TIBCO DataSynapse GridServer® Manager

Speedlink Guide

Version 7.1.0
July 2022

Contents

Introduction

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

Before You Begin

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

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

Installing SpeedLink

This section covers SpeedLink’s system requirements and installation process.

Requirements

For current GridServer requirements see the GridServer Readme.

SpeedLink supports all Services except Command Services which do not benefit from lower latencies, and R Services. All other Services are supported including Java, .NET (`String/byte[]` only), and C++ —all compilers that are supported by standard Services.

Installing SpeedLink

SpeedLink is preinstalled in GridServer. To use SpeedLink, install the GridServer SDK the same way you install it to use other Services.

For more information about installing the SDK, see the *TIBCO GridServer® Developer's Guide*.

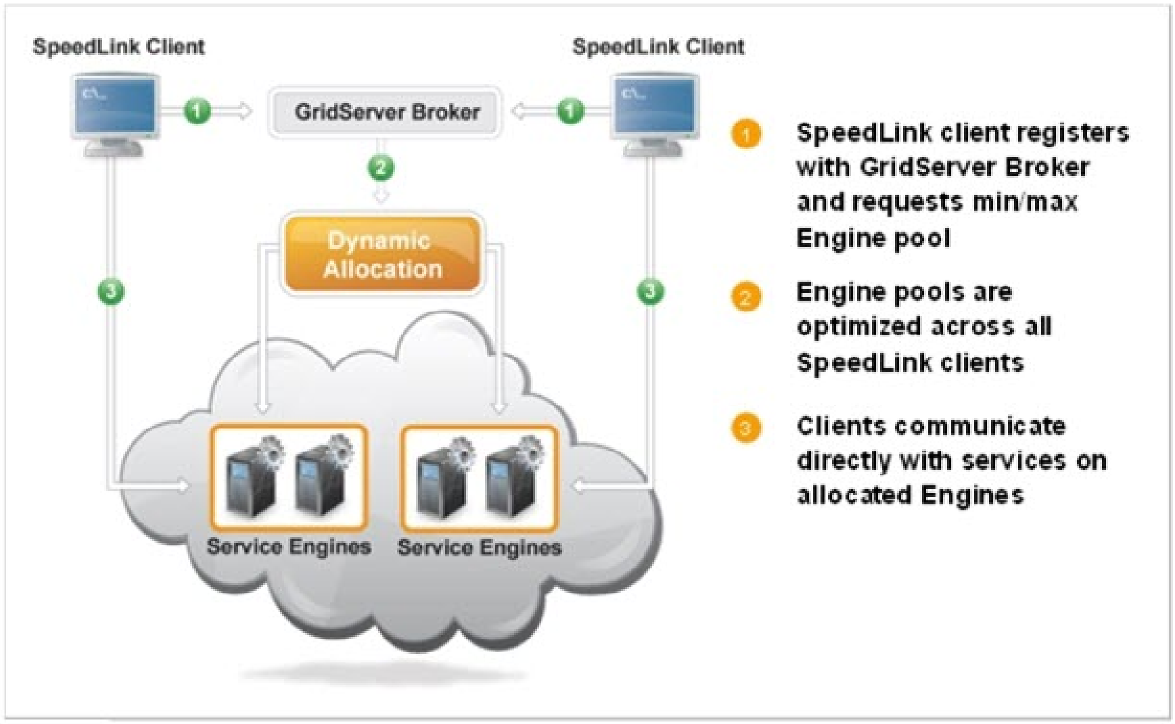
SpeedLink Architecture

This section covers SpeedLink architecture and describes appropriate uses for SpeedLink Services.

Overview

SpeedLink is a high throughput, low latency compute paradigm implemented using GridServer. SpeedLink can deliver end-to-end compute latencies of less than one millisecond, depending on the server and network environment, with no theoretical upper limit on aggregate grid-wide operation throughput. Individual clients can encounter a practical throughput limit based on the client system's performance.

SpeedLink uses the Engine pool of a GridServer installation, sharing Engines seamlessly with other GridServer Services. SpeedLink maintains key GridServer features like Service prioritization and Engine balancing, and standard Services can co-exist with SpeedLink Services, even within the same client or Engine process. SpeedLink Services request Engines from a GridServer Broker, and this allocation operates much like a standard GridServer Service invocation. Unlike standard Services, allocated Engines communicate directly with the SpeedLink client, resulting in the desired throughput and latency characteristics. Allocated Engines are periodically returned to the Engine pool.


SpeedLink Client
SpeedLink Client
GridServer Broker
Dynamic Allocation
Service Engines
Service Engines
1 SpeedLink client registers with GridServer Broker and requests min/max Engine pool
2 Engine pools are optimized across all SpeedLink clients
3 Clients communicate directly with services on allocated Engines

SpeedLink uses GridServer Engines from a common Engine pool.

SpeedLink's design supports easy code migration from standard GridServer Services to SpeedLink Services, and back again to standard Services if desired. The SpeedLink API provides the `Service` interface for SpeedLink Services. The application programmer simply calls the `createService` method on a different factory class. The resulting `Service` interface (or virtual object, depending on the chosen implementation language) can be used the same way you use a standard Service.

Use Cases

The canonical use case for standard GridServer Services is risk analysis. Risk is highly computational, with typical run times in the order of seconds or minutes, and it is infinitely hungry for compute nodes, so horizontal scalability is preferred over low latency.

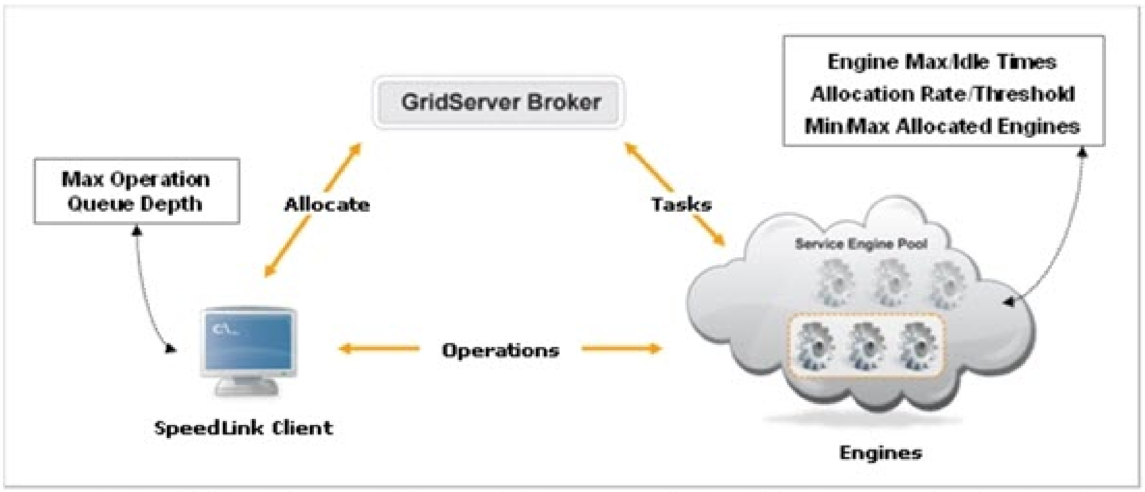
In contrast, a good example of a suitable SpeedLink Service is a Forex trading application, where portfolios, trades, and strategies need to be constantly re-evaluated based on market conditions delivered from a Forex market data feed. This use requires high throughput since Forex rates might change dozens of times a second, and there are hundreds of positions being evaluated. At the same time, the computational requirements for individual positions or strategies are low, so low latency is preferred over horizontal scalability.

SpeedLink Architecture

Like GridServer, SpeedLink uses a client application that calls Services running on many Engines, and then collects and handles the results. But instead of using the Broker as a middleman for all communication, SpeedLink clients only communicate with the Broker when reserving those Engines. Submitted or executed invocations are then treated as low-latency operations and directly communicated to Service code running on GridServer Engines. SpeedLink enables high throughput by bypassing the Broker.

Operations

In GridServer, the term *task* describes the execution of a Service invocation, including reserving an Engine, along with its input and output. For SpeedLink Services, the SpeedLink task executes many invocations over the duration of the Engine reservation. Because of this, we introduce the term *Operation* to refer to each invocation execution. The are many operations within one task.



SpeedLink architecture: Clients and Engines communicate with Broker; operations handled directly.

Clients

The SpeedLink client is the program developed to call a Service implementation running on multiple Engines, and then collate or process the operation results. Your application code in the SpeedLink client creates its Services through the `SpeedLinkServiceFactory`, setting additional options as appropriate. The factory's `createService` method returns a concrete class that implements the Service interface. Internal to the concrete class, `submit`, `execute`, and a subset of other Service methods are implemented. Service Proxies are not supported; client code must use the `Service.submit` or `execute` methods to invoke Service code.

The direct connection between a SpeedLink client and a SpeedLink Engine is initiated from the Engine back to a server socket on the client. Each client requires only one open port that shared between all Service instances created.

Services

The SpeedLink Service implementation, running on the GridServer Engine, corresponds to the SpeedLink client implementation. SpeedLink clients connect to the SpeedLink Service implementation. Like GridServer, SpeedLink operations have the following lifecycle:

1. Your application client tells the SpeedLink client what application Service to use.
2. The SpeedLink client passes that information to the SpeedLink Service implementation.
3. The SpeedLink Service implementation manages the application Service.

Consequently, application Service code need not change to run as a SpeedLink Service. If you deploy a Service with low-latency performance on your GridServer Broker, you can access it as a standard Service by having your client use `ServiceFactory`, or access it as a SpeedLink Service by using `SpeedLinkServiceFactory`. You can even access the same Service code as both standard and SpeedLink Service instances from within the same client process.

In contrast to GridServer Services, the SpeedLink Service itself manages the connection back to the SpeedLink client, receiving Service input and returning output, like a simplified version of the GridServer Engine's connection to a Broker. It also watches the `ALLOCATION_TIME` and `IDLE_LINGER` timers, ending the Engine's reservation when those total or idle timers expire, respectively.

Administration

Administration and maintenance of SpeedLink Services is almost identical to GridServer because SpeedLink uses GridServer Brokers and Engines to maintain an Engine pool. SpeedLink has its own Service Type; SpeedLink clients are managed on the Client Admin page, and SpeedLink Services appear on the Dashboard, as do GridServer Services.

However, there are features not supported in SpeedLink that might change some aspects of your grid management. These include:

- Reporting— due to the large amount of reporting information generated, SpeedLink's task-level reporting is summary only; operation-level reporting must be performed by the client.
- Service Options— for example, a grid configured to use redundant scheduling, which is not supported with SpeedLink, might need to be managed differently.

Despite these differences, an administrator already familiar with GridServer's Administration Tool has no difficulty managing SpeedLink Services.

Developing Applications for SpeedLink

A Service object on a client can create and use a Service implemented in the same or another language. This section discusses how to implement a SpeedLink Service client—the code calling a Service on the Driver, and a SpeedLink Service—the code that runs on the Engine).

Running a Service by Using SpeedLink

Almost any GridServer Service can be executed by using SpeedLink. You can use an existing `ServiceType` and its Grid Libraries. All you need to do to run a Service by using SpeedLink is, in the client code, replace the `ServiceFactory` class with `SpeedLinkServiceFactory`, and `ServiceInvocationHandler` with `SpeedLinkInvocationHandler`.

The factory class returns an object of type `SpeedLinkService`, which extends the `Service` interface. The most important feature of the new type is that it enables access to detailed reporting information. If you are not interested in this information but might to have interchangeability with standard Services, you can instead use the `Service` interface. This factory class also has all additional options needed for the Service declared as string constants.

This example creates a factory object and synchronously calls a Service implementation in Java:

```
// Create a Service instance of JavaAdderExample
Service s =
    SpeedLinkServiceFactory.getInstance().createService(
        "JavaAdderExample", initData, options, description,
        condition);
..........
// Perform Synchronous add
Object[] arguments = new Object[] { new Double(5), new Double(6) };
Double sum = (Double)s.execute("add", arguments);
System.out.println("Result of add: " + sum);
```

Service Options

SpeedLink support many of the existing Service Options, and has additional options for tuning specific to SpeedLink. For a complete list of all available SpeedLink options and any differences for GridServer options with SpeedLink see SpeedLink Service Options. For an explanation of how to set Service Options to tune performance, see the *TIBCO GridServer® Administration*.

Service Options can be set in two ways: in the GridServer Administration Tool in the Service Type Registry page, or when creating 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 the client does not set it, the default value is used. When an option is not available for SpeedLink, a warning message is logged.

GridServer and SpeedLink Service Differences

The following sections detail the differences between standard GridServer Services and SpeedLink Services.

Improvements

The following items are improved in SpeedLink Services versus GridServer Services:

- Typical invocation latency on a generic 100BaseTX network is around one millisecond for SpeedLink and around 25 milliseconds for standard Services.
- Typical maximum client throughput rates are 1,000 invocations per second for SpeedLink and 100 invocations per second for standard Services.
- SpeedLink throughput is limited by client performance, not Broker performance, maximum Broker-wide throughput is limited to about 1,000 invocations per second for standard Services, but could easily scale to 100,000 invocations per second for SpeedLink Services.

Limitations

The following features are limited in SpeedLink Services:

- SpeedLink's higher throughput means that task-level reporting is limited to summary information in the reporting database. Fine-grained operation-level reporting can be performed via the SpeedLink API on the client side.
- SpeedLink's horizontal scalability—the number of Engines per Service— might be limited to hundreds of Engines per client, whereas standard Services are can scale over 5,000 Engines per Service.
- SpeedLink limits the data size returned from an Engine to 1 MB, including the serialization overhead. Returning larger amounts of data significantly reduces SpeedLink's performance.

Not Supported

The following features are not currently supported in SpeedLink:

- Automatic rescheduling of failed Service operations—any failed operation is immediately reported to the application as an invocation failure exception. This is primarily because there is no optimization advantage to having the SpeedLink code retry versus customer code.
- Redundant scheduling, since it has no benefit in low latency applications.
- Data encryption, data compression, or SSL.
- State update.
- Invocation cancellation.
- Task resubmission.
- Operation-level discrimination.
- Deferred collection such as `COLLECT_LATER` and `COLLECT_NEVER`.
- Proxies are not supported; client code must use the `Service.submit` or `execute` methods to invoke Service code.
- SpeedLink does not support the XML Serialization mode, which is typically only used when you require cross-language use of objects, or when a client does not have access to the Service implementation classes.

Administration for SpeedLink

Administering a GridServer grid using SpeedLink Services is essentially identical to managing a grid running standard Services. Knowing the differences can be helpful when running or your grid. This section discusses issues unique to SpeedLink. Refer to the GridServer documentation, and specifically the *TIBCO GridServer® Administration* for more details.

Monitoring and Reporting

SpeedLink Services, like standard Services, are viewable on the GridServer dashboard and Broker console. On the **Services > Services > Service Session Admin** page, there is a Service Type Name column; this is the **SpeedLink** for these Services.

The biggest difference between standard Services and SpeedLink Services is in the granularity of administration and reporting. Since throughput levels can be much higher on SpeedLink Services (it's designed to a nominal 10,000 operations per second per Broker target), it is counter-productive to attempt to report every operation detail to the administration console or reporting database. Instead, Engine reservations are presented as tasks, with operation summary reports presented in the task information column.

Enabling and Configuring TaskInfo Reporting

By default, Task Event reporting is not enabled. To enable reporting, set the value of the SpeedLink Service Option `SUMMARY_INFO_ENABLED` to `true`. Once Task Event reporting is enabled, summary operation information is stored in the TaskInfo field of the reporting database after the Service completes. The following summary information is recorded:

- Number of completed operations
- Number of failed operations
- Service allocation time for task—specified in milliseconds

The format of the database entry is determined by the value of the following SpeedLink Service Option: `SUMMARY_INFO_FORMAT`. The default value is: `operations={0}, failures=`

`{1}, allocationTime={2}`. You can change the value of this option; for more information about the available message format options consult the Javadoc for `java.text.MessageFormat`.

Set the properties described here in the `createService` method of your `SpeedLinkServiceFactory`. You can set reporting and the reporting format independently for each SpeedLink Service.

Performance Tuning

There are three main components to the SpeedLink client implementation that can be configured with Service Options depending on your application, hardware, and network: the Operation Queue, the Allocation Monitor, and the Operation Dispatcher.

Operation Queue

The operation queue contains the low-latency operations that the client needs to have executed on the Grid. When an operation is submitted, it is placed on a queue from which another thread sends the operation to any idle Engine.

The operation queue size is controlled by the `MAX_QUEUED_OPERATIONS` option. If the queue of pending operations not yet submitted to Engines reaches this size, any further submissions cause the client to block until room is available in the queue.

Allocation Monitor

The allocation monitor watches the queue and the number of allocated Engines, and requests Engines to be reserved based on four options.

The minimum and maximum amount of Engines that can be allowed to work concurrently on a Session are specified upon session creation, as `MAX_ALLOCATED_ENGINES`, `MIN_ALLOCATED_ENGINES`. This essentially specifies the number of active reservation tasks at any given time. There are no specific guarantees that Engines are available to take those tasks, however, a session can be given `URGENT` priority to provide for some level of guarantee of availability. The minimum specifies how many idle tasks are maintained if the operation queue is empty; the maximum caps the amount of tasks regardless of the queue.

The size of the operation queue is defined by `ALLOCATION_OPERATION_QUEUE_THRESHOLD`. Once the queue rises above the queue threshold the allocation monitor starts requesting additional Engines at the rate specified in the `ALLOCATION_RATE` option, and continues requesting at that rate until the queue size drops below the threshold, or until the maximum reservation count is reached. The queue monitor might also automatically extend a new reservation when an Engine completes its reservation after reaching one of its timeouts.

If SpeedLink tasks were allowed to remain active indefinitely, then this session could starve the grid of Engines, as this essentially becomes a session with long-running tasks. To prevent this, the `ALLOCATION_TIME` option specifies the maximum amount of time a task can be performing operations. If an operation is in progress when it times out, the operation is allowed to complete first. Also, the SpeedLink implementation adds “noise” to the timeout value so task completion events become randomly staggered. Additionally, the `IDLE_LINGER` time specifies how long a task can remain idle without performing operations before it automatically deallocates itself.

Operation Dispatcher

The operation dispatcher handles sending operations to Engines and correlating results with the original requests. Because all Engines have already been reserved for the Service, prioritization is not necessary; operations are simply assigned to any available Engine. Operation-level conditions are not supported. It tracks the `OPERATION_TIMEOUT` so that an operation that hangs can be terminated.

Failure Modes in SpeedLink

As with GridServer Services, there are several possible points of failure with a SpeedLink Service. This section describes the various issues that can cause a Service to fail, and how they are handled.

Network Failure

From the client point of view, there are two types of failure related to the network and connections between components:

- Socket close, connection failure, and other interruption errors.

- Socket blocks on a read until timeout as defined by the `OPERATION_TIMEOUT` option.

In either case, on failure the client automatically retries an operation up to a total of three times.

Broker Failure

If a client is disconnected from the Broker and the connection times out, the client resubmits outstanding reservation tasks to the new Broker as soon as it can log in.

It is possible that this is due to the Client-Broker connection broken, but the Broker is not down and Engines are still connected and running tasks.

If the Driver timeout is set the same on the Broker and the Driver, the Tasks on the original Broker are canceled, so the existing connections are closed and the Service fully recovers.

Engine Failure

Engine failures generally appear as a network failure from the client when it can't maintain a connection to the Engine. When the Service fails, it is retried on another Engine as if it had timed out.

Operation Failure

If an operation fails, the failure is propagated back to the handler, as it is in standard Services. However, SpeedLink does not provide an auto-retry, like GridServer. (Because there is no Broker middleman, an automatic retry is identical to the client code resubmitting the operation, so it provides no added benefit.)

When an operation fails, there might be some cases in which you might want the reservation task to continue to process operations; in other cases you might also want the task to also fail, such as when using Blacklisting to prevent that Engine from processing any subsequent operations. The Boolean option `TASK_FAIL_ON_OPERATION_FAILURE` sets this option.

SpeedLink Service Options

The tables in this section describe SpeedLink Service options and standard Service options and their use in a SpeedLink Service.

SpeedLink Service Options

The following table lists all SpeedLink Service Options. These options are also further described in the API documentation included in the SpeedLink installation.

Service Option	Description
`SUMMARY_INFO_ENABLED`	When `True`, summary Task Event reporting is enabled for the Service. Default is `False`.
`SUMMARY_INFO_FORMAT`	The default value is: `operations={0}, failures={1}, allocationTime={2}`. For more information about the available message format options consult the Javadoc for `java.text.MessageFormat`.
`ALLOCATION_OPERATION_QUEUE_THRESHOLD`	The operation queue threshold at which more Engines are allocated to this session.
`ALLOCATION_RATE`	The rate in Engines/second, at which Engines are allocated to this session when below the `MAX_ALLOCATED_ENGINES` and above the `ALLOCATION_OPERATION_QUEUE_THRESHOLD`.
`ALLOCATION_TIME`	The amount of time, in milliseconds, that an Engine can remain allocated.
`IDLE_LINGER`	The amount of time, in milliseconds, an allocated Engine remains idle before it

Service Option	Description
	deallocates itself.
`MAX_ALLOCATED_ENGINES`	The maximum number of Engines that are allocated to this session at any time.
`MAX_QUEUED_OPERATIONS`	The maximum total number of operations from this Service Session that can be queued on the client.
`MIN_ALLOCATED_ENGINES`	The minimum number of Engines that are allocated to this session at any time.
`OPERATION_TIMEOUT`	The maximum time an operation can take, in milliseconds.
`TASK_FAIL_ON_OPERATION_FAILURE`	Determines whether an operation failure is considered to be a task failure by the Engine and Scheduler.
`NATIVE_NET_SERVICE`	Specifies whether to run the SpeedLink Service under .NET. Default is `true`. When set to `false` the SpeedLink Service runs under Java.

Standard Service Options in SpeedLink

The following table lists all current standard Service Options, and how they apply to a SpeedLink Service.

Service Option	Applies to SpeedLink
`PRIORITY`	Yes
`GRID_LIBRARY`	Yes

Service Option	Applies to SpeedLink
`KILL_ENGINE_ON_CANCEL`	Yes
`SERVICE_FAIL_RESTART`	Yes
`SERVICE_FAIL_RETRY`	No, retries must be performed by the client.
`ALLOWED_DRIVER_PROFILES`	Yes
`EMAIL`	Yes
`STATE_AFFINITY`	Yes
`AFFINITY_DEPTH`	No, there is no task discrimination.
`AFFINITY_WAIT`	Yes
`INVOCATION_MAX_TIME`	No, automatically calculated as `ALLOCATION_TIME` plus `OPERATION_TIMEOUT`. Prevents a runaway operation from taking an engine out.
`AUTO_CANCEL`	No
`MAX_TASK_RETRIES`	No, retries must be performed by the client.
`MAX_TASK_RESCHEDULES`	No, no redundant rescheduling.
`RESCHEDULE_ON_TIMEOUT`	No, no redundant rescheduling or retries.
`REDUNDANT_RESCHEDULING_ENABLED`	No, no redundant rescheduling.
`RESUBMIT_ON_DDT_FAILURE`	No, no DDT.
`COMPRESS_DATA`	No, the low-latency use-case does not include large data.
`ENCRYPTION_ENABLED`	No

Service Option	Applies to SpeedLink
SHARED_UNIX_DIR	No, no DDT.
SHARED_WIN_DIR	No, no DDT.
MAX_ENGINES	Yes, can be used to limit MAX_ALLOCATED_ENGINES. Overrides that if set.
STATUS_EXPIRES	Yes
ENGINE_BLACKLISTING	Yes
FAILURES_BEFORE_BLACKLIST	Yes
CHECKPOINT	No, no reason to checkpoint very short operations.
PURGE_INVOCATION_DATA	Yes
AUTO_PACK	No
INVOCATIONS_PER_MESSAGE	No
JAR_FILE	No
SHARED_SERVICE	No
COLLECTION_TYPE	No

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

This software may be available on multiple operating systems. However, not all operating system platforms for a specific software version are released at the same time. See the readme file for the availability of this software version on a specific operating system platform.