared library.
- The Service method must either take a `char*`, or a `char**` for multiple arguments. Alternatively, if using the macro it can take a `std::string` or a vector of `std::string`s.
- The method returns data through a `char**` argument, which is set to the returned data. Alternatively, if using the macro it returns a `std::string`.
- Overloaded methods cannot be used.

Command Services

Command Services must comply with these rules:

- The name of the method that is called is appended to the command line.
- Argument values are sent to stdin or an input file. If there is more than one argument, the data is separated by the `argDelimiter`, which can be registered on the Service.
- You can append argument values to the command line instead, if the option is selected. In this case, the client passes the argument values in as strings.
- If your command spawns subprocesses, it must cancel them if the main command is canceled.

R Services

R Services must comply with these rules:

- All functions visible in the R runtime environment can be used as a Service.
- These include all functions defined in any `.R` script files defined in the `R/` directory in the Grid Library and any function exported by third-party libraries.
- Functions exported by libraries can only be accessed when using `functionInterface` as `NATIVE_R`. The `functionInterface` setting is described in R Interoperability.

Python Services

Your Python libraries not only need to be added to the `grid-library.xml` for the Python bridge but also the PIP install needs to be done in your code. Your Python libraries need to use the Python Service Type to ensure that the proper Python script is executed.

You must carefully handle type conversion since the Python driver wraps the C++ Driver and the data is being implicitly converted to a string before being transmitted, and then it is implicitly converted from the string in the Grid Library unless explicit conversion is used.

Client Calling Conventions

Clients must comply with the following rules when calling methods in a Service.

Java/.NET Client

Java/.NET Clients must comply with the following rules when calling methods in a Service:

- Pass arguments into calls as `Object[]`, which corresponds to the arguments of the method. Note that you must use exactly the type `Object[]`. For instance, a set of strings cannot be passed in as a `String[]`. The array length must match the number of arguments.

- For convenience, if the method takes only a single argument, you pass it directly into the call. It is the equivalent of passing in an `Object[1]` with the 0th element being the object.
- If a method takes no arguments, you can call it only with zero-length `Object[]` or a null object.
- Autoboxing is used for primitive types. For example, a Service method that returns a `Double` on the client.

C++ Client

C++ Clients must comply with the following rules when calling methods in a Service:

- Pass arguments into calls as `char**`. Alternatively, if using the macro found in `DynamicLibraryFunctions.h`, it can be a vector of `std::string`.
- If a method takes no arguments, you can call it only with a `NULL` or zero-length `char*` or `string`.

R Client

See R Interoperability for more information.

Python Client

When setting parameters with the Python client, you must initialize the input parameters using `driver.init_input_parameter(`*n*`)` where *n* is the number of parameters to initialize. Each parameter is then set with a call to `driver.set_input_parameter(`*parameter name or literal*`)`. Here is an example from `python_client_example.py`:

```
driver.create_services(b'PythonExample')
driver.init_input_parameter(2)
driver.set_input_parameter(b"8.23")
driver.set_input_parameter(b"1.25")
driver.submit(b"add")
```

In the above example, the literal values are expressed as `'b"8.23"'` and `'b"1.25"'`. These are expressions specific to Python. Refer to Python documentation if you have questions on them.

Registering a Service Type

Service Types are registered in the GridServer Administration Tool. On the primary Director, go to **Services > Services > Service Types**. Service Types registered on the Director are then replicated to Brokers. A list of existing Service Types appears on that page, along with a line for adding a new Service Type. Enter the Service Type name on the blank line. Select a Service Implementation, then click **Add**.

In the window that appears after clicking the **Add** button, enter any name, property or option values for the Service Type.

Container Binding

When you register a Service Type, an associated Container Binding is created. The Container Binding binds the Service implementation (the library or command) to the Container of the Service (the Engine). The Container Binding describes how to use the implementation.

The binding contains the following fields:

Container Binding Fields

Field	Description
`initMethod`	Called when a Service Session is first used on an Engine. It is called prior to any requests or updates.
`destroyMethod`	Called when a Service Session is destroyed.
`cancelMethod`	Called when the Service is canceled if `KILL_CANCELLED_TASKS` is false for this Service. Use this field to interrupt the request if the user does not want the Engine to restart on a cancel.

Field	Description
`serviceMethods`	Methods that perform a request and return a response. These are the actual methods that perform the calculations. Use the * character to denote all methods that are not bound to any other action.
`appendStateMethods`	Updates state, appending to previous updates.
`setStateMethods`	Updates state, and flushes the list of previous updates.

You must bind all methods used to one of these methods. All methods are optional except `serviceMethods`.

The binding also contains the following fields, only applicable to Java, .NET, and R:

Additional Container Binding Fields For Java, .NET, and R

Field	Description
`xmlSerialization`	Whether to use XML serialization to serialize objects. See Interoperable Types for XML Serialization for more details.
`TargetPackage`	The package (Java) or namespace (.NET) into which the generated proxy classes are placed. If not set, the default is the name of the Service. This is only necessary when using generated proxies. In this case, if not set, serialization errors might occur. Note: When you do not set the `targetPackage` name in the **Service Types** page and use a generated proxy, deserialization errors occur. To remedy this, edit the **Service Type** on the **Service Types** page of the GridServer Administration Tool and assign a value for the `targetPackage` property.

The binding for R contains the `functionInterface` field, which is described in R Interoperability.

Python Services need to use the `Python` Service type. The only other critical step is making certain to set the `pythonModule` which is the name of the Python file that contains the implementation, without the file extension.

.NET AppDomains

Services implemented in .NET have full access to .NET's AppDomain functionality. You can manage multiple persistent AppDomains across Service invocations while still having access to the entire DataSynapse Engine-side API. You can specify an AppDomain as part of a Grid Library deployment, and the Engine sets up and manages it automatically.

The **Provider** section in the Service Type Registry for .NET Services supports an `appDomainName` value. Set the `appDomainName` value to specify a unique AppDomain for Services created from this Service Type. If you do not specify an AppDomain for a Service, the Service receives a new anonymous AppDomain with a randomized name. The Service does not run in the Default Domain as it did in previous versions of GridServer. If you create a Service that uses legacy Default Resources rather than a Grid Library, the Service runs in an auxiliary AppDomain. All resources-backed Services use this same auxiliary AppDomain for the lifetime of the Engine.

When you specify an AppDomain as part of a Service Type definition and an Engine creates the Service for the first time, the Assembly search paths used for the AppDomain depend on how you deploy resources:

- When a Service uses a Grid Library, the `assembly-path` path elements for that Grid Library (and any dependent Grid Libraries) are the Assembly search paths for the AppDomain.
- When a Service does not use a Grid Library, the default Assembly search path is used.

The Engine searches the Assembly search path for a valid AppDomain Configuration File, which has the same name as the AppDomain, plus the `.config` suffix.

An Engine restarts if it needs to load a Grid Library to run a Service in an AppDomain that has an Assembly search path that conflicts with the new Grid Library's search path. This is different behavior from previous versions of GridServer.

Use the `unloadAppDomain` property in the Service Type to specify what to do with Grid Library-backed AppDomains after the Service Sessions using them are destroyed. Select `true` to unload AppDomains no longer in use.

.NET Framework Versions

In the **Provider** section of the Service Type Registry is a `frameworkVersion` value. Set this value to the .NET Framework version required by Service. This prevents Services that

specify a Framework version from executing on Engines without that framework version installed.

To determine which Engines have a version of the Framework installed, go to **Grid Components > Engines > Engine Admin**, select the **Engine Details** action, and look for the `NETFrameWorkVersion` and `NETFramework` Engine Properties. The `NETFramework` property is `false` if a Framework is not installed or if it is an unsupported version (such as 1.0).

.NET5 Core

For .NET5 users, select NET5 from the **Service Type** drop-down list.

Language Interoperability

Services provide various levels of interoperability among languages. To provide this interoperability, GridServer can perform conversions on arguments sent to objects. The following sections describe the conversion of arguments between Service Implementations.

Strings and Byte Arrays

All Services can use byte arrays (`byte[]`s) interchangeably with `String`s as arguments. Conversions use UTF-8 encoding. For example, if an argument is of type `String`, and the client passes in a `byte[]`, the `byte[]` is UTF-8 encoded and passed into the method as a `String`.

Because a C++ Service always returns a `string/char*`, you must convert the returned type of an invocation to a `String` or `byte[]`. The first argument to the invocation determines the type of conversion:

- A string first argument returns a string.
- An argument of `byte[]` returns a `byte[]`.
- If you pass no arguments to the invocation, it returns a `byte[]`.

This is most relevant for a .NET, as string I/O must be ASCII. If you are returning binary data, make sure that the first argument is a `byte[]`.

A Command Service converts the output data to a `String` only if the first argument is a `String` and the `appendArgsToCommandline` option is false.

Java and .NET Services do not convert return values if they are `String`s or `byte[]`s.

In the case where a `parameterType` value is specified, GridServer converts to the specified value. For example, if `parameterType` is set as `string`, GridServer returns strings. If set to `byte[]`, GridServer converts the return value to `byte[]`. This applies to both the C++ Service and the Command Service. You can specify the `parameterType` value for DynamicLibrary and Command Services on the Service Type Registry page.

Object Conversion from Strings and Byte Arrays

Java and .NET Services automatically attempt to convert `String/byte[]`s to and from `Object`s when necessary. This is useful when calling these Services from a different language, or when using Service Runners from Batches.

If an argument is not a `String` or `byte[]`, and it is passed in as such, an attempt is made to convert it. If the data is a `byte[]`, it is first converted to a `String`. Then the `String` is converted to the `Object` as follows:

String Argument to Object Conversion

Input Argument	String Argument-to-Object Conversion
Primitives	The primitive wrapper class's parse method
`Date`, `Calendar` (Java)	`DateFormat.getDateTimeInstance().parse`
`DateTime` (.NET)	`DateTime.Parse`
`org.w3c.dom.Document` (Java)	Uses the parse method from the DocumentBuilder given by `DocumentBuilderFactory.newInstance().newDocumentBuilder()`
`XmlDocument` (.NET)	`XmlDocument.loadXml`
Other	If the class has a constructor that takes a single `String` as an argument, it uses that constructor.

If the return value is not a `String/byte[]`, and the client is not of the same language as the Service, the returned value is converted to a `String`, as follows:

Returned Object to String Conversion

Return Type	Returned Object-to-String Conversion
Primitives	The object-equivalent `toString()` method
`Date`, `Calendar` (Java)	`DateFormat.getDateTimeInstance().format`
`DateTime` (.NET)	`date.ToUniversalTime().ToString("r", DateTimeFormatInfo.InvariantInfo)`
`org.w3c.dom.Document` (Java)	The transform method from the Transformer given by `TransformerFactory.newInstance().newTransformer()`
`XmlDocument` (.NET)	`doc.WriteTo(XmlTextWriter)`
Other	The `toString` method

XML Serialization for Java, .NET, and R

XML serialization provides the following features:

- Java, .NET, and R can use rich objects as arguments and return values with each other.
- A client can use a Service with such objects, without needing the original implementation classes. This is because client-side proxy classes are generated.

To use XML serialization, enable it on the Service Type. Note that when you enable this, the other interoperability conversions are no longer used. Additionally, the client must use the proxy that is generated using the Service Type Registry, which contains all user-defined types.

You must use Interoperable Types for the arguments and return values on such Services, as discussed in the following section.

Interoperable Types for XML Serialization

When you use XML Serialization, you must use interoperable, or interop, parameters and return types, as follows:

Interoperable Types

Type	Description
Primitives	`bool`, `byte( Java)`, `sbyte (.NET)`, `byte[]`, `double`, `float`, `short`, `int`, `long`, `string`.
`Calendar` (Java) or `DateTime` (.NET)	Interoperability between Calendar and DateTime in a mixed Java/.NET setup does not support time zones. .NET's DateTime structure does not support the concept of timezone so that information is not maintained. You must send that information separately. The timezone itself is not preserved across serialization so all Calendar objects are created with local timezones. .NET has an additional limitation in that it assumes that all date time data is in its local timezone. Therefore, serialization of XML not in that local timezone produces an incorrect time. If you cannot determine the .NET deserializer's local timezone from Java to set in the Calendar, pass the Calendar data as a String instead.
Arrays	The type can be an array of any interop type.
User-Defined Types	User-defined types can be used, as long as they follow the standard “bean” pattern. For both languages, use interoperable types (including other user-defined types) for all data. For Java, all data must be Java Bean properties; that is, each must have public get/set methods. For .NET, all data must be public fields. When generating a proxy for this Service type, user-defined types result in generated classes. You can have other data and methods in the type, such as private non-interop fields, but they are ignored and not reflected in the generated class. Also, user-defined types must be concrete; abstract classes and interfaces are not allowed.
Data References (Java and .NET)	This type can be used as an argument, return type, or GridCache object, and is interoperable whether XML Serialization is on or off. When generating a proxy for a Service type using Data References, it does not include all methods of the Data References. Normally, the data in a Data Reference is stored and served in the file server on the GridServer component that created it. C++ Data References are not interoperable with Java or .NET. For more information, see Data References.

The following is an example of a Java Interop type:

```
public class Valuation {
    private java.util.Calendar valuationDate;
    private double value;
    private MarketData data;
    private String[] names;
    public Valuation() {}
    public java.util.Calendar getValuationDate() {
        return valuationDate;
    }
    public void setValuationDate(java.util.Calendar
        valuationDate) {
        this.valuationDate = valuationDate;
    }
    public double getValue() {
        return value;
    }
    public void setValue(double value) {
        this.value = value;
    }
    public String[] getNames() {
        return names;
    }
    public String getNames(int index) {
        return names[index];
    }
    public void setNames(String[] names) {
        this.names = names;
    }
    public void setNames(int index, String name ) {
        this.names[index] = name;
    }
}
    public void setValuationDate(java.util.Calendar valuationDate)     {
        this.valuationDate = valuationDate;
    }
    public double getValue() {
        return value;
    }
    public void setValue(double value) {
        this.value = value;
    }
}
```

The following is an example of a .NET Interop type:

```
[Serializable]
    public class Valuation {
```

```
        public DateTime valuationDate;
        public double value;
    }
```

R Interoperability

Before you create an R Service, you must decide which types of clients you intend to support. There are restrictions to the input and return types depending on the client that is used.

There are three main issues to address for each client type:

- Input and return value types
- `ServiceType` definition
- `init` method interface

Input/return Values and the functionInterface Setting

The `functionInterface` setting in the Service Type determines the conversion scheme that is applied to incoming input arguments and the outgoing return value.

There are three possible settings:

- `RAW` (default) - Input arguments and return values are not converted at all.
- `UNWRAP_RETURN` - Input arguments are not converted at all. However, if an R function returns a vector with a single element, the single element in the vector is returned instead. This is convenient for Java and .NET Drivers and a requirement for C++ Drivers. This setting has no effect if the returned value has more than one value.
- `NATIVE_R` - Input arguments and return values are native R types. Therefore they are not subject to any conversion.

Below are examples of how to create R Services depending on the client that has access to it.

Supporting C++ Clients

The C++ client is the most restrictive client to support.

Input and return type

C++ clients only support strings as input arguments and return values. Therefore, if your Service is accessed by a C++ client, all R functions exposed in the Service must only take strings and return a single string value.

ServiceType

Set `functionInterface` to `UNWRAP_RETURN` for C++ clients.

init method

If your Service has an init method, it must take at least one argument. For example:

```
init <- function(unused) {

}
```

The init method requires at least one argument even when `ServiceFactory::createService(const std::string &serviceName)` is used.

The init method can have multiple string arguments and the precise number of parameters must be provided as `initData` when creating a Service. Otherwise, Service initialization fails.

Supporting Java/.NET Clients

Java and .NET clients are less restrictive than the C++ client.

Input and return type

Java and .NET clients support their respective native types, and these types can be used as input arguments and return values.

ServiceType

`functionInterface` can be unset, or set to `UNWRAP_RETURN` depending your preference.

init method

The init method can have multiple arguments and the precise number of parameters must be provided as `initData` when creating a Service. Otherwise, Service initialization fails.

Supporting R Clients in R mode

R clients can access Services exposed using the C++ like Service interface and Services implemented in R and exposed as `NATIVE_R`.

R clients can call Services with the C++ like interface and Services exposed as `NATIVE_R` concurrently. Keep in mind that you cannot expose some functions with the C++ like interface and others as `NATIVE_R` in the same Service Type.

Input and return type

R clients can use native R types as arguments and return values. Refer to the list of supported types below.

ServiceType

`functionInterface` can be set to `NATIVE_R`.

init method

The `init` method can have multiple arguments and the precise number of parameters must be provided as `initData` when creating a Service. Otherwise, Service initialization fails.

Supported R Types

Supported R types are:

- `logical`
- `integer`
- `real`
- `raw`
- `complex`

- `nil` (`NULL` value)
- `string`
- `vector`

`vector` can contain any combination of the above types and `vector` itself.

Python Interoperability

You must carefully handle type conversion since the Python Driver wraps the C++ Driver and the data is being implicitly converted to string before being transmitted, and then it is implicitly converted from string in the Grid Library unless explicit conversion is used.

Maintaining State

To maintain state on a non-virtualized Service, you typically use a field or set of fields in your object to maintain that state. (For C++ or Command Services, state is saved in a slightly different manner.) Because a Service Session can be virtualized on a number of Engines, adjusting a field's value using a Service request adjusts only that value on the Engine that processed that request. Instead, you must declare the appropriate class method as a stateful method in the Service Type Registry and use the `updateState` method to guarantee that all Engines update the state. Register all methods used to update the state as such on the Service type, either as one of the `setStateMethod`s or `appendStateMethod`s.

When an Engine processes a Service request, it first processes all update state calls that it has not yet processed, in the order in which they were called on the Service instance. These calls are made prior to the execution of the request. The append value determines whether to make previous update calls. If append is false (a *set*), all previous update calls are ignored. If append is true, all calls starting from the last set call are performed. Typically, then, append calls update a subset of the state, whereas set calls refresh the entire state. If you intend your Service instance to be a long running state with frequent updates, use a set call on a regular basis so that Engines just coming online do not need to perform a long set of updates the first time they work on this Service instance.

In the case of a task retry when appending state, a task is appended until the correct update state required by the task is reached. Note that this does not occur when using multiplexed Engines — the task receives the most current state.

Initialization

Use the `initMethod`, one of the container binding fields defined above, to initialize the state on a Service that maintains state. You can also use it for other purposes, such as establishing a database connection. The `initMethod` is called with the initialization data the first time an Engine processes a request on a Service instance. It is also called prior to an `updateState` call if it has not already been called.

Cancellation

A request can be canceled for a number of reasons. The Admin API or Administration Tool can directly cancel requests. Canceling a Service Session cancels a request. If the `killCancelledTasks` option is true for this Service, the Engine process exits and the Engine restarts. (By default, the `killCancelledTasks` option is true, except for .NET Services, where it is false by default.) However, in many cases, it is not necessary to do so, and you would prefer to interrupt the calculation so that the Engine becomes immediately available.

In this case, set the `killCancelledTasks` option to false, implement a `cancelMethod`, and register it on the Service type. This method must interrupt any Service method that is in process. It is also possible to call `cancelMethod` after the Service method finishes processing, so the implementer must take this into account.

Note

Use the `cancelMethod` to interrupt the task execution only. Do not use `cancelMethod` for cleanup of any sort. Use the `destroyMethod` for cleanup.

A typical use case is a calculation that periodically checks a cancel flag. The cancel method would set that flag, interrupting the calculation.

Destruction

Often, a Service needs to perform cleanup operations on the Engine when the instance is destroyed, such as closing a database connection. If so, implement and register a `destroyMethod` on the Service type. This method is called whenever a Service instance is

destroyed. It is called on any active Service Sessions on an Engine whenever an Engine shuts down.

Service Instance Caching

Engines maintain a cache of all Engine Service Instances that are currently active on that Engine, set by the Engine Configuration. If an Engine is working on too many Sessions, Engine Service Instances can get dropped from the cache. When this happens, the `destroyMethod` is called, and work that began on that Service is discarded. If an Engine processes a subsequent request, it restarts working on that Service from scratch.

Invocation Variables

While you can implement a Service that is independent of DataSynapse libraries, on certain occasions you might need to refer to properties within the environment of the Engine. This can be accomplished without using additional code with variables that are retrieved in various ways depending on the type of Service:

- **Java** System properties
- **DynamicLibrary** Environment variables, with the same name as Java variables, except with dots replaced with underscores. You can also use symbolic constants provided by `DynamicLibraryFunctions.h`.
- **.NET** `System.AppDomain.CurrentDomain` data values; for example: `System.AppDomain.CurrentDomain.GetData("ds.ServiceSessionID")`
- **Command** Environment variables, with the same name as Java variables, except with the dots replaced with underscores.

The Engine provides the following variables:

Engine Variables

Variable	Description
`ds.ServiceSessionID`	The unique identifier for the Service Session being invoked.

Variable	Description
`ds.ServiceInvocationID`	A number uniquely identifying the invocation (task) of the Service instance.
`ds.ServiceCheckpointDir`	The absolute path to the directory that the Service uses for reading and writing of checkpoint data, if checkpointing is enabled for the Service.
`ds.ServiceInfo`	If this variable is set in an invocation, the value is displayed in the Administration Tool on completion of the invocation.
`ds.WorkDir`	The work directory for the Engine. This variable is set to the directory from which the Service is executed. By default, it is the absolute path to the Engine installation directory, where `machinename` is the name of the machine running the Engine, and `instance` is the number of the Engine instance. For example, a single Engine machine has a *`machinename`*`-0` directory; one with two Engine instances also has a *`machinename`*`-1` directory. Each Engine instance directory has a `log` directory containing Engine logs and a `tmp` directory. Note that the `tmp` directory is periodically deleted by the Engine. The **Temp File Time-to-Live (hours)** setting in each Engine Configuration controls the frequency with which the Engine cleans this directory.
`ds.DataDir`	The data directory for the Engine, which is the absolute path directory in which DDT (Direct Data Transfer) data is stored. The cleanup frequency is also controlled by the Engine Configuration.
`ds.GridLibraryPath`	A list of paths, the contents of which are absolute paths to the root directories of all the expanded and loaded Grid Libraries.

In addition, any environment variables available on the Engine are also available in the Engine Service Instance. Note that any environment variables use absolute paths.

Accessing Services

This section describes how to access and use Services with GridServer.

Services

You can use the GridServer API in Java, C++, .NET, COM, or Python to develop an application that accesses Services. The Java, C++, .NET, and COM Drivers contain API documentation describing how to do this. For example, when you use Java, refer to the Javadocs found in the GridServer Administration Tool, for the package `com.datasynapse.gridserver.client`. Use the `Service` class to access the Service either synchronously or asynchronously.

For examples of developing applications using the GridServer APIs, see the *GridServer Service-Oriented Integration Tutorial*.

Proxy Generation

Proxy Generation is the automatic generation of a client proxy class that mirrors the registered Service type.

Think of the Service as a binding to a virtualized Web Service that can process asynchronous requests in parallel. Additionally, because the proxy does not expose any DataSynapse classes, it provides a standards-compliant approach to integrating applications in a vendor non-specific way.

The following rules apply to the generated proxy:

- The use of the proxy class is completely independent of the DataSynapse API. Client code that uses the proxy class does not need to import or reference any DataSynapse classes.
- If the Service has an `initMethod`, the proxy constructor takes any arguments to that method.

- All Service methods produce synchronous and asynchronous versions of the method on the proxy.
- Each update method has a corresponding update method on the proxy.
- The `cancelMethod` and `destroyMethod` are called implicitly and thus do not generate methods on the proxy.
- The `targetPackage` field identifies the package (in Java) or namespace (in .NET) in which to place the generated classes. Its default value is the name of the Service.
- If you use xmlSerialization, classes are generated for all non-primitive types, which must be interop types. If you do not use xmlSerialization, classes can be any serializable type, they are not generated, and the client must have access to those same classes (by a JAR/Assembly.)
- When generating proxies, a Java `Calendar` object is represented by a `DateTime` object in a .NET proxy, and vice-versa.

The proxy is generated using the **Services > Services > Service Types** page. The proxy is generated on an Engine, so successful proxy generation requires an available idle Engine.

Service Options

Each Service has an Options object that contains configuration parameters and settings. For example, some commonly used options include `PRIORITY` and `GRID_LIBRARY`. To see the options for the `Options` object, refer to the API reference documentation.

Service Options can be set in two ways: on the **Service Types** page, or when you create the Service Session with the client. If an option is set in the registry, the client cannot override it. If it is left as `[not set]` in the registry, and it is not set by the client, Service Options has the default value.

Service Invocation Context

The `ServiceInvocationContext` class provides an interface for interacting with an invocation, such as getting the session and task IDs, while it is running on an Engine. This is an alternative to using, for example, the system properties when running a Java Service. Using this class enables immediate updating of invocation information. In contrast, setting the `INVOCATION_INFO` system property only updates at the end of the invocation.

The `ServiceInvocationContext` object can be reused; the method calls always apply to the currently executing Service Session and invocation. Make all method calls by a `service`, `update`, or `init` method; if not, the method call might throw an `IllegalStateException` or return invalid data. Note that you cannot call this method from a different thread; it fails if it is not called from the main thread.

Setting Task Description

An example use case of the `ServiceInvocationContext` class is how to change the task info field.

You can use the `ServiceInvocationContext.updateInfo()` method on the Engine to set the info, and can be updated a number of times while the task is running to provide real-time info on the task state.

Shared Services

A *Shared Service* is a Service instance that multiple Clients executing on different processes or machines can attach to. State is maintained across the Service instance, and there is only one instance at any given time of a particular Shared Service running on a Broker. For example, you might use a Shared Service when GridServer Service invocation requests are delivered on an Enterprise Service Bus to a middle-tier client. If this middle-tier client is load-balanced across multiple systems, using a Shared Service lets the Broker reaggregate these related Service invocations.

Creating a Shared Service

To create a Shared Service, specify the `SHARED_SERVICE_NAME` option when creating the Service. Once the Shared Service exists, any client of the same type attempting to create a Service with the same name and the same `SHARED_SERVICE_NAME` attaches to the already existing Service. All clients sharing the instance also share the same Service ID.

Limitations to Shared Services

There are certain limitations to Shared Services, listed below:

- If there is an init method, it cannot take any arguments.
- Service options are set when the Service Instance is created initially; you cannot override these Service options on subsequent attaches.
- Clients that share Services must be the same type. For example, a Java Driver cannot share a Service created with CPPDriver.

Warning

Never use the same Shared Service from within the same process, or launch multiple Drivers with the same data/log directories. Shared Services are not designed for such situations and there is no benefit or reason to do so. Doing so can result in premature task data deletion and poor performance.

Ending a Shared Service

To cancel an entire Shared Service, use the Administration tool (at **Services > Servides > Service Session Admin**) or the Web Service Admin interface. When a client cancels a Shared Service instance, it cancels only the tasks it submitted; other clients' tasks continue. The Shared Service becomes inactive when the last client detaches from the Service. The inactive Service closes after waiting for the amount of time specified by `SHARED_SERVICE_INACTIVITY_KEEP_ALIVE_TIME`. If this variable is not set or has a value of 0, the Service closes immediately.

Shared Services and Failover

Do not depend upon Engine state for a Shared Service being maintained through failover. Engine update information can reside on several Drivers running the Shared Service. If, after failover, all of these Drivers are not available or not running the Service, Engine state is indeterminate.

Broker Spanning Services

A normal Service runs on a single Broker and is available to the Engines connected to that Broker. Using Broker Spanning, a single Service can take advantage of all Engines on an

entire grid across all of the grid's Brokers. When Broker Spanning is enabled on a Driver, the Driver concurrently maintains connections to a user-defined set of Brokers, and tasks for Services are submitted to all of those Brokers.

Enabling Broker Spanning on a Driver

Broker Spanning is enabled at the Driver level. When enabled, all Services are panned over all Brokers. To enable Broker Spanning on a Driver, set `DSBrokerList` to a semicolon-delimited string of Broker names for all the Brokers you want to span.

For example:

```
DSBrokerList=London1;London2;Cambridge
```

Note that you must allow the Driver on all Brokers in the list.

You can also set `DSBrokerList` to the wildcard character of `*`. This sets the list to all Brokers on which the Driver has the Execute Services permission. This includes offline Brokers.

When Broker Spanning is enabled, the Driver submits an independent Service for each of the Brokers specified in `DSBrokerList`. The Services submitted to each of the Brokers are separate entities, but for the convenience of the user, have the same Service ID. This allows the user to reference the Services easily across the Brokers when using the Dashboard. While Service IDs for the Broker Spanning Services are the same across all spanned Brokers, the Task IDs for the spanned Service are unique across all spanned Brokers and the Task ID is maintained if the task is moved across spanned Brokers.

Admin API Usage

For the purposes of Broker Spanning, the Admin API provides a way to specify which Manager the Admin API is acting on. `AdminManager.setManager()` takes the name of the Manager it acts on as a string parameter. Admin API components retrieved act only on the Manager most recently specified by `AdminManager.setManager()`. To perform Admin API operations on multiple Managers at the same time, prepare collections of Admin API component objects retrieved after each `AdminManager.setManager()` call. For example, if you want to get all `EngineInfo`s for all spanned Managers, call `AdminManager.setManager()` followed by a call to `AdminManager.getEngineAdmin()` for the Manager last specified by `AdminManager.setManager()`. This must be done for each of the Managers. Each of the

`EngineAdmins` can now be used to call `EngineAdmin.getAllEngineInfo()` to retrieve all the `EngineInfos` associated with that `EngineAdmin's` Manager. See The Admin API for more information.

Scheduling and Task Expiration

When Broker Spanning is enabled, the Driver continuously attempts to maintain connections to all Brokers. It assigns tasks to Brokers using a simple algorithm that attempts to balance the workload. If a connection times out on any Broker, the Driver does Driver-side expiration and resubmits any outstanding tasks to the remaining Brokers. In the event of a Broker failure, long-running tasks continue to run on Engines; when they log in to the Director, they are directed to whichever Broker that has ended up with the expired task.

After submitting a large number of tasks, some Brokers might contain tasks in the queue to execute whereas other Brokers might not. When Task Expiration is enabled, tasks expire on the Broker if they are not scheduled to any Engine after a specified period of time after submission. Task Expiration alleviates the problem of unbalanced Broker queues. Upon expiration, tasks are automatically assigned to the next appropriate Broker. Task Expiration is available for Broker Spanning Services only.

To activate Task Expiration, set the following Service options:

- `PENDING_TASK_TIMEOUT` — To enable Task Expiration, set this to a value indicating the number of minutes to wait before reassigning tasks to another Broker.
- `TASK_EXPIRATION_START_OFFSET` — Optionally enables you to delay the start of Task Expiration by a specified number of minutes. This offset represents a one-time delay in starting the timer to check for Task Expiration. For example, if the value of `PENDING_TASK_TIMEOUT` is set to 5 and `TASK_EXPIRATION_START_OFFSET` is set to 60, the first task to be expired is 65 minutes after the Service was submitted to the Broker. Additional task expiration would happen every 5 minutes after the first expiration event. The Broker first waited for `TASK_EXPIRATION_START_OFFSET` to pass (60 minutes) before starting the `PENDING_TASK_TIMEOUT` timer (5 minutes) for a total of 65 minutes. The default value for `TASK_EXPIRATION_START_OFFSET` is 0.

You can also tune Task Expiration on the Broker by specifying the following values in the Administration Tool at **Admin > System Config > Manager Configuration > Services**:

- **Minimum Tasks to Expire** — You can require that a certain number of tasks expire regardless of the Percentage of Tasks to Expire (Given that there are eligible tasks to expire).

- **Percentage of Tasks To Expire** — You can expire only a percentage of tasks of those eligible for expiration every minute.
- **Idle Broker Only Task Expiration Enabled** — When enabled (the default), this setting prevents tasks from expiring if there is not an idle Broker available.

Administration

Broker Spanning Services can be viewed and administered in the GridServer Administration Tool, at **Dashboard > Spanned Services**. This page is only viewable on Directors. By default, this page shows no Services; you must first perform a search using the Search tool. Administrators can dynamically change Pending Task Timeout and Task Expiration Start Offset on each Service from this page.

Broker Spanning Service Limitations

Broker Spanning Service limitations are as follows:

- Shared Services is not supported. Shared Service functionality conflicts with Broker Spanning Services.
- GridCache is not supported. Since a Service's cache is local to a Broker, the tasks from other Brokers cannot access each other's caches.
- Task Dependencies are not supported. Task Dependency is Broker-scope and therefore cannot work across Brokers.
- Service dependencies are not supported. Service Dependency is Broker-scope and therefore cannot work across Brokers.
- AutoPack is not supported.
- PDriver is not supported.
- The number of priority levels, which is set at **Admin > System Admin > Manager Configuration > Services > Scheduling**, must be the same on all Brokers. If you change this value on a Broker, you must restart the Broker.
- Engine Timeout Minutes > 0 is not supported. That is, if an Engine logs off while working on a Task, it does not continue to work on that while attempting to log back in to a Broker.

Service Groups

Service Sessions can be collected together in a group to aid in administrative tasks. A convenience class called `ServiceGroup` is provided in the API, which enables you to create a Service Group and later create new Service Sessions within the Service Group. Each new Service Session created within a Service Group is automatically assigned `Description.SERVICE_GROUP_ID`, a generated random unique ID for that group.

To view and maintain Service Groups, select **Services > Services > Service Group Admin**. The Service Group Admin page enables you to take actions on an entire group of Services at once, similar to the way you can act on Services on the **Services > Services > Service Session Admin** Page. For example, you could cancel a Service Group, which would cancel all of the Service Sessions within that Service Group.

Data References

Data References are a convenient programming interface for passing lightweight references to data across the network. A Grid client or Service can create Data References, pass them over the network, but leave the data where it created the original Data Reference. If any Grid client or Grid node actually needs the data, it can de-reference the object and the data is automatically downloaded from the original source.

You can use this abstraction for generalizing Grid workflows. A Grid client can receive the results of a particular Service as a reference, and then send another request to the Grid with that reference. The GridServer Engine that services the request de-references the data object, loading it from the original Grid node that produced the data. This is equivalent to passing pointers across the network.

A `DataReference` is an Object that you can pass interoperably if you use it as an argument, return type, or a GridCache object. Note that to work interoperably, it must be the actual object passed; it can't be part of another object.

To create a `DataReference`, use a `DataReferenceFactory`. You cannot create Data References with a null source. To retrieve the actual data, use the `fetch` methods. The data is not cached after a `fetch`, that is, each fetch retrieves data. After a reference expires, a `fetch throws` a FileNotFound exception. A downed Client throws a Connect exception, although in some cases, refused connections are due to other causes, such as socket backlog limitations.

When an object is transmitted over the network, it is serialized on the sending side and then reconstituted completely in memory on the receiving side. The larger the object, the more memory it occupies. In situations with large `DataReference` objects, you must use `DataReferenceOutputStream` instead. Streaming the data can reduce the memory that would otherwise be necessary to reconstitute it in its entirety. As an added benefit, streaming also saves the time it would have taken to reconstruct the entire object.

C++ Data References

Because there is no inherent serialization support in C++, functionality is added to convert a Data Reference to a `byte[]` (and the reverse) so that it can be sent to another client and used by that client.

Because reflection is not available in C++, object Data References are not available from the C++ API.

Python Data References

As a result of Python wrapping the C++ Driver, object Data References are not available from the Python API.

Service Collection

By default, the Driver Service Instance collects results as soon as they are available. In some cases, you want to collect results at a later time, or never collect them. Service collection can be defined with the Service Option `COLLECTION_TYPE`. Possible values for `COLLECTION_TYPE` include `IMMEDIATELY`, `LATER`, `NEVER`, and `AFTER_SUBMIT`.Values used with `COLLECTION_TYPE` are as follows:

Service Collection Values

Value	Description
`AFTER_SUBMIT`	Collection does not begin until all tasks have been submitted. See Collect After Submit for more information. This enables optimal submission without

Value	Description
	network competition from the collection when the Service follows the "submit all, collect results" pattern.
IMMEDIATELY	Task outputs are collected as soon as they are ready. This is the default.
LATER	Task outputs are not collected in this Service. Another Service collects them later. See Deferred Collection (Collect Later) for more information. This is not available from a Service proxy.
NEVER	Task outputs are not collected. See No Collection (Collect Never) for more information. This can be used, for example, in the case where a Service write data directly to a database. This is not available from a Service proxy.

The LATER and NEVER collection modes are not for long-running Services. Use them only for batch submissions that finish quickly. If you use them for Services with indefinite duration, there is no way to clean up the inputs.

Collect After Submit

A common Service pattern is to submit all of your requests, then wait for the results. When using this pattern, the AFTER_SUBMIT collection mode must be used. This enables optimal submission without network competition from the collection.

When this mode is used, the collection of the results begins when either `Service.waitUntilInactive(long)` or `Service.destroyWhenInactive()` is called.

Also, when using this mode or even when using some hybrid pattern that submits many requests at a time, `Options.INVOCATIONS_PER_MESSAGE` must **always** be set greater than one, to speed submission time.

Calling `Service.execute(String methodname, Object data)` does not work with this collection type; an instance of `ServiceException` is thrown.

The following is an example of using the AFTER_SUBMIT collection mode:

```
Properties props = new Properties();
props.setProperty(Options.COLLECTION_TYPE,
                  Options.Collection.AFTER_SUBMIT);
props.setProperty(Options.INVOCATIONS_PER_MESSAGE, "20");
```

```
Service s = ServiceFactory.getInstance().createService(serviceType,
null, props, null);
// submit all the work
for (int i = 0; i < numInputs; i++) {
    s.submit("myMethod", getData(i), handler);
    }
s.waitUntilInactive();
```

Fault Tolerance

In the event of a Driver failure, the Broker automatically cancels the Service after the interval specified by the Client Timeout setting. It is possible to collect tasks from a failed client application for Services of type `Collection.AFTER_SUBMIT` and `Collection.IMMEDIATELY` if cancelslation 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, as described in Deferred Collection (Collect Later).

Note that for fault tolerance to work, the collecting Driver must be the same platform as the original submitting Driver. That is, if the Service was created with Java Driver, it must also be collected with Java Driver.

Deferred Collection (Collect Later)

The `LATER` mode indicates that the client can submit and update data to the Service Session. It does not collect the results, however, as another instance attaches to the Service Session to collect the results. None of the results are removed from the Service Session until it is destroyed by the collector. The `LATER` mode cannot be used from a Service proxy.

There are two reasons to use this method:

1. The architecture requires different applications for submission and results processing.
2. To recover from a failure in the application that embeds the Driver. Since results are not removed until the Session is destroyed, if the application undergoes a failure it can recollect the results when it restarts.

Deferred collection Services require that the submitting Driver call destroy on the Service to indicate that submission is complete. If you are using the submitting Driver in such a way that it exits after submitting the tasks and calling destroy, do not call `System.exit` or

`exit` from the `ServiceLifecycleHandler`, as this stops the destroy message from getting to the Broker. Also note that if you are exiting the submitting Driver immediately, you must set `DirectDataTransfer` to false in the `driver.properties` file.

After creating a Service with the deferred collection, use `ServiceFactory.getService()` to retrieve results. After you collect all results, call destroy to indicate to the Broker:

- that the instance has collected all outputs and
- to destroy the Session

You can create multiple collectors, but keep in mind that if a collector calls `destroy()`, the Service is destroyed and no other collectors can finish collecting outputs.

The following is an example of how to use the `LATER` mode with recovery:

```
// Creates a new Session and submits requests to the Session
// @param serviceType The type
// @param methods The list of methods
// @param args The list of arguments
// @return The id of the Session
// @throws Exception on error.
private String submitService(String serviceType, String methods[],
Object[][] args) throws Exception {
    // create the session as a Collection.LATER type
    Properties props = new Properties();
    props.setProperty(Options.COLLECTION_TYPE,
Options.Collection.LATER);
    Service cs = ServiceFactory.getInstance().createService(serviceType,
        null, props, null);
    // Submit all requests.
    // Note that the handler must be null because this Instance cannot
collect.
    for (int i = 0; i < args.length; i++) {
        cs.submit(methods[i], args[i], null);
        }
    String id = cs.getId();
    // destroy to indicate that submission is complete, and to free
local
    // resources
    cs.destroy();
    // save this ID to a file, for recovery purposes
    saveServiceForRecovery(id);
    return id;
}
// Starts collection of results from a Collection.LATER Session
// @param id The id of the session
// @param handler The invocation handler
// @throws Exception on error.
```

```
private void collectService(final String id, ServiceInvocationHandler
handler) throws Exception {
    // create a handler that removes this Service ID from the list in
the file
    // when it is finished
    ServiceLifecycleHandler slc = new ServiceLifecycleHandler() {
        public void destroyed() {
            removeServiceFromRecovery(id);
        }
        public void destroyed(ServiceException e) {
            removeServiceFromRecovery(id);
        }
    };
    // get an instance of the session, which starts collecting results
    Service cs = ServiceFactory.getInstance().getService(id, handler,
slc);
    // set the Service to be destroyed when it finishes collecting all
output
    cs.destroyWhenInactive();
}
// Runs a Service by first creating a Collection.LATER Session,
submitting all
// requests, then getting the collection instance to collect the
results.
// @param serviceType The type
// @param methods The list of methods
// @param args The list of arguments
// @param handler The invocation handler
// @throws Exception on error.
private void runService(String serviceType, String methods[], Object[][]
args, ServiceInvocationHandler handler) throws Exception{
    String id = submitService(serviceType, methods, args);
    collectService(id, handler);
}
// Recovers from an application failure by starting collection of
Sessions that
// did not complete collection prior to failure
// @param handler The invocation handler
// @throws Exception on error.
//
private void recoverAll(ServiceInvocationHandler handler) throws
Exception {
    String[] recovered = getAllRecoveryServices();
    for (int i = 0; i < recovered.length; i++) {
        collectService(recovered[i], handler);
    }
}
```

Deferred Collection Failover

While other Sessions manage failover by resubmitting all outstanding tasks to a Failover Broker, in this case the submitting Session has already been destroyed, and the Driver might even be offline. So, to handle failover in this case, you must do the following:

Procedure

1. Every standard Broker must have one Failover Broker associated with it.
2. Configure a shared filesystem that can be accessed from each Broker pair.
3. Go to **Admin > System Admin > Manager Configuration > Services**, and under the **DataTransfer** heading, change the **DataTransfer Base Directory** to the location of the shared file system for each pair. Each pair must have its own directory.
4. If the submitting Driver does not use an external web server and it disconnects after submission, disable DDT. The data is then served by the Broker from the shared file system.
5. Since the collecting Driver cannot auto-resubmit when the results cannot be collected from a Daemon that is offline, it's recommended that DDT is also disabled on Engines.
6. The submitting and collecting Drivers must be allowed to log in to both Brokers in the pair.

If a Broker fails while the Service is running, the Failover Broker takes over by recreating the task queues. If the original Broker comes back up, the Failover relinquishes control and the original recreates the task queues.

When the collecting Driver starts to collect, if the original is down, it collects results from the Failover. If that then comes up, it migrates to the original and continues.

No Collection (Collect Never)

The `NEVER` collection mode enables a Service to submit tasks and not collect them. Such a Service, for example, writes results to a database. Services created with `NEVER` collection can only submit and update. Calls to execute throw an Exception. This collection mode is not available when using a Service proxy. To create a `NEVER` collection Service, set the `CollectionType` option to `NEVER`.

Calling destroy releases resources locally on the Driver, and indicates that the Instance is finished with submission. If the Driver is shut down and times out, the session is

considered to be done submitting, just as if the Driver called destroy. After the session finishes submitting due to one of the two prior events, and all tasks complete or fail, the session automatically closes.

No Collection Failover

There are no special issues with failover on No Collection Services, and as many Brokers as you want can be used to recover any running Services. You would, however, need to ensure that any database to which you are sending results has its own failover or redundancy configuration in case of failure.

Engine Pinning

Engine Pinning enables a Service Session to specify that once an Engine has worked on a Service Session, it cannot work on any other Service Sessions as long as that session is in progress. This enables you to quickly replicate 1-to-1 architectures, or if for legacy reasons or lack of availability to source code to maintain massive amounts of expensive-to-replay state on an Engine.

Typically, you use Engine Pinning in conjunction with Max Engines to limit one or a set of Engines to work solely on a Service.

Engine Pinning is available for any type of Service. It is exposed as two Engine-side methods, `pinToService()` and `unpinFromService()`. If an Engine calls `pinToService()` while working on a task for a Service, it is marked as pinned to the session when it completes the task, regardless of whether the task succeeded. From this point, the Engine takes tasks only from this Service. Once the session finishes, or when the Engine calls `unpinFromService()`, the Engine is no longer pinned and can work on other Services.

Engines that are pinned are unpinned automatically in the following circumstances:

- If the Broker loses its connection with the Engine for any reason (such as loss of heartbeat)
- If the Engine process ends for any reason
- If an Engine performs a soft log off due to Engine Sharing
- If a task fails due to an error in DataSynapse code (such as reading input)
- If the session is destroyed

Exceptions issued by user code that result in task failure do not cause an Engine to be unpinned unless the exception specifies Engine restart, in which case the above requirement applies.

You can configure Engine balancing so pinned Engines are always reported as busy by Brokers. This prevents the Engine from being shared with other Brokers. This can be configured at **Admin > System Admin > Manager Configuration > Engines and Clients**, under the **Engine Balancing** heading, with the **Treat Pinned Engines as Busy** parameter. This is false by default.

You can implement other pin/unpin strategies—for instance, unpinning after a certain amount of idle time, or when the Broker queue is empty—as a separate Engine thread that polls for a given condition and unpins as necessary.

Running a Driver from an Engine Service

It's possible to run a recursive Service, which is a Service that runs a Driver on the Engine to call a Service. To do this, you must deploy the Driver packaged in a Grid Library.

To run a Driver from an Engine Service:

1. In the GridServer Administration Tool, go to the top navigation bar, and click the **Downloads** icon.
2. Click the SDK for your platform to download it.
3. Package the `driver.properties` file in a Grid Library. (For more information about creating Grid Libraries, see Creating Grid Libraries.) In the `grid-library.xml`, set the environment variable `DSDRIVER_DIR` to the value `$ds.GridLibraryRoot$`. Also, set the `jar-path` element to `$ds.GridLibraryRoot$`. The `grid-library.xml` must look similar to the following code:

```
<?xml version="1.0" encoding="UTF-8"?>
<grid-library>
    <grid-library-name>
        driver-properties
    </grid-library-name>
   <grid-library-version>7.1</grid-library-version>
   <environment-variables>
        <property>
            <name>DSDRIVER_DIR</name>
            <value>$ds.GridLibraryRoot$</value>
        </property>
```

```
        </environment-variables>
        <jar-path>
            <pathelement>
                $ds.GridLibraryRoot$
            </pathelement>
        </jar-path>
    </grid-library>
```

4. Create a Grid Library that contains all of the DLLs or JAR required to run the Driver. For example, to run CPPDriver for VC14, you would need to create a Grid Library that contained all of the DLLs from the `cppdriver/bin/vc14` directory of the SDK in a `bin/vc14` directory inside the Grid Library. Make this Grid Library dependent on the `driver-properties` Grid Library, and set the `lib-path` to the location of the DLLs. The `grid-library.xml` must look similar to the following code:

```
    <?xml version="1.0" encoding="UTF-8"?>
    <grid-library os="win64">
        <grid-library-name>
            cppdriver-win64-vc14
        </grid-library-name>
        <grid-library-version>7.1</grid-library-version>
        <dependency>
           <grid-library-name>
                driver-properties
            </grid-library-name>
        </dependency>
        <lib-path>
            <pathelement>.\bin\vc14</pathelement>
        </lib-path>
    </grid-library>
```

5. Upload and deploy both Grid Libraries on the **Services > Services > Grid Libraries** page.

PDriver

The *PDriver*, or Parametric Job Driver, can execute command-line programs as a parallel processing service using the GridServer environment. This driver enables you to run a single program on several Engines and return the results to a central location, without writing a program in Java, C++, or .NET.

Overview

The *PDriver*, or Parametric Job Driver, can execute command-line programs as a parallel processing service using the GridServer environment. This driver enables you to run a single program on several Engines and return the results to a central location, without writing a program in Java, C++, or .NET.

PDriver achieves parallelism by running the same program on Engines several times with different parameters. A script defines how these parameters change. For example, a distributed search mechanism using the `grep` command can search a network-attached file system, giving each task in the Service a different directory or piece of the file system to search. Scripts can iteratively change the value of variables that are passed to successive tasks as parameters, step through a range of numbers, and use each value as a parameter for each task that is created, or define variables containing lists of parameters.

PDriver uses its scripting language, called *PDS*, to define Services. You can also use these scripts to set options for a PDriver Service, such as remote logging and exit-code checking.

Installing PDriver

PDriver comes with the GridServer SDK. For installation details, see The Parametric Job Driver (PDriver).

To use PDriver with SSL, you must configure your Manager to use SSL for all connections. For directions, see the *TIBCO GridServer® Installation*.

Resource Deployment

If you plan to run any custom executables using PDriver, deploy them to Engines using GridServer's application resource deployment feature, described in the *TIBCO GridServer® Administration*. Deploy resources in the `deploy/resources/`*platform*`/lib` directory or in another directory referenced by the `execute` command.

PDriver Commands

Use the following commands to run, batch, and cancel PDriver Services on UNIX and Windows systems.

PDriver Commands

Command	Description
`pdriver`	Starts PDriver, and runs a PDS script.
`bsub`	Starts a one-time Service batch submission.
`bcoll`	Collect a batch job on a one-time basis.
`bstatus`	Lists status on pending batch jobs.
`bcancel`	Cancels a pending batch job.

The pdriver Command

Start PDriver with the `pdriver` command:

```
pdriver [ -bsub | -bcoll batchid | -parallel] [-RA] [-noprompt] [-user
driverusername] [-pass driveruserpassword] [-domain windowsdomain] script
```

The `pdriver` command runs the script specified with *script*. The following table describes the arguments you can use:

PDriver Command Arguments

Argument	Description
`bsub` and `bcoll`	Use `bsub` and `bcoll` for batch submission and collection. Use them to • Submit long-running Services without tying up the Driver. Submitting a Service using `bsub` creates a Batch Execution with a Batch ID that you can use to run the Service unattended and collect the outputs later. The Batch Executions created are of the same type as those from the Batch scheduling facility. For more information about Batches, see the *TIBCO GridServer® Administration*. • Run multiple tasks and or prejob/postjob tasks that the PDS script defines.
`parallel`	When you run a single PDS that contains multiple job blocks, this option runs the blocks in parallel.
`RA`	`RA` specifies a Run-as job—a job that runs as the current user and with the current Windows domain. PDriver prompts for a password and passes it with the user and domain for authentication. If you run PDriver on a UNIX machine, you must use the `domain` argument to pass a domain, as shown below. For more information about Run-As, including how to configure your Engines, the *TIBCO GridServer® Administration*.
`noprompt`	`noprompt` turns off the password prompt and sends no password when authenticating for Run-as; this is useful only when you use the `RA` argument and Service RA authentication is disabled on the Broker.
`user` *name* and `pass` *password*	When you use the `RA` argument and run PDriver on a Windows machine, you can use `user` and `pass` to pass a Driver user and password (as defined in the GridServer Administration Tool) to PDriver.
`-domain <windowsdomain>`	When you are using the RA argument and running PDriver on a UNIX machine, you can use the `domain` argument to specify a win32 domain for Windows Engines to use when authenticating.

The bsub Command

You can also run a one-time Service batch submission with the `bsub` command:

```
bsub [ -name name ] [ -priority val ] [ -disc setting ] [ -mail address ] [ -
stdin file ] [ -stdout file ] [ -stderr file ] [ -nfs ] [-RA] [-domain
windowsdomain] [-noprompt] app [args...]
```

This submits a single Service, defined by *app* `[args...]`, to be scheduled and run on the Grid. The arguments are as follows:

bsub Command Arguments

Argument	Description
`name`	Name of the Service.
`priority`	Priority of the Service.
`disc`	Indicates a *name comparator value* discriminator expression that must be satisfied to assign a Service or task to a node. For example, valid settings include `os==win32` or `cpuNo>=4`. You can use this switch multiple times. Put quotes around parameters containing a greater-than or less-than symbol.
`mail`	Email address to send a confirmation message to when the Service is completed.
`stdin`	File to use as input for the Service. This setting is variable based on the `-nfs` switch, described below.
`stdout`	File to use as output when `-nfs` is enabled.
`stderr`	File to use as error output when `-nfs` is enabled.
`nfs`	Indicates whether inputs and outputs are available on a shared file system or whether you must stage them on the Manager. When enabled, `-stdin`, `-stdout`, and `-stderr` function as absolute locators for these files. When not enabled, `-stdout` and `-stderr` are ignored, and files designated as *jobname*`.out` and *jobname*`.err` are placed in the staging directory. When not enabled, `-stdin` refers to a file on the Driver file system which is to be staged

Argument	Description
	on the Manager and retrieved by the Engine performing the Service.
RA	Runs the job on Engines as the current desktop user with the current Windows domain. PDriver first prompts for a password.
domain	Lets you specify a Windows Domain for Windows Engines to use when authenticating. This applies only when running PDriver from a UNIX machine.
noprompt	noprompt turns off the password prompt; this is useful only when Service RA auth is disabled on the Broker.
<app> [args...]	The application to run, and any optional arguments. The application is a binary or script on the Engine and found in the Engine's path. args are any arguments passed to the application.

When you submit the Service, a batch ID is reported to the console. Use this ID when collecting outputs (when using the staging directory for input and output rather than NFS mounts.)

The bcoll Command

To collect batch jobs on a one-time basis, use the bcoll command:

```
bcoll batchid
```

This is a convenient utility for retrieving the *jobname*.out and *jobname*.err files generated by bsub with nfs mode off. The argument is the batch ID indicated when bsub is finished submitting.

To check the status of a Batch Job, you must be running PDriver with a Driver user that has the appropriate Security Role on the Manager. To do this, create a Driver user mapped to a role with access to the **Batch Admin View** feature (such as the default Manage role), and set DSUsername and DSPassword in the driver.properties file to this username.

The bstatus Command

To get status on batch jobs, use the `bstatus` command:

```
bstatus [ -jobs ] [ -engines ] [ -stagedir id ] [ -canceljob id ] [ -
batches]
```

`bstatus` accepts the following arguments:

bstatus Arguments

Argument	Description
`jobs`	Lists all jobs on the Manager. This does not include pending batch jobs. The output displays name, jobid, tasks, priority, and status for each job. If there are no jobs, only the four rows of heading lines are returned.
`engines`	Lists all Engines connected to the Manager. The output displays name, ID, OS, and status of each Engine. If there are no jobs, only the four rows of heading lines are returned.
`stagedir`	Lists all files living on the staging directory for the given job ID or batchjob ID.
`canceljob`	Cancels a job. This only works for job IDs, not batch IDs, and the Driver must have appropriate permissions for this operation.
`batches`	Lists all Batch jobs on the Manager and their status. The output displays name, batchid, and status of each Batch. If there are no batches, no output is returned.

The `bstatus` command prints its output interspersed with Manager log output on `stderr`. To view only the `bstatus` output on a UNIX system, you can redirect `stderr` to null, for example, with `bstatus -jobs 2>/dev/null`. Windows users can use the syntax `bstatus -jobs 2>&0`.

To check the status of a Batch Job, you must be running PDriver with a Driver user that has the appropriate Security Role on the Manager. To do this, create a Driver user mapped to a role with access to the **Batch Admin View** feature (such as the default Manage role), and set `DSUsername` and `DSPassword` in the `driver.properties` file to this username.

The bcancel Command

To cancel a batch job pending on the Manager, use `bcancel`:

```
bcancel batchid
```

The argument is the batch ID returned by `bsub` or `pdriver` in `bsub` mode.

There is an important distinction between batch IDs and Service IDs. A batch ID refers to a Service that is pending execution in the batch queue but has not been handed over to the scheduler. Once that Service is scheduled, a Service with a separate Service ID is launched. The `-canceljob` switch in `bstatus` only works for Service IDs. To remove a pending batch job, `bcancel` must be used. This is unavoidable since running Services and pending batch jobs are separate entities and are tracked differently. Likewise, to perform output collection and to see the contents of the staging directory, the batch ID must be used. This is due to the fact that at batch submission, only the batch ID is known since the Service ID is not generated until run-time. Therefore the batch ID is used as a key for the staging directory.

About PDS Scripts

PDriver uses scripts written in the PDS syntax to define how a Service operates. Aside from defining what programs are run during a Service, PDS scripts enable you to define what happens before and after a task or Service. It also enables you to schedule Services to run in the future, add conditional structure to a Service, and pass custom parameters to a Service or task.

The PDriver script (hereafter referred to as PDS) language enables the expression of distributed computations that are composed of executable programs. It is designed so that typical computations are easy to describe, while providing for advanced features such as conditional execution, iteration, scheduling, and discriminators.

This section is a reference for the PDS language. For sample PDriver scripts, see the `examples/pdriver` directory of the GridServer SDK.

PDS Basics

A single PDS file corresponds to a single GridServer Service. The computation represented by the Service usually does the following:

1. Split up the input data into several pieces, one for each task.
2. Run the tasks in parallel on the Engines.
3. Collect and combine the results.

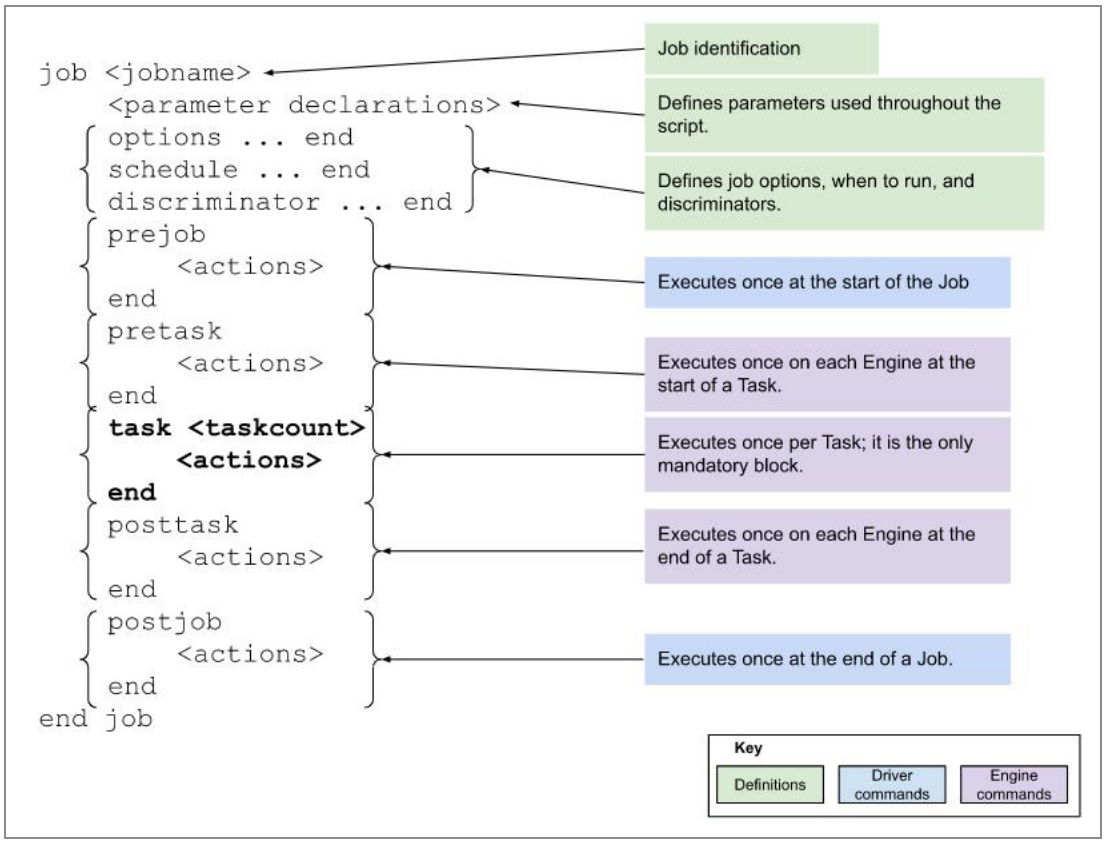
If the data is too large to be passed as command-line arguments to the program running on the Engine, then place it into files. You can locate these files in a shared directory, or copy them to the Engine and copy back the result files.

The PDS language contains constructs for carrying out various statements, such as running executables and copying files, at each point in the lifetime of a Service.

A few words about the lexical structure of PDS: whitespace is not significant, but the case is. All text from the “#” character to the end of the line is a comment and is ignored.

PDS Structure

A PDS file begins with the keyword `job` (a synonym for Service) and ends with the keyword `end`. Supply a name after `job` to identify the Service. Two types of elements can occur in between `job` and `end`: parameter declarations, which assign values to variables, and *blocks,* which describe features or statements of the Service. The `options`, `schedule`, and `discriminator` blocks describe various facets of the Service, such as when to run it and which Engines can accept its tasks. The other five blocks describe statements to be executed at different phases of the Service. All blocks are optional except for the `task` block. Multiple `job` blocks are also defined in a single PDS file, and they run sequentially by default. You can also run the `pdriver` command with the `-parallel` flag to run multiple jobs in one PDS file in parallel.



Structure of a PDS file

Details of the structure of a PDS file:

```
job jobname
options
    onerror ( fail | retry | ignore )
    maxFailedTaskRetries = val (default: 3)
    enableBlacklisting = true|false
    jobPriority = <val> (default: 5)
    autoCancelMode = "always" | "never" | "libloadfailure"
    jobOption "key" "val"
    jobDescription "key" "val"
end
<variable assignment>
schedule
    properties
    [email = "address"]
end
discriminator
    [ affects
        properties
```

```
      end ]
    properties
end
prejob
    statements
end
pretask
    statements
end
task taskcount
    statements
end
posttask
    statements
end
postjob
    statements
end
end
```

The Depends Statement

Multiple `job` blocks can be included in a single translation unit (a PDS file and any included PDS files), and by default, they run sequentially. It is also possible to define jobs that run based on the completion or failure of other jobs using the `depends` statement within a job block. For example, the job containing the following code runs if `firstjob` succeeds, if `secondjob` fails, and following `thirdjob` in either case:

```
depends
    firstjob succeeds
    secondjob fails
    thirdjob succeeds or fails
end
```

The Include Statement

You can include another PDS script within a PDS script by using the `include` statement. For example:

```
include "filename"
```

This includes a PDS script contained within `filename`. The filename can be declared with any string type. The filename is relative to the working directory from which you run PDriver, and can contain a relative or absolute path to a PDS file. The PDS file must contain a full job, and you can use it only outside the top-level `job` block. For example, a single PDS file can contain three include statements, each one including a `job` block stored in another file.

Lifecycle Blocks

Lifecycle blocks include statements that execute during the life of a Service. The following sections describe the lifecycle blocks.

Other blocks include the options block, discriminator block, and schedule block.

prejob

The prejob lifecycle block executes once at the very beginning of the job before any tasks are submitted to the Manager. These commands execute on the Driver.

pretask

The pretask lifecycle block executes once on every Engine that processes tasks for the job before any tasks are processed. Use this block for generic initialization such as obtaining input files common to all tasks.

task

Commands in the task lifecycle block execute once per task. The number of tasks in the Service is determined by the expression following the `task` keyword. Array variables referenced in the task block are indexed by the task ID, as explained in more detail in the Arrays section, below.

Since the same Engine can take more than one task, the statements in the `task` block run many times on the same Engine. The statements in the `pretask` and `posttask` blocks run only once per Engine.

The task block is the only required block in a PDS file. Thus a computation that only required executing a program ten times could be expressed by the PDS program:

```
job simple
    task 10
        execute "myprog"
    end
end
```

posttask

The posttask lifecycle block executes once on every Engine at some point after the job finishes. Place Engine cleanup tasks here. Note that the Engine's file cleaner automatically cleans all files in the `DSWORKDIR` directory. The posttask block typically executes after the postjob block executes on the Driver side, although it is not guaranteed to execute then. Execution order of the blocks in a PDS file is typically `prejob`, `pretask`, `task`, `postjob`, and `posttask`, but this is not guaranteed.

postjob

The postjob lifecycle block executes once on the Driver after all tasks finish. Use this block to obtain outputs from the staging directory, run post-processing scripts, and so on.

The Options Block

Use the options block to set Service options and description information.

Use the following directives for features specific to PDriver jobs:

Options Block Keywords and Arguments

Keyword	Argument	Description
`onerror`	`ignore`, `retry` or `fail`	The default way to handle errors. See "Builtin Commands" below for details.
`maxFailedTaskRetries`	A numeric expression	The number of times to resubmit a failed task.

Keyword	Argument	Description
autoCancelMode	never, always or libloadfailure	Whether to cancel the entire job when a single task fails. The default is libloadfailure, which cancels the job if a task fails due to the inability to load a library on the Engine side. You can set this standard option with jobOption, but the autoCancelMode directive enables you to use mnemonic strings, rather than numbers, as arguments.
enableBlacklisting	An expression evaluating to true or false	This argument is the same as engineBlacklisting for other Services. The default value is false.

Set standard options (described in the C++ API reference documentation for the Options object) with the jobOption directive:

```
jobOption "engineBlacklisting" "true"
jobOption "priority" "8"
```

Set elements of the job description with the jobDescription directive:

```
    jobDescription "serviceName" "Distributed Grep"
```

Each argument to jobOption or jobDescription must be a string or an expression that evaluates to a string. Literal numbers are not allowed.

The following jobOption directives are common to both PDriver and CPPDriver:

jobOption Directive Keywords and Arguments

Keyword	Argument	Description
email	string	An email address that is notified when a Service is completed.
priority	integer	The priority of this Service. The default value is 5.

Keyword	Argument	Description
`taskMaxTime`	integer	If a running task exceeds this amount of time in seconds, the task is rescheduled or retried based on the setting of `rescheduleOnTimeout`. The task is rescheduled or retried when the rescheduler checks for expired tasks after each poll period. This poll period is 60 seconds by default; it can be changed at **Admin > System Admin > Manager Configuration > Services**, under the **Service Rescheduler** heading, with the **Poll Period** property. The default value of `taskMaxTime` is infinite.
`autoCancel`	`0`, `1`, or `2`	Whether to automatically cancel the Service on a task failure. Possible values include 0 (`AUTO_CANCEL_NEVER`), 1 (`AUTO_CANCEL_LIBRARY_LOAD`), or 2 (`AUTO_CANCEL_ALWAYS`). The default value is 1 (`AUTO_CANCEL_LIBRARY_LOAD`).
`compressData`	`true` or `false`	Whether to compress the task, input, and output data. Compression time is minimal and recommended for data sizes greater than 10K for each input or output. The default is `false`.
`killCancelledTasks`	`true` or `false`	Whether to kill and restart an Engine if a task is canceled. If `false`, the canceled method is called rather than killing the Engine, to provide user-defined interruption of the task and any necessary cleanup. The default value is `true`. Tasks are canceled when canceled in the Administration Tool, when a Service is canceled, and when another Engine completes the task due to redundant rescheduling.
`tasksPerMessage`	integer	The maximum number of tasks for each submission or retrieval message. Regardless of this number, messages do not exceed 100 KB.

Keyword	Argument	Description
		The default is 100.
`autoPackNum`	integer	The number of tasks in an auto-packed task. In this mode, a tasklet processes multiple task inputs in one Service routine by packing task inputs into a single task and calling your Service routine on all inputs. Use this mode when there are more inputs than Engines, or tasks are of short duration, to maximize efficient use of memory and Engine processing power. If inputs are added outside of `createTaskInputs`, the Service checks every second if there are any unsubmitted tasks and submits them in a package even if there are fewer than the number requested, to ensure that all tasks are submitted. Task IDs in the Administration Tool are the IDs of the task packages, so they do not directly correspond to the task ID from the Driver and Engine's point of view. `TaskletException` auto-resubmission and task-level discrimination are not supported for this mode, and are ignored. Autopack is also not supported on multiplexed Engines. The default value is 0.
`sharedUnixDir`	string	A directory in which the Driver and UNIX Engines exchange data. This directory must be an NFS mounted directory to which all UNIX Engines working on this job have read/write access. This overrides use of the file servers on the Driver and Engines, and is optimally a directory local to this Driver for minimum network bandwidth. If set and using Windows Engines, the Windows

Keyword	Argument	Description
		shared directory must also be set to the equivalent of this directory.
`sharedWinDir`	string	A directory in which the Driver and Engines exchange data. This directory must be a Windows shared directory to which all Windows Engines working on this job have read/write access. This overrides use of the file servers on the Driver and Engines, and is optimally a directory local to this Driver for minimum network bandwidth. Typically, the share is Windows UNC format, such as `\\server\data`. If set and using UNIX Engines, the UNIX shared directory must also be set to the equivalent of this directory.
`checkpoint`	`true` or `false`	Enables checkpointing for this Service. The default value is `false`.
`maxEngines`	integer	The maximum number of Engines that can be working on a task at a time. The default value is infinite.
`statusExpires`	`true` or `false`	Whether the status of the job in the **Services > Services > Service Session Admin** page expires. If `false`, the status must be manually removed. The default value is `true`.
`engineBlacklisting`	`true` or `false`	Whether to prevent Engines that fail at a task from taking other tasks from that Service. The default value is `false`.
`unloadNativeLibrary`	`true` or `false`	Whether to unload the native library once the Service finishes. Set the value to `false` for sharing global objects in the library. The default value is `true`.

Keyword	Argument	Description
deleteInvocationData	1 or 2	When to purge Service invocation data from display in the Administration Tool. The default value is 1 (SERVICE_COMPLETED), to purge when a Service completes. Set to 2 (SERVICE_REMOVED), to purge when a Service is removed from the Administration Tool.
maxTaskRetries	integer	The maximum number of retries allowed for any task that fails. A retry occurs if the task failed and serviceFailRetry is true, or if the task exceed the taskMaxTime and maxTaskReschedules is false. The default value is 3.
maxTaskReschedules	integer	The maximum number of redundant reschedules allowed for any task, if any of the rescheduler strategies are in effect on the Broker. The default value is 3.
rescheduleOnTimeout	true or false	How a task is dealt with if it exceeds the taskMaxTime. If true, the request is rescheduled, and the current one continues. If false, the Engine running the task is killed, and the task is retried. The default is false.
serviceFailRestart	true or false	Whether an Engine restarts itself on a Service invocation failure. The default is false.
serviceFailRetry	true or false	Whether a Service request is retried on a failure. If true, it is retried up to the maximum number of times, as set by maxTaskRetries. The default is false.
gridLibrary	string	A Grid Library that is used for this Service. The string argument specifies the name of the Grid Library.

Keyword	Argument	Description
gridLibraryVersion	string	The version of the Grid Library that is used for this Service. The string argument specifies the version of the Grid Library.

JobDescriptions

All Services have a JobDescription object created upon instantiation, with default settings. Predefined properties are stored in the database. You can define any other properties. The following JobDescriptions properties are set by default:

JobDescription Properties

Property	Description
appName	The application name.
appDescription	The application description.
deptName	A department name associated with the Service.
groupName	A group name associated with the Service.
individualName	An individual's name associated with the Service.
serviceName	The name of the Service. By default, this is the Service ID.
class	The name of the class in the Service.
serviceType	The type of Service.

The Discriminator Block

This block specifies either a job-level or task-level discriminator for the job. A discriminator without the affects clause is considered a job-level discriminator. You can declare only one job-level discriminator. The affects clause specifies conditions that must be met for the discriminator to be applied to a particular task. You can use multiple discriminator blocks with affects clauses.

Discriminator declarations consist of a property name, a comparator, and a value:

```
param-name  ==|!=|<|>|<=|>=  param-value
```

The property name refers to Engine properties. A predefined set of properties is assigned to all Engines by default. The administrator can assign additional properties to Engines at Engine Properties and Engine Property List. To see an Engine's properties, select **Engines > Engine Admin**, and from the **Actions** list, click **Engine Details**. Property names are case sensitive in PDS scripts. For example, the following discriminates against Engines with a value of less than 350 in the `cpuMHz` Engine property:

```
cpuMHz > 350
```

This example shows the format of the affects clause:

```
discriminator
    affects
        $DSTASKID        #variable
        <                #comparator
        10               #numeric or string
end
```

The variable can be any array type or built-in variable. `$DSTASKID` is the number of the current task, starting with zero. The comparator and numeric or string match against the literal to determine if the discriminator applies against this task. The example above applies a discriminator to the first ten tasks in a job.

The Schedule Block

Parameters in this block have an effect only if the job is submitted asynchronously with `bsub`. You can use these schedule declarations:

relative

```
type = relative
minuteDelay = val
```

absolute

```
type = absolute
startTime = "mm/dd/yy hh:mm AM|PM"
```

With either type, declaring `email="string"` in the Schedule block sends an email to the address given in the string when the job is complete.

Variables, Types, and Expressions

Basics

PDS has two primitive types, string, and floating-point number, with the usual notations. You need not declare variables. Variables can take on values of different types over time. To dereference a variable, precede its name with a dollar sign. Values are converted to the appropriate type depending on context. For instance, a string is converted to a number when it appears in an arithmetic expression. The grammar forbids certain combinations of expressions to catch common mistakes. Some examples:

```
a = 5.2
b = "2.5"
c = $a + $b            # succeeds, value is 7.7
d = 5.2 + "2.5"        # disallowed by the grammar
```

Scoping

Variables assigned outside of any block are global and visible to all blocks. A variable assigned within a block is visible only within that block.

Inside a block, you can assign a variable as a global variable, which is visible within other related blocks. Use the following syntax:

```
global a = 5.2
```

The following table describes what blocks are related with respect to scope:

Global variable set in	Effective in
outside of the lifecycle blocks	all blocks
the `prejob` block	all blocks
the `pretask` block	pretask, task and posttask
the `task` block	task and posttask
the `posttask` and `postjob` blocks	that block

Note that assigning a variable with the same name as a previously defined global variable does not change the value of the global variable. Instead, it creates a new local variable in the block, and that local variable has local scope and does not change the value of the global variable.

Variable Substitution

Variable references are expanded within quoted strings in all contexts. For example, after

```
a = "foo"
b = "$a fighters"
```

the value of `b` is `"foo fighters"`. Use curly braces to separate a variable name from the adjacent non-whitespace text:

```
b = "${a}bar"            # b contains "foobar"
```

Inside quotation marks, represent a quote by escaping it with another quotation mark. Also, inside quotation marks, represent the dollar sign character by escaping it with another dollar sign. For example:

```
a = """hello"""             # a contains "hello"
b = "$$100"            # b contains $100
```

The backquote can also be used to assign the output of a shell command to a variable. For example:

```
datetime = ‘date‘
```

Expressions

PDS supports the usual arithmetic operators (`+`, `-`, `*`, `/`) with standard precedence. All arithmetic operations use double-precision floating-point values.

PDS also supports the standard C-style comparison operators (`==`, `!=`, `>`, `<`, `>=`, `<=`). These operators perform numeric comparison if both arguments are valid numbers; otherwise, they perform string comparison. They evaluate to zero if the comparison is false and to a non-zero value if it is true.

PDS does not support the relational operators `and`, `or`, and `not`.

Backquote expressions

A string enclosed in backquotes (‘`such as this`‘) has variable substitution performed on it, and the result is evaluated in a subshell. The standard output of the command is collected, newlines and linefeed characters are replaced by spaces, and trailing whitespace is removed. The result is the value of the expression.

Arrays

Arrays are fundamental to achieving parametric parallelism in PDS, because an array variable is implicitly indexed in the task block.

Construction

You can construct arrays of primitive values in several ways. Write a literal value as follows:

```
a = [1, 2, 3, 4, 5]
```

You can construct an array with autorange expressions. The expression

```
begin n end m step k
```

constructs an array starting with *n* and proceeding in increments of *k* until *m* is reached. More precisely, it constructs

```
[n, n+k, n+2k, ..., n+rk]
```

where *r* is the largest integer such that *n+rk*<= *m*.

The expression

```
begin n count c step k
```

constructs an array of *c* elements beginning with *n* and proceeding in increments of *k*, that is,

```
[n, n+k, n+2k, ..., n+(c-1)k].
```

For example,

```
begin 10 end 15 step 2
```

and

```
begin 10 count 3 step 2
```

both construct the array

```
[10, 12, 14].
```

The third way to construct an array is to use the `split` function, which divides a string into array elements at whitespace. Quoted elements keep embedded whitespace and strip the quotes upon placement into the array. For example, on UNIX machines:

```
split(‘ls‘)
```

returns an array of the files in the current directory.

Indexing

In most contexts, when a variable containing an array is referenced, the first element is returned. However, in the task block, the element corresponding to the ID of the currently

running task is returned. (If the task ID exceeds the array size, the last element of the array is returned.) This feature makes it easy to write the most common kinds of distributed computations. For example, you can set up an array of values and run a command on each one in parallel with the following PDS script:

```
args = begin 100 end 200 step 5
task sizeof($args)
    execute "doit $args"
end
```

The `$args` in the execute statement expands to 100 for the first task, 105 for the second, and so on.

Another exception to the first element being returned from an array is that inside a `for` loop, the variable containing the array returns the value at the current iteration of the loop. This is redundant with the loop variable. For example, in `for i in $args log "$i" log "$arg" end`, `$i` and `$arg` are the same every time; the loop variable `$i` is there only for convenience.

Explicit array indexing is unsupported: To obtain an array element other than the first, use one of the following:

- Implicit indexing in the task block
- The `for` statement

Even within those contexts, assignment to the variable holding the array changes the entire variable value, not the current element.

Other Features

An array that includes both string and numeric values is legal. Arrays of arrays are not allowed.

You can use the `sizeof` function to determine the number of elements in an array, as shown in the argument to the `task` block in the above example.

You can use the `for` statement to iterate over the elements of an array. See The For and Foreach Statement, for more information.

Built-in Variables

In addition to user-defined parameters, you can use the following built-in variables:

Built-in Variables

Variable	Description
DSTASKID	ID of the current task (meaningless if not in task block)
DSJOBID	ID of the current job. If submitted using the bsub switch, this indicates the batch ID.
DSWORKDIR	The temporary directory for job files. This is an Engine-side variable. This directory is in the Engine installation directory, typically ./work/*machinename*-*i*/tmp/*session-ID*, where machinename is the name of the Engine host; *i* is the instance, starting with 0; and *session-ID* is the Service Session ID for the PDriver Job. The Engine periodically deletes the tmp directory. To set the frequency of deletion, set **Temp File Time-to-Live (hours)** in each Engine Configuration.
DSENGINEHOME	The Engine home. This is an Engine-side variable.
DSSTAGEDIR	The Alias for the Manager staging directory. You can use this as a source or destination directory for the copy command. Files in this directory are automatically deleted periodically.
DSOS	The OS of the current system (for example, “linux”, “win32”, “plinux”)

It is important to put any temporary files in the DSWORKDIR directory. If you put them elsewhere and a task is interrupted, the files stay on the computer and are never automatically removed.

You can use automatic variables such as $1, $2, $3, and so on to reference arguments to the PDriver command that follow the script file name. You can use the automatic variable $* to reference these automatic variables collectively as a string. To create an array of all command-line arguments, use split($*).

Arguments to the PDriver command that follow the script file name be referenced with the automatic variables $1, $2, $3 ... These variables also be referenced collectively as a string using the automatic variable $*. To create an array of all command-line arguments, use split($*).

$? contains the execution status of the last execute command.

Statements

Built-in Commands

You can use the following built-in commands inside any lifecycle block (with the exception of `onerror` and `throw`). These commands are operating-system independent.

Built-in Commands

Command	Description
`mkdir "dir-name"`	Creates a directory.
`copy "src-file" "dest-file"`	Copies a file. Typically, the prejob block is used to copy input files from the Driver machine to the staging directory on the Manager, which can be referenced as `$DSSTAGEDIR`. In the task or pretask block, the input file is copied from the staging directory to the Engine's local filesystem. Output files are copied in the reverse direction.
`rmdir "dir-name"`	Removes a directory. The directory must be empty.
`delete "file"`	Deletes a file. Supports the `*` wildcard.
`log "log-message"`	Writes a message to the Driver log (in prejob or postjob lifecycle blocks) or the Engine log (in pretask, task, or posttask lifecycle blocks).
`execute [stdin="stdin=file"] [stdout="stdout-file"] [stderr="stderr-file"] "command-to-execute"`	Executes a command on the local machine using the shell specified with the `shell` command. A subshell is spawned for the command unless specified otherwise with the `shell` command. The subshell has the path available on the Engine (as set in the **Library Path** property of the Engine Configuration) or Driver. Before running custom executables, deploy those executables to Engines using GridServer's application resource deployment feature. See Resource Deployment for more details.

Command	Description
`shell "shell-type"`	Specifies the shell to spawn for commands run by the `execute` command. The defaults are `/bin/bash` for Linux and `cmd.exe` for Windows. If you set this to `none`, no shell is spawned.
`env (variable)`	Returns the value of the specified environment variable. If the variable is a string literal, enclose it in quotes. Returns an empty string for nonexistent variables.
`onerror <ignore|retry|fail>`	Indicates what happens when an `execute` command returns with a nonzero exit code. `ignore` takes no action; `retry` reschedules the task on another Engine; and `fail` returns an exception back to the Driver. The default is `fail`. When used with `retry`, you can use `onerror` only within the `job` or `task` blocks.
`throw "message"`	Causes the task to fail and throws an exception. The message is displayed on the Driver and written into the Driver log. This cannot be used in the posttask lifecycle block.

Enclose arguments to all statements in quotes, except for arguments to `onerror`. If you include a linebreak in a quoted argument, escape it with a backslash.

The `mkdir`, `copy`, `delete`, and `rmdir` commands are not dependent on the operating system. Pathnames automatically translate to work on the appropriate platform. For example, `mkdir "sample/log1"` creates `sample\log1` on Windows systems.

The If Statement

The syntax of the if statement is

```
if expression comparisonOperator expression then
    statements
end
```

PDS also supports `elsif` and `else` clauses. You can use the `if` statement inside a lifecycle block (`prejob`, `pretask`, `task`, `posttask`, `postjob`) to conditionally execute statements. One typical use is to execute different commands depending on the Engine's operating system:

```
if $DSOS == "win32" then
    cmd = "dir"
else
    cmd = "ls"
end
execute "$cmd"
```

An `if` statement can also appear at the top level of a PDS script, to include or exclude global assignments or entire blocks. An example is

```
if $1 == "alwaysCancel" then
    options
        autoCancelMode "always"
    end
end
```

The `if` statement is not legal in the `options`, `schedule`, or `discriminator` blocks.

The For and Foreach Statement

Explicit looping over an array is not often needed. If you need explicit looping, use the `for` statement:

```
for variable in expression
    statements
end
```

The expression must evaluate to an array. The variable takes on successive elements of the array on each iteration of the loop. Assignment to the loop variable is legal but has an effect only for the remainder of that iteration. It does not alter the array.

Inside a `for` loop, the variable containing the array returns the value at the current iteration of the loop. This is redundant with the loop variable. For example, in

```
for i in $args  log "$i" log "$args" end
```

the `$i` and `$args` are both equal for each iteration. The loop variable `$i` is there only for convenience.

The `foreach` statement lets you implicitly index an array, without the loop variable:

```
foreach expression
    statements
end
```

Shell Directives in Heterogeneous Environments

It is the PDS script writer's responsibility to declare a shell directive that is appropriate to the executing node's platform. For jobs in heterogeneous environments, you can specify different directives within an if-then-else block. For example,

```
if $DSOS == "win32" then shell "cmd.exe" else shell "/bin/bash" end
```

You can specify the argument "none", in which case all following `execute` tasks spawn directly and not in a subshell.

It is also possible to use the bash shell with Windows by utilizing Cygwin, the UNIX environment for Windows.

To use bash and Cygwin on Windows machines:

1. Install Cygwin on another machine. You can get it at `www.cygwin.com`.
2. From the Cygwin installation, copy `bash.exe` and `cygwin1.dll` into the Engine's path. This is either in a directory within the `%PATH%` on the Engine or the replication directory `<engine_home>/resources/win32/bin`. Also, copy any other Cygwin executables you use from the bash shell.
3. If bash is already in the path for the Engine, use the following syntax in the PDS script:

   ```
   task
       shell "bash.exe"
       ...
   end
   ```

   or

   ```
   task
       shell "none"
       execute stdout="outfile"
             "<optional_path>/bash.exe -c ""mycommand.exe"""
   end
   ```

If the path to bash is not set, use the following syntax:

```
task
    shell "c:\cygwin\bin\bash"
    execute stdout="outfile" "echo hello"
end
```

The above example also requires that you copy the Cygwin `echo.exe` to the Engine.

Example

`pi.pds` performs a distributed Monte Carlo calculation of the value of pi. A `picalc` command-line executable resides in the `bin/`*platform* directory of the distribution. The executable is also available on all Engines in your GridServer installation. Running PDriver distributes calculation of Pi across the network, using autoranged parameters to generate differing seed values for the program's pseudorandom number generator. The postjob block runs a script that derives an average of all the values returned.

Creating Grid Libraries

This section provides information about creating Grid Libraries, which are versioned sets of resources that might be used by multiple Services.

Overview

A *Grid Library* provides a solution to the problem of managing versioned sets of resources that might be used by multiple Services. A Grid Library is a set of resources and properties necessary to run a Grid Service, along with configuration information that describes to the GridServer environment how those resources are to be used. For example, a Grid Library can contain JARs, native libraries, configuration files, environment variables, hooks, and other resources.

Grid Libraries are identified by **name** and **version**. All Grid Libraries must have a name and typically have a version. The version is used to detect conflicts between a required library and a library that has already been loaded; it also provides for automatic selection of the latest version of a library. A GridServer Service can specify that it is implemented by a particular Grid Library by specifying the `gridLibrary` and `gridLibraryVersion` Service Options or Service Type Registry Options.

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, but only the appropriate set of package options is applied at run-time.

Grid Library Format

The Grid Library can be an archive file in ZIP (`.zip`) or gzipped TAR format (`.tgz` or `.tar.gz`), with a `grid-library.xml` file in the root. It might also contain any number of directories that contain resources. Although the filename has no inherent meaning, the following format is strongly encouraged:

```
library_name-library_version.[zip|tar.gz|tgz]
```

If a Grid Library has the `OS` attribute set, the following format must be used:

```
library_name-library_version-os.[zip|tar.gz|tgz]
```

To reduce Engine restarts when you deploy a new version, always include the version in the filename, as overwriting existing libraries requires a restart, whereas a new library does not.

The directory structure is completely up to you since the configuration file is used to specify where resources are found within the Grid Library.

The configuration file must be a well-formed XML file named `grid-library.xml`, and be in the root of the Grid Library.

The GridServer SDKs include a `grid-library.dtd` file that can be used to validate the XML file. They also include an example Apache Ant `build.xml` file that can be used to validate and build Grid Libraries. This DTD can also be found in The grid-library.dtd.

Following is a table that specifies all elements and attributes of the `grid-library.dtd` file. It uses the XML schema notation for elements and attributes, such as:

[no tag] (Required)

? (Optional)

* (Optional and Repeatable)

Grid Library DTD Elements and Attributes

Element	Description	Elements and Attributes
`grid-library`	The root element. Note that attributes affect deployment; Engines only download Grid Libraries whose attributes match their own properties and ignore those with non-matching attributes. If no attributes are specified in this element for a particular Grid Library, all Engines download this Grid Library.	**ELEMENTS** • `grid-library-name` • `grid-library-version?` • `arguments?` • `dependency*`

Element	Description	Elements and Attributes
		• conflict* • jar-path* • lib-path* • assembly-path* • command-path* • hooks-path* • environment-variables* • java-system-properties*
		ATTRIBUTES • jre • bridge? • super? • os? • compiler?
grid-library-name	The library name. All libraries must be named.	
grid-library-version	The version. If not specified, 0 is implied. If in comparable format as defined below, it can be used to determine the latest version.	
dependency	A library dependency. If the version is not specified, the latest version is chosen at runtime.	**ELEMENTS** • grid-library-name* • grid-library-version?

Element	Description	Elements and Attributes
		ELEMENTS • os? • compiler?
conflict	Indicates that this library conflicts with the given library. If this Grid Library is NOT a dependency, and grid-library-name="*", then it indicates that this Grid Library conflicts with all other Grid Libraries (aside from its dependencies).	**ELEMENTS** • grid-library-name*
pathelement	An element containing a relative path, typically set to a directory. This element must be in the proper format for the OS.	
jar-path	The JAR path. If specified, all JARs and classes in the path are loaded.	**ELEMENTS** • pathelement* **ATTRIBUTES** • os? • compiler?
lib-path	The native library search path.	**ELEMENTS** • pathelement* **ATTRIBUTES** • os? • compiler?
assembly-path	The .NET assembly search path. Absolute assembly paths, mapped drives, and UNC paths do not work.	**ELEMENTS** • pathelement* • os?

Element	Description	Elements and Attributes
		• compiler?
command-path	The path in which the Engine searches for Command Service executables.	**ELEMENTS** • pathelement* **ATTRIBUTES** • os? • compiler?
hooks-path	Path to Engine Hook XML files. Engine Hooks are initialized at the time the containing Grid Library is loaded.	**ELEMENTS** • pathelement* **ATTRIBUTES** • os? • compiler?
name	The name of a property	
value	The value of a property	
property	A name/value pair, used by environment variables and Java System properties.	**ELEMENTS** • name, value
environment-variables	Environment variables to set.	**ELEMENTS** • property **ATTRIBUTES** • os? • compiler?
java-system-properties	Java system properties, which are set immediately prior to executing a task using this library.	**ELEMENTS** • property

Element	Description	Elements and Attributes
		ATTRIBUTES • os? • compiler?

The following is a list of attributes used above. Valid values can be found in the **About** page in the Administration Tool:

Grid Library Attributes

Attribute	Description
os	The os attribute specifies that it is only applied to this OS. If the attribute is not this operating system (OS), the containing element and its children and content are ignored.
compiler	If the attribute is not this compiler, the containing element and its children and content are ignored.
super	If set to true, the Grid Library is a Super Grid Library, meaning, it is always loaded when the Engine starts up. See Super Grid Libraries for more information.

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 pathelements. 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 `\`.

The implicit property `ds.GridLibraryRoot` can be used in the `grid-library.xml` file. This is useful for including a `driver.properties` file in a Grid Library, and setting the `DSDRIVER_DIR` to the location of the file in the library. This can be done in a C++ or .NET Grid Library as follows:

```
<environment-variables>
<property >
<name>DSDRIVER_DIR</name>
<value>$ds.GridLibraryRoot$</value>
</property>
</environment-variables>
```

The following would be done in a Java Grid Library:

```
<jar-path>
    <pathelement>$ds.GridLibraryRoot$</pathelement>
</jar-path>
```

Versioning

Versioning provides the following functionality:

- It enables deployment of new versions of libraries and deletion of old versions without interrupting currently executing Service Sessions.
- It provides for specifying conflicts, or libraries that cannot coexist with each other.
- It enables a Service Session or dependency to specify the use of the latest version of a Grid Library.
- It enables easy management of multiple applications needing different versions of the same set of libraries.

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 must be a String that follows the proper comparable version format. It can also be used to determine the latest version of the library, for automatic loading. 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 use 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.

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 subsequently 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 was set to use the latest version of a Grid Library, all Engines work on the latest version at the time the Service was started, regardless of whether a newer library has been deployed. This can be changed by setting the `GRID_LIBRARY_STRICT_VERSIONING` Driver option to false. When false, if a newer version of the library is deployed while the Service is running, Engines that have not yet worked on the Service use the newer version, while Engines that worked on it prior to 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 `<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 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 via 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 dynamic versioning is used, and 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.

Grid 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. 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 jars 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 pathelements they contain are added to a ‘master’ list of paths for that type of pathelement. 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 deployed on the grid to ensure that the ordering of library paths does not lead to loading the wrong library. For example, if two different Grid Libraries each provide DLLs in their `lib-path`s that share the same name, because of OS-specific library load 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 such conflicts arise or to use the `conflict` element to explicitly state conflicts.

Grid Library and RunAs State information persists on normal Engine shutdowns, which 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 upon 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 reenabled.

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 upon login. The timeout for this is configurable 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, and 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 would typically be set by Service Type in the **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.

C++ Bridges

C++ Bridges are the native bridges that allow Engines to execute native Services. They are packaged as Grid Libraries, named `cppbridge-[os]-[compiler]-[M]-[m]`, where `M` and `m` are the GridServer major and minor version numbers. All C++ Bridges are pre-packaged and deployed in the Grid Library replication directory upon GridServer Manager installation or upgrade.

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

would then declare 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, `1.8.0.331`. The os is the platform, such as `linux64`, `win64`, `linux`, or `win32`.

For example, for linux 64: `jre-linux64-1.8.0.331.tar.gz`.

To package a JRE Grid Library, create a Grid Library that contains the JRE in a directory in the root of the Grid Library. Then, set the `jre` attribute in the `grid-library` element. For example, `<grid-library ... jre="true" />`

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, just 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.

For an example of how to package a JRE Grid Library, see the "Deploying Services" section of the *TIBCO GridServer® Administration*.

R Grid Libraries

Grid libraries with Services implemented in R must contain a top-level directory named `R`. This means the following Grid Library structure is used:

```
/
/grid-library.xml
/R/
```

The `R` directory is required unless the Grid Library contains only R packages. All files ending in `.R` are sourced. The order that the R files are sourced is not guaranteed.

Grid libraries with R Services must depend on the `rbridge-`*platform* Grid Library. That dependency can be direct or indirect.

Grid Libraries can also contain R packages in non-source form. Packages root directory within the Grid Library is defined using the `lib-path` element.

The following is an example `grid-library.xml` with R packages:

```
<?xml version="1.0" encoding="UTF-8"?>
<grid-library os="win64">
    <grid-library-name>Rpscl-win64</grid-library-name>
    <grid-library-version>3.0.3</grid-library-version>
    <lib-path>
        <pathelement>library</pathelement>
    </lib-path>
    <dependency os="win64">
        <grid-library-name>rbridge-win64</grid-library-name>
    </dependency>
    <dependency os="win64">
        <grid-library-name>Rextras-win64</grid-library-name>
    </dependency>
</grid-library>
```

Building TERR Runtime Grid Libraries

A tool to create a TERR runtime Grid Library is provided in the SDK's `$SDK_HOME/tools/bin` directory.

The `createTERRGL.bat` and `createTERRGL.sh` scripts are a wrapper for the Java implementation. Therefore, Windows and Linux versions take the same parameters:

Option	Description
`--terrHome=${TERR_HOME}`	Specify the `TERR_HOME` directory. Required.
`--version=${gl version}`	Override the version information deduced from `${TERR_HOME}/bin/TERR --version`.

Option	Description
`--32|--64`	Generate 32-bit or 64-bit Grid Library. By default, Windows generates a 32-bit runtime, and Linux generates a 64-bit runtime.
`--complete`	Generate Grid Library with all files. By default, smaller Grid Libraries are generated without some unused files.
`--no-build-info`	Generate Grid Library without build information. By default, build information is in the Grid Library descriptor.
`--verbose`	Generate more output.
`--help`	Show a list of all options.

Python Bridges

Python Bridges are the native bridges that enable Engines to execute Python scripts using the packaged Python runtime. They are packaged as Grid Libraries, named `pybridge-`*os*`-`*compiler*`-`*M*`.`*m*, where *M* and *m* are the GridServer major and minor version numbers. All Python Bridges are pre-packaged and deployed in the Grid Library replication directory upon GridServer Manager installation or upgrade.

Python Grid Libraries

Grid Libraries with Services implemented in Python must contain a top-level folder where the individual `.py` files are stored. In the example below, this folder is named `commands`. In the `grid-library.xml`, you must reference this folder in a `lib-path` element.

The Grid Library must either contain the binary ("pybridge-win64-vc14" for Windows or "pybridge-linux64-gcc34" for Linux) or reference a dependent grid-library containing it.

Lastly, it is necessary to set an environment variable `ds.PYTHON_USER_FILE_PATH` which references the folder containing the `.py` files set above. The value of this environment variable must be `$ds.GridLibraryRoot$/`*folder-name* where *folder-name* is the folder containing the `.py` files.

Here is an example of a `grid-library.xml` for Python:

```
<?xml version="1.0" encoding="UTF-8"?>
<grid-library>
    <grid-library-name>Python-example</grid-library-name>
    <grid-library-version>1.0.0</grid-library-version>
       <lib-path>
        <pathelement>commands</pathelement>
    </lib-path>
    <dependency os="win64">
      <grid-library-name>pybridge-win64-vc14</grid-library-name>
      <grid-library-version>7.1</grid-library-version>
   </dependency>
   <dependency os="linux64">
      <grid-library-name>pybridge-linux64-gcc34</grid-library-name>
      <grid-library-version>7.1</grid-library-version>
   </dependency>
   <environment-variables>
    <property>
      <name>ds.PYTHON_USER_FILE_PATH</name>
      <value>$ds.GridLibraryRoot$/commands</value>
      </property>
   </environment-variables>

</grid-library>
```

Windows Application Deployment

The Windows Deployment Scripting Language provides a mechanism by which programs can be executed in conjunction with file updating on Windows Engines. This can be used for such purposes as registering COM DLLs and .NET assemblies, running Microsoft Installer packages, and so on. It runs an installation command when the script is added, and when any dependent files are modified. It can also run an uninstallation command when the script is removed.

A deployment script is a file named `dsinstall.conf`. This is a reserved filename, and the Engine Daemon interprets any file with this name as a deployment script. The script is a properties file, with name and value pairs that govern the command execution.

The script is placed, with associated files, in its own subdirectory of a Grid Library. This is referred to as the installation directory. In situations where a remote application installation must take place when the Engine is first started, it can be packaged in a Super Grid Library.

The following properties are provided:

Deployment Script Properties

Property	Description
`install_cmd`	The installation command. The command must be in the current directory; you can also specify the full path to a command. This command is run when the `dsinstall.conf` file is added, modified, and when any dependency is modified. Note that the Engine's `PATH` environment variable is not set within the installation command. You must set the path to use some commands, such as `ping` or `find`.
`workdir`	Working directory from which the commands are launched. The directory is relative to the installation directory.
`uninstall_cmd`	Optional. The uninstall command. This is executed when the script is deleted, or prior to subsequent runs of the install command if `uninstall_first` is true. Supporting files for the uninstall script might be deleted along with the script; the command is executed prior to local deletion of the files. Typically an uninstall is performed by simply removing the entire installation directory. Note that the path is not set within the uninstall command. You must set the path to use some commands, such as `ping` or `find`.
`waittime`	Number of seconds to wait for the install/uninstall command to finish. The default is 30 seconds. If this time is exceeded, the process running the command is killed.
`uninstall_first`	Optional. If true, the uninstall command always runs prior to the install command, except for the first time the install command is run. This is for situations in which you need to uninstall software prior to reinstallation.
`success_exit_codes`	Optional. Comma-delimited list of exit code values that indicate successful command execution. If the exit code does not match any value, an error logs with the failure code, and the next time the Daemon restarts it retries the installation. If this property is not set, exit codes are ignored.
`disable_on_fail`	If an Engine Daemon disables itself on the failure of an install. The default is false if not specified in the conf file. When the value is true, the Engine

Property	Description
	Daemon disables itself if the installation returned exit code is not in the success exit codes.

The `:` and `\` characters must be escaped with a backslash (`\`) character in the `dsinstall.conf` file. Also, you must not rename the `dsinstall.conf` file.

The following code is an example of a script that installs a Microsoft Installer package:

```
workdir=.
waittime=30
uninstall_first=true
install_cmd=install.bat
uninstall_cmd=uninstall.bat
success_exit_codes=0
install.bat:
%SystemRoot%\system32\msiexec /q /i mypackage.msi ALLUSERS=1
uninstall.bat:
%SystemRoot%\system32\msiexec /q /x mypackage.msi ALLUSERS=1
```

These three files, plus the `mypackage.msi` file, are all placed in a Grid Library. Note that the `uninstall_first` property is used to uninstall the previous version of the software whenever the package is changed. To uninstall the software, simply undeploy the Grid Library; the uninstallation is performed prior to deleting the files.

Grid Library Example

The following example `grid-library.xml` is for a mixed Java/C++ application that runs on Windows, and both `gcc` and `gcc34` for Linux:

```
<?xml version="1.0" encoding="UTF-8"?>
<grid-library>
    <grid-library-name>MyLib</grid-library-name>
    <grid-library-version>1.0.0.1</grid-library-version>
       <!-- Example of how to use both gcc and gcc34 libraries -->
    <lib-path os="linux" compiler="gcc">
        <pathelement>lib/gcc</pathelement>
    </lib-path>
    <lib-path os="linux" compiler="gcc34">
        <pathelement>lib/gcc34</pathelement>
```

```
    </lib-path>
<!-- All three C++ bridges are included here -->
    <dependency>
        <grid-library-name>cppbridge-vc14</grid-library-name>
    </dependency>
    <dependency>
        <grid-library-name>cppbridge-gcc</grid-library-name>
    </dependency>
    <dependency>
        <grid-library-name>cppbridge-gcc34</grid-library-name>
    </dependency>
<!-- Specifies that win32 use this JRE Grid Library, others use default
-->
    <dependency>
        <grid-library-name>jre-win32</grid-library-name>
        <grid-library-version>1.8.0.161</grid-library-version>
    </dependency>
<!-- Specifies JVM options in a JRE Grid Library -->
    <arguments>
        <property>
            <name>-Xdebug</name>
        </property>
        <property>
            <name>-Xmx512m</name>
        </property>
        <property>
            <name>-Dfoo</name>
            <value>bar</value>
        </property>
    </arguments>
<!-- Example of linking to another of my Grid Libraries-->
    <dependency>
        <grid-library-name>MyCalculator</grid-library-name>
    </dependency>
    <hooks-path>
        <pathelement>hooks</pathelement>
    </hooks-path>
<!-- Example of multiple jar paths -->
    <jar-path>
        <pathelement>jars</pathelement>
        <pathelement>morejars</pathelement>
    </jar-path>
<!-- Example of a lib path with absolute dir -->
    <lib-path os="win32">
        <pathelement>lib\win</pathelement>
    </lib-path>
<!-- Example of OS-dependent env vars, using a property sub -->
    <environment-variables os="win32">
```

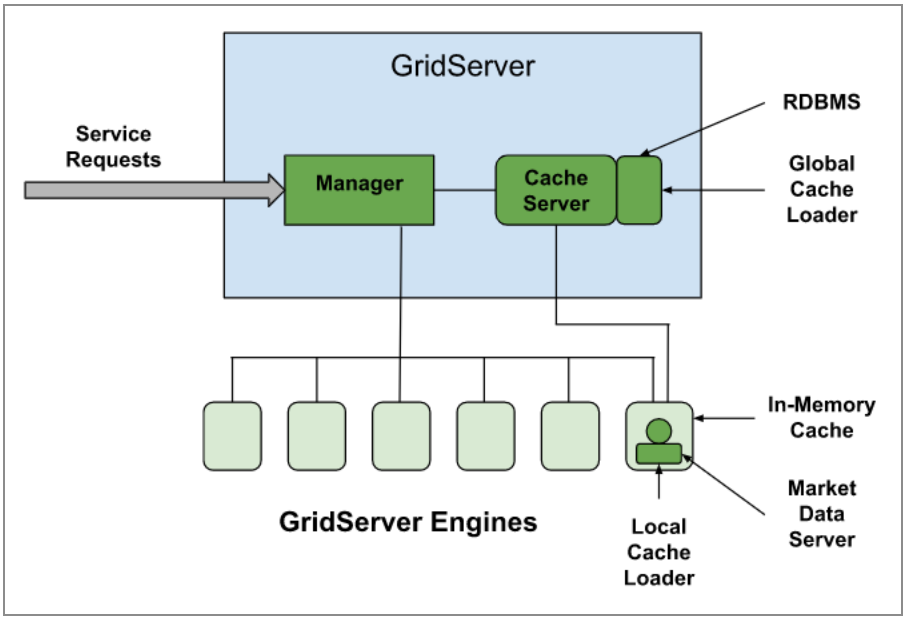
```
        <property >
            <name>MY_WIN_VAR</name>
            <value>$WinVar$</value>
        </property>
    </environment-variables>
    <environment-variables os="linux" compiler="gcc">
    <property >
            <name>MY_GCC_VAR</name>
            <value>$LinuxDriverDir$</value>
        </property>
    </environment-variables>
    <java-system-properties>
        <property>
            <name>foo</name>
            <value>bar</value>
        </property>
    </java-system-properties>
</grid-library>
```

GridCache

This section provides information about using GridCache, a dynamically updateable distributed object cache that any GridServer Driver or Engine can use to store data for later retrieval by other GridServer components.

Overview

GridCache is a dynamically updateable distributed object cache that any GridServer Driver or Engine can use to store data for later retrieval by other GridServer components. While GridServer extensively uses object caches internally, applications can access the GridCache object cache directly through an API, to reduce the load on backend datastores and decrease access time to data. GridCache is similar to the proposed JSR-107 JCache, with a similar interface and a subset of features.



A typical GridCache implementation

GridCache meets the requirements of many informational market data systems, where a consistent view of object state is extremely important but it is not necessary to guarantee that all participants process every individual state change (for instance, every individual quote move) as a transaction.

General Capabilities

GridCache is a general distributed cache that provides a consistent view of data to all clients (Drivers and Engines) in the Grid. Data resides in unique *regions* of the cache. Data can be serializable Java objects, .NET objects, strings, and byte arrays for C++. The global cache of data can be arbitrarily large, limited only by the amount of disk space on the Manager. Each component locally caches only the data requested by users on that component. The local cache of each client, designed to speed up access to frequently used data, is in-memory with the option in Java Drivers and Engines to spool to disk. The size of the local cache is configurable through the Engine distribution configuration or the Driver properties file.

API

You can access GridCache through a client API available on Drivers and Engines. The API follows the JCache specification where appropriate. The API is available in Java, .NET, and C++, and provides cross-platform access to data where appropriate. That is, C++ and Java applications can share XML documents (strings), but have little use for sharing Java or .NET objects. You can also use Data References across different platforms to support streaming very large objects. See Data References for more information.

Modes

GridCache operates in one of the following modes:

- **Local** This mode lets you cache data locally by putting elements into the cache. This mode does not synchronize clients that are accessing local cache regions with the same name. This is similar to having a local hashtable with LRU and eviction based on time-to-live from creation time.
- **Local with loader** This mode lets you load data into the local cache using a loader specified at create time. Puts (cache writes) are not allowed in this mode. Users can manually synchronize clients' local caches using clear and invalidate methods.
- **Global** Data that users put into the cache is then available globally. Full automatic synchronization occurs in this mode. All components have access to a synchronized view of all entries.

- **Global with loader** Users can use a global loader to load data into the cache. Full automatic synchronization occurs in this mode.

Cache Configuration and Access

To configure cache regions, use the GridServer Administration Tool at **Services > Services > Cache Regions**. Define the region names or regular expressions with a set of attributes. The `getRegion` method in `CacheFactory` provides access to the region if it already exists or creates the region with the mapped attributes. A region name throws an exception when it matches multiple regular expressions from different schemas. A second Cache access method is available and takes the schema name to provide a dynamic mapping of regions to attribute schemas.

When you configure Local and Global loaders, the class name of the loader and the type are arguments to the constructor. Users can define bean properties for loaders. Loaders are available only in Java and .NET, but can be used by the C++ API. JNI, or managed C++ can be used to implement loaders to access native resources. Each schema that requires a loader defines the loader within the configuration page.

Changes to the cache configuration take effect the next time you create regions with that cache configuration. Pre-existing regions that require configuration changes must be manually destroyed and recreated.

Data Storage

All data put to a global cache is stored in a persistent backend datastore on the Broker’s file system. You can limit the size of the Broker’s file cache. The default is that there is no limit. If the Broker’s file cache is size limited and full, the Broker silently removes data to make room for the incoming data. If the cache entry is too big, it first goes to the memory cache, and if it cannot fit there it goes to the disk cache. If it is too big for the disk cache too, then it is dropped. There is no global resilience when using a memory-based cache; however, you can use a loader for that purpose.

Attributes

Define the GridCache Configuration Attributes in schemas. Then, apply the attributes to newly created regions. You can define the following types of attributes:

- **TimeToLive** Regions can define a time-to-live attribute. Data that is stored in the cache for longer than the time-to-live attribute is evicted, and you must reload that data from the backend datastore. Data in local caches is evicted locally. Data for distributed regions is evicted from the distributed cache, and the Broker and all clients delete that data if they have cached it locally.
- **Local/GlobalLoader** Loads data from a backend datastore. See Cache Loaders, below.
- **KeepAlive** Specifies how long the client keeps the region and its keys in its local cache after the last reference to the region goes away.

Consistency/Synchronization

The cache synchronizes by propagating update notifications to all clients listening on a region. These notifications occur any time the region changes. Specifically, they occur on a put (when an element already exists), clear, remove, or invalidate. This applies to different region types differently:

- **Engines** GridCache guarantees that all Engines receive all update notifications by the time they take the next task or Service request.
- **Drivers** There are no synchronization guarantees for the Driver. The Driver receives notification messages the next time it performs a request, polls the server for results, or sends a heartbeat.

Cache Loaders

Loaders provide an optional mechanism for loading data into the cache from a backend datastore, such as a relational database. Users can implement and associate Cache Loaders with a region of the cache. These Cache Loaders can be installed locally on the client (Driver or Engine) or globally, in the GridServer Broker.

Global

Use a Global Cache Loader for synchronized regions from which all clients can access data.

- Global loaders are defined and configured in the schemas.

- When other clients get access to that region, they automatically are using that loader.
- A client can specifically pre-load data into the cache by explicitly calling the load method with a single key or a list of keys.
- If a get does not find data, the loader then attempts to load for that key by calling the loader's load method.
- Puts are not allowed on regions with loaders.
- Global loaders are written in Java, but can be bridged to native or .NET code through JNI.
- Global CacheLoader JAR files are deployed to the `lib` directory in the cache directory. By default, the cache directory is `DS_DATA/cache`. Configure the cache directory in the **Cache** section of the **Manager Configuration** page in the Administration Tool.

Local

Use a local loader to cache data locally from an external database. Clients do not share local loaders.

- Puts (writes) are not allowed, as it is a local cache, and data is not propagated to other clients or regions.
- Removing an item is not allowed. Instead, invalidate the item. Invalidating causes the item to be removed from other clients' caches.
- Local loaders can only be .NET or Java. You can adapt CPP loaders through JNI or managed C++.

Cache Loader Write-through and Bulk Operations

To simplify using backend datastores with GridCache, it's sometimes desirable to add a means for supporting the store and remove methods on the loader rather than just supporting it purely as a data fetching mechanism. It's also sometimes desirable to be able to use such methods in a batch form, such that a cache can be "primed" with entries through a bulk load method, or that multiple objects can be stored, removed, and invalidated with a single method to cut down on per-store operation overhead.

Loaders can inherit from two interfaces that support methods for supporting the store and remove methods on the loader:

- The preload methods are exposed in the `BulkCacheLoader` interface.
- The store, remove, and clear methods are exposed in another interface, `CacheStore`, and can be used by the underlying cache mechanism for modifying the content of cache puts and removals on the backend datastore. The cache mechanism can then invalidate the corresponding entry or entries for all other caches listening on that region, if applicable.

It is possible that the backend store gets updated without sending an invalidation message to all other clients. If this scenario is detected, it throws an exception indicating a loss of synchronization, but the cache client must handle recovery from that point on.

The caching system in GridServer does not provide a mechanism to auto-update data in the cache when it changes in the backend, if done so by a mechanism other than those offered by the CacheStore interface.

Support is available for datastore write-through, bulk write-through, remove, bulk remove and bulk load, on both global and local loaders, in Java and .NET.

Notification

GridCache provides an optional mechanism whereby you can implement a class that listens for update notifications. An update is defined to be either an invalidation call on a loaded object or on a put call on a key that exists in the cache already. You can then take any action desired such as updating local copies of the object or data to the new version or ignoring the update completely. The next time that the data is requested from the cache, GridCache fetches and locally caches the most current version of that data.

Disk/Memory Caching

When the cache is full, cache puts (or writes) push the oldest element out of the cache. Elements are then put into the backing disk cache or removed entirely. This makes the caches LRU caches. You can configure the size of the local cache and the size of the backing disk cache:

- In the `driver.properties` file, for Drivers.
- In the Engine configuration, for Engines.

If you configure disk caching, then any puts into the memory cache when the memory cache is full force the oldest element out of the memory cache into the disk cache. Any access to a cache element that has to get the element from the disk cache brings the element into the memory cache. CPP Drivers do not have disk-backed cache.

Cache Region Scope

Global cache regions exist until they are destroyed through the destroy method regardless of whether any client has a reference to that region. Unnecessary global cache regions impact eviction performance, so it is important to destroy global regions when they are no longer needed.

After all references to a cache region on a client go out of scope, local cache regions persist on clients until their keepalive timeout. At that point, the region is swept from the cache. Use the `close()` method to explicitly release a reference to a region. If you do not use the close method, garbage collection handles decrementing references to the region. However, garbage collection is never guaranteed so the keepalive timeout is not a guaranteed timeout. Using the close method is recommended.

Data Conversion Matrix

The following table outlines the results of putting a data type into GridCache and then getting it with a different Driver.

Input	.NET Output	Java Output	C++ Output
String	String	String	std::string
Java or .NET byte[]	byte[]	byte[]	std::string
.NET Object	Object	undefined	undefined
Java Object	undefined	Object	undefined
DataReference	DataReference	DataReference	DataReference

Using The GridCache API

Details about the GridCache API are in the GridServer API JavaDoc, which is available from the Documentation page of the Administration tool. Documentation for the `Cache` interface covers the use of GridCache.

The GridCache API supports the following seven primitives:

GridCache constructor with CacheFactory

To create a new GridCache instance, use the CacheFactory to get a reference to a particular region. On a particular client component, you can construct multiple instances of a GridCache with the same region, but each exposed instance with the same region shares the same underlying implementation. This lets multiple Sessions share the same view of a cache without having to duplicate the storage or the code.

Put and Get

The `put` method writes to the cache a new entry for a key and object. The `get` method returns the object stored in the cache, for a given key. If you use the get method on a key that does not exist and the region has an associated loader, the loader attempts to load the data for that key. A second get method, when given a mapping of keys, bulk-gets a mapping of keys and their objects. Bulk get methods use a single HTTP request for each map, which can lower transaction overhead.

Region

Region names are any printable low ASCII character (32-126), except for characters prohibited in XML attribute values (', ", &, <).

Keys

A Key is a string that refers to an object in the cache. The `keys` method gets:

- For a global region type, a list of all keys currently stored on the Manager for this cache.
- For a local region type, a list of locally cached keys.

Remove

Removes this object from the region, from the Manager, and from all distributed caches.

Clear

Clears all objects from the region, the Manager, and regions on other components.

Invalidation handlers

By default, GridCache implements a lazy invalidation mechanism where callers are told only that their version of an object is out-of-date when they make a fresh “get” call for the object. The invalidation handler interface lets the caller register or deregister to receive asynchronous notification that a get, put, remove, or clear has invalidated the caller’s local copy of an object.

Note that .NET cache loaders for GridCache must be packaged as a Super Grid Library. For more information about using Super Grid Libraries, see the *TIBCO GridServer® Administration*.

Fault Tolerance and GridCache

GridCache supports fault-tolerance. For details, see the *TIBCO GridServer® Administration*.

GridServer Design Guidelines

This section discusses two important aspects to consider when designing an application to run on GridServer: data movement, and task or Service request duration. There are a variety of ways to move data among the machines involved in an application; the first section considers their characteristics and suggests which to choose under various circumstances. When you divide a problem into a set of tasks or Service requests, you can usually select how many to use, or equivalently, how much time each one can take. The second section discusses factors that can influence this decision.

Data Movement

Every distributed computation ultimately executes as a local computation—a single computing process. You must move every piece of input data across the network from wherever it resides to the machine that needs to process it, and every piece of output data must travel over the network from the machine that produced it to its ultimate destination. Additionally, you can use caching to optimize data movement, providing a strategy for lowering the amount of data transfer. Moving large amounts of data over a network efficiently is a crucial aspect in the design of most distributed applications. Efficient data movement can often make a dramatic difference in performance.

Principles of Data Movement

Good data movement design can be summarized in two principles:

- Move each piece of data over the network as few times as possible—preferably just once.

  The less that data is moved, the less time it takes to move it. But the many layers of abstraction offered by modern computer systems can hide data movement, making it harder to see the bottlenecks. Network file systems are a good example: there is no way to tell from reading the code whether a file is being read from the local disk or over a network, but the performance difference can be significant.

- Move data as early as possible—preferably before the computation starts.

Doing so improves the performance of the computation because the stopwatch that times the computation is started after the data movement has already occurred. But this is more than a mere accounting trick. Consider a nightly report that must run after 5 PM to avoid conflicting with daytime Services. If the data for the report is available at 4 PM, it can be distributed to Engines in the hour before the report runs.

Data Movement Mechanisms

The GridServer software uses the following data movement mechanisms:

- Service Request Argument and Return Value
- Service Session State
- Shared Directories and Direct Data Transfer
- Resource Update
- GridCache
- Data References

Service Request Argument and Return Value

The most direct way to transmit data between a Grid client and an Engine is through:

- The argument to a Service request and
- The return value from the Service request.

If you enable Direct Data Transfer, the data travels directly between Driver and Engine.

Each request is handled efficiently, but the aggregate data transfer across hundreds of requests adds up significantly. Therefore, factor data common to all requests into session state or init data, or distribute it by another mechanism.

Service Session State

Any Service Session can have an associated state. As described in the Services sections, this state resides on the Driver as well as on each Engine hosting the instance, so it is fault-tolerant with respect to Engine failure.

Service Session state is ideal for data that is specific to a session. Service Session state is easy to work with because it fits the standard object-oriented programming model; it is downloaded once per Engine.

Transmission of the Service Session state from Driver to Engine is peer to peer and is GridServer's Direct Data Transfer (DDT) feature. DDT is enabled by default. When DDT is enabled and a Service creation or Service request is initiated on a Driver, the initialization data or request argument resides on the Driver, sending only a URL (and not data) to the Manager. When an Engine receives the request, it downloads the data directly from the Driver rather than the Manager. This mechanism saves one network trip for the data and can result in significant performance improvements when the data is much larger than the URL that points to it, as is usually the case. It also greatly reduces the load on the Manager, improving Manager throughput and robustness.

Shared Directories and DDT

Some network configurations are more efficient using a shared directory for DDT rather than the internal file servers included in the Drivers and Engines. In this case, configure the Driver and Engines to read and write requests and results to the same shared network directory, rather than to transfer data over HTTP. All Engines and the Driver must have read and write permissions on this directory. Configure shared directories at the Service level with the `SHARED_UNIX_DIR` and `SHARED_WIN_DIR` options. If you use both Windows and UNIX Engines and Drivers, configure both options to be directories that resolve to the same directory location for the respective operating systems.

Resource Update

GridServer's Resource Update mechanism replicates Grid Libraries, or archives of versioned sets of resources, with Engines. It also replicates the contents of a directory on the Manager to a corresponding directory on each Engine. When you use Resource Update, use the **Services > Services > Grid Libraries** page in the GridServer Administration Tool to upload files to the Manager. After all currently running Services finish, the Engines download the new files. For more on Resource Update, see the TIBCO GridServer® Administration.

Resource Update is the best way to guarantee that the same file is on the disk of every Engine in your Grid. File Update is ideal for distributing application code, but it is also a good way to deliver configuration files or static data to Engines before your computation starts. Any kind of data that changes infrequently, like historical data, is a good candidate for distribution in this fashion.

GridCache

GridServer's GridCache feature is a repository on the Manager that is aggressively cached by components (Drivers and Engines). The repository comprises a set of regions, each of which is a map from string keys to arbitrary values. The GridCache API supports reads, writes, removing key-value pairs, and getting a list of all keys in a catalog. For more information about GridCache, see GridCache.

A GridCache component caches every value that it gets or puts. If a component changes a key's value or removes it, the Manager asks all components to invalidate their cached copy of that key's value.

GridCache is fault-tolerant with respect to Engine failure because the data is stored on the Manager. When an Engine fails, its cached data is lost and its task is rescheduled. The Engine that picks up the rescheduled task gradually builds up its cache as it gets data from the Manager.

GridCache is a flexible and efficient way for Engines and Drivers to share data. Like File Update, an Engine needs only a single download to obtain a piece of constant data. Unlike File Update, GridCache supports data that changes over the life of a computation.

You can use GridCache for having Engines post results. This is generally only useful if those results are to be used as inputs to subsequent computations.

Data References

GridServer Data References are objects that represent data existing on a GridServer client. You can use them to pass lightweight data from one client to another so that only the destination needing the data performs the data transfer. Typically, one client filesystem stores the data and another client's file server serves the data.

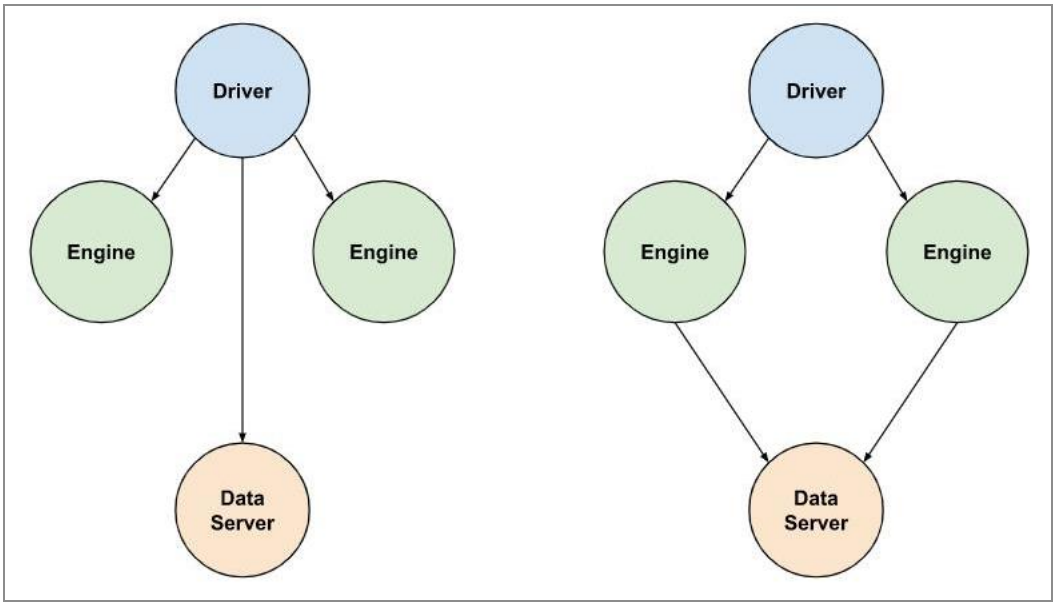
Data Movement Examples

As an example of using the data movement mechanisms discussed above, consider the problem of determining the value of a financial instrument. This example uses a computation method named value. The value method takes two arguments: a deal and a pricing scenario. The deal argument contains all information specific to a financial instrument needed to determine its value, such as coupon and maturity date. The pricing scenario argument contains all other determinants of the deal's value, such as interest rates and prices of underlying instruments. The output of the value function is a single number representing the value of the deal under the given pricing scenario.

Typical applications require the value of many deals over one or several pricing scenarios. To distribute and parallelize this computation, we execute the value function simultaneously on many Engines. We assume the code for the value function is available to each Engine (whether by Resource Update or over a network file system). We also assume that the numbers returned by the value function make their way back to the client through the standard Service return value mechanism. The question we want to consider is how to get the deal and pricing scenario information to the Engines.

Database Access

We first look at the deal information itself, stored in a database or data server somewhere on the network. Compare the two scenarios in Figure 8-1. In the first diagram, on the left, the Driver loads the deal information from the data server and sends it to the Engines. In the second diagram, on the right, the Driver sends just the unique identifier and has each Engine access the data server on its own.



Data flow between a Driver, two Engines, and a Data Server

The second choice is better because it requiresfewer data moves across the network to accomplish the same result. In the first choice, the data moves across the network twice, once from the data server to the Driver and second from the Driver to the Engine. In the second choice, the data moves across the network only once from the data server to the Engine. Also, the data needs marshaling and unmarshaling only once.

The second choice also increases parallelism at the data server. In the first choice, only the Driver is attempting to load data from the data server. In the second, multiple Engines

attempt to load data concurrently. Assuming that the data server can handle the load, the second choice increases parallelism.

Single Pricing Scenario

We now consider the case in which you use a single pricing scenario to evaluate many deals. Here is one (suboptimal) way to organize this computation. We assume throughout that you already deployed and registered a Service containing the value function.

Algorithm 1 (suboptimal):

1. Create a Service Session of the value Service.
2. For each deal, submit the **deal identifier** and the **pricing scenario** as an asynchronous request to the Service Session.
3. Wait for results.

Although this algorithm gets the job done, it needlessly sends the same pricing scenario multiple times.

This is an ideal application of Service Session state:

Algorithm 2:

Procedure

1. Create a Service Session of the value Service, initialized with the pricing scenario.
2. For each deal, submit the deal identifier as an asynchronous request to the Service Session.
3. Wait for results.

By making the pricing scenario be part of the session's state, it is transmitted only as many times as there are Engines that implement the session, rather than once per request. GridServer never allocates more Engines to a Service session than there are requests for that instance, so Algorithm 2 never moves more data than Algorithm 1. And in the likely event that there are many more requests than Engines (we argue below in the Task Duration section why this is a good idea), Algorithm 2 moves much less data than Algorithm 1.

Several Pricing Scenarios

What if the application needs to value the portfolio of deals for more than one pricing scenario? One approach is simply to repeat Algorithm 2 several times, creating a new Service session for each pricing scenario. It is also possible to use a single session and employ the `updateState` method of the Service client API to transmit each successive pricing scenario to the Engines running the session. If the differences between pricing scenarios are small and they are used to perform the update instead of the pricing scenarios themselves, then using `updateState` can result in considerable data movement savings; even if the pricing scenarios themselves are used as updates, this approach is still likely to be superior to using separate instances.

Multiple Pricing Scenarios Available Early

Now let us add the following wrinkle: we still want to compute the value of many deals over many pricing scenarios, but the pricing scenarios are available to us sometime before we can run the application. For instance, pricing scenario information is available at 4 PM, but we cannot start the nightly report until 5:30 PM, to avoid interfering with daily work. In this situation, we can exploit the time gap to push information to the Engines before the computation starts. One approach would be to use File Update to put all the pricing scenario data on all the Engines. Another would be to put the pricing scenario data into GridCache and run a “primer” Service that copies the data to the Engines. The trade-offs between these two approaches were discussed above under Data Movement Mechanisms.

Deal-Pricing Scenario Symmetry

Finally, we point out that deals and pricing scenarios are for the most part symmetric in these examples (the main difference being that pricing scenarios are less likely to be indexed by primary key in a database, so the discussion of deal identifiers versus deal data does not apply to them). For instance, if deals are available to you early, you can use File Update or GridCache to push deal information to Engines before your application starts.

Service or Task Duration

Service or Task duration has an important impact on the performance of distributed computations. Recall that a task corresponds to a single Service request when using Services. Make tasks long enough to compensate for communication overhead, but not so long that their interruption seriously delays the overall computation. Dividing the work into

more tasks, each of which takes less time, can also mitigate the performance degradation that can arise from having tasks of different sizes. We discuss these issues in detail in the following sections.

As a running example, we use the deal valuation problem discussed in the previous section on data movement. There we assumed that each task was responsible for pricing a single deal. But this is unlikely to be efficient for most types of deals; instead, group several deals together in a single task.

Engine Interruption and Smoothing

If an Engine is interrupted or fails during a task, that task must run again from the beginning. Therefore, divide work into tasks with short execution times. The shorter the task, the less work you lose when an Engine fails.

Additionally, shorter tasks result in better performance. This is because you are reducing the variability of task durations in a computation.

For example, suppose that you divide the work of computation so that 10 Engines each have one task. You expect that this minimizes communication overhead and that Engines do not fail. However, what if you wrongly estimate one task and it takes twice as long as the others? Since all tasks must finish for the computation to be complete, the longest task determines the computation time. If you have nine one-minute tasks and one two-minute task on 10 Engines, the computation takes two minutes, with the last minute consisting of nine idle Engines and one Engine still working on the two-minute task. With exactly as many tasks as Engines, your program runs as long as the longest task. (This section simplifies this discussion by ignoring communication time.)

Suppose you use twice as many tasks as Engines. This significantly improves the expected running time. To understand why, continue the above example. If you divide each of the 10 tasks in two, you have 20 tasks for 10 Engines: 18 30-second tasks, and two one-minute tasks. Each Engine takes two tasks at random. The chance of the same Engine receiving both long tasks is fairly small, so this program is likely to take one and a half minutes most of the time.

Similarly, more, shorter tasks smooth out the effect of different processor speeds. Assume that all tasks take the same time, but that one Engine is slower than the others. With exactly one task per Engine, the slow Engine determines the computation time. With many short tasks, the slow Engine takes fewer tasks than the other Engines, and all Engines finish at close to the same time, minimizing the time for the whole computation.

Summary

Communication overhead dictates using long tasks, but the possibility of Engine failure and the opportunity to smooth over differences in task durations and processor speeds suggest using many quick tasks. The best compromise is to choose a task running time between 30 seconds and several minutes and to choose a number of tasks that is three or four times the number of available Engines.

These are just a few examples of how to improve the performance of your Services. For a more complete list, see the *Performance and Tuning* section of the *TIBCO GridServer® Administration*.

The Admin API

The GridServer Admin API offers programmatic access to administrative tasks or information normally performed or presented in the web-based GridServer Administration Tool. The API is available through Java, C++, and .NET Drivers and Services, on Managers through a Server Hook, from a SOAP Web Service, with JMX, and REST.

Documentation for the GridServer Admin API

Detailed documentation on GridServer Admin API is in the API documentation. The WSDL is available in the GridServer Administration Tool at **Grid Components > Drivers > Web Services**.

The following components are defined:

- `BatchAdmin`
- `BrokerAdmin`
- `DriverAdmin`
- `DriverManager`
- `EngineAdmin`
- `EngineDaemonAdmin`
- `ManagerAdmin`
- `ServiceAdmin`
- `UserAdmin`
- `Version`

The methods allowed are based on the Security Roles of the user. When called in a Service, the user is considered to be the user that created the Service Session. When called in a Server Hook, all methods are available. Note that methods return null if there is no output, as opposed to returning a zero-length array. For example, `EngineAdmin.getAllEngineInfo()` returns null if there are no Engines currently logged in to the Broker.

See the **Admin > User Admin > Role Admin** page on the Administration Tool for a list of all permissions.

Using the Admin API over SOAP

The following example uses the Admin API over SOAP with Java:

1. Locate the WSDL for the Service from the Manager's Web Service List. For example, for the `EngineDamonAdmin` class use, `http://example:8080/livecluster/webservices/EngineDaemonAdmin?wsdl`
2. Generate Java Stubs for the Service. For example, using Axis:

```
   org.apache.axis.wsdl.WSDL2Java
   http://example:8080/livecluster/webservices/EngineDaemonAdmin?
wsdl
```

3. Use the Stubs. For example:

```
   // Get the interface to the Admin Service
   EngineDaemonAdmin server = (new
   EngineDaemonAdminServiceLocator()).getEngineDaemonAdmin();
   // Required when Driver authentication is enabled
   ((Stub)server).setUsername("admin");
   ((Stub)server).setPassword("admin");
   // Maintain the session ID for each request
   ((Stub)server).setMaintainSession(true);
   // Query the Admin Service
   EngineDaemonInfo[] info = server.getAllEngineDaemonInfo();
```

Using Server Hooks

The entire Java Admin API is available within a Server Hook. For details about implementing Server Hooks, see the JavaDoc documentation for the `ServerHook` class.

Using JMX

The Java Admin API is also available using JMX. Most API components are exposed as MBeans within the `com.datasynapse.gridserver.admin` tree. See the JavaDoc documentation for the `com.datasynapse.gridserver.admin` package for more information about each object.

Using Conditions

In a typical Grid environment, machines are not all identical. Some machines are slower, or have less RAM; other machines are faster but work to capacity during the day. Depending on the Services you have and the general demographics of your computing environment, the scheduling of Services to Engines might not be clearly deterministic. And sometimes, a specific Service might require special handling to ensure that optimal resources are available for it.

Conditions

Conditions are a feature of GridServer. Conditions affect how Service Sessions and tasks are scheduled to Engines. Conditions enable you to use Engines selectively based on their properties.

GridServer provides the following types of Conditions:

- **Discriminator Conditions** specify a subset of Engine that can work on a task or Session, typically based on Engine properties.
- **Affinity Conditions** enable you to set an affinity for tasks or Sessions for Engines, based on Service state and typically properties. Unlike Discriminators, they do not prevent tasks from going to any Engine, it only attempts the best match.
- **Dependency Conditions** enable submitting workflows to a Broker without an active Grid client to manage Dependencies (wait for completed tasks, submit more based on a successful outcome, and so on).
- **QueueJump Conditions** enable the Driver to specify that a Task is placed at the front of the waiting queue, rather than at the back.
- **Descriptor Conditions** set descriptive information about an Invocation request.
- **EXTRAConditions** prevent Engines from taking Tasks when a necessary resource, such a database connection, is not available. The current resource counts are updated with a simple REST API.

Discriminator Conditions

Use Discriminator Conditions to select Engines for particular Services based on Engine properties. Discriminators have many uses:

- Limit a Service to run only on Engines whose usernames come from a specified set, to confine the Service to machines under your jurisdiction.
- Limit a resource-intensive task to run only on Engines whose processors are faster than a certain threshold, or that have more than a specified amount of memory or disk space.
- Direct a task that requires operating-system-specific resources to Engines that run that operating system.

Discriminator Conditions can be dynamically attached to a Service based on the Service Description on the Manager or set programmatically with the Driver.

Setting Discriminators in the Administration Tool

You can also attach Discriminators to Services in the GridServer Administration Tool, at **Services > Services > Service Conditions**. This page enables you to create Discriminators by entering three defining factors: the Services that the Discriminator affects, the types of Engines that can run on those Services, and optionally, if an EXTRACondition must be met. (See EXTRAConditions for more information about EXTRAConditions.) This differs from programmatic Discriminators because they aren't explicitly attached to a Service at its creation; instead, a group of Services is defined as being attached to that Discriminator, by Service name, application name, department name, or wildcards on that or other criteria.

By default, changes made to Discriminators in the GridServer Administration Tool are applied to Services immediately. This can be changed so that Discriminator changes only apply to subsequently created Services. To change this behavior, go to **Admin > System Admin > Manager Configuration > Services**, and under the **Scheduling** heading, change the value of **Apply Condition Admin Changes Immediately**.

Setting Discriminators Programmatically

You can use the GridServer API to set Discriminators for a Service or task. Create Discriminators with the `SchedulingConditionFactory`. In Java, this is located in `com.datasynapse.gridserver.client`. You can use the factory to create several Conditions. The method to use to create Discriminators is `createPropertyDiscriminator`.

The following example in Java code creates a `SchedulingConditionFactory`, which you then use to create a Discriminator Condition to run a Service or task only on Engines where the operating system is Linux:

```
SchedulingConditionFactory schedFactory =
SchedulingConditionFactory.getInstance();
Condition isLinux = schedFactory.createPropertyDiscriminator
(EngineProperties.OS, SchedulingConditionFactory.CONTAINS, "Linux",
false);
```

The `createPropertyDiscriminator` method takes four parameters:

- The first parameter is the name of an Engine property.
- The second parameter is a comparator used to compare the Engine property to the property value. You can use several comparators to compare numbers or strings. (See details in the `SchedulingConditionFactory` API documentation.) For example, the `CONTAINS` comparator is true if an Engine property matches any one of a comma-delimited list of properties. `MATCHES` compares against a Java regular expression; `EQUAL` checks if parsed numerical values are equal.
- The third parameter is a property value.
- The fourth parameter defines what happens if the property is not set on an Engine. In this case, the Engine is not considered. In some situations, you might want the opposite behavior, such as when you are using a Discriminator to exclude a subset of Engines with a specific property, but allowing any other Engines.

After you create a Condition such as a Discriminator, you can use it when creating or submitting Services. The following example in Java code creates a Discriminator and sets it in a Service:

```
SchedulingConditionFactory schedFactory =
SchedulingConditionFactory.getInstance();
//Only run on Linux Engines
Condition isLinux = schedFactory.createPropertyDiscriminator
```

```
(EngineProperties.OS, SchedulingConditionFactory.CONTAINS, "Linux",
false);
Service s = ServiceFactory.getInstance().createService("MyService",
null, null, null, isLinux);
```

If you change the properties of Engines, the changes take effect during the execution of a Service. For example, if you configure a Discriminator attached to a running Service to look for a certain Engine property, changing this property can change what Engines work on that Service.

Discriminators are, however, attached to a Service at Service creation, so changes you make to the Discriminator affect only subsequently submitted Services, not Services that are already running.

PDriver Discriminators

When writing a PDS script, you can create job-level or task-level Discriminators to limit which Engines work on a PDriver job or task. The `discriminator` block specifies either a job-level or task-level Discriminator for a job.

For more information about PDriver Discriminators, see The Discriminator Block.

Affinity Conditions

When a Broker is assigning tasks to Engines, one of the methods used to make optimal matches is *affinity*, or the degree to which an Engine has initialization data and updates from a particular Service. If there are a number of possible matches at a point in the scheduling decision, the Broker assigns tasks to the match with the highest affinity score. Affinity is a number that is calculated between an Engine and a Service Session. By default, the affinity number is based on the amount of state the Engine has for the session, whether it has loaded its Grid libraries, and whether it is on a Home Broker.

It is possible to use *Affinity Condition* to add affinity, typically based on Engine properties. This Condition is similar to a Discriminator Condition, except when it is satisfied, it adds a defined number to the affinity score.

If you plan to use affinity, aside from Affinity Conditions, you also need to tune how and how much the scheduler uses affinity. For more information see the *TIBCO GridServer® Administration*.

Setting Affinity Conditions Programmatically

Create Property Affinity Conditions with the `SchedulingConditionFactory`, using the `createPropertyAffinity` method.

The following example in Java code creates a `SchedulingConditionFactory`, which is then used to create a Property Affinity Condition that would add four to the affinity of any Engine-Service pairing where the Engine has two CPUs:

```
SchedulingConditionFactory schedFactory =
     SchedulingConditionFactory.getInstance();
Condition dualAffinity =
     schedFactory.createPropertyAffinity(EngineProperties.CPU_NO,
     SchedulingConditionFactory.EQUALS, "2", 4.0);
```

Setting Affinity Conditions in the Administration Tool

You can also attach Property Affinity Conditions to Services in the GridServer Administration Tool. Go to **Services > Services > Service Conditions**. This page enables you to create Affinity Conditions by entering two defining factors: the Services that the Condition affects, and the affinity change that occurs based on an Engine's properties.

Task Affinity

Task Affinity provides the ability to run a set of Tasks on the same Engine or set of Engines. It is primarily used for data awareness and locality. For example, you might have a large dataset, and have Tasks that work on subsets of that dataset, so you would prefer that an Engine works on Tasks from the same subset.

Create Task Affinity Conditions with the `SchedulingConditionFactory`, using the `createTaskAffinity` method.

When set on a Task, it specifies that this Task must run only on an Engine that has already worked on a Task with the same `TaskAffinity` as specified by a tag.

The `wait` value indicates how long it must wait for an Engine with affinity before being allowed on any Engine. This wait countdown starts the first time that the task becomes available for scheduling, and only if it is the next task to be assigned to this set.

The first `TaskAffinity` for a given tag must be assigned a wait value of 0 so that it is immediately assigned to an Engine. Typically the remaining in the set would use the same value, for example, 10000 for a wait of 10s. An Engine retains the affinity for the duration of the Service Session but not beyond.

The tag is not guaranteed to be unique among all Services, so you would typically append a unique value to the tag, such as the Service ID. This is left to the caller, since there might be a unique string shorter than the Service ID which results in quicker string comparisons.

`TaskAffinity` can be used in a `ConditionSet`. For example, you can use this along with a `PropertyDiscriminator` and a `Descriptor` using an `AND ConditionSet`.

Custom Discriminator and Affinity Conditions

In addition to Conditions based on Engine Properties, you can implement your own custom Conditions. For example, you might have a number of Services that require a connection to a database and might have more Engines working on those Services than available connections. You could write a Server Hook that keeps track of how many tasks are currently running, and write a Custom Discriminator to prevent all Engines from taking a Task if all connections are in use by other Tasks.

To write a Custom Condition:

1. Implement the `CustomDiscriminator` or `CustomAffinity` interface in `SchedulingConditionFactory`.
2. Package your classes into a JAR file, and for each Broker on which this is used, place that file in `DS_MANAGER/webapps/livecluster/WEB-INF/lib`.
3. At Service creation time, create it using the `SchedulingConditionFactory.createCustom[Affinity/Discriminator]` call. Note that it is not necessary to have the JAR in a Java Driver classpath; likewise, this is also available for C++ and .NET Services, since the class is only used on Brokers.

Dependency Conditions

Dependency Conditions allow you to submit workflows to a Broker without an active Grid client to manage Dependencies (wait for completed tasks, submit more based on a successful outcome, and so on). When you submit a Session, you can require one or more tasks or entire Services to complete before the scheduled Service. These Dependencies can be Sessions or tasks already submitted, or ones that have not yet been created. This way, multiple tasks in different Services can be submitted, but they are not eligible for scheduling until certain conditions are met—namely the successful completion of specific tasks or Sessions.

Creating Dependencies

Dependencies are Conditions that are applied to a Service or a Task. This Condition has methods for adding Dependencies in the form of a Session ID, and for an optional Task ID. Dependencies also allow a Boolean operation that dictates whether to cancel an entire Task/Service when a dependent Task/Service fails.

You can create a forward Dependency for a Session, and optional Task ID, that does not yet exist. To create a forward Dependency, generate a reference ID to a Session, and then use that ID when creating the Session.

In Java, you use `com.datasynapse.gridserver.client.DependencyFactory` to create Dependency Conditions. You can assign more than one Dependency by using a `ConditionSet`. C++ and .NET APIs are similar. See the GridServer API documentation for more details.

If there is no session with the dependent Service ID on the Broker when the Session or task with Dependencies is added, it is automatically canceled because the Dependency does not exist. If it is a forward Dependency, no such cancellation is made.

You can specify that a Session or task be canceled if it has a Dependency that fails.

If a non-forward Dependency is made, and the session does not exist, the task is always canceled.

Administering Task Dependencies

You can view Dependencies in the GridServer Administration Tool. Select **Services > Services > Service Sessions**, or from the Task Admin page, available from the **Actions** list on the Service Session Admin page. On either page, select **Service Session Details** or **Task Details** from the **Actions** list. View the details for a list of Dependencies showing which are pending and which are complete.

You can remove Dependencies from a task or Service with the **Remove Dependencies** action on the **Actions** list on the Task Admin page. This removes the entire Dependency object, which removes all pending Dependencies; there is no a way to remove a single Dependency from the Administration Tool.

Note Task Dependencies are Broker-scope, and rely on Service and task events on a Broker. They do not work across Brokers.

Because the default for the `PURGE_INVOCATION_DATA` option is `SERVICE_COMPLETED`, task information can be lost on a Service, making a dependent Service unavailable to find the information. In this situation, you can set `PURGE_INVOCATION_DATA` to `SERVICE_REMOVED` instead.

Queue Jump Conditions

The *Queue Jump Condition* is used to specify that a Task is placed at the front of the waiting queue, rather than at the back. It cannot be set on a Session. A typical use case is a Service that submits some work, waits for results, and submits more work based on earlier results. If an early task fails and you need to resubmit it, you use this condition to make sure it is executed as soon as possible instead of waiting for all other work to be complete.

Queue Jump Conditions are created programmatically using the `SchedulingConditionFactory` class. For more information about using this class, see Setting Discriminators Programmatically

Descriptor Conditions

Use Descriptor Conditions to display descriptive information about an invocation request.

Descriptive information about an invocation request is also available on the Task Admin page and Admin API. Also, the Descriptor name is part of any log message for a task with a Descriptor Condition.

Create Descriptors with the `DescriptorFactory`. For example:

```
DescriptorFactory descFactory = DescriptorFactory.getInstance();
descFactory.create("C++ Driver Linux Build");
```

You cannot create a Condition Set of Descriptors. A Descriptor Condition can be only a sole Condition in a Service or Service Set.

EXTRAConditions

An *EXTRACondition* is an External Resource Advisory Condition, and is used to prevent Engines from taking Tasks when a necessary external resource, such as a database connection, is not available. When using EXTRAConditions, the scheduler ensures that a task's required resources are available before assigning it to an Engine.

There are two parts involved in using EXTRAConditions: defining the Service Condition, and maintaining the EXTRACondition count. Like other Conditions, this type of Condition is defined on the Service Conditions page, and consists of one or more named resources with a numerical comparison. The current resource counts are then updated with a simple REST API.

Using the EXTRACondition REST Interface

Each EXTRACondition consists of a key and value pair that specifies a resource count. To use an EXTRACondition, you must create a key and then periodically update its count as the resource count changes.

Resource counts for EXTRAConditions are created and updated with a REST API.

The EXTRACondition REST Interface root is:

```
http://host:port/livecluster/restadmin/extracondition-admin/
```

The REST interface resides on the Primary Director. Values are backed up to the Secondary Director but updates to the Secondary Director are not replicated back if the Primary Director comes back up. In the event of a Primary Director failure, you must use the Secondary Director's URL to update the EXTRACondition values until the Primary Director comes back up.

The API includes the following resources:

- `extracondition` — for operations on a single EXTRACondition
- `extraconditions` — for operations on all EXTRAConditions.

REST requests use basic authentication.

Creating and Updating an EXTRACondition

HTTP `PUT` requests are used to create and modify an EXTRAConditon. The following syntax is used:

```
/restadmin/extracondition-admin/extracondition/key
```

The value for the key must be passed in the body of the request. The request returns the value of the EXTRACondition as a string with an HTTP Response code of 201 upon success. Any other server-side error returns exception text with an HTTP Response code of 500.

For example, the following `PUT` request, when made with a value in the body of the request, creates a new EXTRACondition for the key `db.connections`, or updates the existing `db.connections` EXTRACondition:

```
/restadmin/extracondition-admin/extracondition/db.connections
```

Getting an Existing EXTRACondition

An HTTP `GET` request can be used to retrieve the value of an EXTRACondition. The following syntax is used:

```
/restadmin/extracondition-admin/extracondition/key
```

If the EXTRACondition is not found, an HTTP Response code of 404 is returned. Any other server-side error returns exception text with an HTTP Response code of 500.

For example, the following `GET` request returns the value of the EXTRACondition of the key `db.connections` as a string with an HTTP Response code of 200 upon success:

```
/restadmin/extracondition-admin/extracondition/db.connections
```

Deleting an existing EXTRACondition

An HTTP `DELETE` request can be used to remove an EXTRACondition. The following syntax is used:

```
/restadmin/extracondition-admin/extracondition/key
```

The request returns with an HTTP Response code of 200 even if the key does not exist. It only returns an exception text with an HTTP Response code of 500 for other server errors or an existing EXTRACondition was not able to be deleted.

For example, the following `DELETE` request deletes the EXTRACondition with the key `db.connections`.

```
/restadmin/extracondition-admin/extracondition/db.connections
```

Batch Operations

There is also an `extraconditions` resource that enables you to make an HTTP `GET` to retrieve all EXTRAConditions. The following syntax is used

```
/restadmin/extracondition-admin/extraconditions
```

The request returns a JSON array of key-value pairs formatted like the following:

```
{
    "Boston.BigFatDB.Connections":600,
    "SF.BigFatDB.Connections":500,
    "NYC.BigFatDB.Connections":1000
}
```

Similarly, an HTTP `PUT` with the same JSON in the body of the request updates or creates all pairs in the list.

To batch delete EXTRAConditions, you must use an HTTP `PUT` using the following syntax:

```
/restadmin/extracondition-admin/extraconditions/delete
```

The keys being deleted must be passed in the body of the request, like the following:

```
{
    "Boston.BigFatDB.Connections",
    "SF.BigFatDB.Connections",
    "NYC.BigFatDB.Connections"
}
```

Batch operation requests return with an HTTP Response code of 200 upon success. Any other server-side error returns exception text with an HTTP Response code of 500.

Setting EXTRAConditions

EXTRAConditions can be specified in Discriminators, which are attached to Services in the GridServer Administration Tool at **Services > Services > Service Conditions**. You can specify that one or more comparisons of EXTRACondition to resource criteria must be true, or the Task is not given to an Engine.

When creating a Discriminator using an EXTRACondition, there is a **modifier** column for each Condition, which you can set to a positive or negative number. When a Task is assigned to an Engine, the scheduler modifies the EXTRACondition's external resource value by adding the modifier value. Note that the external resource value is not automatically changed when a Task completes; you must script this using the REST interface, as described above.

Condition Sets

In addition to using a single Condition, you can also combine Conditions. A *Condition Set* is a set of one or more Conditions; the set is treated as a single Condition.

Create Condition Sets with `ConditionSetFactory`. (In Java, this is `com.datasynapse.gridserver.client.ConditionSetFactory`.) After you create a Condition

Set, use its `add` method to add Conditions to it. There are three types of Condition Sets, described below, which dictate how the Conditions are evaluated, and what Conditions can be added.

AND set

The AND set creates a set of ANDed Conditions. Two possible combinations of Conditions are legal in an AND set:

- An AND set can contain Discriminators, Dependencies, or both. If all the Conditions in the set are true, the Condition Set is true.
- Or, an AND set can contain Property Affinity Conditions. If all Affinity Conditions are true, the scores that match are added to the affinity score.

You cannot mix Discriminators, Dependencies, and Property Affinity Conditions in one AND set. Note that you cannot add Descriptor Conditions to an AND set.

OR Set

The OR set creates a set of Conditions for which any Condition can be true for the set to be true. The OR set can contain only Discriminators, Dependencies, and sets of either or both.

Service Set

The Service Set creates a Condition Set for adding different Condition types to an Invocation or Session. You can add this set only to `ServiceFactory.createService(...)`, `Service.execute(...)`, and `Service.execute(...)`. You can add only one Condition of each type (Discriminator/Dependency, Affinity, Descriptor, and QueueJump).

You can, however, add a Condition Set containing only one type of Condition. For example, a Service Set could include a Condition Set of Discriminators and Dependencies, a Condition Set of Property Affinity Conditions, and a Descriptor. (You cannot create a set of Descriptor Conditions.)

Engine Properties

Within GridServer, each Engine has a set of properties. GridServer sets some Engine properties automatically, such as the Engine's operating system and the estimated speed of the Engine's processor. You can also create custom properties for Engines.

Intrinsic Engine Properties

An Engine has a number of properties that reflect machine-specific information, such as OS, hostname, free memory, and so on. For information about these properties, see `EngineProperties` in the `com.datasynapse.gridserver.engine` package of the API documentation.

Custom Engine Properties

You can create your own custom Engine properties, and give them values using the GridServer Administration Tool. To do so, first, create the property on the Manager, and then give it a value for each Engine, either from the GridServer Administration Tool or programmatically with the Admin API.

To create new custom Engine properties and give them values with the Administration Tool:

1. In the GridServer Administration Tool, go to **Grid Components > Engines > Engine Properties**.
2. In the upper right corner, click **Add**.
3. Enter a property name and a brief description, then click **Add**. You can now set a value to this property on any Engine.
4. Go to **Grid Components > Engines > Daemon Admin** and select **Set Property for Daemons on Page** or **Set Property for All Daemons** from the **Global Actions** list. This displays the user-defined properties you can set on Engines started by a Daemon.
5. Select one or more Engine Daemons from the list, then select a predefined property and enter a value, or enter a new property name and assign a value.

Instead of 3-4, you can also use the Admin API on a Driver to programmatically set them.

Engine Session Properties

Session Properties are properties that last for the duration of an Engine session on the Manager. They are set on an Engine when it logs in and reset when the Engine logs off.

This example sets a Session Property:

```
public void init() {
    // initialize some static data for use by another service
    EngineSession.setProperty("inited", "true");
    // this property can now be used by the Discriminator
    // of the other Service
}
```

See the API documentation for the `EngineSession` class for more information.

GPU Services Engine Properties

One set of intrinsic Engine properties is the CUDA GPU Engine properties set. On Windows and Linux systems, the Engine Daemon detects the presence and characteristics of GPU processors with the CUDA runtime library and provides those characteristics as Engine properties. This allows for the development of Services that can take advantage of machines that have GPU cards.

If the Engine Daemon detects a CUDA GPU, it sets the following Engine properties:

- `CUDA_DEVICES` – The number of CUDA devices detected on an Engine.
- `CUDA_FIRST_GPU_NAME` – The name of the first CUDA device.
- `CUDA_GLOBAL_MEMORY` – The amount of CUDA global memory supported. If there is more than one device, this is the minimum amount supported.
- `CUDA_VERSION` – The CUDA capability version. If there is more than one device, this is the minimum version.
- `CUDA_PROCESSORS` – The number of CUDA processors supported. If there is more than one device, this is the minimum number of processors.

Note that on 64-bit Windows machines, CUDA properties are only detected when the Engine Daemon is started on the physical console. Engine Daemons started via RDP do not detect any GPUs.

On Linux machines, you must ensure that the device files `/dev/nvidia*` exist and have read/write file permissions for the user running the Engine. This can be done by creating a startup script to load the driver kernel module and create the entries at boot time. For more information, see the Linux documentation at the NVIDIA support site.

See the CUDA example in the GridServer SDK for more information.

CUDA detection was tested on the GeForce 210 device with 3.0 drivers, GeForce 9800 GT device with 3.10 drivers, the Quadro NVS 295 with 3.0 and 3.20 drivers, and the Quadro FX 380 with 3.10 drivers.

MIC Processor Engine Properties

TIBCO GridServer® supports Intel Many Integrated Cores (MIC) coprocessors. This includes MIC detection and Engine properties that provide information that can be used for discrimination.

Intel MIC coprocessors, such as the Xeon PHI, are a multi-core processor architecture capable of running a large number of tasks in parallel due to a large number of physical cores available in the card. MIC coprocessors support two execution models: *native*, where programs are executed directly in the co-processor; and *offload*, where programs are executed in the host machine and some parts are executed in the coprocessor. Support for the MIC coprocessor in GridServer is restricted to the offload model.

Requirements

Support for MIC coprocessors requires the following:

- Linux on X86_64

The following compilers are supported:

- Intel Composer XE 13

MIC support has the following limitations:

- The Engine requires a pre-load of shared libraries to run Services with offload code.
- Services with offload code share global objects across Service instances. This is a limitation due to how Intel handles shared libraries with offload code.
- The availability of the MIC coprocessor is determined at Engine Daemon startup time. Changes to the status of coprocessors are not visible to the Engines.

Properties

If the Engine Daemon detects a MIC device, it sets the following Engine properties:

- `MICDevices` – The number of MIC devices enabled at the Engine Daemon startup time.
- `MICModel` – The identifier of the coprocessor model. This information is based on information from the `cpuid` opcode.
- `MICCores` – The number of logical cores.
- `MICMemory` – The amount of memory in bytes.

`MICModel`, `MICCores`, and `MICMemory` refer to the oldest co-processor available in the host machine. Oldest is defined based on this information from the `cpuid` opcode.

Configuration

To use MIC in your Service, you must do the following:

Engine Configuration

To preload the required shared libraries:

1. In the TIBCO GridServer® Administration Tool, go to **Grid Components > Engines> Engine Configurations** and select the appropriate Engine configuration.
2. Set this value under Engine Daemon and Instance Process Settings: Environment Variables:

   ```
   LD_PRELOAD=libjsig.so liboffload.so.5
   ```

   `libjsig.so` is part of the JRE and `liboffload.so.5` is part of the Intel runtime libraries.

Service Type

Services with offload code require `unloadNativeLibrary` set to false in the Service Type registry.

Grid Libraries

Services with the above Service Type setting share global objects across Services. Use the `<conflict>` setting in your `grid-library.xml` to avoid sharing of global objects with other Services:

```
<?xml version="1.0" encoding="UTF-8"?>
<grid-library>
    ...
    <conflict>
        <grid-library-name>[name of grid-library that shares global
objects]</grid-library-name>
    </conflict>
    ...
</grid-library>
```

See the MIC example in the GridServer SDK for more information.

NUMA Engine Properties and Configuration

Non-uniform memory access (NUMA) can increase processor speed without increasing the load on the processor bus. In a NUMA system, CPUs are arranged in smaller systems called nodes. A node has its own processors and memory and is connected to the larger system. The system attempts to improve performance by scheduling processes on processors that are in the same node as the memory being used.

Requirements

NUMA is only supported on Windows Engines.

Windows 7 and Windows 2008 R2 require the following hotfix:
http://support.microsoft.com/kb/2417038

32-bit Engine installations in 64-bit Windows are only able to address processors #0 through #31 in any given processor group.

Properties

The following Engine properties are related to NUMA support:

- `HIGHEST_NUMA_NODE` – The highest NUMA node available in the system. 0 is reported in non-NUMA systems.

- `PROCESS_SCHEDULING_POLICY` – The process scheduling policy in effect. The possible values are described below.

Configuration

To change the process scheduling policy to use NUMA:

1. In the TIBCO GridServer® Administration Tool, go to **Grid Components > Engines> Engine Configurations** and select the appropriate Engine configuration.
2. Change the Process Scheduling Policy to the desired value, which is described below.

There are three possible process scheduling policy settings in the Engine Configuration:

- **Native** (the default) uses the process scheduling as assigned by Windows.
- **Balanced** makes the best effort to balance the number of Engine processes and processors. The Engine process is allowed to run in all processors in a given processor group. Engines can run in processors from different NUMA nodes.
- **NUMA** makes the best effort to assign Engine processes to the ideal NUMA node. This option is implicitly 'balanced'. An Engine process is only allowed to run in processors in the ideal NUMA node.

Balanced Versus Native Policies

Windows assigns processes to processor groups at creation time in a round-robin fashion. This might be undesirable when you have an unbalanced number of logical processors in processor groups. Processor groups with fewer logical processors might end up with the same number of Engines as processor groups with more logical processors.

The balanced policy attempts to assign Engines proportionally to the number of logical processors in the processor groups. The processor group with more logical processors is assigned more Engines.

The native policy is preferred when the number of logical processors in processor groups is balanced.

Balanced and native policies behave identically in machines with at most one processor group or no support for processor groups.

Extending GridServer

You can extend the GridServer Manager and Engine with Manager and Engine Hooks. A Manager Hook enables you to interface your own Java object directly with the Manager's event processing mechanism and interact with any Server Event. An Engine Hook can perform user-defined operations on Engine startup or termination, or before or after Service invocation.

A Hook consists of two parts: the class implementation of the Hook, and the Hook registration. Register Manager Hooks on the Hook Admin page. Register Engine Hooks with an XML file that is deployed to Engines.

Manager Hooks

You can create Manager Hooks on the Broker or the Director. Depending on where the hook resides, it receives a different subset of the Server Events broadcast by the Manager. See the `ServerEvents` class JavaDoc for available events and the components for which they are relevant. For example, a hook that needs to see the addition or removal of an Engine must run on the Broker, because the `ENGINE_ADDED` and `ENGINE_REMOVED` events are Broker specific.

For details on Manager Hook implementation, see the JavaDoc documentation for the `ServerHook` class. After implementing a Manager Hook, contain its class definition in a JAR file in the shared classes directory (`hooks/`*component*, where *component* is either `broker` or `director`).

To register a Manager Hook in the GridServer Administration Tool, go to **Admin > System Admin > Manager Hooks**. The Hook Admin page enables you to edit, enable, or disable hooks on the Manager. To add a new hook, select **Create New Hook** from the Global Actions list. This opens a Hook Editor in a new window. Enter a filename for the hook XML file, and select whether to apply the hook to the Director or Broker. Enter the name of a class in the hooks directory and click Update Properties to display an updateable property list. After you enter properties, click **Save** to edit the hook or **Cancel** to revert to the last saved version of the hook.

From the Actions control of each hook, you can Enable, Disable, Edit, or Delete an existing hook. Note that you do not need to restart the Manager after deploying the JAR file.

However, if you redeploy a JAR file, you must remove and re-add the hook for any new changes to take effect.

Engine Hooks

Engine Hooks are implemented to perform operations on an Engine:

- On library initialization
- Before or after Service invocation
- On Engine termination

Engine Hooks are useful for performing AOP operations such as setting up a global environment, administering a database or other external component, or collecting metrics on performance or utilization without requiring Services to call a library on their own. You can also use them to do things like restart an Engine after a certain number of invocations.

Engine Hook implementation details are in the JavaDoc documentation for the `EngineHook` class.

Add Engine Hooks by adding the XML and JAR file containing the class definition to a Grid Library. In order for an Engine Hook to always be loaded at Engine startup, you can define it as a Super Grid Library. For more information about Super Grid Libraries, see the *TIBCO GridServer® Administration*. You can use multiple XML files in Grid Libraries (as opposed to the method of having a single `hooks.xml` file, used in previous releases). Note that if you are using Grid Libraries, hooks in the `deploy` directory do not work. For an example of the XML format to use for your Hook, see the `EngineHook` JavaDoc.

Engine Hook Example

The following example initializes a JDBC database.

```
package examples.hook;
import com.datasynapse.gridserver.engine.*;
import java.sql.*;
import java.util.*;
/*
 * This is an example of a hook that initializes data from a database.
 * The property "initialized" can be used to discriminate on Services,
```

```
so that
 * only Engines that have initialized the data will take tasks.
 */
public class JDBCHook extends EngineHook {
    public void initialized() {
        initializeData();
        EngineSession.setProperty("initialized", "true");
    }
    // static method is used by the task
    public static Vector getData() {
        return vData;
    }
 private void initializeData() throws ClassNotFoundException {
        System.out.println("initializing");
        Class cl = Class.forName(getDriver());
        System.out.println("Driver class:" + cl);
 boolean successful = false;
        do {
        try {
                Connection conn = DriverManager.getConnection(getUrl(),
                        getUsername(), getPassword());
                PreparedStatement ps = conn.prepareStatement("select *
from
                    people");
                ResultSet rs = ps.executeQuery();
                System.out.println("rs:" + rs);
                while ( rs.next() )
                    vData.add( rowToLine(rs) );
                successful = true;
            } catch (SQLException e) {
                System.out.println("JDBCHook: failed to retrieve data,
will try
                                    again.");
            }
            if (!successful) {
                try { Thread.sleep(getFrequency()); } catch
(InterruptedException ie) { break; }
            }
} while (!successful);
    }
static String rowToLine( ResultSet input ) throws SQLException {
        StringBuffer buf = new StringBuffer();
        int cols = input.getMetaData().getColumnCount();
        for ( int i=1; i <= cols; i++ ) {
            buf.append(input.getString(i));
            buf.append(' ');
        }
 buf.append('\n');
```

```
        return buf.toString();
    }
    public final void setUrl(String url) {
        _url = url;
    }
    public final String getUrl() {
        return _url;
    }
    public final String getDriver() {
        return _driver;
    }
public final void setDriver(String driver) {
        _driver = driver;
    }
 public final String getUsername() {
        return _user;
    }
    public final void setUsername(String user) {
        _user = user;
    }
    public final String getPassword() {
        return _pass;
    }
 public final void setPassword(String password) {
        _pass = password;
    }
    public final void setFrequency(long frequency) {
        _frequency = frequency;
    }
    public final long getFrequency() {
        return _frequency;
    }
    private String _url;
    private String _driver;
    private String _user;
    private String _pass;
    private long _frequency = 5000;
    private static Vector vData = new Vector();
}
```

The following is also an example of the XML to add to the `hooks.xml` file for the JDBC example given above.

```
<hook class="examples.hook.JDBCHook">
        <property name="username" value="sa"/>
        <property name="password" value=""/>
        <property name="url"
```

```
            value="jdbc:HypersonicSQL:hsql://%server%:2034"/>
        <property name="driver" value="org.hsql.jdbcDriver"/>
</hook>
```

Implementing Engine Hooks as a Grid Library

To implement an Engine Hook as a Grid Library you must create a type of Grid Library called a Super Grid Library. If you are unfamiliar with implementing a Grid Library, review Creating Services.

Super Grid Libraries are very similar to regular Grid Libraries. You still need to use a ZIP archive to contain the code and XML documents just like a regular Grid Library. In the `grid-library.xml`, you must add an attribute to the `<grid-library>` element called `super` which is set to `true`. Additionally, for the code to be used as an Engine Hook, you must specify a folder where the XML descriptor for the hook is contained. Below is an example of a `grid-library.xml` file that has been set up for an Engine Hook:

```
<?xml version="1.0" encoding="UTF-8"?>
<grid-library super="true">
    <grid-library-name>AddPropHook</grid-library-name>
    <grid-library-version>1.0</grid-library-version>
    <jar-path>
        <pathelement>jars</pathelement>
    </jar-path>
    <hooks-path>
        <pathelement>hook</pathelement>
    </hooks-path>
</grid-library>
```

Populating the Super Grid Library is also very similar to a regular Grid Library. The folder structure includes the folder you specified in the `grid-library.xml` for the hook's XML descriptor and the `grid-library.xml` is placed at the top level of the folder structure within the Grid Library. For example, the following is the top level folder structure for an example Engine Hook:

```
    jars/
    AddPropHook.xml
    grid-library.xml
```

Once the Super Grid Library has been constructed, upload it to the Manager in exactly the same way as a standard Grid Library. After it is deployed, the Engines discover it is a Super Grid Library and immediately execute the `initialized()` method.

Task Instrumentation

This Appendix describes the instrumentation phases produced by enabling task instrumentation.

Overview

This Appendix describes the instrumentation phases produced by enabling task instrumentation. To enable task instrumentation, see the *TIBCO GridServer® Administration*.

> ⚠ **Warning** Use task instrumentation only for development, not for production environments. Task instrumentation slows down the Manager significantly, and also requires additional disk space, so it is important to disable it after you finish using it.

Syntax

All instrumentation phases have an absolute time marker, which is the time at the start of the action. Actions also have a relative duration marker, if it is possible to measure the duration. The times are marked according to the client's clock.

Instrumentation phases have the following syntax:

```
[Client] [Action] [Object]
```

Client

The Client of an instrumentation phase can be one of the following:

- Engine

- Driver
- Broker

Action

The Action of an instrumentation phase can be one of the following:

Instrumentation Phase Actions

Action	Description
Send	A send of a message. The absolute time is the start time of the send, and there is no duration value.
Receive	A receive of a message. The absolute time is the end of the retrieval. There is no duration value.
Retrieve	A receive, with a measurement of the duration. The absolute time is the time at which the retrieval started.
Serialize	The conversion of an in-memory object to its serialized format, for transfer to another client.
Deserialize	The conversion of a serialized object to an in-memory object.
Write	The writing of data to a file, typically for DDT (Direct Data Transfer).
Download	The downloading of data from another client.
Call	A call to a user-implemented method.
Load	A native library load.

Object

The Object of an instrumentation phase can be one of the following

Instrumentation Phase Objects

Object	Description
Jar	The JAR file, which is only used for dynamic class loading.
Instance	The instance object, which is either the task or the initialization data.
Input	The input data or message.
Output	The output data or message.
Update	The update data, message, or call.
Checkpoint	Checkpoint data, if checkpointing is enabled.
Library	A native library.
Initialize	The initialization call.
Service	The Service call.
Completed	The callback on completion.
Failed	The callback on failure.
Serialize	The call to a user-implemented native serialize method.
Deserialize	The call to a user-implemented native deserialize method.

Phases

The following is the complete list of all phases.

Driver-side

The following are Driver-side phases:

Instrumentation Driver-side Phases

Phase	Description
Driver Serialize Jar	The serialization of the JAR file when the JAR file is set.
Driver Serialize Instance	The serialization of the Service instance object.
Driver Serialize Input	The serialization of the Service input.
Driver Send Input	The time the Driver sends the input message to the Broker. Keep in mind that more than one input can be sent in one message.
Driver Call Completed	The callback of a successful task.
Driver Call Failed	The callback of a failed task.
Driver Download Output:	The download of output over DDT.
Driver Deserialize Output	
Driver Retrieve Output	The time at which the Driver receives the output message from the Broker. Keep in mind that more than one output can be retrieved in one message.

Engine-side

The following are Engine-side phases:

Instrumentation Engine-side Phases

Phase	Description
Engine Receive Input	The time at which the Engine receives the input message from the Broker.
Engine Deserialize Instance	The deserialization of the Service instance object.
Engine Call Initialize	The initialization call.
Engine Download Update	The download of update data.
Engine Deserialize Update	The deserialization of update data.
Engine Call Update	The update call.
Engine Deserialize Input	The conversion of the serialized input to an in-memory object.
Engine Download Checkpoint	The download of checkpoint data from another Engine.
Engine Call Service	The Service call.
Engine Serialize Output	The serialization of the output.
Engine Send Output	The time at which the Engine sends the output message to the Broker.

Broker-side

The following are Broker-side phases:

Instrumentation Broker-side Phases

Phase	Description
Broker Receive	The time at which the Broker received the input from the Driver.

Phase	Description
Input	
Broker Send Input	The time at which the Broker sent the input to the Engine.
Broker Receive Output	The time at which the Broker received the output from the Engine.
Broker Send Output	The time at which the Broker sent the output to the Driver.
Broker Remove Output	The time at which the Broker removed the output due to the acknowledgment from the Driver.

DDT file write

The following are DDT file write phases:

DDT File Write Phases

Phase	Description
[Client] Write Input:	The input file write.
[Client] Write Output:	The output file write.
[Client] Write Instance	The instance object write.
[Client] Write Jar	The JAR file write.
[Client] Write Update	The update data write.

Client can refer to the Engine, Driver, or Broker.

Native

The following are native phases:

Native Phases

Phase	Description
Engine Load Library	The load of a native dynamic library.
Driver Call Serialize	The native object serialize call.
Driver Call Deserialize	The native object deserialize call.

Example Phases in a Service Execution

The following is a reference example of a typical list of native Service execution phases. Not all possible phases are shown.

Example Service Execution Phases

Phase	Description
Driver Serialize Instance	The serialization of the Service instance object.
Driver Serialize Input	The serialization of the Service input.
Driver Write Input	Driver writes input data to the driver machine file system.
Driver Write Instance	Driver writes task data to the driver machine file system.
Driver Send Input	Driver sends the input message to the Broker. Keep in mind that more than one input can be sent in one message.

Phase	Description
Broker Receive Input:	Broker receives the new task entry.
Broker Send Input	Broker sends the input to the Engine.
Engine Receive Input	Engine receives the input message from the Broker.
Engine Deserialize Instance	Engine deserializes task data.
Engine Load Library	Engine loads all necessary libraries for the task.
Engine Call Initialize	Engine calls init method.
Engine Deserialize Input	Engine deserializes input data.
Engine Call Service	Engine executes the task.
Engine Serialize Output	Engine serializes output data.
Engine Write Output	Engine writes output data to disk.
Engine Send Output	Broker receives the task complete message from Engine.
Broker Receive Output	Broker marks the task complete.

Phase	Description
Broker Send Output	Broker notifies the driver that the task is ready to collect.
Driver Retrieve Output	Driver retrieves completed tasks info from Broker.
Driver Download Output	Driver gets output data from Engine.
Driver Deserialize Output	Driver deserializes output data.
Driver Call Completed	Driver completes the task.
Broker Remove Output	Broker removes the task from the task entry.

The grid-library.dtd

The `grid-library.xml` configuration file in the root of a Grid Library must be a well-formed XML file. The GridServer SDKs include a `grid-library.dtd` file that can be used to validate the XML file. The DTD is also shown in this section

The grid-library.dtd

The following is the `grid-library.dtd` file:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!-- Copyright 2022 TIBCO Software, Inc. All Rights Reserved. -->
<!-- Grid-Library is in the root of the GL. -->
<!ELEMENT grid-library (grid-library-name, grid-library-version?,
arguments?, dependency*, conflict*, jar-path*, lib-path*,
assembly-path*, command-path*, hooks-path*, environment-variables*,
java-system-properties*)>
<!ATTLIST grid-library jre (true|false) "false">
<!ATTLIST grid-library bridge (true|false) "false">
<!ATTLIST grid-library super (true|false) "false">
<!ATTLIST grid-library os CDATA #IMPLIED >
<!ATTLIST grid-library compiler CDATA #IMPLIED >
<!-- The library name. -->
<!ELEMENT grid-library-name (#PCDATA)>
<!-- The version. If not specified, 0 is implied. -->
<!ELEMENT grid-library-version (#PCDATA)>
<!-- Additional arguments to the JVM. -->
<!ELEMENT arguments (property*)>
<!-- A library dependency. Dependencies can be specified by package name
and
     optional version.
     If the version is not specified, the latest version is chosen at
load time.  -->
<!ELEMENT dependency (grid-library-name, grid-library-version?)>
<!-- A library conflict. Indicates that this library conflicts with the
given
     library.
     If this library is NOT a dependency, and grid-library-name="*",
     then it indicates that this library conflicts with all other
libraries
```

```
     (aside from its own dependencies).   -->
<!ELEMENT conflict (grid-library-name)>
<!-- The JAR path. If specified, all jars and classes in the path are
loaded. -->
<!ELEMENT jar-path (pathelement*)>
<!ATTLIST jar-path os CDATA #IMPLIED>
<!ATTLIST jar-path compiler CDATA #IMPLIED>
<!-- An element of a path, typically a directory.  -->
<!ELEMENT pathelement (#PCDATA)>
<!-- Load library path. If not specified, it is assumed that no native
libraries are
     loaded by this GL.
     If this is specified and it the library was not loaded at init
time, the Engine      will restart, adding this path to the current
path. -->
<!ELEMENT lib-path (pathelement*)>
<!ATTLIST lib-path os CDATA #IMPLIED>
<!ATTLIST lib-path compiler CDATA #IMPLIED>
<!-- .NET assembly path.
System.AppDomain.CurrentDomain.AppendPrivatePath(path) will be called on
this path,
    which add it to the lookup location for assemblies.  -->
<!ELEMENT assembly-path (pathelement*)>
<!ATTLIST assembly-path os CDATA #IMPLIED>
<!ATTLIST assembly-path compiler CDATA #IMPLIED>
<!-- The path in which the Engine will search for Command Service
executables.  -->
<!ELEMENT command-path (pathelement*)>
<!ATTLIST command-path os CDATA #IMPLIED>
<!ATTLIST command-path compiler CDATA #IMPLIED>
<!-- Engine hooks library path. Hook will be initialized as libraries
are loaded. -->
<!ELEMENT hooks-path (pathelement*)>
<!ATTLIST hooks-path os CDATA #IMPLIED>
<!ATTLIST hooks-path compiler CDATA #IMPLIED>
<!-- Environment variables to set. Environment variables are set via JNI
immediately prior to executing a task using this library. -->
<!ELEMENT environment-variables (property*)>
<!ATTLIST environment-variables os CDATA #IMPLIED>
<!ATTLIST environment-variables compiler CDATA #IMPLIED>
<!-- A property, used by env vars & system props. -->
<!ELEMENT property (name,value)>
<!-- The name for a property element. -->
<!ELEMENT name (#PCDATA)>
<!-- The value for a property element. -->
<!ELEMENT value (#PCDATA)>
<!-- Java system properties, which are set upon load. -->
<!ELEMENT java-system-properties (property*)>
```

```
<!ATTLIST java-system-properties os CDATA #IMPLIED>
<!ATTLIST java-system-properties compiler CDATA #IMPLIED>
<!-- end of grid-library dtd -->
```

REST API Reference

Web services that can be used by Grid Server components such as Director, Broker, Engine, and Driver are explained here. By using these web services, you can perform different types of operations over the grid components.

APIs for the following classes are available:

- BatchAdmin
- BrokerAdmin
- DriverAdmin
- DriverManager
- EngineAdmin
- EngineDaemonAdmin
- ManagerAdmin
- ServiceAdmin
- UserAdmin
- Version

BatchAdmin

Provides administrative access to the Batches and Batch executions on a Broker. Methods in class BatchAdmin are listed in the following table:

Method	Method Type	Description
batch-definition	POST	Stores the XML contents to a Batch definition on the

Method	Method Type	Description
		Broker.
batch-definition	DELETE	Removes a Batch definition stored on the Broker.
all-batch-execution-info	GET	Retrieves all the Batch execution information stored on the Broker.
all-batch-info	GET	Retrieves information about all the Batch entries on the Broker.
batch-count	GET	Retrieves the total number of Batch entries stored on the Broker.
batch-definition	GET	Retrieves XML contents of the Batch definition.
batch-definition-names	GET	Retrieves the Batch definition names stored on the Broker.
batch-execution-count	GET	Retrieves the total number of Batch executions stored on the Broker.
batch-execution-ids	GET	Retrieves the list of Batch execution IDs stored on the Broker.
batch-execution-info	GET	Retrieves information about a given Batch execution ID stored on the Broker.
batch-execution-info-by-batch-id	GET	Retrieves all the Batch execution information about a given Batch entry stored on the Manager.
batch-ids	GET	Retrieves the list of Batch entry IDs stored on the Broker.
batch-info	GET	Retrieves the Batch entry information about a given Batch entry ID stored on the Broker.
running-batch-	GET	Retrieves the total number of running batch executions

Method	Method Type	Description
execution-count		stored on the Broker.
scheduled-batch-count	GET	Retrieves the total number of scheduled Batch entries on the Broker.
selected-batch-execution-info	POST	Retrieves information about given Batch execution IDs stored on the Broker.
selected-batch-info	POST	Retrieves information about given Batch entry IDs stored on the Broker.
available	GET	Retrieves whether methods are available.
batch	DELETE	Removes the information about a finished or suspended Batch entry.
batch-execution	DELETE	Removes the information about a finished or failed Batch execution.
finished-batch-executions	DELETE	Removes the information about all finished Batch executions on the Broker.
finished-batches	DELETE	Removes the information about all finished Batch entries on the Broker.
resume-batch	PUT	Resumes a suspended Batch entry.
schedule-batch-definitions	GET	Runs the specified Batch definition.
suspend-all-batches	PUT	Suspends all Batch entries running on the Broker.
suspend-batch	PUT	Suspends a scheduled Batch entry.

batch-definition

Stores the XML contents to a Batch definition on the Broker.

Example Request

```
POST http://example.com:8080/livecluster/rest/batch/batch-definition
```

Example Input

Parameters

Parameter Name	Data Type	Description	Sample Value
`name`	String	Name of Batch definition	`NewCalculatorServiceExample`
`contents`	String	XML contents of Batch definition	`<?xml version="1.0" encoding="UTF-8"?> <Batch class="com.datasynapse. gridserver.batch.Batch"> <property name="name" value="CalculatorServiceExample" /> <property name="description" value="Batch Example that runs the Calculator Service" /> <property name="type" value="serial" /> <Schedule class="com.datasynapse. gridserver.batch.Schedule"> <property name="type" value="Immediate" /> </Schedule> <Command class="com.datasynapse. gridserver.batch.command.`

Parameter Name	Data Type	Description	Sample Value
			LogCommand"><property name="message" value="Starting Java Calculator Service Example" /> </Command> <ServiceRunnerReference class="com.datasynapse. gridserver.batch. ServiceRunnerReference"><property name="name" value="JavaCalculatorRunnerExampl e" /> </ServiceRunnerReference> <Command class="com.datasynapse. gridserver.batch.command. LogCommand"><property name="message" value="Done running Java Calculator Service Example" /> </Command> </Batch>

Example Response

```
204 no content
```

Result: The Batch definition is added to the Broker.

batch-definition

Removes a Batch definition stored on the Broker.

Example Request

```
DELETE http://example.com:8080/livecluster/rest/batch/batch-definition
```

Example Input

Parameters

Parameter Name	Data Type	Description	Sample Value
`name`	String	Name of batch definition whose definition needs to be removed from the Broker	`CalculatorServiceExample`

Example Response

```
204 no content
```

Result: Batch definition is removed from the Broker.

all-batch-execution-info

Retrieves all the Batch execution information stored on the Broker.

Example Request

```
GET http://example.com:8080/livecluster/rest/batch/all-batch-execution-info
```

Example Response

```
[
 {
 "name": "CalculatorServiceExample",
 "startTime": 1578568486582,
 "status": "Failed",
 "batchId": 4562207508471809452,
```

```
 "batchExecutionId": 6976777355170490963
 }
]
```

all-batch-info

Retrieves information about all the Batch entries on the Broker.

Example Request

```
GET http://example.com:8080/livecluster/rest/batch/all-batch-info
```

Example Response

```
[
 {
 "name": "CalculatorServiceExample",
 "type": "serial",
 "definition": "CalculatorServiceExample.xml",
 "description": "Batch Example that runs the Calculator Service",
 "suspended": false,
 "status": "Finished",
 "batchId": 4562207508471809452,
 "nextRuntime": null,
 "scheduleType": "Immediate",
 "localFileName": "CalculatorServiceExample.xml",
 "submitTime": 1578568485925
 }
]
```

batch-count

Retrieves the total number of Batch entries stored on the Broker.

Example Request

```
GET http://example.com:8080/livecluster/rest/batch/batch-count
```

Example Response

0

batch-definition

Retrieves XML contents of the Batch definition.

Example Request

```
GET http://example.com:8080/livecluster/rest/batch/batch-definition
```

Example Input

Parameters

Parameter Name	Data Type	Description	Sample Value
`name`	String	Name of batch definition whose XML content needs to be retrieved	`CalculatorServiceExample`

Example Response

```
<?xml version="1.0" encoding="UTF-8"?>
<Batch class="com.datasynapse.gridserver.batch.Batch">
<property name="name" value="CalculatorServiceExample" />
<property name="description" value="Batch Example that runs the
```

```
Calculator Service" />
<property name="type" value="serial" />
<Schedule class="com.datasynapse.gridserver.batch.Schedule"><property
name="type" value="Immediate" />
</Schedule>
<Command
class="com.datasynapse.gridserver.batch.command.LogCommand"><property
name="message" value="Starting Java Calculator Service Example" />
</Command>
<ServiceRunnerReference
class="com.datasynapse.gridserver.batch.ServiceRunnerReference"><propert
y name="name" value="JavaCalculatorRunnerExample"
/></ServiceRunnerReference>
<Command
class="com.datasynapse.gridserver.batch.command.LogCommand"><property
name="message" value="Done running Java Calculator Service Example" />
</Command>
</Batch>
```

batch-definition-names

Retrieves the Batch definition names stored on the Broker.

Example Request

```
GET http://example.com:8080/livecluster/rest/batch/batch-definition-
names
```

Example Response

```
[
    "CalculatorServiceExample.xml"
]
```

batch-execution-count

Retrieves the total number of Batch executions stored on the Broker.

Example Request

```
GET http://example.com:8080/livecluster/rest/batch/batch-execution-count
```

Example Response

0

batch-execution-ids

Retrieves the list of Batch execution IDs stored on the Broker.

Example Request

```
GET http://example.com:8080/livecluster/rest/batch/batch-execution-ids
```

Example Response

```
[
    6976777355170490963
]
```

batch-execution-info

Retrieves information about a given Batch execution ID stored on the Broker.

Example Request

```
GET http://example.com:8080/livecluster/rest/batch/batch-execution-info
```

Example Input

Parameters

Parameter Name	Data Type	Description	Sample Value
`instanceId`	Long	Batch execution ID of the Batch whose information needs to be retrieved	`4534714832987845913`

Example Response

```
{
    "name": "CalculatorServiceExample",
    "startTime": 1578902422107,
    "batchExecutionId": 4534714832987845913,
    "status": "Finished",
    "batchId": 7270548312726024832
}
```

batch-execution-info-by-batch-id

Retrieves all the Batch execution information about a given Batch entry stored on the Manager.

Example Request

```
GET http://example.com:8080/livecluster/rest/batch/batch-execution-info-by-batch-id
```

Example Input

Parameters

Parameter Name	Data Type	Description	Sample Value
`batchId`	Long	Batch entry ID of the Batch whose execution information needs to the retrieved	`4562207508471809452`

Example Response

```
[
    {
        "name": "CalculatorServiceExample",
        "startTime": 1578568486582,
        "status": "Failed",
        "batchId": 4562207508471809452,
        "batchExecutionId": 6976777355170490963
    }
]
```

batch-ids

Retrieves the list of Batch entry IDs stored on the Broker.

Example Request

```
GET http://example.com:8080/livecluster/rest/batch/batch-ids
```

Example Response

```
[
    4562207508471809452
]
```

batch-info

Retrieves the Batch entry information about a given Batch entry ID stored on the Broker.

Example Request

```
GET http://example.com:8080/livecluster/rest/batch/batch-info
```

Example Input

Parameters

Parameter Name	Data Type	Description	Sample Value
`batchId`	Long	Batch entry ID of the Batch whose batch entry information needs to the retrieved	`4562207508471809452`

Example Response

```
{
    "name": "CalculatorServiceExample",
    "type": "serial",
    "definition": "CalculatorServiceExample.xml",
    "description": "Batch Example that runs the Calculator Service",
    "suspended": false,
    "status": "Finished",
    "batchId": 4562207508471809452,
    "nextRuntime": null,
    "scheduleType": "Immediate",
    "localFileName": "CalculatorServiceExample.xml",
    "submitTime": 1578568485925
}
```

running-batch-execution-count

Retrieves the total number of running batch executions stored on the Broker.

Example Request

```
GET http://example.com:8080/livecluster/rest/batch/running-batch-
execution-count
```

Example Response

0

scheduled-batch-count

Retrieves the total number of scheduled Batch entries on the Broker.

Example Request

```
GET http://example.com:8080/livecluster/rest/batch/scheduled-batch-count
```

Example Response

0

selected-batch-execution-info

Retrieves information about given Batch execution IDs stored on the Broker.

Example Request

```
POST http://example.com:8080/livecluster/rest/batch/selected-batch-
execution-info
```

Example Input

Parameters

Parameter Name	Data Type	Description	Sample Value
—	JSON	Array of Batch Execution IDs whose information needs to be retrieved	`[ 39333184909810832, 4534714832987845913 ]`

Example Response

```
[
    {
        "name": "CalculatorServiceExample",
        "startTime": 1578900113522,
        "batchExecutionId": 39333184909810832,
        "status": "Finished",
        "batchId": 58389110300650212 73
    },
    {
        "name": "CalculatorServiceExample",
        "startTime": 1578902422107,
        "batchExecutionId": 4534714832987845913,
        "status": "Finished",
        "batchId": 7270548312726024832
    }
]
```

selected-batch-info

Retrieves information about given Batch entry IDs stored on the Broker.

Example Request

```
POST http://example.com:8080/livecluster/rest/batch/selected-batch-info
```

Example Input

Parameters

Parameter Name	Data Type	Description	Sample Value
—	JSON	Array of Batch Entry IDs whose information needs to be retrieved	`[ 5838911030065021273, 7270548312726024832 ]`

Example Response


```
[
    {
        "name": "CalculatorServiceExample",
        "type": "serial",
        "definition": "CalculatorServiceExample.xml",
        "suspended": false,
        "description": "Batch Example that runs the Calculator Service",
        "status": "Finished",
        "submitTime": 1578900113129,
        "batchId": 5838911030065021273,
        "nextRuntime": null,
        "scheduleType": "Immediate",
        "localFileName": "CalculatorServiceExample.xml"
    },
    {
        "name": "CalculatorServiceExample",
        "type": "serial",
        "definition": "CalculatorServiceExample.xml",
        "suspended": false,
        "description": "Batch Example that runs the Calculator Service",
        "status": "Finished",
        "submitTime": 1578902422059,
        "batchId": 7270548312726024832,
        "nextRuntime": null,
        "scheduleType": "Immediate",
        "localFileName": "CalculatorServiceExample.xml"
    }
]
```


available

Retrieves whether methods are available.

Example Request

```
GET http://example.com:8080/livecluster/rest/batch/available
```

Example Response

`True` or `False`

batch

Removes the information about a finished or suspended Batch entry.

Example Request

```
DELETE http://example.com:8080/livecluster/rest/batch/batch
```

Example Input

Parameters

Parameter Name	Data Type	Description	Sample Value
`batchIds`	Long	Batch Entry ID of Batch whose information needs to be removed	`926167698331434208`

Example Response

```
204 no content
```

Result: Information about the finished or suspended Batch entry is removed.

batch-execution

Removes the information about a finished or failed Batch execution.

Example Request

```
DELETE http://example.com:8080/livecluster/rest/batch/batch-execution
```

Example Input

Parameters

Parameter Name	Data Type	Description	Sample Value
`instanceId`	Long	Execution ID of Batch whose information needs to be removed	`39333184909810832`

Example Response

```
204 no content
```

Result: Information about the finished or failed Batch execution is removed.

finished-batch-executions

Removes the information about all finished Batch executions on the Broker.

Example Request

```
DELETE http://example.com:8080/livecluster/rest/batch/finished-batch-
executions
```

Example Response

```
204 no content
```

Result: Information about all finished Batch executions is removed from the Broker.

finished-batches

Removes the information about all finished Batch entries on the Broker.

Example Request

```
DELETE http://example.com:8080/livecluster/rest/batch/finished-batches
```

Example Response

```
204 no content
```

Result: Information about all finished Batch entries on the Broker is removed.

resume-batch

Resumes a suspended Batch entry.

Example Request

```
PUT http://example.com:8080/livecluster/rest/batch/resume-batch
```

Example Input

Parameters

Parameter Name	Data Type	Description	Sample Value
`bacthId`	Long	Batch Entry ID of the Batch that needs to be resumed	`5838911030065021273`

Example Response

```
204 no content
```

Result: The suspended Batch entry is resumed.

schedule-batch-definitions

Runs the specified Batch definition.

Example Request

```
GET http://example.com:8080/livecluster/rest/batch/schedule-batch-definition
```

Example Input

Parameters

Parameter Name	Data Type	Description	Sample Value
`name`	String	Name of the batch definition that must be run	`CalculatorServiceExample`

Example Response

`3668132062277102496`.

Result: The running Batch Definition ID is returned.

suspend-all-batches

Suspends all Batch entries running on the Broker.

Example Request

```
PUT http://example.com:8080/livecluster/rest/batch/suspend-all-batches
```

Example Response

```
204 no content
```

Result: All Batch entries running on the Broker are suspended.

suspend-batch

Suspends a scheduled Batch entry.

Example Request

```
PUT http://example.com:8080/livecluster/rest/batch/suspend-batch
```

Example Input

Parameters

Parameter Name	Data Type	Description	Sample Value
`batchId`	Long	Batch entry ID of the Batch that needs to be suspended	`5838911030065021273`

Example Response

```
204 no content
```

Result: The scheduled Batch entry is suspended.

BrokerAdmin

BrokerAdmin APIs provide administrative access to Brokers on a Director. They are listed in the following table:

Method	Method Type	Description
service-discriminator	POST	Adds the service discriminator.
service-discriminator	DELETE	Deletes the Service Discriminator.
all-broker-info	GET	Retrieves information about all Brokers logged in to the Director.
broker-count	GET	Retrieves the total number of Brokers logged in to the Director.
broker-info	GET	Retrieves information about a given Broker.
engine-router	GET	Retrieves the Engine Router assigned to this Broker.
service-discriminator	GET	Retrieves the PropertyConditionSet used by the Service Discriminator.
service-discriminator-names	GET	Retrieves the names of all Service Discriminators.
shared-brokers	GET	Retrieves Broker sharing.
available	GET	Retrieves whether the methods are available.
driver-weight	PUT	Sets the Driver distribution weighing relative to other Brokers.
engine-router	POST	Sets an Engine router for a given Broker.
engine-weight	PUT	Sets the Engine distribution weighting relative to other Brokers.
maximum-engines	PUT	Sets the maximum number of Engines the Broker can

Method	Method Type	Description
		manage.
min-idle-home-engines	PUT	Sets the minimum number of Idle Home Machines that the Broker is left to manage.
minimum-engines	PUT	Sets the minimum number of Engines the Broker is left to manage.
shared-brokers	PUT	Sets the Broker sharing.

service-discriminator

Adds the service discriminator.

Example Request

```
POST http://example.com:8080/livecluster/rest/broker/service-discriminator
```

Example Input

Parameters

Parameter Name	Data Type	Description	Sample Value
`name`	String	Name of the Service Discriminator that needs to be added	`TestDiscriminatorNew`

Parameter Name	Data Type	Description	Sample Value

—	JSON	Data of Service Discriminator that needs to be added	{
			"descriptionCondition":{
			"sets": [],
			"conditions": [
			{
			"name": "serviceName",
			"comparison": 1,
			"value": "Lin",
			"nullCompare": false
			}
			],
			"type": 0
			},
			"engineCondition":{
			"sets": [],
			"conditions": [
			{
			"name": "os",
			"comparison": 3,
			"value": "linux64",
			"nullCompare": false
			}
			],
			"type": 0
			}
			}

Example Response

```
204 no content
```

Result: A new Service Discriminator is added.

service-discriminator

Deletes the Service Discriminator.

Example Request

```
DELETE http://example.com:8080/livecluster/rest/broker/service-
discriminator
```

Example Input

Parameters

Parameter Name	Data Type	Description	Sample Value
`name`	String	Name of the Service Discriminator that needs to be deleted	`TestDiscriminatorNew`

Example Response

```
204 no content
```

Result: The specified Service Discriminator is deleted.

all-broker-info

Retrieves information about all Brokers logged in to the Director.

Example Request

```
GET http://example.com:8080/livecluster/rest/broker/all-broker-info
```

Example Response


```
[
        {
                "minIdleHomeEngines": 0,
                "name": "abcd",
                "baseUrl": "http://win64vm091:8000/livecluster",
                "engineWeight": 1.0,
                "driverWeight": 1.0,
                "maxEngines": 2500,
                "minEngines": 0,
                "brokerId": 899860077,
                "driverCount": 1,
                "failover": false,
                "engineCount": 0,
                "busyEngineCount": 0,
                "engineRoutingConditions": [
                        {
                                "name": "AMIAvailZone",
                                "comparison": 1,
                                "value": "fvszgvsa",
                                "nullCompare": false
                        }
                ],
                "engineRoutingComparators": [
                        {
                                "methodNum": 0,
                                "value": "fvszgvsa",
                                "name": "AMIAvailZone",
                                "nullComparison": false
                        }
                ],
                "driverRoutingComparators": null,
                "driverRoutingConditions": null,
                "hostname": "http://win64vm091:8000/livecluster"
        }
]
```


broker-count

Retrieves the total number of Brokers logged in to the Director.

Example Request

```
GET http://example.com:8080/livecluster/rest/broker/broker-count
```

Example Response

1.

broker-info

Retrieves information about a given Broker.

Example Request

```
GET http://example.com:8080/livecluster/rest/broker/broker-info
```

Example Input

Parameters

Parameter Name	Data Type	Description	Sample Value
`id`	Long	Broker ID of Broker whose information needs to be retrieved	`899860077`

Example Response

```
{
    "minIdleHomeEngines": 0,
        "name": "abcd",
        "baseUrl": "http://win64vm091:8000/livecluster",
```

```
        "engineWeight": 1.0,
        "driverWeight": 1.0,
        "maxEngines": 2500,
        "minEngines": 0,
        "brokerId": 899860077,
        "driverCount": 1,
        "failover": false,
        "engineCount": 0,
        "busyEngineCount": 0,
        "engineRoutingConditions": [
                {
                        "name": "AMIAvailZone",
                        "comparison": 1,
                        "value": "fvszgvsa",
                        "nullCompare": false
                }
        ],
        "engineRoutingComparators": [
                {
                        "methodNum": 0,
                        "value": "fvszgvsa",
                        "name": "AMIAvailZone",
                        "nullComparison": false
                }
        ],
        "driverRoutingComparators": null,
        "driverRoutingConditions": null,
        "hostname": "http://win64vm091:8000/livecluster"
}
```


engine-router

Retrieves the Engine Router assigned to this Broker.

Example Request

```
GET http://example.com:8080/livecluster/rest/broker/engine-router
```

Example Input

Parameters

Parameter Name	Data Type	Description	Sample Value
`id`	Long	Broker ID of Broker for which information of assigned Engine Router needs to be retrieved	`899860077`

Example Response

```
{
    "sets": null,
    "conditions": [
        {
            "name": "AMIAvailZone",
            "comparison": 1,
            "value": "fvszgvsa",
            "nullCompare": false
        }
    ],
    "type": 0
}
```

service-discriminator

Retrieves the PropertyConditionSet used by the Service Discriminator.

Example Request

```
GET http://example.com:8080/livecluster/rest/broker/service-
discriminator
```

Example Input

Parameters

Parameter Name	Data Type	Description	Sample Value
`name`	String	Name of Service Discriminator for which PropertyConditionSet needs to be retrieved	`TestDiscriminator`

Example Response

```
[
        {
                "sets": [],
                "conditions": [
                        {
                                "name": "serviceName",
                                "comparison": 1,
                                "value": "Lin",
                                "nullCompare": false
                        }
                ],
                "type": 0
        },
        {
                "sets": [],
                "conditions": [
                        {
                                "name": "os",
                                "comparison": 3,
                                "value": "linux64",
                                "nullCompare": false
                        }
                ],
                "type": 0
        },
        {
                "sets": [],
                "conditions": [],
                "type": 0
```

```
        }
]
```

service-discriminator-names

Retrieves the names of all Service Discriminators.

Example Request

```
GET http://example.com:8080/livecluster/rest/broker/service-
discriminator-names
```

Example Response

```
 [
    "testdiscriminator"
 ]
```

shared-brokers

Retrieves Broker sharing.

Example Request

```
GET http://example.com:8080/livecluster/rest/broker/shared-brokers
```

Example Input

Parameters

Parameter Name	Data Type	Description	Sample Value
`id`	Long	Broker ID of Broker for which Broker sharing needs to be retrieved	`899860077`

Example Response

```
1
```

available

Retrieves whether the methods are available.

Example Request

```
GET http://example.com:8080/livecluster/rest/broker/available
```

Example Response

`True` or `False`

driver-weight

Sets the Driver distribution weighing relative to other Brokers.

Example Request

```
PUT http://example.com:8080/livecluster/rest/broker/driver-weight
```

Example Input

Parameters

Parameter Name	Data Type	Description	Sample Value
`id`	Long	Broker ID of Broker for which Driver distribution weighting needs to be set	`1961263499`
`weight`	Double	Driver weight	`2`

Example Response

```
204 no content
```

Result: The driver-weight value is updated for the corresponding Broker.

engine-router

Sets an Engine router for a given Broker.

Example Request

```
POST http://example.com:8080/livecluster/rest/broker/engine-router
```

Example Input

Parameters

Parameter Name	Data Type	Description	Sample Value
`id`	Long	Broker ID of Broker for which Engine Router needs to be set	`1580108171`

—	JSON	Engine Router data	<pre>{
			  "sets": null,
			  "conditions": [
			  {
			  "name": "os",
			  "comparison": 1,
			  "value": "win64",
			  "nullCompare": false
			  },
			  {
			  "name": "osName",
			  "comparison": 1,
			  "value": "WIN2K12",
			  "nullCompare": false
			  }
			  ],
			  "type": 0
			}</pre>

Example Response

`204 no content`

Result: Engine router is set for the specified Broker.

engine-weight

Sets the Engine distribution weighting relative to other Brokers.

Example Request

```
PUT http://example.com:8080/livecluster/rest/broker/engine-weight
```

Example Input

Parameters

Parameter Name	Data Type	Description	Sample Value
`id`	Long	Broker ID of Broker for which Engine Distribution weighting needs to be set	`1961263499`
`weight`	Double	Engine weight	`2`

Example Response

```
207 no content
```

Result: Engine weight is updated for the specified Broker.

maximum-engines

Sets the maximum number of Engines the Broker can manage.

Example Request

```
PUT http://example.com:8080/livecluster/rest/broker/maximum-engines
```

Example Input

Parameters

Parameter Name	Data Type	Description	Sample Value
`id`	Long	Broker ID of Broker	`1580108171`
`maxEngines`	Int	Maximum number of Engines that the given Broker can manage	`6`

Example Response

```
207 no content
```

Result: The maximum number of Engines that the Broker can Manage is set.

min-idle-home-engines

Sets the minimum number of Idle Home Machines that the Broker is left to manage.

Example Request

```
PUT http://example.com:8080/livecluster/rest/broker/min-idle-home-engines
```

Example Input

Parameters

Parameter Name	Data Type	Description	Sample Value
`id`	Long	Broker ID of Broker	`1580108171`
`minEngines`	Int	Minimum number of Idle Home Engines on the given Broker	`1`

Example Response

```
204 no content
```

Result: The minimum number of Idle Home Engines for the Broker is set.

minimum-engines

Sets the minimum number of Engines the Broker is left to manage.

Example Request

```
PUT http://example.com:8080/livecluster/rest/broker/minimum-engines
```

Example Input

Parameters

Parameter Name	Data Type	Description	Sample Value
`id`	Long	Broker ID of Broker	`1580108171`
`minEngines`	Int	Minimum number of Engines that the given Broker can manage	`5`

Example Response

```
204 no content
```

Result: The minimum number of Engines for the Broker is set.

shared-brokers

Sets the Broker sharing.

Example Input

```
PUT http://example.com:8080/livecluster/rest/broker/shared-brokers
```

Example Input

Parameters

Parameter Name	Data Type	Description	Sample Value
`id`	Long	Broker ID of Broker	`1580108171`
`brokerSharing`	String	Broker ID that needs to be shared	`1961263499`

Example Response

```
204 no content
```

Result: Broker sharing is set.

DriverAdmin

Provides administrative access to the Drivers on a Broker. Methods in class DriverAdmin are listed in the following table:

Method	Method Type	Description
all-driver-info	GET	Retrieves information about all Drivers logged in to the Broker.
driver-count	GET	Retrieves the total number of Drivers logged in to the Broker.
driver-info	GET	Retrieves information about Drivers with a given hostname.
available	GET	Retrieves whether the methods are available.

all-driver-info

Retrieves information about all Drivers logged in to the Broker.

Example Request

```
GET http://example.com:8080/livecluster/rest/driver/all-driver-info
```

Example Response

```
{
        "properties": [
                {
                        "name": "Username",
                        "value": "internal"
                },
                {
                        "name": "maxPriority",
                        "value": "None"
                },
                {
                        "name": "Hostname",
                        "value": "10.128.88.91"
                },
                {
                        "name": "roles",
```

```
                        "value": "internal"
                },
                {
                        "name": "logToSystem",
                        "value": "true"
                },
                {
                        "name": "baseDir",
                        "value": "F:/datasynapse/manager-data"
                },
                {
                        "name": "ClientJavaVersion",
                        "value": "7.1.0.172681"
                },
                {
                        "name": "Version",
                        "value": "7.1.0.172681"
                }
        ],
        "username": "internal",
        "serviceCount": 0,
        "loginDate": 1578562190177,
        "hostname": "10.128.88.91"
}
```


driver-count

Retrieves the total number of Drivers logged in to the Broker.

Example Request

```
GET http://example.com:8080/livecluster/rest/driver/driver-count
```

Example Response

```
1
```

driver-info

Retrieves information about Drivers with a given hostname.

Example Request

```
GET http://example.com:8080/livecluster/rest/driver/driver-info
```

Example Input

Parameters

Parameter Name	Data Type	Description	Sample Value
`hostname`	String	Name of the host whose driver information needs to be retrieved	`10.128.88.91`

Example Response

```
{
        "properties": [
            {
                "name": "Username",
                "value": "internal"
            },
            {
                "name": "maxPriority",
                "value": "None"
            },
            {
                "name": "Hostname",
                "value": "10.128.88.91"
            },
            {
                "name": "roles",
                "value": "internal"
            },
            {
                "name": "logToSystem",
```

```
                "value": "true"
            },
            {
                "name": "baseDir",
                "value": "F:/datasynapse/manager-data"
            },
            {
                "name": "ClientJavaVersion",
                "value": "7.1.0.172681"
            }
            {
                "name": "Version",
                "value": "7.1.0.172681"
            }
        ],
        "username": "internal",
        "serviceCount": 0,
        "loginDate": 1578562190177,
        "hostname": "10.128.88.91"
}
```


available

Retrieves whether the methods are available.

Example Request

```
GET http://example.com:8080/livecluster/rest/driver/available
```

Example Response

`True` or `False`

EngineAdmin

EngineAdmin APIs provide administrative access to Engines on a Broker. They are listed in the following table:

Method	Method Type	Description
all-engine-info	GET	Retrieves information about all Engines logged in to the Broker.
busy-engine-count	GET	Retrieves the total number of Engines logged in to the Broker and are currently executing a task.
engine-count	GET	Retrieves the total number of Engines logged in to the Broker.
engine-ids	GET	Retrieves the list of Engine IDs logged in to the Broker.
engine-info	GET	Retrieves information about a given Engine.
engine-info-by-properties	POST	Retrieves information about Engines logged in to the Broker that match the Engine Condition.
log-url-list	GET	Retrieves the log file URLs on a given Engine when the Engine logging is enabled on the Server.
selected-engine-info	POST	Retrieves information, which includes all instances for the given IDs, about given Engines logged in to the Broker.
available	GET	Retrieves whether the methods are available.
kill-all-engines	PUT	Logs out all the Engines from the Broker.
kill-engine	PUT	Logs out a given Engine from the Broker.
park-engines	POST	Sets Engine's parked flag to true for passed Engine IDs.
unpark-engines	POST	Sets Engine's parked flag to false for passed Engine IDs.
parked-engines	GET	Retrieves the list of Engine IDs that have the parked flag set to true.

all-engine-info

Retrieves information about all Engines logged in to the Broker.

Example Request

```
GET http://example.com:8080/livecluster/rest/engine/all-engine-info
```

Example Response

```
{
        "instance": "0",
        "properties": [
                {
                        "name": "AMIProductCode",
                        "value": "null"
                },
                {
                        "name": "dataDir",
                        "value":
"C:\\TIBCO\\DataSynapse\\Engine\\data\\win64vm091-0\\ddt"
                },
                {
                        "name": "availableDiskInMB",
                        "value": "12503261184"
                },
                {
                        "name": "cpuIdString",
                        "value": "GenuineIntel"
                },
                {
                        "name": "osVersion",
                        "value": "6.1"
                },
                {
                        "name": "workUrl",
                        "value":
"http://10.128.88.91:27159/work/win64vm091-0"
                },
                {
                        "name": "QuarantineStatus",
                        "value": "New Engine"
                },
```

```
                {
                        "name": "CUDAGlobalMemory",
                        "value": "null"
                },
                {
                        "name": "osUsername",
                        "value": "SYSTEM"
                },
                {
                        "name": "CUDAProcessors",
                        "value": "null"
                },
                {
                        "name": "CUDAFirstGPUName",
                        "value": "null"
                },
                {
                        "name": "dataUrl",
                        "value":
"http://10.128.88.91:27159/data//win64vm091-0/ddt/"
                },
                {
                        "name": "additionalPlatforms",
                        "value": "null"
                },
                {
                        "name": "MICMemory",
                        "value": "null"
                },
                {
                        "name": "ClientJavaVersion",
                        "value": "7.1.0.172681"
                },
                {
                        "name": "homeBrokers",
                        "value": ""
                },
                {
                        "name": "configurationName",
                        "value": "win64:default"
                },
                {
                        "name": "AMIInstanceId",
                        "value": "null"
                },
                {
                        "name": "cpuSocketCount",
```

```
                        "value": "1"
                },
                {
                        "name": "IP",
                        "value": "10.128.88.91"
                },
                {
                        "name": "totalMemInKB",
                        "value": "2096696"
                },
                {
                        "name": "lastUpdated",
                        "value": "Mon Jan 06 06:54:35 PST 2020"
                },
                {
                        "name": "guid",
                        "value": "deadbeef8091"
                },
                {
                        "name": "RTLDGlobal",
                        "value": "null"
                },
                {
                        "name": "CUDADevices",
                        "value": "0"
                },
                {
                        "name": "GridLibs",
                        "value": "python-win64-
vc14,3.7.0;SpeedLink,7.0.0.165519;pybridge-win64-vc14,7.1"
                },
                {
                        "name": "multicore",
                        "value": "false"
                },
                {
                        "name": "os",
                        "value": "win64"
                },
                {
                        "name": "MICModel",
                        "value": "null"
                },
                {
                        "name": "NETFrameworkVersions",
                        "value": "4.5,4.5FullProfile"
                },
```

```
                {
                        "name": "MICCores",
                        "value": "null"
                },
                {
                        "name": "username",
                        "value": "win64vm091"
                },
                {
                        "name": "GridLibUpdateTime",
                        "value": "1578322476609"
                },
                {
                        "name": "id",
                        "value": "4848394891525229394"
                },
                {
                        "name": "freeMemInKB",
                        "value": "163244"
                },
                {
                        "name": "cpuNo",
                        "value": "1"
                },
                {
                        "name": "pid",
                        "value": "792"
                },
                {
                        "name": "cpuCoreCount",
                        "value": "1"
                },
                {
                        "name": "AMIAvailZone",
                        "value": "null"
                },
                {
                        "name": "cpuMHz",
                        "value": "2597.0"
                },
                {
                        "name": "CUDAVersion",
                        "value": "null"
                },
                {
                        "name": "cpuTotal",
                        "value": "2794.447000"
```

```
                },
                {
                        "name": "Version",
                        "value": "7.1.0.172681"
                },
                {
                        "name": "freeDiskInMB",
                        "value": "12503261184"
                },
                {
                        "name": "MICDevices",
                        "value": "null"
                },
                {
                        "name": "instance",
                        "value": "0"
                },
                {
                        "name": "crtVersions",
                        "value": "VC12,VC14,VC15,VC16"
                },
                {
                        "name": "HighestNumaNode",
                        "value": "0"
                },
                {
                        "name": "homeDir",
                        "value": "C:\\TIBCO\\DataSynapse\\Engine"
                },
                {
                        "name": "ProcessSchedulingPolicy",
                        "value": "native"
                },
                {
                        "name": "NETFramework",
                        "value": "true"
                },
                {
                        "name": "gridLibrary",
                        "value": "null"
                },
                {
                        "name": "cpuThreadCount",
                        "value": "1"
                },
                {
                        "name": "osName",
```

```
                        "value": "WIN2K12"
                }
        ],
        "username": "win64vm091",
        "engineId": 4848394891525229394,
        "serviceId": -1,
        "invocationId": -1,
        "busy": false,
        "elapsedTime": -1
}
```

busy-engine-count

Retrieves the total number of Engines logged in to the Broker and are currently executing a task.

Example Request

```
GET http://example.com:8080/livecluster/rest/engine/busy-engine-count
```

Example Response

1.

Result: The number of Engines that are busy is returned.

engine-count

Retrieves the total number of Engines logged in to the Broker.

Example Request

```
GET http://example.com:8080/livecluster/rest/engine/engine-count
```

Example Response

1.

Result: The total number of Engines that are logged in to the Broker is returned.

engine-ids

Retrieves the list of Engine IDs logged in to the Broker.

Example Request

```
GET http://example.com:8080/livecluster/rest/engines/engine-ids
```

Example Response

```
[
    4848394891525229394,
    4848394891525229394,
    4848394891525229394,
    4848394891525229394,
    4848394891525229394,
    4848394891525229394,
    4848394891525229394,
    4848394891525229394,
    4848394891525229394,
    4848394891525229394
]
```

engine-info

Retrieves information about a given Engine.

Example Request

```
GET http://example.com:8080/livecluster/rest/engine/engine-info
```

Example Input

Parameters

Parameter Name	Data Type	Description	Sample Value
`id`	Long	ID of Engine whose information needs to be retrieved	`4848394891525229394`
`instance`	String	Engine instance whose information needs to be retrieved	`1`

Example Response

```
{
    "instance": "0",
    "properties": [
        {
            "name": "AMIProductCode",
            "value": "null"
        },
        {
            "name": "dataDir",
            "value": "C:\\TIBCO\\DataSynapse\\Engine\\data\\win64vm091-
0\\ddt"
        },
        {
            "name": "availableDiskInMB",
            "value": "11030"
        },
        {
            "name": "cpuIdString",
            "value": "GenuineIntel"
        },
        {
            "name": "osVersion",
            "value": "6.1"
        },
        {
            "name": "workUrl",
            "value": "http://10.128.88.91:27159/work/win64vm091-0"
```

```
        },
        {
            "name": "QuarantineStatus",
            "value": "New Engine"
        },
        {
            "name": "CUDAGlobalMemory",
            "value": "null"
        },
        {
            "name": "osUsername",
            "value": "SYSTEM"
        },
        {
            "name": "CUDAProcessors",
            "value": "null"
        },
        {
            "name": "CUDAFirstGPUName",
            "value": "null"
        },
        {
            "name": "dataUrl",
            "value": "http://10.128.88.91:27159/data//win64vm091-0/ddt/"
        },
        {
            "name": "additionalPlatforms",
            "value": "null"
        },
        {
            "name": "MICMemory",
            "value": "null"
        },
        {
            "name": "ClientJavaVersion",
            "value": "7.1.0.172681"
        },
        {
            "name": "homeBrokers",
            "value": ""
        },
        {
            "name": "configurationName",
            "value": "win64:default"
        },
        {
            "name": "AMIInstanceId",
            "value": "null"
```

```
        },
        {
            "name": "cpuSocketCount",
            "value": "1"
        },
        {
            "name": "IP",
            "value": "10.128.88.91"
        },
        {
            "name": "totalMemInKB",
            "value": "2096696"
        },
        {
            "name": "lastUpdated",
            "value": "Wed Jan 08 23:08:10 PST 2020"
        },
        {
            "name": "guid",
            "value": "deadbeef8091"
        },
        {
            "name": "RTLDGlobal",
            "value": "null"
        },
        {
            "name": "CUDADevices",
            "value": "0"
        },
        {
            "name": "GridLibs",
            "value": "python-win64-
vc14,3.7.0;SpeedLink,7.0.0.165519;pybridge-win64-vc14,7.1"
        },
        {
            "name": "multicore",
            "value": "false"
        },
        {
            "name": "os",
            "value": "win64"
        },
        {
            "name": "MICModel",
            "value": "null"
        },
        {
            "name": "NETFrameworkVersions",
```

```
            "value": "4.5,4.5FullProfile"
        },
        {
            "name": "MICCores",
            "value": "null"
        },
        {
            "name": "username",
            "value": "win64vm091"
        },
        {
            "name": "GridLibUpdateTime",
            "value": "1578553692407"
        },
        {
            "name": "id",
            "value": "4848394891525229394"
        },
        {
            "name": "freeMemInKB",
            "value": "1019692"
        },
        {
            "name": "cpuNo",
            "value": "1"
        },
        {
            "name": "pid",
            "value": "5012"
        },
        {
            "name": "cpuCoreCount",
            "value": "1"
        },
        {
            "name": "AMIAvailZone",
            "value": "null"
        },
        {
            "name": "cpuMHz",
            "value": "2597.0"
        },
        {
            "name": "CUDAVersion",
            "value": "null"
        },
        {
            "name": "cpuTotal",
```

```
            "value": "1676.7"
        },
        {
            "name": "Version",
            "value": "7.1.0.172681"
        },
        {
            "name": "freeDiskInMB",
            "value": "11030"
        },
        {
            "name": "MICDevices",
            "value": "null"
        },
        {
            "name": "instance",
            "value": "0"
        },
        {
            "name": "crtVersions",
            "value": "VC12,VC14,VC15,VC16"
        },
        {
            "name": "HighestNumaNode",
            "value": "0"
        },
        {
            "name": "homeDir",
            "value": "C:\\TIBCO\\DataSynapse\\Engine"
        },
        {
            "name": "ProcessSchedulingPolicy",
            "value": "native"
        },
        {
            "name": "NETFramework",
            "value": "true"
        },
        {
            "name": "gridLibrary",
            "value": "null"
        },
        {
            "name": "cpuThreadCount",
            "value": "1"
        },
        {
            "name": "osName",
```

```
            "value": "WIN2K12"
        }
    ],
    "username": "win64vm091",
    "engineId": 4848394891525229394,
    "serviceId": -1,
    "invocationId": -1,
    "busy": false,
    "elapsedTime": -1
}
```

engine-info-by-properties

Retrieves information about Engines logged in to the Broker that match the Engine Condition.

Example Request

```
POST http://example.com:8080/livecluster/rest/engine/engine-info-by-properties
```

Example Input

Parameters

Parameter Name	Data Type	Description	Sample Value
—	JSON	Engine condition data depending on which Engines are retrieved	`{ "sets": null, "conditions": [ { "name": "IP",`

Parameter Name	Data Type	Description	Sample Value

			<pre>"comparison": 1,
			
			"value":
			"10.128.88.142",
			
			"nullCompare":
			false
			        },
			        {
			
			"name": "os",
			
			"comparison": 1,
			
			"value": "win64",
			
			"nullCompare":
			false
			        }
			    ],
			    "type": 0
			}</pre>

Example Response

```
[
    {
        "instance": "0",
        "properties": [
            {
                "name": "AMIProductCode",
                "value": "null"
            },
            {
                "name": "dataDir",
                "value":
"C:\\TIBCO\\DataSynapse\\Engine\\data\\win64vm142-0\\ddt"
            },
            {
                "name": "availableDiskInMB",
```

```
                "value": "13793"
            },
            {
                "name": "cpuIdString",
                "value": "GenuineIntel"
            },
            {
                "name": "osVersion",
                "value": "6.1"
            },
            {
                "name": "workUrl",
                "value": "http://10.128.88.142:27159/work/win64vm142-0"
            },
            {
                "name": "QuarantineStatus",
                "value": "New Engine"
            },
            {
                "name": "CUDAGlobalMemory",
                "value": "null"
            },
            {
                "name": "osUsername",
                "value": "SYSTEM"
            },
            {
                "name": "CUDAProcessors",
                "value": "null"
            },
            {
                "name": "CUDAFirstGPUName",
                "value": "null"
            },
            {
                "name": "dataUrl",
                "value": "http://10.128.88.142:27159/data//win64vm142-
0/ddt/"
            },
            {
                "name": "additionalPlatforms",
                "value": "null"
            },
            {
                "name": "MICMemory",
                "value": "null"
            },
            {
```

```
                "name": "ClientJavaVersion",
                "value": "7.1.0.172681"
            },
            {
                "name": "homeBrokers",
                "value": ""
            },
            {
                "name": "configurationName",
                "value": "win64:default"
            },
            {
                "name": "AMIInstanceId",
                "value": "null"
            },
            {
                "name": "cpuSocketCount",
                "value": "1"
            },
            {
                "name": "IP",
                "value": "10.128.88.142"
            },
            {
                "name": "totalMemInKB",
                "value": "2096696"
            },
            {
                "name": "lastUpdated",
                "value": "Tue Jan 14 05:37:32 EST 2020"
            },
            {
                "name": "guid",
                "value": "deadbeef8142"
            },
            {
                "name": "RTLDGlobal",
                "value": "null"
            },
            {
                "name": "CUDADevices",
                "value": "0"
            },
            {
                "name": "GridLibs",
                "value": "calculator,1.0.0.1;python-win64-
vc14,3.7.0;SpeedLink,7.0.0.165519;pybridge-win64-vc14,7.1"
            },
```

```
            {
                "name": "multicore",
                "value": "false"
            },
            {
                "name": "os",
                "value": "win64"
            },
            {
                "name": "MICModel",
                "value": "null"
            },
            {
                "name": "NETFrameworkVersions",
                "value": "3.5,3.5SP1,4,4FullProfile"
            },
            {
                "name": "MICCores",
                "value": "null"
            },
            {
                "name": "username",
                "value": "win64vm142"
            },
            {
                "name": "GridLibUpdateTime",
                "value": "1578998255589"
            },
            {
                "name": "id",
                "value": "6153431282294490153"
            },
            {
                "name": "freeMemInKB",
                "value": "1356648"
            },
            {
                "name": "cpuNo",
                "value": "1"
            },
            {
                "name": "pid",
                "value": "2072"
            },
            {
                "name": "cpuCoreCount",
                "value": "1"
            },
```

```
            {
                "name": "AMIAvailZone",
                "value": "null"
            },
            {
                "name": "cpuMHz",
                "value": "2597.0"
            },
            {
                "name": "CUDAVersion",
                "value": "null"
            },
            {
                "name": "cpuTotal",
                "value": "2704.3"
            },
            {
                "name": "Version",
                "value": "7.1.0.172681"
            },
            {
                "name": "freeDiskInMB",
                "value": "13793"
            },
            {
                "name": "MICDevices",
                "value": "null"
            },
            {
                "name": "instance",
                "value": "0"
            },
            {
                "name": "crtVersions",
                "value": "VC12,VC14,VC15,VC16"
            },
            {
                "name": "HighestNumaNode",
                "value": "0"
            },
            {
                "name": "homeDir",
                "value": "C:\\TIBCO\\DataSynapse\\Engine"
            },
            {
                "name": "ProcessSchedulingPolicy",
                "value": "native"
            },
```

```
            {
                "name": "NETFramework",
                "value": "true"
            },
            {
                "name": "gridLibrary",
                "value": "null"
            },
            {
                "name": "cpuThreadCount",
                "value": "1"
            },
            {
                "name": "osName",
                "value": "WIN2K12"
            }
        ],
        "username": "win64vm142",
        "engineId": 6153431282294490153,
        "serviceId": -1,
        "invocationId": -1,
        "busy": false,
        "elapsedTime": -1
    }
```


log-url-list

Retrieves the log file URLs on a given Engine when the Engine logging must is enabled on the Server.

Example Request

```
GET http://example.com:8080/livecluster/rest/engine/log-url-list
```

Example Input

Parameters

Parameter Name	Data Type	Description	Sample Value
`id`	Long	Engine ID for which logs need to be retrieved	`4848394891525229394`
`instance`	String	Instance ID for corresponding Engine ID	`0`

Example Response

```
[
    "http://10.128.88.91:27159/work/win64vm091-0/log/engine-
0.01.08.2020-23.08.11.log",
    "http://10.128.88.91:27159/work/win64vm091-0/log/engine-
0.01.06.2020-06.54.36.log",
    "http://10.128.88.91:27159/work/win64vm091-0/log/engine-
0.01.06.2020-06.54.18.log",
    "http://10.128.88.91:27159/work/win64vm091-0/log/engine-
0.01.06.2020-06.54.02.log",
    "http://10.128.88.91:27159/work/win64vm091-0/log/engine-
0.01.06.2020-06.53.42.log",
    "http://10.128.88.91:27159/work/win64vm091-1/log/engine-
1.01.08.2020-22.54.01.log",
    "http://10.128.88.91:27159/work/win64vm091-2/log/engine-
2.01.08.2020-22.54.02.log",
    "http://10.128.88.91:27159/work/win64vm091-3/log/engine-
3.01.08.2020-22.53.52.log",
    "http://10.128.88.91:27159/work/win64vm091-4/log/engine-
4.01.08.2020-22.54.01.log",
    "http://10.128.88.91:27159/work/win64vm091-5/log/engine-
5.01.08.2020-22.53.52.log",
    "http://10.128.88.91:27159/work/win64vm091-6/log/engine-
6.01.08.2020-22.54.02.log",
    "http://10.128.88.91:27159/work/win64vm091-7/log/engine-
7.01.08.2020-22.54.01.log",
    "http://10.128.88.91:27159/work/win64vm091-8/log/engine-
8.01.08.2020-22.53.55.log",
    "http://10.128.88.91:27159/work/win64vm091-9/log/engine-
9.01.08.2020-22.53.52.log"
]
```

selected-engine-info

Retrieves information, which includes all instances for the given IDs, about given Engines logged in to the Broker.

Example Request

```
POST http://example.com:8080/livecluster/rest/engine/selected-engine-info
```

Example Input

Parameters

Parameter Name	Data Type	Description	Sample Value
—	JSON	Engine IDs of Engines that are logged in to the Broker	`[4848394891525229394,484839489152522939 4]`

Example response

```
{
        "instance": "0",
        "properties": [
            {
                "name": "AMIProductCode",
                "value": "null"
            },
            {
                "name": "dataDir",
                "value":
"C:\\TIBCO\\DataSynapse\\Engine\\data\\win64vm091-0\\ddt"
            },
            {
                "name": "availableDiskInMB",
                "value": "11566219264"
            },
```

```
            {
                "name": "cpuIdString",
                "value": "GenuineIntel"
            },
            {
                "name": "osVersion",
                "value": "6.1"
            },
            {
                "name": "workUrl",
                "value": "http://10.128.88.91:27159/work/win64vm091-0"
            },
            {
                "name": "QuarantineStatus",
                "value": "New Engine"
            },
            {
                "name": "CUDAGlobalMemory",
                "value": "null"
            },
            {
                "name": "osUsername",
                "value": "SYSTEM"
            },
            {
                "name": "CUDAProcessors",
                "value": "null"
            },
            {
                "name": "CUDAFirstGPUName",
                "value": "null"
            },
            {
                "name": "dataUrl",
                "value": "http://10.128.88.91:27159/data//win64vm091-
0/ddt/"
            },
            {
                "name": "additionalPlatforms",
                "value": "null"
            },
            {
                "name": "MICMemory",
                "value": "null"
            },
            {
                "name": "ClientJavaVersion",
                "value": "7.1.0.172681"
```

```
            },
            {
                "name": "homeBrokers",
                "value": ""
            },
            {
                "name": "configurationName",
                "value": "win64:default"
            },
            {
                "name": "AMIInstanceId",
                "value": "null"
            },
            {
                "name": "cpuSocketCount",
                "value": "1"
            },
            {
                "name": "IP",
                "value": "10.128.88.91"
            },
            {
                "name": "totalMemInKB",
                "value": "2096696"
            },
            {
                "name": "lastUpdated",
                "value": "Wed Jan 08 23:08:10 PST 2020"
            },
            {
                "name": "guid",
                "value": "deadbeef8091"
            },
            {
                "name": "RTLDGlobal",
                "value": "null"
            },
            {
                "name": "CUDADevices",
                "value": "0"
            },
            {
                "name": "GridLibs",
                "value": "python-win64-
vc14,3.7.0;SpeedLink,7.0.0.165519;pybridge-win64-vc14,7.1"
            },
            {
                "name": "multicore",
```

```
            "value": "false"
        },
        {
            "name": "os",
            "value": "win64"
        },
        {
            "name": "MICModel",
            "value": "null"
        },
        {
            "name": "NETFrameworkVersions",
            "value": "4.5,4.5FullProfile"
        },
        {
            "name": "MICCores",
            "value": "null"
        },
        {
            "name": "username",
            "value": "win64vm091"
        },
        {
            "name": "GridLibUpdateTime",
            "value": "1578553692407"
        },
        {
            "name": "id",
            "value": "4848394891525229394"
        },
        {
            "name": "freeMemInKB",
            "value": "1019692"
        },
        {
            "name": "cpuNo",
            "value": "1"
        },
        {
            "name": "pid",
            "value": "5012"
        },
        {
            "name": "cpuCoreCount",
            "value": "1"
        },
        {
            "name": "AMIAvailZone",
```

```
            "value": "null"
        },
        {
            "name": "cpuMHz",
            "value": "2597.0"
        },
        {
            "name": "CUDAVersion",
            "value": "null"
        },
        {
            "name": "cpuTotal",
            "value": "1676.7"
        },
        {
            "name": "Version",
            "value": "7.1.0.172681"
        },
        {
            "name": "freeDiskInMB",
            "value": "11566219264"
        },
        {
            "name": "MICDevices",
            "value": "null"
        },
        {
            "name": "instance",
            "value": "0"
        },
        {
            "name": "crtVersions",
            "value": "VC12,VC14,VC15,VC16"
        },
        {
            "name": "HighestNumaNode",
            "value": "0"
        },
        {
            "name": "homeDir",
            "value": "C:\\TIBCO\\DataSynapse\\Engine"
        },
        {
            "name": "ProcessSchedulingPolicy",
            "value": "native"
        },
        {
            "name": "NETFramework",
```

```
                "value": "true"
            },
            {
                "name": "gridLibrary",
                "value": "null"
            },
            {
                "name": "cpuThreadCount",
                "value": "1"
            },
            {
                "name": "osName",
                "value": "WIN2K12"
            }
],
        "username": "win64vm091",
        "engineId": 4848394891525229394,
        "serviceId": -1,
        "invocationId": -1,
        "busy": false,
        "elapsedTime": -1
}
```

available

Retrieves whether the methods are available.

Example Request

```
GET http://example.com:8080/livecluster/rest/engine/available
```

Example Response

`True` or `False`

kill-all-engines

Logs out all the Engines from the Broker.

Example Request

```
PUT http://example.com:8080/livecluster/rest/engine/kill-all-engines
```

Example Response

```
204 no content
```

Result: All Engines are logged out of the Broker.

kill-engine

Logs out a given Engine from the Broker.

Example Request

```
PUT http://example.com:8080/livecluster/rest/engine/kill-engine
```

Example Input

Parameters

Parameter Name	Data Type	Description	Sample Value
`id`	Long	ID of the Engine that needs to be logged out	`4848394891525229394`
`instance`	String	Engine instance that needs to be logged out	`0`

Example Response

```
204 no content
```

Result: The specified Engine is logged out of the Broker.

park-engines

Sets Engine's parked flag to true for passed Engine IDs.

Example Request

```
POST http://example.com:8080/livecluster/rest/engine/park-engines
```

Example Input

Parameters

Parameter Name	Data Type	Description	Sample Value
—	JSON	Array of Engine Daemon IDs for which the parked flag needs to be set to true	`[ 6153431282294490153, 3574664343712327623 ]`

Example Response

```
204 no content
```

Result: Specified Engine Daemons are parked.

unpark-engines

Sets Engine's parked flag to false for passed Engine IDs.

Example Request

```
POST http://example.com:8080/livecluster/rest/engine/unpark-engines
```

Example Input

Parameters

Parameter Name	Data Type	Description	Sample Value
—	JSON	Array of Engine Daemon IDs for which the parked flag needs to be set to false	`[ 6153431282294490153 ]`

Example Response

```
204 no content
```

Result: Specified Engine Daemons are unparked.

parked-engines

Retrieves the list of Engine IDs that have the parked flag set to true.

Example Request

```
GET http://example.com:8080/livecluster/rest/engine/parked-engines
```

Example Response

```
[
    "9007929106493580894"
]
```

EngineDaemonAdmin

EngineDaemonAdmin APIs provide administrative access to the Engine Daemons on a Director. They are listed in the following table:

Method	Method Type	Description
all-engine-daemon-info	GET	Retrieves information about all the Engine Daemons logged in to the Director.
default-properties	GET	Retrieves the names and descriptions from the default Engine property list.
engine-daemon-count	GET	Retrieves the total number of Engine Daemons logged in to the Director.
engine-daemon-ids	GET	Retrieves the list of Engine Daemon IDs logged in to the Director.
engine-daemon-info	GET	Retrieves information about the given Engine Daemon logged in to the Director.
engine-daemon-info-by-properties	POST	Retrieves information about all the Engine Daemons logged in to the Director that match the Engine Condition.
log-url-list	GET	Retrieves the log file URLs from the given Engine Daemon when Engine Logging is enabled on the Server.
selected-engine-daemon-info	POST	Retrieves information about a given Engine Daemon logged in to the Director.
available	GET	Retrieves whether the methods are available.
default-property	DELETE	Removes the default property.
property	DELETE	Removes a property from the Engine Daemon.

Method	Method Type	Description
property-by-properties	DELETE	Removes a property from selected Engine Daemons that match the Engine Condition.
restart-engine-daemon	PUT	Forces the Engine Daemon to log off and restart.
restart-engine-daemon-by-properties	POST	Forces those Engine Daemons that match the Engine Condition to log off and restart.
all-enabled	PUT	Sets whether all Engine Daemons can run Engines.
all-start-mode	PUT	Sets whether all Engines have to be started manually or automatically.
configuration	PUT	Sets the configuration to run on the Engines.
configuration-by-properties	POST	Sets the configuration to run on the Engines that match the Engine Condition.
default-property	PUT	Creates a new default Engine Property and its description.
directors	PUT	Sets the primary and secondary Directors for the Daemons, overriding the values in the configuration.
directors-by-properties	POST	Sets the primary and secondary Directors on the Daemons for the Daemons that match the Engine condition, overriding the values in the configuration.
enabled	PUT	Sets whether the Engine Daemon can run Engines.
enabled-by-properties	POST	Sets whether the Engine Daemon that matches the Engine Condition can run Engines.
instances	PUT	Sets the number of Engine instances to run, overriding the value in the configuration.

Method	Method Type	Description
instances-by-properties	POST	Sets the number of Engine instances to run, overriding the value in the configuration for Engines that match the Engine Conditions.
property	PUT	Sets a property on the Engine Daemon.
property-by-properties	POST	Sets a property on selected Engine Daemons that match the Engine Condition.
start-mode	PUT	Sets whether the Engines have to be started manually or automatically.
start-mode-by-properties	POST	Sets whether the Engines that match the Engine Condition have to be started manually or automatically.

all-engine-daemon-info

Retrieves information about all the Engine Daemons logged in to the Director.

Example Request

```
GET http://example.com:8080/livecluster/rest/engineDaemon/all-engine-daemon-info
```

Example Response

```
{
        "properties": [
                {
                        "name": "ClientPlatform",
                        "value": "win64"
                },
                {
                        "name": "cpuCoreCount",
```

```
                "value": "1"
        },
        {
                "name": "availableDiskInMB",
                "value": "10965"
        },
        {
                "name": "IP",
                "value": "10.128.88.91"
        },
        {
                "name": "numInstances",
                "value": "10"
        },
        {
                "name": "QuarantineStatus",
                "value": "New Engine"
        },
        {
                "name": "username",
                "value": "win64vm091"
        },
        {
                "name": "workUrl",
                "value": "http://10.128.88.91:27159/work"
        },
        {
                "name": "freeMemInKB",
                "value": "125520"
        },
        {
                "name": "ClientEngineDaemonVersion",
                "value": "7.1.0.172681"
        },
        {
                "name": "Version",
                "value": "7.1.0.172681"
        },
        {
                "name": "os",
                "value": "win64"
        },
        {
                "name": "timeStamp",
                "value": "1578558278135"
        },
        {
```

```
                "name": "cpuUtilization",
                "value": "98"
        },
        {
                "name": "cpuSocketCount",
                "value": "1"
        },
        {
                "name": "id",
                "value": "4848394891525229394"
        },
        {
                "name": "freeDiskInMB",
                "value": "10965"
        },
        {
                "name": "dsCpuUtilization",
                "value": "2"
        },
        {
                "name": "cpuNo",
                "value": "1"
        },
        {
                "name": "guid",
                "value": "deadbeef8091"
        },
        {
                "name": "totalMemInKB",
                "value": "2096696"
        },
        {
                "name": "cpuThreadCount",
                "value": "1"
        },
        {
                "name": "Timestamp",
                "value": "1578553686304"
        },
        {
                "name": "osName",
                "value": "WIN2K12"
        },
        {
                "name": "cpuTotal",
                "value": "1676.7"
        },
```

```
                {
                        "name": "instance",
                        "value": "0"
                },
                {
                        "name": "osUsername",
                        "value": "SYSTEM"
                }
        ],
        "enabled": true,
        "engineId": 4848394891525229394,
        "instances": 10,
        "lastLoginDate": "Jan 8, 2020 11:08:06 PM",
        "installDate": "2020-01-06 06:53:25.803",
        "autoStart": false,
        "manualStart": true,
        "lastFileUpdate": "Jan 8, 2020 11:07:44 PM",
        "primaryDirector": null,
        "secondaryDirector": null,
        "configuration": "default"
}
```


default-properties

Retrieves the names and descriptions from the default Engine property list.

Example Request

```
GET http://example.com:8080/livecluster/rest/engineDaemon/default-
properties
```

Example Response


```
[
    {
        "name": "Location",
        "value": "machine location"
    },
    {
        "name": "Group",
        "value": "work group to attach engine"
```

```
    },
    {
        "name": "QuarantineStatus",
        "value": "quarantine status of the engine"
    },
    {
        "name": "Description",
        "value": "brief description of machine"
    }
]
```

engine-daemon-count

Retrieves the total number of Engine Daemons logged in to the Director.

Example Request

```
GET http://example.com:8080/livecluster/rest/engineDaemon/engine-daemon-count
```

Example Response

1.

Result: The count of Engine Daemons logged in to the Director is returned.

engine-daemon-ids

Retrieves the list of Engine Daemon IDs logged in to the Director.

Example Request

```
GET http://example.com:8080/livecluster/rest/engineDaemon/engine-daemon-ids
```

Example Response

```
[
    4848394891525229394
]
```

engine-daemon-info

Retrieves information about the given Engine Daemon logged in to the Director.

Example Request

```
GET http://example.com:8080/livecluster/rest/engineDaemon/engine-daemon-info
```

Example Input

Parameters

Parameter Name	Data Type	Description	Sample Value
`id`	Long	Engine daemon ID of the Engine Daemon whose information needs to be retrieved	`6153431282294490153`

Example Response

```
"{
    "properties": [
        {
            "name": "ClientPlatform",
            "value": "win64"
        },
        {
            "name": "cpuCoreCount",
            "value": "1"
```

```
        },
        {
            "name": "availableDiskInMB",
            "value": "13788"
        },
        {
            "name": "IP",
            "value": "10.128.88.142"
        },
        {
            "name": "numInstances",
            "value": "0"
        },
        {
            "name": "QuarantineStatus",
            "value": "New Engine"
        },
        {
            "name": "username",
            "value": "win64vm142"
        },
        {
            "name": "workUrl",
            "value": "http://10.128.88.142:27159/work"
        },
        {
            "name": "freeMemInKB",
            "value": "1324600"
        },
        {
            "name": "ClientEngineDaemonVersion",
            "value": "7.1.0.172681"
        },
        {
            "name": "Version",
            "value": "7.1.0.172681"
        },
        {
            "name": "os",
            "value": "win64"
        },
        {
            "name": "timeStamp",
            "value": "1579081956825"
        },
        {
            "name": "cpuUtilization",
            "value": "1"
```

```
        },
        {
            "name": "cpuSocketCount",
            "value": "1"
        },
        {
            "name": "id",
            "value": "6153431282294490153"
        },
        {
            "name": "freeDiskInMB",
            "value": "13788"
        },
        {
            "name": "dsCpuUtilization",
            "value": "1"
        },
        {
            "name": "cpuNo",
            "value": "1"
        },
        {
            "name": "guid",
            "value": "deadbeef8142"
        },
        {
            "name": "totalMemInKB",
            "value": "2096696"
        },
        {
            "name": "cpuThreadCount",
            "value": "1"
        },
        {
            "name": "Timestamp",
            "value": "1579005517350"
        },
        {
            "name": "osName",
            "value": "WIN2K12"
        },
        {
            "name": "cpuTotal",
            "value": "2704.3"
        },
        {
            "name": "instance",
            "value": "0"
```

```
        },
        {
            "name": "osUsername",
            "value": "SYSTEM"
        }
    ],
    "enabled": true,
    "lastFileUpdate": "N/A",
    "primaryDirector": null,
    "secondaryDirector": null,
    "configuration": "default",
    "instances": -2,
    "engineId": 6153431282294490153,
    "installDate": "2020-01-14 00:31:17.682",
    "lastLoginDate": "Jan 14, 2020 4:38:37 AM",
    "autoStart": false,
    "manualStart": true
}"
```


engine-daemon-info-by-properties

Retrieves information about all the Engine Daemons logged in to the Director that match the Engine Condition.

Example Request

```
POST http://example.com:8080/livecluster/rest/engineDaemon/engine-
daemon-info-by-properties
```

Example Input

Parameters

Parameter Name	Data Type	Description	Sample Value

—	JSON	Engine Daemon condition data based on which the Engines need to be retrieved	{
			"sets": null,
			"conditions":
			[
			{
			"name": "IP",
			"comparison": 1,
			"value":
			"10.128.88.142",
			"nullCompare":
			false
			},
			{
			"name": "os",
			"comparison": 1,
			"value": "win64",
			"nullCompare":
			false
			}
			],
			"type": 0
			}

Example Response


```
[
    {
        "properties": [
            {
                "name": "ClientPlatform",
                "value": "win64"
```

```
            },
            {
                "name": "cpuCoreCount",
                "value": "1"
            },
            {
                "name": "availableDiskInMB",
                "value": "13792"
            },
            {
                "name": "IP",
                "value": "10.128.88.142"
            },
            {
                "name": "numInstances",
                "value": "1"
            },
            {
                "name": "QuarantineStatus",
                "value": "New Engine"
            },
            {
                "name": "username",
                "value": "win64vm142"
            },
            {
                "name": "workUrl",
                "value": "http://10.128.88.142:27159/work"
            },
            {
                "name": "freeMemInKB",
                "value": "1263052"
            },
            {
                "name": "ClientEngineDaemonVersion",
                "value": "7.1.0.172681"
            },
            {
                "name": "Version",
                "value": "7.1.0.172681"
            },
            {
                "name": "os",
                "value": "win64"
            },
            {
                "name": "timeStamp",
                "value": "1578998880502"
```

```
            },
            {
                "name": "cpuUtilization",
                "value": "1"
            },
            {
                "name": "cpuSocketCount",
                "value": "1"
            },
            {
                "name": "id",
                "value": "6153431282294490153"
            },
            {
                "name": "freeDiskInMB",
                "value": "13792"
            },
            {
                "name": "dsCpuUtilization",
                "value": "1"
            },
            {
                "name": "cpuNo",
                "value": "1"
            },
            {
                "name": "guid",
                "value": "deadbeef8142"
            },
            {
                "name": "totalMemInKB",
                "value": "2096696"
            },
            {
                "name": "cpuThreadCount",
                "value": "1"
            },
            {
                "name": "Timestamp",
                "value": "1578997789590"
            },
            {
                "name": "osName",
                "value": "WIN2K12"
            },
            {
                "name": "cpuTotal",
                "value": "2704.3"
```

```
            },
            {
                "name": "instance",
                "value": "0"
            },
            {
                "name": "osUsername",
                "value": "SYSTEM"
            }
        ],
        "enabled": true,
        "lastFileUpdate": "N/A",
        "primaryDirector": null,
        "secondaryDirector": null,
        "configuration": "default",
        "instances": -2,
        "engineId": 6153431282294490153,
        "installDate": "2020-01-14 00:31:17.682",
        "lastLoginDate": "Jan 14, 2020 2:29:49 AM",
        "autoStart": false,
        "manualStart": true
    },
    {
        "properties": [
            {
                "name": "ClientPlatform",
                "value": "win64"
            },
            {
                "name": "cpuCoreCount",
                "value": "1"
            },
            {
                "name": "availableDiskInMB",
                "value": "14090"
            },
            {
                "name": "IP",
                "value": "10.128.88.108"
            },
            {
                "name": "numInstances",
                "value": "1"
            },
            {
                "name": "QuarantineStatus",
                "value": "New Engine"
            },
```

```
            {
                "name": "username",
                "value": "win64vm108"
            },
            {
                "name": "workUrl",
                "value": "http://10.128.88.108:27159/work"
            },
            {
                "name": "freeMemInKB",
                "value": "7412"
            },
            {
                "name": "ClientEngineDaemonVersion",
                "value": "7.1.0.172681"
            },
            {
                "name": "Version",
                "value": "7.1.0.172681"
            },
            {
                "name": "os",
                "value": "win64"
            },
            {
                "name": "timeStamp",
                "value": "1578998850072"
            },
            {
                "name": "cpuUtilization",
                "value": "7"
            },
            {
                "name": "cpuSocketCount",
                "value": "1"
            },
            {
                "name": "id",
                "value": "9007929106493580894"
            },
            {
                "name": "freeDiskInMB",
                "value": "14090"
            },
            {
                "name": "dsCpuUtilization",
                "value": "0"
            },
```

```
            {
                "name": "cpuNo",
                "value": "1"
            },
            {
                "name": "guid",
                "value": "deadbeef8108"
            },
            {
                "name": "totalMemInKB",
                "value": "2096696"
            },
            {
                "name": "cpuThreadCount",
                "value": "1"
            },
            {
                "name": "Timestamp",
                "value": "1578974352009"
            },
            {
                "name": "osName",
                "value": "WIN2K12"
            },
            {
                "name": "cpuTotal",
                "value": "2794.4"
            },
            {
                "name": "instance",
                "value": "0"
            },
            {
                "name": "osUsername",
                "value": "SYSTEM"
            }
        ],
        "enabled": true,
        "lastFileUpdate": "N/A",
        "primaryDirector": null,
        "secondaryDirector": null,
        "configuration": "default",
        "instances": -2,
        "engineId": 9007929106493580894,
        "installDate": "2020-01-12 22:55:40.340",
        "lastLoginDate": "Jan 13, 2020 7:59:12 PM",
        "autoStart": false,
        "manualStart": true
```

```
    }
]
```

log-url-list

Retrieves the log file URLs from the given Engine Daemon when Engine Logging is enabled on the Server.

Example Request

```
GET http://example.com:8080/livecluster/rest/engineDaemon/log-url-list
```

Example Input

Parameters

Parameter Name	Data Type	Description	Sample Value
`id`	Long	Engine Daemon ID for which the log file URLs need to be retrieved	`4848394891525229394`

Example Response

```
[
    "http://10.128.88.91:27159/work/win64vm091-0/log/engine-
0.01.08.2020-23.08.11.log",
    "http://10.128.88.91:27159/work/win64vm091-0/log/engine-
0.01.06.2020-06.54.36.log",
    "http://10.128.88.91:27159/work/win64vm091-0/log/engine-
0.01.06.2020-06.54.18.log",
    "http://10.128.88.91:27159/work/win64vm091-0/log/engine-
0.01.06.2020-06.54.02.log",
    "http://10.128.88.91:27159/work/win64vm091-0/log/engine-
0.01.06.2020-06.53.42.log",
    "http://10.128.88.91:27159/work/win64vm091-1/log/engine-
```

```
1.01.08.2020-22.54.01.log",
    "http://10.128.88.91:27159/work/win64vm091-2/log/engine-
2.01.08.2020-22.54.02.log",
    "http://10.128.88.91:27159/work/win64vm091-3/log/engine-
3.01.08.2020-22.53.52.log",
    "http://10.128.88.91:27159/work/win64vm091-4/log/engine-
4.01.08.2020-22.54.01.log",
    "http://10.128.88.91:27159/work/win64vm091-5/log/engine-
5.01.08.2020-22.53.52.log",
    "http://10.128.88.91:27159/work/win64vm091-6/log/engine-
6.01.08.2020-22.54.02.log",
    "http://10.128.88.91:27159/work/win64vm091-7/log/engine-
7.01.08.2020-22.54.01.log",
    "http://10.128.88.91:27159/work/win64vm091-8/log/engine-
8.01.08.2020-22.53.55.log",
    "http://10.128.88.91:27159/work/win64vm091-9/log/engine-
9.01.08.2020-22.53.52.log"
]
```

selected-engine-daemon-info

Retrieves information about a given Engine Daemon logged in to the Director.

Example Request

```
POST http://example.com:8080/livecluster/rest/engineDaemon/selected-
engine-daemon-info
```

Example Input

Parameters

Parameter Name	Data Type	Description	Sample Value
—	JSON	Array of Engine Daemon IDs whose information needs to be retrieved	`[`

Parameter Name	Data Type	Description	Sample Value
			6153431282294490153, 9007929106493580894]

Example response


```
[
    {
        "properties": [
            {
                "name": "ClientPlatform",
                "value": "win64"
            },
            {
                "name": "cpuCoreCount",
                "value": "1"
            },
            {
                "name": "availableDiskInMB",
                "value": "13793"
            },
            {
                "name": "IP",
                "value": "10.128.88.142"
            },
            {
                "name": "numInstances",
                "value": "0"
            },
            {
                "name": "QuarantineStatus",
                "value": "New Engine"
            },
            {
                "name": "username",
                "value": "win64vm142"
            },
            {
                "name": "workUrl",
                "value": "http://10.128.88.142:27159/work"
```

```
            },
            {
                "name": "freeMemInKB",
                "value": "1355704"
            },
            {
                "name": "ClientEngineDaemonVersion",
                "value": "7.1.0.172681"
            },
            {
                "name": "Version",
                "value": "7.1.0.172681"
            },
            {
                "name": "os",
                "value": "win64"
            },
            {
                "name": "timeStamp",
                "value": "1578999181503"
            },
            {
                "name": "cpuUtilization",
                "value": "0"
            },
            {
                "name": "cpuSocketCount",
                "value": "1"
            },
            {
                "name": "id",
                "value": "6153431282294490153"
            },
            {
                "name": "freeDiskInMB",
                "value": "13793"
            },
            {
                "name": "dsCpuUtilization",
                "value": "0"
            },
            {
                "name": "cpuNo",
                "value": "1"
            },
            {
                "name": "guid",
                "value": "deadbeef8142"
```

```
            },
            {
                "name": "totalMemInKB",
                "value": "2096696"
            },
            {
                "name": "cpuThreadCount",
                "value": "1"
            },
            {
                "name": "Timestamp",
                "value": "1578997789590"
            },
            {
                "name": "osName",
                "value": "WIN2K12"
            },
            {
                "name": "cpuTotal",
                "value": "2704.3"
            },
            {
                "name": "instance",
                "value": "0"
            },
            {
                "name": "osUsername",
                "value": "SYSTEM"
            }
        ],
        "enabled": true,
        "lastFileUpdate": "N/A",
        "primaryDirector": null,
        "secondaryDirector": null,
        "configuration": "default",
        "instances": -2,
        "engineId": 6153431282294490153,
        "installDate": "2020-01-14 00:31:17.682",
        "lastLoginDate": "Jan 14, 2020 2:29:49 AM",
        "autoStart": false,
        "manualStart": true
    },
    {
        "properties": [
            {
                "name": "ClientPlatform",
                "value": "win64"
            },
```

```
            {
                "name": "cpuCoreCount",
                "value": "1"
            },
            {
                "name": "availableDiskInMB",
                "value": "14090"
            },
            {
                "name": "IP",
                "value": "10.128.88.108"
            },
            {
                "name": "numInstances",
                "value": "1"
            },
            {
                "name": "QuarantineStatus",
                "value": "New Engine"
            },
            {
                "name": "username",
                "value": "win64vm108"
            },
            {
                "name": "workUrl",
                "value": "http://10.128.88.108:27159/work"
            },
            {
                "name": "freeMemInKB",
                "value": "53068"
            },
            {
                "name": "ClientEngineDaemonVersion",
                "value": "7.1.0.172681"
            },
            {
                "name": "Version",
                "value": "7.1.0.172681"
            },
            {
                "name": "os",
                "value": "win64"
            },
            {
                "name": "timeStamp",
                "value": "1578999461332"
            },
```

```
            {
                "name": "cpuUtilization",
                "value": "6"
            },
            {
                "name": "cpuSocketCount",
                "value": "1"
            },
            {
                "name": "id",
                "value": "9007929106493580894"
            },
            {
                "name": "freeDiskInMB",
                "value": "14090"
            },
            {
                "name": "dsCpuUtilization",
                "value": "0"
            },
            {
                "name": "cpuNo",
                "value": "1"
            },
            {
                "name": "guid",
                "value": "deadbeef8108"
            },
            {
                "name": "totalMemInKB",
                "value": "2096696"
            },
            {
                "name": "cpuThreadCount",
                "value": "1"
            },
            {
                "name": "Timestamp",
                "value": "1578974352009"
            },
            {
                "name": "osName",
                "value": "WIN2K12"
            },
            {
                "name": "cpuTotal",
                "value": "2794.4"
            },
```

```
            {
                "name": "instance",
                "value": "0"
            },
            {
                "name": "osUsername",
                "value": "SYSTEM"
            }
        ],
        "enabled": true,
        "lastFileUpdate": "N/A",
        "primaryDirector": null,
        "secondaryDirector": null,
        "configuration": "default",
        "instances": -2,
        "engineId": 9007929106493580894,
        "installDate": "2020-01-12 22:55:40.340",
        "lastLoginDate": "Jan 13, 2020 7:59:12 PM",
        "autoStart": false,
        "manualStart": true
    }
]
```


available

Retrieves whether the methods are available.

Example Request

```
GET http://example.com:8080/livecluster/rest/engineDaemon/available
```

Example Response

`True` or `False`

default-property

Removes the default property.

Example Request

```
DELETE http://example.com:8080/livecluster/rest/engineDaemon/default-
property
```

Example Input

Parameters

Parameter Name	Data Type	Description	Sample Value
`property`	String	Default property that needs to be removed	`NewProp`

Example Response

```
new property description
```

Result: The description of the removed property is returned.

property

Removes a property from the Engine Daemon.

Example Request

```
DELETE http://example.com:8080/livecluster/rest/engineDaemon/property
```

Example Input

Parameters

Parameter Name	Data Type	Description	Sample Value
`id`	Long	Engine Daemon ID whose property needs to be removed	`6153431282294490153`
`key`	String	Key of the property that needs to be removed	`Location`

Example Response

```
machine location
```

Result: The description of the removed property is returned.

property-by-properties

Removes a property from selected Engine Daemons that match the Engine Condition.

Example Request

```
DELETE http://example.com:8080/livecluster/rest/engineDaemon/property-by-properties
```

Example Input

Parameters

Parameter Name	Data Type	Description	Sample Value
—	JSON	Engine Condition data depending on which engines are selected	`{ "sets":`

Parameter Name	Data Type	Description	Sample Value

			`null,`
			
			`"conditions": [`
			`{`
			
			`"name":`
			`"osName",`
			
			`"comparison": 1,`
			
			`"value":`
			`"RHEL6",`
			
			`"nullCompare":`
			`false`
			`}`
			`],`
			`"type": 0`
			`}`

`key`	String	Key of the property that needs to be removed	`NewProp`

Example Response

1

Result: Returns 1 if there is no exception.

restart-engine-daemon

Forces the Engine Daemon to log off and restart.

Example Request

```
PUT http://example.com:8080/livecluster/rest/engineDaemon/restart-
engine-daemon
```

Example Input

Parameters

Parameter Name	Data Type	Description	Sample Value
`id`	Long	Engine Daemon ID of Engine Daemon that is forced to log off and restart	`4848394891525229394`

Example Response

```
204 no content
```

Result: Engine Daemon is restarted.

restart-engine-daemon-by-properties

Forces those Engine Daemons that match the Engine Condition to log off and restart.

Example Request

```
POST http://example.com:8080/livecluster/rest/engineDaemon/restart-
engine-daemon-by-properties
```

Example Input

Parameters

Parameter Name	Data Type	Description	Sample Value
—	JSON	Engine Condition data depending on which engines need to be selected for log off	{ "sets": null, "conditions": [{ "name": "osName", "comparison": 1, "value": "RHEL6", "nullCompare": false }], "type": 0 }

Example Response

```
204 no content
```

Result: The Engine Daemon matching the specified condition is restarted.

all-enabled

Sets whether all Engine Daemons can run Engines.

Example Request

```
PUT http://example.com:8080/livecluster/rest/engineDaemon/all-enabled
```

Example Input

Parameters

Parameter Name	Data Type	Description	Sample Value
`enable`	Boolean	True or false depending on which it is decided whether all Engine Daemons can run the Engines or not	`true`

Example Response

```
204 no content
```

Result: All Engine Daemons are enabled or disabled.

all-start-mode

Sets whether all Engines have to be started manually or automatically.

Example Request

```
PUT http://example.com:8080/livecluster/rest/engineDaemon/all-start-mode
```

Example Input

Parameters

Parameter Name	Data Type	Description	Sample Value
`start`	Boolean	True or false. True states manual start and false states automatic start.	`true`

Example Response

```
204 no content
```

Result: The manual start property of all Engine Daemons is set to true or false, depending on what you have specified.

configuration

Sets the configuration to run on the Engines.

Example Request

```
PUT http://example.com:8080/livecluster/rest/engineDaemon/configuration
```

Example Input

Parameters

Parameter Name	Data Type	Description	Sample Value
`id`	Long	Engine ID for which configuration needs to be changed	`3574664343712327623`
`config`	String	The configuration which needs to be updated	`testlin64`

Example Response

```
204 no content
```

Result: Configuration of the specified Engine Daemon is updated.

configuration-by-properties

Sets the configuration to run on the Engines that match the Engine Condition.

Example Request

```
POST
http://example.com:8080/livecluster/rest/engineDaemon/configuration-by-properties
```

Example Input

Parameters

Parameter Name	Data Type	Description	Sample Value
—	JSON	Engine condition data depending on which engines need to be selected	`{ "sets": null, "conditions": [ { "name": "os", "comparison": 1, "value": "linux64", "nullCompare": false } ], "type": 0 }`
`config`	String	Configuration which needs to be updated	`testlin64`

Example Response

```
204 no content
```

Result: Configuration of the Engine Daemons matching the given condition is updated.

default-property

Creates a new default Engine Property and its description.

Example Request

```
PUT http://example.com:8080/livecluster/rest/engineDaemon/default-
property
```

Example Input

Parameters

Parameter Name	Data Type	Description	Sample Value
`property`	String	Name of the new default Engine property that needs to be created	`NewProp`
`description`	String	Description on the new default Engine property	`New Property description`

Example Response

```
204 no content
```

Result: A new default property is added.

directors

Sets the primary and secondary Directors for the Daemons, overriding the values in the configuration.

Example Request

```
PUT http://example.com:8080/livecluster/rest/engineDaemon/directors
```

Example Input

Parameters

Parameter Name	Data Type	Description	Sample Value
`id`	Long	Engine Daemon ID whose Primary and Secondary Directors whose primary and secondary Directors need to be overridden	`3574664343712327623`
`primary`	String	Name of the Primary Director	`http://example.com:8080`
`secondary`	String	Name of Secondary Director	`http://win64vm218.rofa.tibco.com:8080`

Example Response

```
204 no content
```

Result: The primary and secondary Directors for the specified Engine Daemon are set.

directors-by-properties

Sets the primary and secondary Directors on the Daemons for the Daemons that match the Engine condition, overriding the values in the configuration.

Example Request

```
POST http://example.com:8080/livecluster/rest/engineDaemon/directors-by-properties
```

Example Input

Parameters

Parameter Name	Data Type	Description	Sample Value
—	JSON	Engine condition data depending on which Engines you select	`http://example.com:8080`
`primary`	String	Name of the Primary Director	`http://example.com:8080`
`secondary`	String	Name of the Secondary Director	`http://example.com:8080`

Example Response

```
204 no content
```

Result: The primary and secondary Directors are set for the Engine Daemons matching the given condition.

enabled

Sets whether the Engine Daemon can run Engines.

Example Request

```
PUT http://example.com:8080/livecluster/rest/engineDaemon/enabled
```

Example Input

Parameters

Parameter Name	Data Type	Description	Sample Value
`id`	Long	Engine Daemon ID	`6153431282294490153`
`enable`	Boolean	Engine Daemon enabled flag	`False`

Example Response

```
204 no content
```

Result: The specified Engine Daemon is enabled or disabled.

enabled-by-properties

Sets whether the Engine Daemon that matches the Engine Condition can run Engines.

Example Request

```
POST http://example.com:8080/livecluster/rest/engineDaemon/enabled-by-
properties
```

Example Input

Parameters

Parameter Name	Data Type	Description	Sample Value

—	JSON	Engine condition data depending on which engines are selected	<pre>{
			"sets": null,
			"conditions": [
			{
			"name": "os",
			"comparison": 1,
			"value": "win64",
			"nullCompare": false
			},
			{
			"name": "id",
			"comparison": 1,
			"value": "9007929106493580894",
			"nullCompare": false
			}
			],
			"type": 0
			}</pre>

`enable`	Boolean	Engine Daemon enabled flag. True or false.	`true`

Example Response

```
204 no content
```

Result: The Engine Daemons matching condition are enabled or disabled.

instances

Sets the number of Engine instances to run, overriding the value in the configuration.

Example Request

```
PUT http://example.com:8080/livecluster/rest/engineDaemon/instances
```

Example Input

Parameters

Parameter Name	Data Type	Description	Sample Value
`id`	Long	Engine Daemon ID for which the number of instances need to be updated	`6153431282294490153`
`instances`	Int	Number of instances that need to be overridden	`2`

Example Response

```
204 no content
```

Result: The number of Engine instances to run for Engine Daemons is set.

instances-by-properties

Sets the number of Engine instances to run, overriding the value in the configuration for Engines that match the Engine Conditions.

Example Request

```
POST http://example.com:8080/livecluster/rest/engineDaemon/instances-by-properties
```

Example Input

Parameters

Parameter Name	Data Type	Description	Sample Value
—	JSON	Engine Condition data depending on which Engines need to be selected	`{ "sets": null, "conditions": [ { "name": "os", "comparison": 1, "value": "linux64", "nullCompare": false } ], "type": 0 }`
`instances`	Int	Number of instances which need to be overridden	`3`

Example Response

```
204 no content
```

Result: The number of Engine instances to run is set for Engine Daemons matching the condition.

property

Sets a property on the Engine Daemon.

Example Request

```
PUT http://example.com:8080/livecluster/rest/engineDaemon/property
```

Example Input

Parameters

Parameter Name	Data Type	Description	Sample Value
`id`	Long	Engine Daemon ID for which a new property needs to be set	`9007929106493580894`
`key`	String	Key of the new property that needs to be added	`instances`
`value`	String	Value of the new property that needs to be added	`4`

Example Response

```
204 no content
```

Result: A new property of the Engine Daemon is set.

property-by-properties

Sets a property on selected Engine Daemons that match the Engine Condition.

Example Request

```
POST http://example.com:8080/livecluster/rest/engineDaemon/property-by-properties
```

Example Input

Parameters

Parameter Name	Data Type	Description	Sample Value
—	JSON	Engine Condition data depending on which engines need to be selected	`{ "sets": null, "conditions": [ { "name": "osName", "comparison": 1, "value": "RHEL6", "nullCompare": false } ], "type": 0 }`
`key`	String	Key of the new property that needs to be added	`New Prop`
`value`	String	Value of the new property that needs to be added	`New Prop Value`

Example Response

```
204 no content
```

Result: A property is set for the selected Engine Daemons that match the Engine condition.

start-mode

Sets whether the Engines have to be started manually or automatically.

Example Request

```
PUT http://example.com:8080/livecluster/rest/engineDaemon/start-mode
```

Example Input

Parameters

Parameter Name	Data Type	Description	Sample Value
`id`	Long	Engine Daemon ID for which start mode needs to be set	`9007929106493580894`
`start`	Boolean	True or False. True states manual start and false states automatic start.	`true`

Example Response

```
204 no content
```

Result: The property of all the specified Engine Daemons to start manually is set to true or false.

start-mode-by-properties

Sets whether the Engines that match the Engine Condition have to be started manually or automatically.

Example Request

```
POST http://example.com:8080/livecluster/rest/engineDaemon/start-mode-
by-properties
```

Example Input

Parameters

Parameter Name	Data Type	Description	Sample Value
—	JSON	Engine Condition data depending on which engines need to be selected	`{ "sets": null, "conditions": [ { "name": "os", "comparison": 1, "value": "linux64", "nullCompare": false } ], "type": 0 }`
`manual`	Boolean	Manual startup flag. True or false.	`true`

Example Response

```
204 no content
```

Result: The property to start manually for Engine Daemons that match the condition is set to true or false.

DriverManager

DirverManager API provides a method to retrieve the base context URL. The method is listed in the following table:

Method	Method Type	Description
broker-url	GET	Retrieves the base context URL of the Brokers

broker-url

Retrieves the base context URL of the Brokers

Example Request

```
GET http://example.com:8080/livecluster/rest/driverManager/broker-url
```

Example Response

`http://example.com:8080/livecluster/.`

Result: All the base context URLs of the Brokers are returned.

ManagerAdmin

Provides administrative access to the Manager. Methods in class ManagerAdmin are listed in the following table:

Method	Method Type	Description
broker-id	GET	Retrieves the Broker ID if it exists else returns -1.

Method	Method Type	Description
broker-name	GET	Retrieves the Broker name if it exists else returns null.
broker-url	GET	Retrieves the Broker URL; returns null if the Manager does not contain a Broker.
build-version	GET	Retrieves the full Manager version including the build number.
busy-engine-count	GET	Retrieves the total number of busy Engines from all the Brokers logged in to the Director.
category	GET	Retrieves the specified configuration category.
category-names	GET	Retrieves the Manager configuration category names.
director-id	GET	Gets the Director ID if it exists else returns -1.
engine-configuration-names	GET	Retrieves a list of names of the Engine configuration profiles available on the specified Manager.
engine-count	GET	Retrieves the total number of Engines from all the Brokers logged in to the Director.
events	GET	Retrieves the available subscription events.
finished-service-count	GET	Retrieves the total number of finished Services from all the Brokers logged in to the Director.
license-info	GET	Retrieves the Manager license information.
manager-value	GET	Retrieves a Manager property value specified by category name, property group name, and property name on the specified manager.
pending-	GET	Retrieves the total number of pending Service invocations

Method	Method Type	Description
invocation-count		from all the Brokers logged in to the Director.
running-invocation-count	GET	Retrieves the total number of running Service invocations from all the Brokers logged in to the Director.
running-service-count	GET	Retrieves the total number of running Services from all the Brokers logged in to the Director.
service-count	GET	Retrieves the total number of Services from all the Brokers logged in to the Director.
subscriber-events	POST	Retrieves the events that the given subscriber is registered to receive.
subscribers	GET	Retrieves the registered event subscribers.
value	GET	Gets a Manager property value specified by the category name, property group name, and property name.
version	GET	Retrieves the Manager version in M.m format.
available	GET	Retrieves whether the methods are available.
manager-value	PUT	Sets a Manager property value specified by the category name, property group name, and property name on the specified manager.
value	PUT	Sets a Manager property value specified by the category name, property group name, and property name.
subscribe	POST	Subscribes to the given events.
unsubscribe	POST	Unsubscribes from the given events.

broker-id

Retrieves the Broker ID if it exists else returns -1.

Example Request

```
GET http://example.com:8080/livecluster/rest/manager/broker-id
```

Example Response

```
899860077
```

broker-name

Retrieves the Broker name if it exists else returns null.

Example Request

```
GET http://example.com:8080/livecluster/rest/manager/broker-name
```

Example Response

```
800860077.
```

Result: The name of the Broker is returned.

broker-url

Retrieves the Broker URL; returns null if the Manager does not contain a Broker.

Example Request

```
GET http://example.com:8080/livecluster/rest/manager/broker-url
```

Example Response

```
http: //<hostname>:8000/livecluster
```

build-version

Retrieves the full Manager version including the build number.

Example Request

```
GET http://example.com:8080/livecluster/rest/manager/build-version
```

Example Response

```
7.1.0.172681
```

busy-engine-count

Retrieves the total number of busy Engines from all the Brokers logged in to the Director.

Example Request

```
GET http://example.com:8080/livecluster/rest/manager/busy-engine-count
```

Example Response

```
0
```

category

Retrieves the specified configuration category.

Example Request

```
GET http://example.com:8080/livecluster/rest/manager/category
```

Example Input

Parameters

Parameter Name	Data Type	Description	Sample Value
`name`	String	Name of the category depending on which configuration needs to be retrieved	`Logging`

Example Response

```
{
    "name": "Logging",
    "groups": [
        {
            "name": "General",
            "properties": [
                {
                    "name": "Default Debug Level",
                    "value": "INFO"
                },
                {
                    "name": "Use the Fully Qualified Class Name",
                    "value": "false"
                },
                {
                    "name": "Log Level List",
                    "value": ""
                }
            ]
        },
        {
            "name": "File",
            "properties": [
                {
                    "name": "Stack Trace",
```

```
                    "value": "true"
                },
                {
                    "name": "Log To File",
                    "value": "true"
                },
                {
                    "name": "Max File Length",
                    "value": "1000000"
                },
                {
                    "name": "Log File Directory",
                    "value": "F:/datasynapse/manager-data/logs/server"
                },
                {
                    "name": "Max Log Level",
                    "value": "ALL"
                },
                {
                    "name": "Time Zone",
                    "value": ""
                }
            ]
        },
        {
            "name": "Log to REST",
            "properties": [
                {
                    "name": "Server URL",
                    "value": ""
                },
                {
                    "name": "Enabled",
                    "value": "false"
                },
                {
                    "name": "Username",
                    "value": ""
                },
                {
                    "name": "Password",
                    "value": null
                },
                {
                    "name": "Max Log Level",
                    "value": "ALL"
                },
```

```
                {
                    "name": "Backlog",
                    "value": "100"
                },
                {
                    "name": "Discard when Backlogged",
                    "value": "true"
                },
                {
                    "name": "Timestamp format",
                    "value": "yyyy-MM-dd'T'HH:mm:ss.SSSZZ"
                },
                {
                    "name": "Excluded LogRecord Properties",
                    "value":
"millis,sequenceNumber,sourceClassName,sourceMethodName"
                },
                {
                    "name": "HTTP Timeout",
                    "value": "5000"
                }
            ]
        },
        {
            "name": "Server Events to REST",
            "properties": [
                {
                    "name": "Enabled",
                    "value": "false"
                },
                {
                    "name": "Server URL",
                    "value": ""
                },
                {
                    "name": "Username",
                    "value": ""
                },
                {
                    "name": "Password",
                    "value": null
                },
                {
                    "name": "Backlog",
                    "value": "1000"
                },
                {
```

```
                    "name": "Discard when Backlogged",
                    "value": "false"
                },
                {
                    "name": "Timestamp format",
                    "value": "yyyy-MM-dd'T'HH:mm:ss.SSSZZ"
                },
                {
                    "name": "Excluded Events",
                    "value": ""
                },
                {
                    "name": "HTTP Timeout",
                    "value": "5000"
                },
                {
                    "name": "Include name",
                    "value": "false"
                },
                {
                    "name": "Include type",
                    "value": "true"
                }
            ]
        },
        {
            "name": "Syslog",
            "properties": [
                {
                    "name": "Enabled",
                    "value": "false"
                },
                {
                    "name": "Time Zone",
                    "value": ""
                },
                {
                    "name": "Syslog Server Address",
                    "value": ""
                },
                {
                    "name": "Syslog Server Port",
                    "value": "514"
                },
                {
                    "name": "Components",
                    "value": "*"
```

```
                },
                {
                    "name": "Max Log Level",
                    "value": "ALL"
                },
                {
                    "name": "Tag",
                    "value": "GridServerManager"
                },
                {
                    "name": "Facility Level",
                    "value": "1"
                }
            ]
        },
        {
            "name": "Component-level Logging",
            "properties": [
                {
                    "name": "Engine Events",
                    "value": "DEFAULT"
                },
                {
                    "name": "Driver Events",
                    "value": "DEFAULT"
                },
                {
                    "name": "Service Events",
                    "value": "DEFAULT"
                },
                {
                    "name": "Scheduler",
                    "value": "DEFAULT"
                },
                {
                    "name": "Engine Sharing",
                    "value": "DEFAULT"
                },
                {
                    "name": "GridCache",
                    "value": "DEFAULT"
                },
                {
                    "name": "Batch Events",
                    "value": "DEFAULT"
                },
                {
```

```
                "name": "Web Services",
                "value": "DEFAULT"
            }
        ]
    },
    {
        "name": "Log File Cleaner",
        "properties": [
            {
                "name": "File Cleaner Timeout",
                "value": "0"
            },
            {
                "name": "Cleaner Cron Expression",
                "value": "0 50 3 * * ?"
            },
            {
                "name": "File Time To Live",
                "value": "120.0"
            },
            {
                "name": "File Cleaner Enabled",
                "value": "true"
            }
        ]
    },
    {
        "name": "Tomcat Log File Cleaner",
        "properties": [
            {
                "name": "File Cleaner Timeout",
                "value": "0"
            },
            {
                "name": "Cleaner Cron Expression",
                "value": "0 0 * * * ?"
            },
            {
                "name": "File Time To Live",
                "value": "120.0"
            },
            {
                "name": "File Cleaner Enabled",
                "value": "true"
            }
        ]
    },
```

```
        {
            "name": "Memory Logging",
            "properties": [
                {
                    "name": "Log Period",
                    "value": "1.0"
                },
                {
                    "name": "Warning Threshold",
                    "value": "80"
                },
                {
                    "name": "Total Allocated Heap",
                    "value": "192"
                },
                {
                    "name": "Free Heap",
                    "value": "52"
                },
                {
                    "name": "Maximum Heap",
                    "value": "1015"
                }
            ]
        }
    ]
}
```

category-names

Retrieves the Manager configuration category names.

Example Request

```
GET http://example.com:8080/livecluster/rest/manager/category-names
```

Example Response

```
[
    "Logging",
```

```
    "Services",
    "Engines and Clients",
    "Admin",
    "Database",
    "Resource Deployment",
    "Cache",
    "Security",
    "Communication",
    "Hotfixes"
]
```

director-id

Gets the Director ID if it exists else returns -1.

Example Request

```
GET http://example.com:8080/livecluster/rest/manager/director-id
```

Example Response

```
899860077
```

engine-configuration-names

Retrieves a list of names of the Engine configuration profiles available on the specified Manager.

Example Request

```
GET http://example.com:8080/livecluster/rest/manager/engine-
configuration-names
```

Example Response

```
[

    "linux64:default",
    "win64:default",
    "win32:default",
    "linux:default"
]
```

engine-count

Retrieves the total number of Engines from all the Brokers logged in to the Director.

Example Request

```
GET http://example.com:8080/livecluster/rest/manager/engine-count
```

Example Response

```
10
```

events

Retrieves the available subscription events.

Example Request

```
GET http://example.com:8080/livecluster/rest/manager/events
```

Example Response

```
[
    "ServiceEvent",
```

```
    "ServicePriorityChangedEvent",
    "EngineDiedEvent",
    "EngineAddedEvent",
    "EngineRemovedEvent",
    "EngineBlacklistedEvent",
    "EngineGreylistedEvent",
    "EngineDaemonAddedEvent",
    "EngineDaemonRemovedEvent",
    "DriverAddedEvent",
    "DriverRemovedEvent",
    "BrokerAddedEvent",
    "BrokerRemovedEvent",
    "LicenseEvent",
    "ServerInitializedEvent",
    "MemoryWarningEvent"
]
```

finished-service-count

Retrieves the total number of finished Services from all the Brokers logged in to the Director.

Example Request

```
GET http://example.com:8080/livecluster/rest/manager/finished-service-
count
```

Example Response

```
3
```

license-info

Retrieves the Manager license information.

Example Request

```
GET http://example.com:8080/livecluster/rest/manager/license-info
```

Example Response


```
{
    "properties": [
        {
            "name": "licenseVersion",
            "value": "5.0"
        },
        {
            "name": "maxDaemons",
            "value": "632000"
        },
        {
            "name": "productname",
            "value": "GridServer"
        },
        {
            "name": "maxBrokers",
            "value": "632000"
        },
        {
            "name": "maxCpuThreadNo",
            "value": "632000"
        },
        {
            "name": "maxCpuCoreNo",
            "value": "632000"
        },
        {
            "name": "maxEngines",
            "value": "632000"
        },
        {
            "name": "maxCpuSocketNo",
            "value": "632000"
        },
        {
            "name": "maxCpuNo",
            "value": "632000"
        }
    ],
```

```
    "errorMessage": null,
    "customerName": null,
    "allowedHostnames": [
        "*:*"
    ],
    "expirationDate": 1787447752000,
    "featureSets": [
        {
            "name": "drivers",
            "value": "jdriver pdriver cppdriver NETdriver SOAPdriver
DWSdriver COMdriver"
        },
        {
            "name": "serviceTypes",
            "value": "*"
        },
        {
            "name": "schedulers",
            "value": "SLA"
        },
        {
            "name": "engineOS",
            "value": "*"
        },
        {
            "name": "serverOS",
            "value": "*"
        }
    ]
}
```


manager-value

Retrieves a Manager property value specified by category name, property group name, and property name on the specified manager.

Example Request

```
GET http://example.com:8080/livecluster/rest/manager/manager-value
```

Example Input

Parameters

Parameter Name	Data Type	Description	Sample Value
`manager`	String	Manger ID for which property needs to be retrieved	`1580108171`
`category`	String	Name of category	`Logging`
`propertyGroup`	String	Name of property group	`General`
`name`	String	Name of property	`Default debug level`

Example Response

```
INFO
```

Result: The corresponding value of the property is returned.

pending-invocation-count

Retrieves the total number of pending Service invocations from all the Brokers logged in to the Director.

Example Request

```
GET http://example.com:8080/livecluster/rest/manager/pending-invocation-count
```

Example Response

```
0
```

running-invocation-count

Retrieves the total number of running Service invocations from all the Brokers logged in to the Director.

Example Request

```
GET http://example.com:8080/livecluster/rest/manager/running-invocation-count
```

Example Response

```
0
```

running-service-count

Retrieves the total number of running Services from all the Brokers logged in to the Director.

Example Request

```
GET http://example.com:8080/livecluster/rest/manager/running-service-count
```

Example Response

```
0
```

service-count

Retrieves the total number of Services from all the Brokers logged in to the Director.

Example Request

```
GET http://example.com:8080/livecluster/rest/manager/service-count
```

Example Response

2

subscriber-events

Retrieves the events that the given subscriber is registered to receive.

Example Request

```
POST http://example.com:8080/livecluster/rest/manager/subscriber-events
```

Example Input

Parameters

Parameter Name	Data Type	Description	Sample Value
—	JSON	Subscriber data depending on which events need to be retrieved	`{ "filter": "pqr", "email": "test@tibco.com" }`

Example Response

```
[
    "ServiceEvent",
    "ServicePriorityChangedEvent"
]
```

subscribers

Retrieves the registered event subscribers.

Example Request

```
GET http://example.com:8080/livecluster/rest/manager/subscribers
```

Example Response

```
[
    {
        "filter": null,
        "email": "test"
    }
]
```

value

Gets a Manager property value specified by the category name, property group name, and property name.

Example Request

```
GET http://example.com:8080/livecluster/rest/manager/value
```

Example Input

Parameters

Parameter Name	Data Type	Description	Sample Value
`category`	String	Name of the category	`Logging`
`propertyGroup`	String	Name of the property group	`General`
`name`	String	Name of the property	`Default Debug Level`

Example Response

```
INFO
```

version

Retrieves the Manager version in M.m format.

Example Request

```
GET http://example.com:8080/livecluster/rest/manager/version
```

Example Response

```
7.1
```

available

Retrieves whether the methods are available.

Example Request

```
GET http://example.com:8080/livecluster/rest/manager/available
```

Example Response

`True` or `False`

manager-value

Sets a Manager property value specified by the category name, property group name, and property name on the specified manager.

Example Request

```
PUT http://example.com:8080/livecluster/rest/manager/manager-value
```

Example Input

Parameters

Parameter Name	Data Type	Description	Sample Value
`manager`	String	Manager ID for which property needs to be retrieved	`1580108171`
`category`	String	Name of the category	`Logging`
`propertyGroup`	String	Name of the property group	`General`

Parameter Name	Data Type	Description	Sample Value
`name`	String	Name of the property	`Default Debug Level`
`value`	String	Value of the property	`FINE`

Example Response

```
204 no content
```

Result: Property value is updated correctly.

value

Sets a Manager property value specified by the category name, property group name, and property name.

Example Request

```
PUT http://example.com:8080/livecluster/rest/manager/value
```

Example Input

Parameters

Parameter Name	Data Type	Description	Sample Value
`category`	String	Name of the category	`Logging`

Parameter Name	Data Type	Description	Sample Value
`propertyGroup`	String	Name of the property group	`General`
`name`	String	Name of the property	`Default Debug Level`
`value`	String	Value of the category	`FINE`

Example Response

```
204 no content
```

Result: Property value is updated correctly.

subscribe

Subscribes to the given events.

Example Request

```
POST http://example.com:8080/livecluster/rest/manager/subscribe
```

Example Input

Parameters

Parameter Name	Data Type	Description	Sample Value

—	JSON	Subscriber data that subscribes to given events	<pre>{
			    "subscriber": {
			        "filter": "xyz",
			        "email":
			"test@example.com"
			    },
			    "events": [
			        "ServiceEvent",
			
			"ServicePriorityChangedEvent"
			    ]
			}</pre>

Example Response

```
204 no content
```

Result: Subscribed to given event.

unsubscribe

Unsubscribes from the given events.

Example Request

```
POST http://example.com:8080/livecluster/rest/manager/unsubscribe
```

Example Input

Parameters

Parameter Name	Data Type	Description	Sample Value

—	JSON	Subscriber data that unsubscribes from given events	<pre>{
			    "subscriber": {
			        "filter": "xyz",
			        "email":
			"test@example.com"
			    },
			    "events": [
			        "ServiceEvent",
			
			"ServicePriorityChangedEvent"
			    ]
			}</pre>

Example Response

```
204 no content
```

Result: Unsubscribed from given event.

ServiceAdmin

ServiceAdmin APIs provide administrative access to the Services on a Broker. They are listed in the following table:

Method	Method Type	Description
all-services	PUT	Cancels all active Services on the Broker.
invocation	PUT	Cancels a specific invocation specified by the given Service ID and Invocation ID.

Method	Method Type	Description
service	PUT	Cancels the Service specified by the given Service ID.
resources	DELETE	Removes resource files from both the staging and deployed areas.
deploy-resources	POST	Deploys resources.
all-service-info	GET	Retrieves Service information of all the Services on the Broker.
blacklisted-engines	GET	Retrieves a list of IDs of Engines that have been blacklisted for a specified Service.
completed-service-invocation-count	GET	Retrieves the number of completed invocations of the specified Service.
finished-service-count	GET	Retrieves the total number of finished Services on the Broker.
invocation-count	GET	Retrieves the total number of pending and running invocations on the Broker.
invocation-info	GET	Retrieves the information about a particular invocation of a Service, for the given Service ID and Invocation ID.
pending-invocation-count	GET	Retrieves the number of pending invocations on the Broker.
pending-service-invocation-count	GET	Retrieves the number of pending invocations of the specified Service.
registered-services	GET	Retrieves a list of all registered Services; returns null if no Services are registered.
running-invocation-count	GET	Retrieves the number of invocations currently executed by Engines on the Broker.

Method	Method Type	Description
running-service-count	GET	Retrieves the total number of active Services on the Broker.
running-service-invocation-count	GET	Retrieves the number of invocations currently executed by Engines of the specified Service.
selected-invocation-info	POST	Retrieves information of the given Service and invocation IDs.
selected-service-info	POST	Retrieves information of the given Service IDs running on the Broker.
service-binding	GET	Retrieves the binding of the specified Service.
service-count	GET	Retrieves the total number of Services on the Broker.
service-ids	GET	Retrieves the list of all Service IDs on the Broker.
service-info	GET	Retrieves the Service information of the given Service ID.
service-info-by-properties	POST	Retrieves the Service information of all Services on the Broker that match the description condition.
service-invocation-count	GET	Retrieves the total number of invocations of the specified Service.
task-expiration-event-count	GET	Retrieves the total number of task expiration events of the specified Service.
available	GET	Retrieves whether the methods are available.
list-resources	GET	Lists resources on the Server.
register-service	POST	Registers the binding in the Service Registry; this must be done on the Primary Director.

Method	Method Type	Description
all-finished-services	DELETE	Removes the information of all finished Services from the Broker.
finished-service	DELETE	Removes the information of finished Services from the Broker.
resource-exists	GET	Checks whether a file exists in the designated area.
expires	PUT	Sets whether Service information is removed during Service cleanup.
priority	PUT	Sets the priority of the specified Service.
unregister-service	DELETE	Unregisters the binding in the Service Registry; this must be done on the Primary Director.

all-services

Cancels all active Services on the Broker.

Example Request

```
PUT http://example.com:8080/livecluster/rest/service/all-services
```

Example Response

```
204 no content
```

Result: All active services on the broker are canceled.

invocation

Cancels a specific invocation specified by the given Service ID and Invocation ID.

Example Request

```
PUT http://example.com:8080/livecluster/rest/service/invocation
```

Example Input

Parameters

Parameter Name	Data Type	Description	Sample Value
`jobId`	Long	Service ID of the Service that needs to be canceled	`4697058754240048269`
`taskId`	Long	Invocation ID of the Service that needs to be canceled	`5`

Example Response

```
204 no content
```

Result: Specific invocation for given service ID and Invocation ID is canceled.

service

Cancels the Service specified by the given Service ID.

Example Request

```
PUT http://example.com:8080/livecluster/rest/service/service
```

Example Input

Parameters

Parameter Name	Data Type	Description	Sample Value
`jobId`	Long	Service ID of the Service that needs to be canceled	`6344484810359622656`

Example Response

```
204 no content
```

Result: Service specified by the given Service ID is canceled.

resources

Removes resource files from both the staging and deployed areas.

Example Request

```
DELETE http://example.com:8080/livecluster/rest/service/resources
```

Example Input

Parameters

Parameter Name	Data Type	Description	Sample Value
—	JSON	Array of resource names that needs to be removed.	`["calculator-1.0.0.1.tar.gz"]`

Example Response

```
204 no content
```

Result: Corresponding resources are deleted.

deploy-resources

Deploys resources.

Example Request

```
POST http://example.com:8080/livecluster/rest/service/deploy-resources
```

Example Input

Parameters

Parameter Name	Data Type	Description	Sample Value
—	JSON	Array of resource names that need to be deployed	`["calculator-1.0.0.1.tar.gz"]`
`verifyGridLibraryStructure`	Boolean	Grid library structure flag. True or false.	`True or False`
`verifyGridLibraryDependencies`	Boolean	Grid library dependency flag. True or false.	`True or False`
`verifyGridLibraryNaming`	Boolean	Grid library naming flag. True or false.	`True or False`

Example Response

```
204 no content
```

Result: Corresponding resources are deployed.

all-service-info

Retrieves Service information of all Services on the Broker.

Example Request

```
GET http://example.com:8080/livecluster/rest/service/all-service-info
```

Example Response


```
[
    {
        "finishTime": -1,
        "pendingCount": 2,
        "runningCount": 8,
        "completedCount": 0,
        "totalCount": 10,
        "serviceName": "Linpack Test",
        "cancelled": false,
        "description": [
            {
                "name": "appName",
                "value": "DataSynapse Job/Service Test"
            },
            {
                "name": "serviceTypeName",
                "value": "LinpackServiceTest"
            },
            {
                "name": "serviceName",
                "value": "Linpack Test"
            },
            {
                "name": "class",
                "value": "java:examples.linpack.LinpackService"
            },
            {
                "name": "serviceType",
                "value": "Service"
```

```
            },
            {
                "name": "slaGroupName",
                "value": ""
            },
            {
                "name": "providerType",
                "value": "JavaProvider"
            },
            {
                "name": "username",
                "value": "internal"
            }
        ],
        "finished": false,
        "status": "Running",
        "options": [
            {
                "name": "autoPackMode",
                "value": "0"
            },
            {
                "name": "compressData",
                "value": "true"
            },
            {
                "name": "resultsPerMessage",
                "value": "100"
            },
            {
                "name": "resubmitOnDDTFailure",
                "value": "true"
            },
            {
                "name": "statusExpires",
                "value": "true"
            },
            {
                "name": "tasksPerMessage",
                "value": "20"
            },
            {
                "name": "deleteInvocationData",
                "value": "2"
            },
            {
                "name": "maxEngines",
```

```
                "value": "2147483647"
            },
            {
                "name": "taskMaxTime",
                "value": "9223372036854775807"
            },
            {
                "name": "priority",
                "value": "5"
            },
            {
                "name": "killCancelledTasks",
                "value": "true"
            },
            {
                "name": "checkpoint",
                "value": "false"
            },
            {
                "name": "unloadNativeLibrary",
                "value": "true"
            },
            {
                "name": "autoCancel",
                "value": "1"
            },
            {
                "name": "autoPackNum",
                "value": "0"
            },
            {
                "name": "collectionType",
                "value": "1"
            }
        ],
        "invocationInfo": [],
        "submitTime": 1643633925014,
        "driverHostname": "TIBCO-PF379VJL",
        "driverUsername": "internal",
        "resultProperties": null,
        "serviceId": 565597320224386901,
        "completed": false
    }
]
```


Result: Information about all Services is returned.

blacklisted-engines

Retrieves a list of IDs of Engines that have been blacklisted for a specified Service.

Example Request

```
GET http://example.com:8080/livecluster/rest/service/blacklisted-engines
```

Example Input

Parameters

Parameter Name	Data Type	Description	Sample Value
`serviceId`	Long	Service ID for which black-listed Engine IDs need to be retrieved	`7655032223441706693`

Example Response

```
[
    4848394891525229394
]
```

Result: IDs of blacklisted Engines are returned.

completed-service-invocation-count

Retrieves the number of completed invocations of the specified Service.

Example Request

```
GET http://example.com:8080/livecluster/rest/service/completed-service-
invocation-count
```

Example Input

Parameters

Parameter Name	Data Type	Description	Sample Value
`serviceId`	Long	Service ID for which completed Service invocation count needs to be retrieved	`7655032223441706693`

Example response

```
0
```

finished-service-count

Retrieves the total number of finished Services on the Broker.

Example Request

```
GET http://example.com:8080/livecluster/rest/service/finished-service-
count
```

Example Response

```
1
```

invocation-count

Retrieves the total number of pending and running invocations on the Broker.

Example Request

```
GET http://example.com:8080/livecluster/rest/service/invocation-count
```

Example Response

```
0
```

invocation-info

Retrieves the information about a particular invocation of a Service, for the given Service ID and Invocation ID.

Example Request

```
Get http://example.com:8080/livecluster/rest/service/invocation-info
```

Example Input

Parameters

Parameter Name	Data Type	Description	Sample Value
`jobId`	Long	Service ID of the Service whose information needs to be retrieved	`4697058754240048269`
`taskId`	Long	Invocation ID of the Service whose information needs to be retrieved	`5`

Example Response


```
{
    "description": null,
    "engine": null,
    "startTime": 0,
    "elapsedTime": 0,
    "submitTime": 1578912618068,
    "exception": 0,
    "id": 5,
    "info": null,
    "status": "Submitted",
    "computationalTime": 0,
    "inputSize": 533,
    "outputSize": 0,
    "reschedules": 0,
    "retries": 0
}
```


pending-invocation-count

Retrieves the number of pending invocations on the Broker.

Example Request

```
GET http://example.com:8080/livecluster/rest/service/pending-invocation-
count
```

Example Response

```
0
```

pending-service-invocation-count

Retrieves the number of pending invocations of the specified Service.

Example Request

```
GET http://example.com:8080/livecluster/rest/service/pending-service-
invocation-count
```

Example Input

Parameters

Parameter Name	Data Type	Description	Sample Value
`serviceId`	Long	Service ID for which Service invocation count needs to be retrieved	`7655032223441706693`

Example Response

```
0
```

registered-services

Retrieves a list of all registered Services; returns null if no Services are registered.

Example Request

```
GET http://example.com:8080/livecluster/rest/service/registered-services
```

Example Response

```
[
    "CPPCalculatorExample",
    "CUDAExample",
    "JavaAdderExample",
    "JavaCalculatorExample",
    "MICExample",
```

```
    "NETCalculatorExample",
    "NETSpeedLink",
    "PythonExample",
    "RadhocExample",
    "RCalculatorExample",
    "RPICalculatorExample",
    "SpeedLink"
]
```

running-invocation-count

Retrieves the number of invocations currently executed by Engines on the Broker.

Example Request

```
GET http://example.com:8080/livecluster/rest/service/running-invocation-
count
```

Example Response

```
0
```

running-service-count

Retrieves the total number of active Services on the Broker.

Example Request

```
GET http://example.com:8080/livecluster/rest/service/running-service-
count
```

Example Response

```
0
```

running-service-invocation-count

Retrieves the number of invocations currently executed by Engines of the specified Service.

Example Request

```
GET http://example.com:8080/livecluster/rest/service/running-service-
invocation-count
```

Example Input

Parameters

Parameter Name	Data Type	Description	Sample Value
`serviceId`	Long	Service ID for which running Service invocation count needs to be retrieved	`7655032223441706693`

Example Response

```
10
```

selected-invocation-info

Retrieves information of the given Service and invocation IDs.

Example Request

```
POST http://example.com:8080/livecluster/rest/service/selected-
invocation-info
```

Example Input

Parameters

Parameter Name	Data Type	Description	Sample Value
`serviceId`	Long	Service ID for which information needs to be retrieved	`4625254489192920053`
—	JSON	Invocation IDs for which information needs to be retrieved	`[1,5]`

Example Response


```
[
    {
        "description": null,
        "engine": {
            "instance": "0",
            "properties": null,
            "username": "win64vm108",
            "engineId": 9007929106493580894,
            "serviceId": -1,
            "invocationId": -1,
            "busy": false,
            "elapsedTime": -1
        },
        "startTime": 1578911378974,
        "elapsedTime": 10131,
        "submitTime": 1578911368704,
        "exception": 0,
        "id": 1,
        "info": "2,572.13 MFlops",
        "status": "Completed",
        "computationalTime": 10033,
        "inputSize": 533,
        "outputSize": 328,
        "reschedules": 0,
        "retries": 0
    },
    {
```

```
        "description": null,
        "engine": {
            "instance": "0",
            "properties": null,
            "username": "win64vm108",
            "engineId": 9007929106493580894,
            "serviceId": -1,
            "invocationId": -1,
            "busy": false,
            "elapsedTime": -1
        },
        "startTime": 1578911419455,
        "elapsedTime": 10082,
        "submitTime": 1578911368799,
        "exception": 0,
        "id": 5,
        "info": "2,180.66 MFlops",
        "status": "Completed",
        "computationalTime": 10065,
        "inputSize": 533,
        "outputSize": 328,
        "reschedules": 0,
        "retries": 0
    }
]
```


selected-service-info

Retrieves information of the given Service IDs running on the Broker.

Example Request

```
POST http://example.com:8080/livecluster/rest/service/selected-service-info
```

Example Input

Parameters

Parameter Name	Data Type	Description	Sample Value
—	JSON	List of Service IDs for which information needs to be retrieved	`8640091942181654218`

Example Response


```
[
    {
        "finishTime": -1,
        "pendingCount": 5,
        "runningCount": 0,
        "completedCount": 0,
        "totalCount": 5,
        "cancelled": false,
        "serviceName": "Linpack Test",
        "description": [
            {
                "name": "appName",
                "value": "DataSynapse Job/Service Test"
            },
            {
                "name": "serviceTypeName",
                "value": "LinpackServiceTest"
            },
            {
                "name": "serviceName",
                "value": "Linpack Test"
            },
            {
                "name": "class",
                "value": "java:examples.linpack.LinpackService"
            },
            {
                "name": "serviceType",
                "value": "Service"
            },
            {
                "name": "slaGroupName",
                "value": ""
            },
            {
                "name": "providerType",
                "value": "JavaProvider"
```

```
            },
            {
                "name": "username",
                "value": "internal"
            }
        ],
        "status": "Running",
        "options": [
            {
                "name": "autoPackMode",
                "value": "0"
            },
            {
                "name": "compressData",
                "value": "true"
            },
            {
                "name": "resultsPerMessage",
                "value": "100"
            },
            {
                "name": "resubmitOnDDTFailure",
                "value": "true"
            },
            {
                "name": "statusExpires",
                "value": "true"
            },
            {
                "name": "tasksPerMessage",
                "value": "20"
            },
            {
                "name": "deleteInvocationData",
                "value": "2"
            },
            {
                "name": "maxEngines",
                "value": "2147483647"
            },
            {
                "name": "taskMaxTime",
                "value": "92233720368547758O7"
            },
            {
                "name": "priority",
                "value": "5"
            },
```

```
            {
                "name": "killCancelledTasks",
                "value": "true"
            },
            {
                "name": "checkpoint",
                "value": "false"
            },
            {
                "name": "unloadNativeLibrary",
                "value": "true"
            },
            {
                "name": "autoCancel",
                "value": "1"
            },
            {
                "name": "autoPackNum",
                "value": "0"
            },
            {
                "name": "collectionType",
                "value": "1"
            }
        ],
        "finished": false,
        "resultProperties": null,
        "invocationInfo": [],
        "completed": false,
        "submitTime": 1578909679845,
        "serviceId": 8640091942181654218,
        "driverHostname": "mkamdar-p50.apac.tibco.com",
        "driverUsername": "internal"
    }
]
```


service-binding

Retrieves the binding of the specified Service.

Example Request

```
GET http://example.com:8080/livecluster/rest/service/service-binding
```

Example Input

Parameters

Parameter Name	Data Type	Description	Sample Value
`name`	String	Name of the Service	`LinpackServiceTest`

Example Response


```
{
    "parameters": [
        {
            "name": "className",
            "value": "examples.linpack.LinpackService"
        }
    ],
    "options": null,
    "description": [
        {
            "name": "serviceTypeName",
            "value": "LinpackServiceTest"
        },
        {
            "name": "providerType",
            "value": "JavaProvider"
        }
    ],
    "name": "LinpackServiceTest",
    "uidescription": "",
    "type": "java",
    "containerBinding": [
        {
            "name": "serviceMethods",
            "value": "*"
        }
    ]
}
```

service-count

Retrieves the total number of Services on the Broker.

Example Request

```
GET http://example.com:8080/livecluster/rest/service/service-count
```

Example Response

```
1
```

service-ids

Retrieves the list of all Service IDs on the Broker.

Example Request

```
GET http://example.com:8080/livecluster/rest/service/service-ids
```

Example Response

A list of Service IDs on the Broker is returned.

```
[
    4371703018904140926
]
```

service-info

Retrieves Service information of the given Service ID.

Example Request

```
GET http://example.com:8080/livecluster/rest/service/service-info
```

Example Input

Parameters

Parameter Name	Data Type	Description	Sample Value
`jobId`	Long	Service ID of the Service whose information needs to be retrieved	`8656732499466570974`

Example Response


```
{
    "finishTime": 1641820932754,
    "pendingCount": 0,
    "runningCount": 0,
    "completedCount": 10,
    "totalCount": 10,
    "serviceName": "Linpack Test",
    "cancelled": false,
    "description": [
        {
            "name": "appName",
            "value": "DataSynapse Job/Service Test"
        },
        {
            "name": "serviceTypeName",
            "value": "LinpackServiceTest"
        },
        {
            "name": "serviceName",
            "value": "Linpack Test"
        },
        {
            "name": "class",
            "value": "java:examples.linpack.LinpackService"
        },
```

```
        {
            "name": "serviceType",
            "value": "Service"
        },
        {
            "name": "slaGroupName",
            "value": ""
        },
        {
            "name": "providerType",
            "value": "JavaProvider"
        },
        {
            "name": "username",
            "value": "internal"
        }
    ],
    "status": "Finished",
    "finished": true,
    "options": [
        {
            "name": "autoPackMode",
            "value": "0"
        },
        {
            "name": "compressData",
            "value": "true"
        },
        {
            "name": "resultsPerMessage",
            "value": "100"
        },
        {
            "name": "resubmitOnDDTFailure",
            "value": "true"
        },
        {
            "name": "statusExpires",
            "value": "true"
        },
        {
            "name": "tasksPerMessage",
            "value": "20"
        },
        {
            "name": "deleteInvocationData",
            "value": "2"
```

```
        },
        {
            "name": "maxEngines",
            "value": "2147483647"
        },
        {
            "name": "taskMaxTime",
            "value": "92233720368547758O7"
        },
        {
            "name": "priority",
            "value": "5"
        },
        {
            "name": "killCancelledTasks",
            "value": "true"
        },
        {
            "name": "checkpoint",
            "value": "false"
        },
        {
            "name": "unloadNativeLibrary",
            "value": "true"
        },
        {
            "name": "autoCancel",
            "value": "1"
        },
        {
            "name": "autoPackNum",
            "value": "0"
        },
        {
            "name": "collectionType",
            "value": "1"
        }
    ],
    "invocationInfo": [],
    "completed": true,
    "serviceId": 4371703018904140926,
    "resultProperties": null,
    "driverHostname": "TIBCO-PF379VJL",
    "driverUsername": "internal",
    "submitTime": 1641820912300
}
```

service-info-by-properties

Retrieves the Service information of all Services on the Broker that match the description condition.

Example Request

```
POST http://example.com:8080/livecluster/rest/service/service-info-by-properties
```

Example Input

Parameters

Parameter Name	Data Type	Description	Sample Value

—	JSON	Description condition data depending on information of which Services need to be retrieved	<pre>{
			    "sets": null,
			    "conditions": [
			        {
			            "name": "serviceType",
			            "comparison": 1,
			            "value": "Service",
			            "nullCompare": false
			        },
			        {
			            "name": "serviceTypeName",
			            "comparison": 1,
			            "value": "LinpackServiceTest",
			            "nullCompare": false
			        }</pre>

Parameter Name	Data Type	Description	Sample Value
			], "type": 0 }

Example Response


```
"[
    {
        "finishTime": 1578980454833,
        "pendingCount": 0,
        "runningCount": 0,
        "completedCount": 100,
        "totalCount": 100,
        "cancelled": false,
        "serviceName": "Linpack Test",
        "description": [
            {
                "name": "appName",
                "value": "DataSynapse Job/Service Test"
            },
            {
                "name": "serviceTypeName",
                "value": "LinpackServiceTest"
            },
            {
                "name": "serviceName",
                "value": "Linpack Test"
            },
            {
                "name": "class",
                "value": "java:examples.linpack.LinpackService"
            },
            {
                "name": "serviceType",
                "value": "Service"
            },
            {
                "name": "slaGroupName",
                "value": ""
            },
```

```
            {
                "name": "providerType",
                "value": "JavaProvider"
            },
            {
                "name": "username",
                "value": "internal"
            }
        ],
        "status": "Finished, Task Errors",
        "options": [
            {
                "name": "autoPackMode",
                "value": "0"
            },
            {
                "name": "compressData",
                "value": "true"
            },
            {
                "name": "resultsPerMessage",
                "value": "100"
            },
            {
                "name": "resubmitOnDDTFailure",
                "value": "true"
            },
            {
                "name": "statusExpires",
                "value": "true"
            },
            {
                "name": "tasksPerMessage",
                "value": "20"
            },
            {
                "name": "deleteInvocationData",
                "value": "2"
            },
            {
                "name": "maxEngines",
                "value": "2147483647"
            },
            {
                "name": "taskMaxTime",
                "value": "9223372036854775807"
            },
```

```
            {
                "name": "priority",
                "value": "5"
            },
            {
                "name": "killCancelledTasks",
                "value": "true"
            },
            {
                "name": "checkpoint",
                "value": "false"
            },
            {
                "name": "unloadNativeLibrary",
                "value": "true"
            },
            {
                "name": "autoCancel",
                "value": "1"
            },
            {
                "name": "autoPackNum",
                "value": "0"
            },
            {
                "name": "collectionType",
                "value": "1"
            }
        ],
        "finished": true,
        "resultProperties": null,
        "driverUsername": "internal",
        "driverHostname": "10.128.88.108",
        "invocationInfo": [],
        "serviceId": 1185560448718947021,
        "completed": true,
        "submitTime": 1578979437654
    },
    {
        "finishTime": 1578980058243,
        "pendingCount": 0,
        "runningCount": 0,
        "completedCount": 1,
        "totalCount": 1,
        "cancelled": false,
        "serviceName": "Linpack Test",
        "description": [
```

```
            {
                "name": "appName",
                "value": "DataSynapse Job/Service Test"
            },
            {
                "name": "serviceTypeName",
                "value": "LinpackServiceTest"
            },
            {
                "name": "serviceName",
                "value": "Linpack Test"
            },
            {
                "name": "class",
                "value": "java:examples.linpack.LinpackService"
            },
            {
                "name": "serviceType",
                "value": "Service"
            },
            {
                "name": "slaGroupName",
                "value": ""
            },
            {
                "name": "providerType",
                "value": "JavaProvider"
            },
            {
                "name": "username",
                "value": "internal"
            }
        ],
        "status": "Finished",
        "options": [
            {
                "name": "autoPackMode",
                "value": "0"
            },
            {
                "name": "compressData",
                "value": "true"
            },
            {
                "name": "resultsPerMessage",
                "value": "100"
            },
```

```
            {
                "name": "resubmitOnDDTFailure",
                "value": "true"
            },
            {
                "name": "statusExpires",
                "value": "true"
            },
            {
                "name": "tasksPerMessage",
                "value": "20"
            },
            {
                "name": "deleteInvocationData",
                "value": "2"
            },
            {
                "name": "maxEngines",
                "value": "2147483647"
            },
            {
                "name": "taskMaxTime",
                "value": "9223372036854775807"
            },
            {
                "name": "priority",
                "value": "5"
            },
            {
                "name": "killCancelledTasks",
                "value": "true"
            },
            {
                "name": "checkpoint",
                "value": "false"
            },
            {
                "name": "unloadNativeLibrary",
                "value": "true"
            },
            {
                "name": "autoCancel",
                "value": "1"
            },
            {
                "name": "autoPackNum",
                "value": "0"
```

```
            },
            {
                "name": "collectionType",
                "value": "1"
            }
        ],
        "finished": true,
        "resultProperties": null,
        "driverUsername": "internal",
        "driverHostname": "10.128.88.108",
        "invocationInfo": [],
        "serviceId": 4863259086811987322,
        "completed": true,
        "submitTime": 1578980045220
    }
]"
```


service-invocation-count

Retrieves the total number of invocations of the specified Service.

Example Request

```
GET http://example.com:8080/livecluster/rest/service/service-invocation-count
```

Example Input

Parameters

Parameter Name	Data Type	Description	Sample Value
`serviceId`	Long	Service ID for which Service invocation count needs to be retrieved	`3801168623022738886`

Example Response

```
10
```

task-expiration-event-count

Retrieves the total number of task expiration events of the specified Service.

Example Request

```
GET http://example.com:8080/livecluster/rest/service/task-expiration-
event-count
```

Example Input

Parameters

Parameter Name	Data Type	Description	Sample Value
`serviceId`	Long	Service ID for which total number of task expiration events need to be retrieved	`3801168623022738886`

Example Response

```
0
```

available

Retrieves whether the methods are available.

Example Request

```
GET http://example.com:8080/livecluster/rest/service/available
```

Example Response

`True` or `False`

list-resources

Lists resources on the Server.

Example Request

```
GET http://example.com:8080/livecluster/rest/service/list-resources
```

Example Input

Parameters

Parameter Name	Data Type	Description	Sample Value
`staged`	Boolean	Staged flag. Can be true or false.	`False`

Example Response

```
[
    "calculator-1.0.0.1.tar.gz",
    "cppbridge-linux-gcc34-7.1.zip",
    "cppbridge-linux64-gcc34-7.1.zip",
    "cppbridge-linux64-gcc49-7.1.zip",
    "cppbridge-linux64-gcc83-7.1.zip",
    "cppbridge-win32-vc12-7.1.zip",
    "cppbridge-win32-vc14-7.1.zip",
    "cppbridge-win64-vc12-7.1.zip",
    "cppbridge-win64-vc14-7.1.zip",
    "cppbridge-win64-vc15-7.1.zip",
    "cppbridge-win64-vc16-7.1.zip",
    "jre-linux-1.8.0.311.tar.gz",
    "jre-linux64-1.8.0.311.tar.gz",
    "jre-win32-1.8.0.311.zip",
```

```
    "jre-win64-1.8.0.311.zip",
    "net5bridge-linux64-gcc49-7.1.zip",
    "net5bridge-win64-7.1.zip",
    "netbridge-7.1.zip",
    "netbridge-win64-7.1.zip",
    "pybridge-linux64-gcc34-7.1.tar.gz",
    "pybridge-win64-vc14-7.1.tar.gz",
    "python-linux64-3.7.0.zip",
    "python-win64-3.7.0.zip",
    "rbridge-linux64-7.1.tar.gz",
    "rbridge-win64-7.1.tar.gz",
    "SpeedLink-7.1.0.179676.tar.gz"
]
```

register-service

Registers the binding in the Service Registry; this must be done on the Primary Director.

Example Request

```
POST http://example.com:8080/livecluster/rest/service/register-service
```

Example Input

Parameters

Parameter Name	Data Type	Description	Sample Value

—	JSON	Service binding date that needs to be registered	<pre>{ "parameters": [
			{
			"name": "className",
			"value":
			"examples.calculator.service.</pre>

Parameter Name	Data Type	Description	Sample Value


```
JavaCalculator"
        }
    ],
    "options": [
        {
            "name": "gridLibrary",
            "value": "calculator"
        }
    ],
    "description": [
        {
            "name":
"serviceTypeName",
            "value":
"JavaCalculatorExample"
        },
        {
            "name": "serviceName",
            "value": "Java
Calculator Example"
        },
        {
            "name": "providerType",
            "value": "JavaProvider"
        }
    ],
    "name":
"JavaCalculatorExample",
    "uidescription": "Cross-
language Java Service example that
performs basic calculator
operations on strings",
    "type": "java",
    "containerBinding": [
        {
            "name":
"targetPackage",
            "value":
"examples.calculator.client" },
        {
```

Parameter Name	Data Type	Description	Sample Value
			<pre>"name": "xmlSerialization", "value": "false" }, { "name": "serviceMethods", "value": "*" }, { "name": "setStateMethods", "value": "setMemory" }, { "name": "appendStateMethods", "value": "addToMemory" }] }"</pre>

Example Response

```
204 no content
```

Result: Service type is created.

all-finished-services

Removes the information of all finished Services from the Broker.

Example Request

```
DELETE http://example.com:8080/livecluster/rest/service/all-finished-services
```

Example Response

```
204 no content
```

Result: All finished services are removed.

finished-service

Removes the information of finished Services from the Broker.

Example Request

```
DELETE http://example.com:8080/livecluster/rest/service/finished-service
```

Example Input

Parameters

Parameter Name	Data Type	Description	Sample Value
`jobId`	Long	Service ID for which finished Service needs to be removed	`3801168623022738886`

Example Response

```
204 no content
```

Result: Service with the specified ID is deleted.

resource-exists

Checks whether a file exists in the designated area.

Example Request

```
GET http://example.com:8080/livecluster/rest/service/resource-exists
```

Example Input

Parameters

Parameter Name	Data Type	Description	Sample Value
`name`	String	Name of the resource that needs to be checked	`calculator-1.0.0.1.tar.gz`
`staged`	Boolean	Staged flag. True or false.	`false`

Example Response

`True` or `False`.

expires

Sets whether Service information is removed during Service cleanup.

Example Request

```
PUT http://example.com:8080/livecluster/rest/service/expires
```

Example Input

Parameters

Parameter Name	Data Type	Description	Sample Value
`jobId`	Long	Job ID for which the expiration flag needs to be updated	`3001310211432770499`
`expires`	Boolean	Expired flag. True or false	`true`

Example Response

```
204 no content
```

Result: Service information is removed during Service cleanup when set to true.

priority

Sets the priority of the specified Service.

Example Request

```
PUT http://example.com:8080/livecluster/rest/service/priority
```

Example Input

Parameters

Parameter Name	Data Type	Description	Sample Value
`jobId`	Long	Service ID whose priority needs to be set	`3001310211432770499`
`priority`	Int	Priority for the Service.	`2`

Example Response

```
204 no content
```

Result: Service priority is set to the specified value.

unregister-service

Unregisters the binding in the Service Registry; this must be done on the Primary Director.

Example Request

```
DELETE http://example.com:8080/livecluster/rest/service/unregister-
service
```

Example Input

Parameters

Parameter Name	Data Type	Description	Sample Value
`name`	String	Name of the Service that needs to be unregistered	`JavaCalculatorExample`

Example Response

```
204 no content
```

Result: Service is removed from Service Types.

UserAdmin

UserAdmin APIs provide administrative access to the Users and Roles on a Manager. They are listed in the following table:

Method	Method Type	Description
user	POST	Creates a new user with an initial password.
role	DELETE	Deletes the named role.
user	DELETE	Deletes the user with the given username.
all-roles	GET	Retrieves all roles.
all-users	GET	Retrieves all users.
role	GET	Retrieves the named role.
user	GET	Retrieves the user with the given username.
available	GET	Retrieves whether the methods are available.
role	POST	Adds or updates a role.
user	PUT	Updates the user.

user

Creates a new user with an initial password.

Example Request

```
POST http://example.com:8080/livecluster/rest/user/user
```

Example Input

Parameters

Parameter Name	Data Type	Description	Sample Value
—	JSON	User data that needs to be created	`{ "username": "admin1", "firstName": "", "lastName": "", "email": "", "roles": "Configure" }`
`password`	String	Password for the user.	`admin1`

Example Response

`204 no content`

Result: User is created.

role

Deletes the named role.

Example Request

```
DELETE http://example.com:8080/livecluster/rest/user/role
```

Example Input

Parameters

Parameter Name	Data Type	Description	Sample Value
`name`	String	Name of the role you want to delete	`test`

Example Response

```
204 no content
```

Result: The specified role is deleted.

user

Deletes the user with the given username.

Example Request

```
DELETE http://example.com:8080/livecluster/rest/user/user
```

Example Input

Parameters

Parameter Name	Data Type	Description	Sample Value
`name`	String	Name of the user to be deleted.	`admin1`

Example Response

```
204 no content
```

Result: The specified user is deleted.

all-roles

Retrieves all roles.

Example Request

```
GET http://example.com:8080/livecluster/rest/user/all-roles
```

Example Response

```
"[
    {
        "name": "Configure",
        "description": "The Configure role, all permissions.",
        "managerAccess": "*",
        "ldapGroup": "",
        "maxPriority": -1,
        "permissions": "Batch Manage,Batch Admin View,Batch Definition
View,Batch Registry View,Batch Definition Edit,Batch Schedule
View,Broker Admin View,Broker Admin Manage,Broker Configuration
View,Broker Configuration Manage,Broker Monitor,Broker Reports,Broker
Routing Manage,Broker Routing View,Credential Repository View,Credential
Repository Edit,Current Log,Diagnostics,Service Condition View,Service
Condition Edit,ExtraCondition Modify,Documentation,Driver View,Driver
Admin,Driver Download,Driver Events,Engine View,Engine Manage,Engine
Configuration View,Engine Configuration Edit,Engine Daemon View,Engine
Daemon Manage,Engine Events,Engine Install,Engine Log Files,Engine
Properties View,Engine Properties Edit,Engine Properties List
View,Engine Properties List Edit,Engine Daemon Reports,Event
Subscription View,Event Subscription Edit,Resource Deployment
View,Resource Deployment Maker,Resource Deployment Checker,GridCache
View,GridCache Manage,GridCache Schema View,GridCache Schema Edit,Grid
Monitor,Server Hook View,Server Hook Manage,Server Hook Edit,Import
Export,Service Session View,Service Session View Group,Service Session
Manage,Service Session Manage Group,Service Reports,License View,License
Upload,Manager Reconfigure,Manager Configuration View,Manager
Configuration Edit,Service Group View,Service Group Manage,Service Type
List,Service Type Manage,Service Type Edit,Service Type View,Service
Runner View,Service Runner Manage,SNMP View,SNMP Edit,Task Reports,Test
Job,User Manage,User View,User Events,Role View,Role Edit,Auth View,Auth
Edit,Service Access to All Users,Dashboard View,Dashboard Grid
View,Dashboard Services View,Dashboard Brokers View,Dashboard Drivers
View,Dashboard Daemons View,Dashboard Engines View,Web Services List
```

```
View,Specify Additional RunAs User,Execute Services,Quarantine
Engine,Service Diagnostics"
    },
    {
        "name": "Manage",
        "description": "Management of all services and other
components",
        "managerAccess": "*",
        "ldapGroup": "",
        "maxPriority": -1,
        "permissions": "Batch Manage,Batch Admin View,Batch Schedule
View,Broker Admin View,Broker Admin Manage,Broker Configuration
View,Broker Configuration Manage,Broker Monitor,Broker Reports,Broker
Routing Manage,Broker Routing View,Current Log,Diagnostics,Service
Condition View,Service Condition Edit,ExtraCondition
Modify,Documentation,Driver View,Driver Admin,Driver Download,Driver
Events,Engine View,Engine Manage,Engine Configuration View,Engine Daemon
View,Engine Daemon Manage,Engine Events,Engine Install,Engine Log
Files,Engine Properties View,Engine Properties Edit,Engine Properties
List View,Engine Properties List Edit,Engine Daemon Reports,Event
Subscription View,Resource Deployment View,Resource Deployment
Maker,Resource Deployment Checker,GridCache View,GridCache
Manage,GridCache Schema View,GridCache Schema Edit,Grid Monitor,Server
Hook View,Service Session View,Service Session View Group,Service
Session Manage,Service Session Manage Group,Service Reports,License
View,Service Group View,Service Group Manage,Service Type List,Service
Type Manage,Service Type View,SNMP View,Task Reports,Test Job,User
View,User Events,Role View,Auth View,Dashboard View,Dashboard Grid
View,Dashboard Services View,Dashboard Brokers View,Dashboard Drivers
View,Dashboard Daemons View,Dashboard Engines View,Web Services List
View,Execute Services,Quarantine Engine,Service Diagnostics"
    },
    {
        "name": "Service",
        "description": "View plus management of own services",
        "managerAccess": "*",
        "ldapGroup": "",
        "maxPriority": -1,
        "permissions": "Broker Admin View,Broker Configuration
View,Broker Monitor,Broker Routing View,Diagnostics,Service Condition
View,Documentation,Driver View,Driver Download,Engine View,Engine
Configuration View,Engine Daemon View,Engine Install,Engine Log
Files,Engine Properties View,Engine Properties List View,Resource
Deployment View,GridCache View,GridCache Schema View,Grid
Monitor,Service Session View,Service Session View Group,Service Session
Manage,Service Session Manage Group,License View,Service Group
View,Service Type List,Service Type View,Test Job,User View,Role
```

```
View,Auth View,Dashboard View,Dashboard Grid View,Dashboard Services
View,Dashboard Brokers View,Dashboard Drivers View,Dashboard Daemons
View,Dashboard Engines View,Web Services List View,Execute
Services,Quarantine Engine,Service Diagnostics"
    },
    {
        "name": "View",
        "description": "Default View Role",
        "managerAccess": "*",
        "ldapGroup": "",
        "maxPriority": -1,
        "permissions": "Broker Admin View,Broker Configuration
View,Broker Monitor,Broker Routing View,Service Condition
View,Documentation,Driver View,Engine View,Engine Configuration
View,Engine Daemon View,Engine Install,GridCache View,Grid
Monitor,Service Session View,Service Session View Group,License
View,Service Group View,Service Type List,User View,Role View,Auth
View,Dashboard View,Dashboard Grid View,Dashboard Services
View,Dashboard Brokers View,Dashboard Drivers View,Dashboard Daemons
View,Dashboard Engines View"
    }
]"
```

all-users

Retrieves all users.

Example Request

```
GET http://example.com:8080/livecluster/rest/users/all-users
```

Example Response

```
[
    {
        "username": "admin",
        "firstName": "",
        "lastName": "",
        "email": "",
        "roles": "Configure"
```

```
    }
]
```

role

Retrieves the named role.

Example Request

```
GET http://example.com:8080/livecluster/rest/user/role
```

Example Input

Parameters

Parameter Name	Data Type	Description	Sample Value
`name`	String	Name of the role whose information needs to be retrieved	`Service`

Example Response

```
{
    "name": "Service",
    "description": "View plus management of own services",
    "managerAccess": "*",
    "ldapGroup": "",
    "maxPriority": -1,
    "permissions": "Broker Admin View,Broker Configuration View,Broker
Monitor,Broker Routing View,Diagnostics,Service Condition
View,Documentation,Driver View,Driver Download,Engine View,Engine
Configuration View,Engine Daemon View,Engine Install,Engine Log
Files,Engine Properties View,Engine Properties List View,Resource
Deployment View,GridCache View,GridCache Schema View,Grid
Monitor,Service Session View,Service Session View Group,Service Session
```

```
Manage,Service Session Manage Group,License View,Service Group
View,Service Type List,Service Type View,Test Job,User View,Role
View,Auth View,Dashboard View,Dashboard Grid View,Dashboard Services
View,Dashboard Brokers View,Dashboard Drivers View,Dashboard Daemons
View,Dashboard Engines View,Web Services List View,Execute
Services,Quarantine Engine,Service Diagnostics"
}
```

user

Retrieves the user with the given username.

Example Request

```
GET http://example.com:8080/livecluster/rest/user/user
```

Example Input

Parameters

Parameter Name	Data Type	Description	Sample Value
`name`	String	Name of the user whose information you want to retrieve	`admin`

Example Response

```
{
    "username": "admin",
    "firstName": "",
    "lastName": "",
    "email": "",
    "roles": "Configure"
}
```

available

Retrieves whether the methods are available.

Example Request

```
GET http://example.com:8080/livecluster/rest/user/available
```

Example response

`True` or `False`

role

Adds or updates a role.

Example Request

```
POST http://example.com:8080/livecluster/rest/user/role
```

Example Input

Parameters

Parameter Name	Data Type	Description	Sample Value
—	JSON	Role data that needs to be added or updated	`{ "name": "Service1234", "description": "View plus management of own services", "managerAccess": "*",`

Parameter Name	Data Type	Description	Sample Value
			"ldapGroup": "", "maxPriority": -1, "permissions": "Broker Admin View,Broker Configuration View,Broker Monitor,Broker Routing View,Diagnostics,Service Condition View,Documentation,Driver View,Driver Download,Engine View,Engine Configuration View,Engine Daemon View,Engine Install,Engine Log Files,Engine Properties View,Engine Properties List View,Resource Deployment View,GridCache View,GridCache Schema View,Grid Monitor,Service Session View,Service Session View Group,Service Session Manage,Service Session Manage Group,License View,Service Group View,Service Type List,Service Type View,Test Job,User View,Role View,Auth View,Dashboard View,Dashboard Grid View,Dashboard Services View,Dashboard Brokers View,Dashboard Drivers View,Dashboard Daemons View,Dashboard Engines View,Web Services List View,Execute Services,Quarantine Engine,Service Diagnostics" }

Example Response

New role is created.

user

Updates the user.

Example Request

```
PUT http://example.com:8080/livecluster/rest/user/user
```

Example Input

Parameters

Parameter Name	Data Type	Description	Sample Value
—	JSON	Role data that needs to be updated	`{ "username": "admin", "firstName": "", "lastName": "", "email": "xyz@like.com", "roles": "Configure" }`

Example Response

User data is updated successfully.

Version

Version APIs provide release version information. They are listed in the following table:

Method	Method Type	Description
version-release-name	GET	Retrieves release version.
build-version	GET	Retrieves build version.

version-release-name

Retrieves release version.

Example Request

```
GET http://example.com:8080/livecluster/rest/version/version-release-
name
```

Example Response

```
7.1.0
```

build-version

Retrieves build version.

Example Request

```
GET http://example.com:8080/livecluster/rest/version/build-version
```

Example Response

```
7.1.0.172681
```

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 TIBCO Product Documentation website, mainly in HTML and PDF formats.

The TIBCO 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.

The following documents for this product can be found in the TIBCO Documentation site:

- TIBCO GridServer® Release Notes
- TIBCO GridServer® Installation
- TIBCO GridServer® Introducing TIBCO GridServer®
- TIBCO GridServer® Administration
- TIBCO GridServer® Developer's Guide
- TIBCO GridServer® Upgrade
- TIBCO GridServer® Security
- TIBCO GridServer® COM Integration Tutorial
- TIBCO GridServer® PDriver Tutorial
- TIBCO GridServer® Speedlink
- TIBCO GridServer® Service-Oriented Integration Tutorial

How to Contact TIBCO Support

Get an overview of TIBCO Support. You can contact TIBCO Support in the following ways:

- For accessing the Support Knowledge Base and getting personalized content about products you are interested in, visit the TIBCO Support website.
- For creating a Support case, you must have a valid maintenance or support contract with TIBCO. You also need a user name and password to log in to TIBCO Support website. If you do not have a user name, you can request one by clicking **Register** on the website.

How to Join TIBCO Community

TIBCO Community is the official channel for TIBCO customers, partners, and employee subject matter experts to share and access their collective experience. TIBCO Community offers access to Q&A forums, product wikis, and best practices. It also offers access to extensions, adapters, solution accelerators, and tools that extend and enable customers to gain full value from TIBCO products. In addition, users can submit and vote on feature requests from within the TIBCO Ideas Portal. For a free registration, go to TIBCO Community.

Legal and Third-Party Notices