TIBCO Spotfire® Statistics Services Users Guide

Software Release 12.0 LTS (12.0.0)

Contents

TIBCO Spotfire Statistics Services. 4

Developer roles. 4

Access Spotfire Statistics Services. 5

Server domain packages and namespaces. 6

Supported programming languages. 6

WebDAV file management. 6

Deploy a package to Spotfire Statistics Services. 7

A custom R function. 7

R code for the server. 9

Managing files. 9

Access databases using R code. 9

Code for the Expression client interface. 10

Deploy R code for the Expression client interface. 10

R language differences and limitations. 10

Deploying R code for the Function client interface. 10

Function and package resources. 12

Manage packages between Spotfire and Spotfire Statistics Services. 12

Code for the Function client interface. 15

The AsterDB package. 15

SAS and MATLAB code for the server. 16

Spotfire data inputs and outputs for SAS. 16

Non-Spotfire data in SAS. 17

Known issues with SAS. 17

Spotfire data inputs and outputs for MATLAB. 17

Non-Spotfire data in MATLAB. 18

Known issues with MATLAB. 19

Java, C#, and URL APIs. 20

Write Java and C# code for Spotfire Statistics Services. 20

Function evaluations. 20

Expression evaluations. 21

Pass in a data object. 21

Authentication on the server. 23

Use notifications. 24

Write requests using the URL API. 26

Server information. 27
URL authentication. 27
Administration tasks. 28
Retrieve the job list. 28
Expression and function requests. 29

Interpret and use returned objects. 31
The SplusDataResult object. 31
The data results XML. 31
The SplusObject. 34
Return results as JSON. 35

Transient and persistent data. 37
Access transient and persistent data on the server using WebDAV. 37
Transient data directories. 37
Persistent data directories. 37
Spotfire Statistics Services WebDAV API. 38
The spserver.* functions. 39

Troubleshoot SSL certificates. 41
Bypassing the certificate validation. 41
Installing a certificate. 41
Creating the new truststore. 41
Certificate parameters for testing. 42

Addressing engine issues. 44

Documentation and support services. 45

Legal and Third-Party Notices. 46

Index. 47

TIBCO Spotfire Statistics Services

TIBCO Spotfire® Statistics Services is a light-weight, flexible server that provides a communication layer, a service layer, and a TIBCO® Enterprise Runtime for R (TERR™) engine pool or an open-source R engine pool, among other features.

Open-source R is available under separate open source software license terms and is not part of TIBCO Spotfire Statistics Services. As such, open-source R is not within the scope of your license for TIBCO Spotfire Statistics Services. Open-source R is not supported, maintained, or warranted in any way by TIBCO Software Inc. Download and use of open-source R is solely at your own discretion and subject to the free open source license terms applicable to open-source R.

You can also use an external SAS® or MATLAB® engine if you have access to the corresponding software. See SAS and MATLAB Code for the Server for more information. However, the main part of this documentation is focused on the R engines. Spotfire Statistics Services does not include user interface features (such as an IDE).

Spotfire Statistics Services includes:

- Support and help for functions.
- Several language APIs for creating tools to communicate with the server from a desktop client or browser.

The server is designed to work in a cluster environment. It provides load balancing and the capacity for multiple R engines to handle requests from clients.

Client applications that implement the application programming interfaces (APIs) gain access to the R engine through the web server. R developers write functions and expressions that the users call via the APIs.

Any R code that is run by client applications is run by R engines on the server using R libraries that have been published on the server. R developers who are writing functions or expressions for client applications can use their development environments to write and test their code, and then deploy it to the server. These requests produce jobs, or the output of requests to R engines on the server. See Access Spotfire Statistics Services for more information.

Developer roles

Spotfire Statistics Services is designed for two types of developers: The R developer working in TERR or open-source R, and the client application developer, working in C# or Java.

- The R developer: This developer writes R functions, deploys the functions' packages to the server, and then calls them on the server via the Function client using the APIs. Alternatively, the developer writes expressions and sends them to the server using the Expression client APIs. See R code for the server for more information.
- The Client Application developer (or systems integrator): This developer uses the APIs and documentation provided with Spotfire Statistics Services to design and develop customized desktop or web applications. These applications send R evaluation requests to the server, and then examine and store the results. See Java C# and URL APIs and Access Spotfire Statistics Services for more information about these two developer roles.

[1] SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

MATLAB is a trademark or registered trademark of The MathWorks, Inc.

Access Spotfire Statistics Services

Enterprises can access Spotfire Statistics Services with Spotfire or by using the Spotfire Statistics Services client APIs.

Access	Description
Tools available in Spotfire Analyst	Using Spotfire Analyst, R developers can employ predictive models, or they can register TERR functions in the Spotfire Library, making them available for reuse in appropriately-configured Spotfire applications. • During application prototyping and configuration, R developers can use theTERR engine that is installed with Spotfire Analyst. • For production deployment to other business users, R developers can create packages to hold functions, and then deploy them to Spotfire Statistics Services so that Spotfire can call Spotfire Statistics Services remotely. See the Help provided with each of these features for more information.
client APIs	The server includes the following APIs in three programming languages (C# API, or Java API, and a URL-based API: • An Administration client interface. Use to examine the server properties (such as version, name, and time), perform administrative tasks on the server (such as checking to see if authentication is enabled), and manage server jobs (such as examining, interrupting, or deleting jobs). • A Function client interface. Use to call R functions on the server either synchronously or asynchronously, optionally starting at a specified time. • You can pass R objects or data as an argument in a request. For more information about using this feature, see Pass in a data object. • An Expression client interface. Use to send R expressions to the server either synchronously, asynchronously, or at a specified time. • A Utilities interface. Use to help create R expressions and adapt the results to the appropriate view model. • A WebDAV client interface. Use to examine or manipulate files, folders, properties, or methods stored in the WebDAV repository. The APIs include objects representing returned R objects and server exceptions in the Domain package (Java) or namespace (C#). Objects are represented in SPXML, described in The SplusObject. • C# and Java programmers can use those APIs to design client applications or web pages that users can use to call R functions or send expressions to the server, and then examine and store the results. If you are using the Spotfire Statistics Services Java or C# API to send data through Spotfire Statistics Services, you must limit the data to a size that can be sent using SPXML. If the data exceeds the allowed size of 512,000 bytes, the API throws an exception. • Server administrators and programmers can use the URL API to send simple requests to the server to check on server status.

Server domain packages and namespaces

In addition to the objects, Spotfire Statistics Services includes domain packages and namespaces. For more information about the package and namespace classes, see the Spotfire Statistics Services API reference.

Domain Package/ Namespace	Description
Binary	Contains classes that determine how binary data is managed in Spotfire Statistics Services. These classes include domain objects to act as containers for multi-part binary request and result objects, and functions to manage binary serialization.
Cluster	Contains classes for managing jobs and results in a Spotfire Statistics Services cluster environment.
Configuration	Contains classes and enums to discover server and engine configuration details, such as server type and engine type.
Converter	Contains classes that help properly convert Spotfire data types to the corresponding nullable C# data types so that “NA” values in Spotfire are represented as null in C#.
Exceptions	Contains top-level API classes for Spotfire Statistics Services exceptions.
Job	Contains classes defining how to run jobs on Spotfire Statistics Services immediately, scheduled, or asynchronously.
Packages	Contains classes for the Spotfire Statistics Services for adding, deploying, updating, and removing packages.
Representation	Contains classes that provide generic object representation of a value returned as a result of job execution on the server.

Supported programming languages

If you use multiple languages, this language list might help you decide which programming language or technology to select for developing the type of application you need.

Language/ Technology	Desktop Application	Web Application
C#, .NET	Yes	ASP.NET (web container, for example, IIS)
Java	Yes	JSP, JSF (web container, for example, Tomcat)
URL	No	See Write requests using the URL API

WebDAV file management

Spotfire Statistics Services uses the WebDAV protocol to manage transient and persistent data, results directories, and files on the server.

Spotfire Statistics Services includes WebDAV interfaces in the C# and Java APIs.

For more information, see Transient and persistent data.

Deploy a package to Spotfire Statistics Services

The developer can upload, update, or remove packages on the server using the C# or Java APIs.

1. A TERR or open-source R developer writes custom functions.
2. The developer combines the functions in a package.

   See Deploying R code for the Function client interface for a list of resources providing more information about building and deploying open-source R language packages to Spotfire Statistics Services.

3. The developer or the server administrator deploys the package to the server via a tool or an API for system-wide deployment.

For more information about managing packages, see Deploying R code for the Function client interface.

For more information about developing packages for open-source R, see the R documentation. Spotfire Statistics Services does not ship open-source R code or open-source R documentation.

Open-source R is available under separate open source software license terms and is not part of TIBCO Spotfire Statistics Services. As such, open-source R is not within the scope of your license for TIBCO Spotfire Statistics Services. Open-source R is not supported, maintained, or warranted in any way by TIBCO Software Inc. Download and use of open-source R is solely at your own discretion and subject to the free open source license terms applicable to open-source R.

A custom R function

The Function Client API contains an evaluation method, which sends the function to be evaluated to the server.

For example, one version of the C# `Eval()` method takes the following arguments:

- The name of the function to run.
- The package containing the function. (The package must be deployed to the server's package repository and loaded in the open-source R or the TERR engine for the function to be found.)
- The function arguments, represented by the `SplusDataRequest` object that contains a collection of objects and/or an encapsulated data set.
- A `JobStartup` argument specifying whether to run the function synchronously or asynchronously, and (optionally) the time to run it.

Java provides a similar evaluation implementation.

See Java C# and URL APIs and Function evaluations.

Finding help

Help for using the APIs is available via your web browser on Spotfire Statistics Services.

Prerequisites

You must know the service name and port number. Spotfire Statistics Services service must be running to access the service landing page.

Find help from a web browser.

Procedure

- In the browser address box, type `http://`*`servername`*`:`*`port`*`/`*`service_name`*

  The *`servername`*, the *`port`* number, and the *`service_name`* are assigned by the administrator.

  If you are developing on Microsoft Windows®, you might not be able to open a CHM file containing Spotfire Statistics Services help or C# API help from the server. (This behavior is by design, due to a security enhancement added to Windows after XP version SP2. See the Microsoft web site for more information and potential workarounds.) We recommend copying the CHM file from your server installation to your Windows development machine, and then unblocking the file in the properties dialog as the safest workaround.

  Developers can find more information about writing functions for the Function Client Interface or writing expressions for the Expression Client Interface in R code for the server.

  The landing is displayed from this address, and the landing page links to the API help sets, as well as the programming interfaces, for each language.

R code for the server

Whether you are writing expressions or functions, your R language scripts (either TERR or open-source R) for the server are similar to those you write for the desktop. However, you might need to account for some differences.

For example, you should plan to write code that can be run on any platform.

Open-source R is available under separate open source software license terms and is not part of TIBCO Spotfire Statistics Services. As such, open-source R is not within the scope of your license for TIBCO Spotfire Statistics Services. Open-source R is not supported, maintained, or warranted in any way by TIBCO Software Inc. Download and use of open-source R is solely at your own discretion and subject to the free open source license terms applicable to open-source R.

Development and production server platforms can differ. Therefore, when you write your code to run on the server, make sure that it can be run on any platform.

Important By default, certain Spotfire Statistics Services capabilities are disabled. If you have functions that you created or used in a previous release, you might find that they no longer work as expected. Additionally, any expression that TERR determines to be potentially malicious is disabled. Below is a non-exhaustive example of some of the capabilities disabled by default in Spotfire Statistics Services.

- Sending TERR expressions that perform I/O to the file system or the internet.
- Spawning new OS processes by calling the `system` function.
- Calling into Java using the terrJava package, or using functions in the parallel package.
- Loading new packages, except for those included with TERR.
- Calling `.C` or `.Fortran`.
- Send expressions to the server using the Expression Service.
- Calling `ExtendedServerInfo` or sending other expressions that read from, or write to, your server.
- Sending potentially malicious expressions to the server using the URL API.

Additionally, the Function Service can execute only one function, which allows a connection from Spotfire. If you have functions that you want to have enabled on Spotfire Statistics Services, or if you have additional questions about the configuration settings that control these capabilities, see your server administrator.

Managing files

When you construct an R function to run on the server, take care to manage the file output appropriately.

Procedure

- Specify relative paths rather than absolute paths when you access inputs and save results. This practice allows the server to manage the results files for you.

Access databases using R code

You can access data stored in databases from your script.

Writing import and export code for accessing data on a database should be identical for both the server environment and the desktop environment.

Direct database drivers and JDBC are cross-platform; ODBC is Windows-only.

For information on using JDBC, see the sjdbc library documentation for TERR.

Code for the Expression client interface

The Expression client interface is designed to be used in client applications. As long as the expression is valid, the Expression client interface passes it to the engine.

Using the Expression client interface does not necessarily mean that you have to deploy custom functions in packages on the server and ensure that the functions are loaded in the engine before calling them.

By default, the Expression client is disabled in Spotfire Statistics Services because it can send any valid expression to the server. For more information, talk to your server administrator..

You or your designated C# or Java developer can use the APIs for one of these available programming languages to send expressions to the server.

For more information on using these specific APIs, see Java C# and URL APIs.

Deploy R code for the Expression client interface

When you use the Expression Client interface, you can put your custom functions in a package, and then load the package explicitly as part of the expression before calling the function.

For example, if an application uses a custom function `doGraph35()`, you could put the code for this function into a package (MyGraphPkg) and call it with an expression, such as follows.

```
{library(“MyGraphPkg”); doGraph35(arg1=...)}
```

The above code assumes that the package exists in the package repository on the server. Note that the TERR engine does not support graph functions.

This approach does not work with the Function Client interface, because the Function Client service can execute only one function contained within a package that is automatically attached to the R engine.

R language differences and limitations

Differences exist between TERR and open-source R. However, we strive for near-total compatibility. The help for TERR includes a list of known differences between it and open-source R. To see this document, from the TERR console, type `help.start()` and follow the link to the document Differences Between TIBCO Enterprise Runtime for R and Open-Source R.

Version 6.0 of TERR does not have graphic support; however, you can use packages from CRAN such as htmlwidgets and leaflet to produce graphs in TERR. If you are writing functions or expressions to be run using the TERR engine in Spotfire Statistics Services from TIBCO Spotfire® (either through the Predictive Modeling or Data Functions features), TIBCO Spotfire® provides data visualizations.

From Spotfire 7.0 or later, you can access `TERR.exe`, which launches the TERR console from the Spotfire **Tools** > **Terr Tools** menu.

Deploying R code for the Function client interface

Most client applications that use the Spotfire Statistics Services Function Client Interface call custom functions that were written for the client application. Typically, they are not part of the packages included with TERR. The Spotfire Statistics Services Function Client Interface calls one function, and then it releases the engine. To use the Function Client Interface, follow these steps.

Important By default, certain Spotfire Statistics Services capabilities are disabled. If you have functions that you created or used in a previous release, you might find that they no longer work as expected. Additionally, any expression that TERR determines to be potentially malicious is disabled. Below is a non-exhaustive example of some of the capabilities disabled by default in Spotfire Statistics Services.

- Sending TERR expressions that perform I/O to the file system or the internet.
- Spawning new OS processes by calling the `system` function.
- Calling into Java using the terrJava package, or using functions in the parallel package.
- Loading new packages, except for those included with TERR.
- Calling `.C` or `.Fortran`.
- Send expressions to the server using the Expression Service.
- Calling `ExtendedServerInfo` or sending other expressions that read from, or write to, your server.
- Sending potentially malicious expressions to the server using the URL API.

Additionally, the Function Service can execute only one function, which allows a connection from Spotfire. If you have functions that you want to have enabled on Spotfire Statistics Services, or if you have additional questions about the configuration settings that control these capabilities, see your server administrator.

Procedure

1. Develop a set of custom functions containing your complete request.
2. Create a binary package to hold your custom functions.
3. Upload the package to Spotfire Statistics Services.

   Packages you create to deploy on the server must be compiled as binaries and uploaded as zipped files (with a .zip or tar.gz extension).
4. Make sure the client application function begins with a call to load the package (`library(`*`packagename`*`)`).

   Remember that each time you call a function, a new R engine is started, and when the function is completed, the engine is released. This means that the package must be loaded with each function call.

   Optionally, you can ask your system administrator to add your package to the engine init script, so the package is loaded automatically whenever the engine starts. In a clustered environment, when a package is deployed to a Manager node, the package is automatically made available to all of the cluster's Worker nodes.
5. Provide the client developer with the function names, the package name, and all the function arguments.

   The client developer calls each function using the Function Client Interface API `eval()`.
6. After your package is deployed, if you find you need to change it, you can make any necessary changes, and then redeploy it easily.

Function and package resources

The TIBCO Spotfire® product documentation includes comprehensive guidance on developing, packaging, uploading, and maintaining functions. Additionally, resources for R packages are readily available.

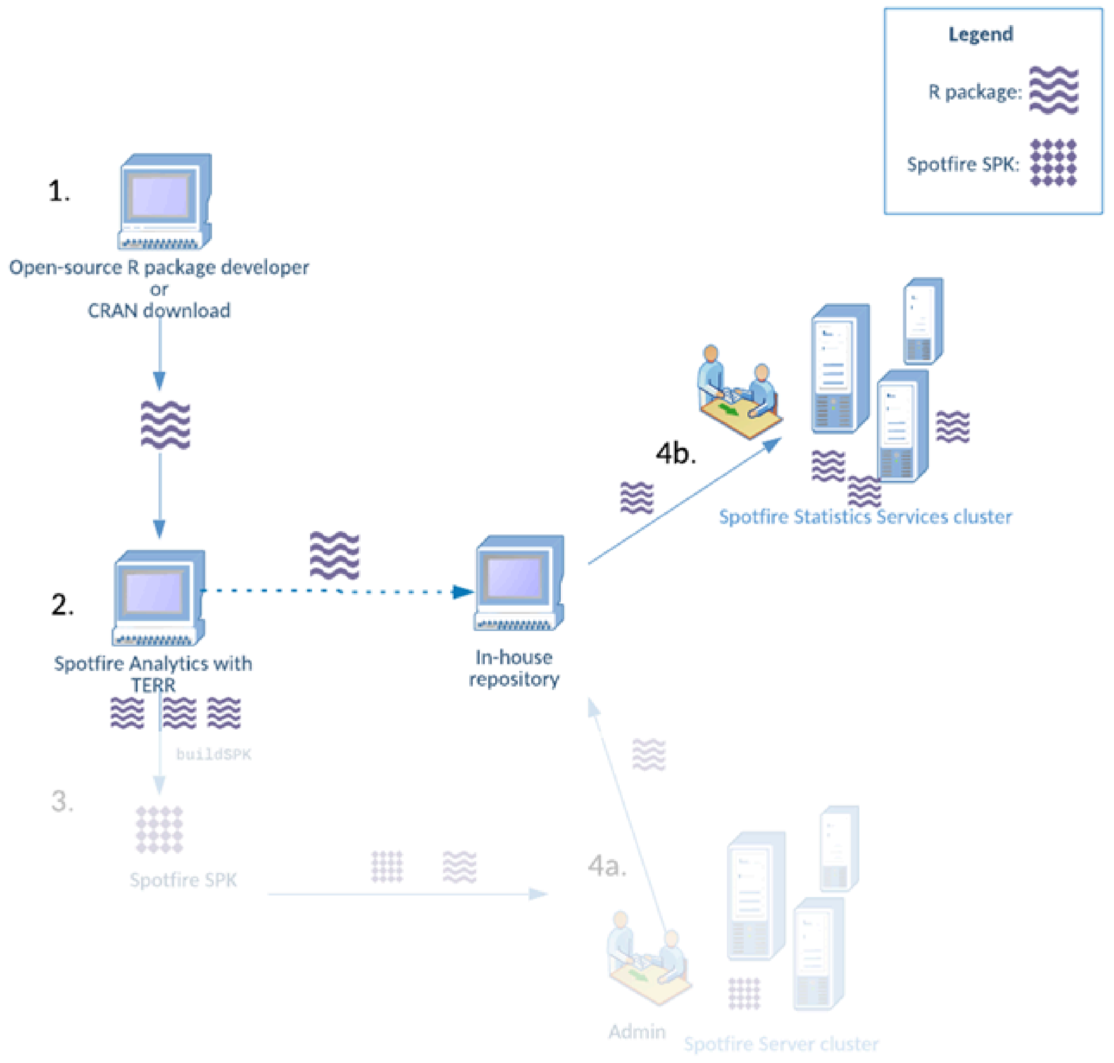
Document Name	Description
TIBCO Spotfire® Package Management	Contains instructions for creating a Spotfire package (SPK) containing R pack Analyst and Spotfire Web Player users. Also contains instructions for upload for R package to Spotfire Statistics Services for use with the Spotfire Web Play
TIBCO® Enterprise Runtime for R (TERR™) Language Reference	Contains help for functions in the packages available to the TERR engine. Yo opening the TERRconsole, and at the prompt, type `?functionname` (for func start() to display the TERR help in a browser window. From Spotfire 7.0 or later, you can access `TERR.exe`, which launc **Terr Tools** menu.
Writing R Extensions	Contains instructions for creating packages in open-source R.

Open-source R is available under separate open source software license terms and is not part of TIBCO Spotfire Statistics Services. As such, open-source R is not within the scope of your license for TIBCO Spotfire Statistics Services. Open-source R is not supported, maintained, or warranted in any way by TIBCO Software Inc. Download and use of open-source R is solely at your own discretion and subject to the free open source license terms applicable to open-source R.

Manage packages between Spotfire and Spotfire Statistics Services

You can share Spotfire visualizations that use R language packages. To share such visualizations widely in a web browser, your server configuration must include the Spotfire web client and Spotfire Statistics Services deployed and configured to work with Spotfire Server.

Spotfire Statistics Services includes a repository for the packages containing functions that can be used by Spotfire analyses. These packages must be identical to those packages distributed to the installations of Spotfire Analyst.



You can add packages to your Spotfire Statistics Services installation. For more information, see "Install packages on Spotfire Statistics Services" in the *Package Management for the TIBCO Spotfire® Environment*.

Validating the package upload

After you upload a package, run a quick validation to ensure that your package is on the server.

Prerequisites

You must have administrative privileges to perform this task.

Perform this task from the TERR console in your installation of Spotfire Statistics Services

Procedure

1. From the browser, type the following command.
   ```
   installed.packages()
   ```
   A list of installed packages is printed in the console.

2. Review the list (including the path) to ensure that the package you uploaded is installed.

   ImportantAlthough the package is on Spotfire Statistics Services, it is not loaded into the engine. With each call to Spotfire Statistics Services, the engine is started anew.

   To use the package, you can either load the library as part of your function scripts, or you can include it in the `Packages` field of your data function.

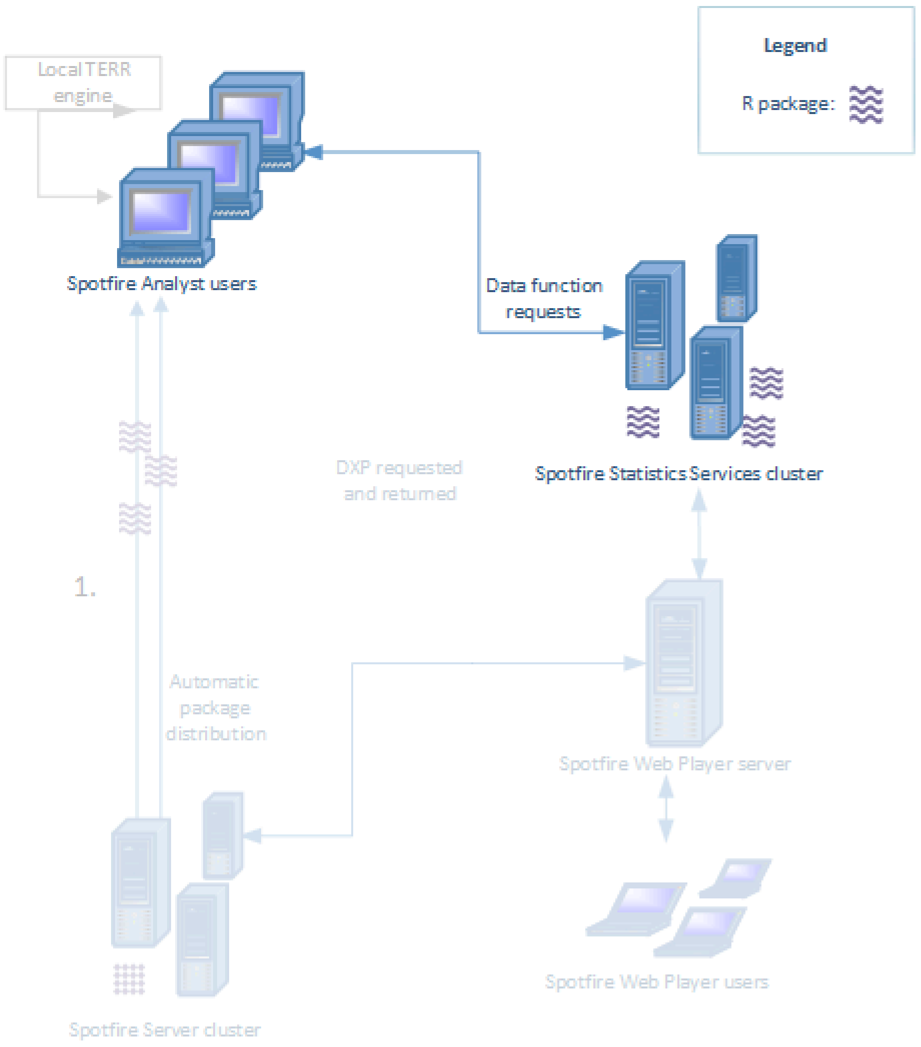
Changing the local engine option

You can configure Spotfire Analyst to use the TERR engine that is installed in your organization's Spotfire Statistics Services deployment.

Prerequisites

Remember that you must have the same package version on your local installation and on the server. See your organization's package curator for help.

You can use Spotfire Statistics Services in deployments where web client users access Spotfire analyses. Changing from the local TERR engine to the one on Spotfire Statistics Services is useful for testing the analyses the Spotfire web client users access.


Legend
R package:
Local TERR engine
Spotfire Analyst users
Data function requests
Spotfire Statistics Services cluster
DXP requested and returned
1.
Automatic package distribution
Spotfire Web Player server
Spotfire Web Player users
Spotfire Server cluster

Procedure

1. In Spotfire Analyst, click **Tools** > **Options**.
2. In the left pane, scroll down and select **Data Functions**.
3. In the **Data Functions** pane, select **Custom URL**, and then provide the URL to the Spotfire Statistics Services (for example, `http://CoTSSS:8080/TERRServer`).
4. Clear the checkbox **Use locally installed TIBCO Enterprise Runtime for R**, and then click **OK** to accept.

The following advanced analytic tools available in the Spotfire Analyst installation always use the local engine, regardless of this setting.

- Classification modeling.
- Regression modeling (linear regression or regression tree).

Code for the Function client interface

The Function client interface provides more structure and protection for your server than does the Expression client interface.

Because the Expression client interface processes any legitimate expression sent to the R engine, using it can pose some risks.

A function's structure makes managing it as a unit much easier. The Function client interface imposes the following design considerations:

- The Function client interface calls only one function at a time. The next time you send a request to the Function client interface , it goes to the server to be delegated to the engines as an entirely new request. Therefore, it is likely that your function will not even use the same R engine for a second request. That is, you must write functions that contain the entire, discrete job request.
- The call you send to the Function client interface must be to a function that is currently loaded in the R engine. This section includes information about deploying functions in packages to theSpotfire Statistics Services package repository.

The following example demonstrates calling a simple core R function `rnorm()`.

```
rnorm(100)
```

In most cases, an analysis requires more than such a simple call to a function that ships with the engine. You would want to do something with the results. Taking these points into consideration, you probably have already discerned that you must perfom the following tasks.

1. Design a "wrapper" function (one that contains your complete analytic).
2. Deploy your custom function—in a package—to the server, ensuring that it is loaded in the engine before the client application calls it.

The AsterDB package

The AsterDB package, which is provided in Spotfire Statistics Services, is a specialized package designed for working with the Teradata® Aster Database.

TIBCO Spotfire® provides out-of-the-box TERR data functions that use the Aster Database and the AsterDB package. If you plan to create analyses that access an Aster Database, you must get the preconfigured package from your Spotfire Statistics Services administrator and install it on your machine. Additionally, you must have Java 7 or higher installed, with the JVM, and you must set *JAVA_HOME* to that installation location.

For more information about getting this package, see your server administrator.

SAS and MATLAB code for the server

You can configure the server to evaluate scripts written using the SAS®, or MATLAB® scripting languages instead of evaluating using TERR or R code.

SAS or MATLAB engines must be installed on the server.

Unlike the open-source R or TERR code engines, the server supports evaluating a script using only the Expression Client interface. It does not support calling a named function using the Function Client interface.

By default, the Expression Client interface is not enabled on the Spotfire Statistics Services server. See your server administrator for more information.

This section assumes that anyone using the SAS, or MATLAB engines already knows how to write scripts in those languages. The following subsections give details of how to access input and output data within the script when you are accessing the server from TIBCO Spotfire® or by using the other APIs.

Spotfire data inputs and outputs for SAS

The only part of the SAS software required by Spotfire Statistics Services is Base SAS. Other SAS software packages are not required, but they can be installed on the machine.

For general information regarding scripting using SAS scripts, refer to your SAS software documentation.

An input to a Spotfire data function can have a type of Table (a whole data set), Column (a single column), or Value (a single scalar). A Table or Column input are accessible in a SAS script as a SAS data set with the name given by the data function input variable.

After the script is executed, the data function outputs are retrieved from SAS data sets with the names of the output variables. If one of these data sets is not found (because the script did not create it), an error is generated. If Spotfire expects an output to contain a column or a scalar value, it generates an error if the actual data has multiple columns (for a column output) or multiple rows and columns (for a scalar value output).

The following is a SAS script that takes an input data table named myinput, adds two new columns (rnum and cnum), and produces an output data set named myoutput:

```
DATA myoutput;
SET myinput;
rnum = UNIFORM(-1);
cnum = 10*_N_;
RUN;
```

A Value data function input is interpreted differently than a Table or Column input. Rather than using the single scalar value to create a SAS data set, it is used to define a SAS macro variable whose name is the data function input variable, and whose value is the character string produced from the Spotfire scalar value. Using macro variables is a common way to parameterize SAS scripts, which matches the purpose of Spotfire Value inputs.

For example, if you define a Value input named `filename`, and its value is the string `D:/temp.txt`, then the following macro variable definition is added to the beginning of the SAS script:

```
%LET filename D:/temp.txt;
```

[2] SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

MATLAB is a trademark or registered trademark of The MathWorks, Inc.

This macro variable can be referenced within the SAS script using SAS code such as the following, which reads from the specified text file, producing the output data set filedata.

```
DATA filedata;
INFILE '&filename';
INPUT A $ B $ Num;
RUN;
```

Non-Spotfire data in SAS

When you use one of the non-Spotfire APIs to send a SAS script to the server, you can specify input data objects that are converted into SAS data sets and available as SAS input data sets with the specified names.

When a SAS script using one of the non-Spotfire APIs runs, the last data set created by the script is captured (by accessing the SAS special variable `_LAST_`), and returned as the result of the script execution.

Known issues with SAS

If you plan to use a SAS engine in Spotfire Statistics Services, you should known issues.

- Sending a zero-row, zero-column data set to the SAS engine generates an error. If the SAS script creates a zero-row, multiple-column output data set, it is read as a zero-row, zero-column data set.
- SAS engines cannot handle all Spotfire data types. Sending raw (binary) data to a SAS engine generates an error. Spotfire `SingleReal` and `Integer` values are sent to the SAS engine as doubles. The SAS engine reads the double infinity value as `NA`. An empty string value (`“”`) is returned to Spotfire as `NA`.
- Variable names and column names must respect the SAS scripting restrictions (that is, they cannot start with a digit, or have less than 32 characters).

Spotfire data inputs and outputs for MATLAB

For general information regarding scripting using MATLAB, refer to your MATLAB software documentation.

The common MATLAB data structures are somewhat different than the rectangular data sets Spotfire supports, so it is necessary to convert them when sending data inputs to MATLAB, or when reading outputs from a MATLAB script. In particular:

- The basic MATLAB engine does not include a table data type with named columns. A Spotfire data set is represented in MATLAB as a MATLAB struct with named elements, where each name is the name of a table column, and each element is a column vector containing the values in the column.

  A vector of arbitrary-length string values is represented as a MATLAB cell array (1 column, N rows) containing N string objects.

  It is not represented as a MATLAB char array, because this type of array requires all of the strings to have the same length.
- A list of raw (binary) vectors is represented as a MATLAB cell array (1 column, N rows) containing N UINT8 vectors.

An input to a Spotfire data function can have a type of Table (a whole data set), Column (a single column), or Value (a single scalar). A Table input is sent into MATLAB as a variable (with the same name as the data function input) containing a MATLAB struct with named elements. A Column input is sent into MATLAB as a variable containing a column array. A Value input is sent into MATLAB as a variable containing a scalar value (which is the same as an 1x1 array in MATLAB).

For example, suppose that Spotfire sends an input table variable named in with one numeric column and one string column:

```
Num   Str
1.2   aa
3.4   bbb
```

This would be available in MATLAB as the following structure:

```
>> in = struct('Num', [1.2;3.4], 'Str', {{'aa';'bbb'}});
>> in
in =
    Num: [2x1 double]
    Str: {2x1 cell}
>> in.Num
ans =
    1.2000
    3.4000
>> in.Str
ans =
    'aa'
    'bbb'
>>
```

All outputs from Spotfire data functions are retrieved from MATLAB variables with the names of the data function outputs. If one of these variables is not found (because the script did not set it), an error is generated. If Spotfire expects an output to contain a column or a scalar value, it generates an error if the actual data has multiple columns (for a column output) or multiple rows and columns (for a scalar value output).

If an output is a MATLAB struct, it is turned into a multi-column data set, with column names from the names of the struct elements. If one element is of a different size than the others, it will be replicated as necessary to produce a rectangular data set.

Output data types other than struct are converted into rectangular Spotfire data sets, with column names derived from the output variable name. For example, if an output variable *aa* is a single-column array, it generates a single column named “`aa`”. If an output variable *bb* is a multiple-column matrix, it generates columns named “`bb.1`”, “`bb.2`”, and so on. If the exact column names are important, it is best to set the output variable to a struct with specified column names.

The following MATLAB script takes an input data table named `myinput`, adds two new columns (`rnum` and `cnum`), and produces an output data set named `myoutput`. It starts with a few useful utility functions for handling input tables represented as structs.

```
% fns to get the number of rows and columns of a table
dfrows = @(s) max(structfun(@length, s));
dfcols = @(s) length(fieldnames(s));
% fn to convert a table to a matrix
% (error if all columns are not of same type)
df2mat = @(s) reshape(cell2mat(struct2cell(s)),dfrows(s),dfcols(s));
% copy input to output variable
myoutput=myinput;
% add rnum column, with random numbers
myoutput.rnum=randn(dfrows(myinput),1);
% add cnum column, with 10,20,...
myoutput.cnum=transpose(10*(1:dfrows(myinput)));
```

Non-Spotfire data in MATLAB

When you use one of the non-Spotfire APIs to send a MATLAB script to the server, you can specify input data objects which are converted into MATLAB objects and available as MATLAB input variables with the specified names.

When you run a MATLAB script using one of the non-Spotfire APIs, the last expression evaluated by the script is captured (by accessing the MATLAB special variable *ans*), and returned as the result of the script execution. This process can be counter-intuitive, because the *ans* variable is not set when a simple variable is evaluated. Thus, the following script would return 3, because the last expression is `1+2`:

```
x=10+5;1+2;x;
```

The following script returns the value of variable x, namely 15:

```
x=10+5;1+2;ans=x;
```

Known issues with MATLAB

If you plan to send MATLAB calls to Spotfire using Spotfire Statistics Services, you should review the known issues.

- MATLAB cannot handle all Spotfire data types. When date or time values are sent into MATLAB, they are converted to double values (whose integer part is the number of days since 1/1/1960, and whose fractional part is the fraction of the day). An error occurs when a MATLAB output variable contains strings with non-ASCII characters (with a byte value greater than 0x7F). An error occurs when a MATLAB output variable contains any of the types (U)INT16, (U)INT32, (U)INT64, or sparse arrays. The imaginary part of any complex output values are discarded.
- Variable names and column names must respect MATLAB's restrictions (only alphanumerics and underscore; it cannot start with a digit).

Java, C#, and URL APIs

Spotfire Statistics Services provides classes and interfaces for writing Java and C#-based web and desktop applications for sending requests to the server.

Important As of version 12.0.0 of Spotfire Statistics Services, the Java and C# APIs are deprecated. They will be removed in a future release.

The Java and C# APIs are implemented using a standard Factory design pattern for client instance creation. `ClientFactory` is the class that implements a factory and provides a way to obtain a client object of a given type by calling a relevant method. For example, to create an `AdministrationClient` object in C#:

```
string serviceUrl = "http://servername:port/service_name";
string username   = "myusername";
string password   = "mypassword";
IAdministrationClient api = ClientFactory.GetAdministrationClient(serviceUrl, username,
 password);
```

(where *`servername`* is the name of your Spotfire Statistics Services server, *`port`* is the port number for your Spotfire Statistics Services installation, and *`service_name`* is the Service Name provided to you by the administrator.)

The following sections provide information and examples for using the Java and C# APIs with Spotfire Statistics Services, highlighting features that are specific to the server, including:

- Passing open-source R or TERR objects in an evaluation request.
- Providing authentication support.
- Creating a notification listener to monitor job status.

Spotfire Statistics Services also includes a URL API, which is designed to accept simple requests to check server health. This API is discussed in further detail in Write requests using the URL API.

Write Java and C# code for Spotfire Statistics Services

This section includes an overview on using the features specific to Spotfire Statistics Services Java and C# APIs. A knowledge of either Java or C# is assumed.

For information about sending simple requests to the server via a web browser, see the Write requests using the URL API.

Important As of version 12.0.0 of Spotfire Statistics Services, the Java and C# APIs are deprecated. They will be removed in a future release.

We also discuss sending Function client and Expression client evaluation requests, implementing authentication, and creating a notification listener.

For more information about using the Administration, Expression, or Function Client APIs, see their respective overviews in the language help, available from the Spotfire Statistics Services landing page, at `http://`*`servername`*`:8080/`*`service_name`*`/`.

Function evaluations

The Function client contains three versions of the evaluation method (`eval` in Java; `Eval` in C#).

Important As of version 12.0.0 of Spotfire Statistics Services, the Java and C# APIs are deprecated. They will be removed in a future release.

This section addresses the Function client's evaluation methods in both languages. The following list shows the three evaluation method versions supplied in C#. (Java's evaluation methods are the same, except for different naming conventions inherent in the language):

```
SplusDataResult Eval(
string functionName,
string packageName,
Dictionary<string, string[]> args,
JobStartup jobStartup
SplusDataResult Eval(
 string functionName,
 string packageName,
 SplusDataRequest inputData,
 JobStartup jobStartup
)
SplusDataResult Eval(
 string functionName,
 string packageName,
 BinaryDataRequest inputData,
 JobStartup jobStartup
)
```

Each of these methods returns an `SplusDataResult` object, which is described in more detail in Interpret and use returned objects.

Expression evaluations

The Expression client also contains three current versions and four deprecated versions of the evaluation method (`eval` in Java; `Eval` in C#).

Important As of version 12.0.0 of Spotfire Statistics Services, the Java and C# APIs are deprecated. They will be removed in a future release.

This section addresses the Expression client's evaluation methods in both languages. The following list shows the three current evaluation method versions supplied in C#. (Java's evaluation methods are the same, except for different naming conventions inherent in the language.)

```
SplusDataResult Eval(
 string command,
 JobStartup jobStartup,
 SplusDataRequest inputData
)
SplusDataResult Eval(
 string command,
 JobStartup jobStartup,
 SplusDataRequest inputData
)
SplusDataResult Eval(
 string command,
 JobStartup jobStartup,
 BinaryDataRequest inputData
)
```

Each of these methods returns an `SplusDataResult` object, which is described in more detail in Interpret and use returned objects.

Pass in a data object

Notice the versions of the evaluation functions in both the Function and Expression clients that take as an argument an `SplusDataRequest` object. This object is constructed from the Domain: it is a collection of C# or Java representations of R language objects (either TERR or open-source R) and/or an encapsulated data set (along with the `ResultSerializationType` property).

You can use the following option.

- If your data is not located on the server, because you do not require the evaluation request to read the data from the server repository.

- If you need to perform further analysis on results received from a previous call by passing in the resulting object.
- If you receive results from several previous requests (that is, several objects) by passing in all of the objects in an `SplusDataRequest` argument.

You can use these versions of the evaluation methods with either the TERR engine or the open-source R engine.

Notice that the third versions of the evaluation methods in both the Function and Expression clients accept as the input data a `BinaryDataRequest` object, which contains the binary version of the `SplusDataRequest` object (that is, an array of binary "serialized" arguments) and the serialization type. The evaluation request is sent to the server using the appropriate serialization type, where it is loaded into the engine and executed. The results are returned as an `SplusDataResult` object. This object contains binary serialized results, as specified by the `ResultSerializationType` property in the request.)

The APIs do not include methods for serializing or deserializing binary objects. These tasks require using either the TERR engine or the open-source R engine.

For example, you can send the object containing binary arguments to the R engine, specifying `TerrSerialize`, `SBDF`, or `RSerialize` as the `SerializationType`. Using binary serialization requires fewer computer resources (memory and CPU), because it does not have to parse the SPXML.

If you are using the Spotfire Statistics Services Java or C# API to send data through Spotfire Statistics Services, you must limit the data to a size that can be sent using SPXML. If the data exceeds the allowed size of 512,000 bytes, the API throws an exception.

For more information about using the `SplusDataRequest` or the `BinaryDataRequest` object, see the help for the programming language of your choice, available from the landing page.

Creating a binary serialization request for R using Java and rJava

The example described in this topic uses example test code in the Java API and the rJava package to demonstrate how one might build a request using the binary API.

Procedure

1. Create an object and then serialize the object using R serialization.
2. Invoke the Java API `eval` method to create and send to the server a synchronous job request. (This `eval` method can accept binary arguments (that is, a `BinaryArgument` object).
3. The request is sent to the server, processed in the R engine there, and the results are returned (serialized and deserialized as needed).
4. The results are returned as a serialized byte array.

Example

```
public void PassBinaryArgument() throws Exception {
    //Serialize an object using TERR serialization.
    byte[] asByteArray = getSerializedBytes("1:10");
    // Invoke the overloaded eval method that accepts
    // and returns binary arguments.
    SplusDataResult result = mApi.eval("as.integer((x+1))",
        new SynchronousJobStartup(), new BinaryDataRequest(
        new BinaryArgument("x", asByteArray,
            SerializationType.RSerialize)));
 // NOTE: The following code uses the rJava library,
 // which includes rengine and its eval method (NOT to
 // be confused with the eval method found in the
 // Spotfire Statistics Services Java APIs.
protected byte[] getSerializedBytes(String expression) {
    rengine.eval("a<-serialize(" + expression + ",NULL)");
    rengine.eval(
        ".jcall('com/insightful/splusserver/r/binary/
        RBinaryArgumentExample'," +
        "'V','setA',a)");
    return serializedArray;
}
protected REXP unserializeInREngine(byte[] serializedData) {
    serializedArray = serializedData;
    rengine.eval("a<-.jcall('com/insightful/splusserver/
        r/binary/RBinaryArgumentExample'," +
        "'[B','getA')");
    return rengine.eval("unserialize(a)");
}
private static byte[] serializedArray;
public static void setA(byte[] a) {
        serializedArray = a;
}
public static byte[] getA() {
    return serializedArray;
}
```

The above shows just the relevant sections. Note that it uses the rJava package to transfer data between Java and R, so it is necessary to have knowledge of rJava to write such code.

Authentication on the server

When authentication is activated in the server, user credentials are authenticated against LDAP or a local database. (If you need configuration details, see your system administrator.)

You can use the Administration client utility method `isAuthenticationEnabled()` or `IsAuthenticationEnabled()` (in Java or C#, respectively) to determine the authentication state on the server. If this method returns `true`, authentication is enabled for your server.

Create authenticated objects

When you are developing an application for a server that has authentication enabled, use the constructor that specifies `username` and `password`.

For example, in C#:

```
IAdministrationClient api = ClientFactory.GetAdministrationClient(serviceUrl, username,
 password);
```

Even if authentication is not enabled on the server, we recommend that you use the version of the Administration client that requires user name and password so you can find the jobs executed by user names. If you supply no user name, the jobs are listed under the default username "nobody".

When the user is successfully authenticated, the credentials become a property of the object, and the user can use the API object to invoke the required API. The server throws a `NotAuthenticatedException` exception if authentication fails.

For more information about objects using authentication credentials, see the JavaDoc or the C# CHM file, available from your server landing page, by default,

```
http://servername:8080/SplusServer.
```

For information about checking authentication using the URL API, see URL authentication.

Use notifications

When the client instance is first instantiated (using the `ClientFactory.getserviceClient()` methods, where service is the type of service), a listening port is opened on the client to receive messages from Spotfire Statistics Services. When any change to the job status (such as job completion, failure, or interruption) occurs, the server sends a message to the client, which can opt to handle the message as appropriate.

You can get or set the notification type property in the `ClientFactory` to either Polling or UDP. By default, it is set to UdpCallback.

As of Apache Tomcat version 8.5, Comet notification is no longer supported; therefore, it is no longer included in the Spotfire Statistics Services Java and C# API.

Notification is supported only in the C# and Java APIs. It is not available in the URL API.

For the UDP protocol on the server, the port is assigned randomly by default. However, it is likely that your server administrator has designated a port or a range of ports specifically for the UDP protocol. You can specify a port using the `ClassFactory.NotificationReceiverPort` property. Check with your server administrator to get more information about your notification protocol, and to learn which ports to use in your client.

User Datagram Protocol (UDP) is a connectionless communication protocol used frequently by services and other special-use applications. Some network routers and switches are configured to either limit UDP traffic or block it entirely. As such, it is usually only reliable in a local area network (LAN) environment where the clients reside on the same subnet as the server. If you are having trouble using the notification features described in this section, consult your Network Administrator to determine whether UDP communication is fully supported on your network.

We strongly recommend against using `GetJobs()` to poll the server to see if the status of any of your asynchronously-run jobs has changed. This practice is resource-intensive. Rather, we recommend that you use the notification APIs provided for C# or Java.

Notification examples

C# and Java implement the notification API slightly differently. This section contains examples for each.

You can create a notification listener that provides information about the job's current status. The status returned via the notification listener can be one of the following:

- `Running`
- `Done`
- Done with Error
- Failed

You can create a single generic notification listener that provides status messages for all jobs, or you can create job-specific notification listeners. If you create a job-specific notification listener, be sure to release it when your job completes. Otherwise, it can become a drain on your memory resources. The following examples show both generic and job-specific listeners.

These examples use the default *service_name* SplusServer. The *service_name* for your Spotfire Statistics Services installation is set by the server administrator and might be different. See your server administrator for more information.

C# notification listener example

```
public class NotificationExample
{
   public NotificationExample()
   {
      IExpressionClient Client = ClientFactory.GetExpressionClient("http://localhost:8080/
SplusServer");
      // execute command asynchronously (return
      // immediately)
      SplusDataResult job = Client.Eval("sleep(10)",
   new SynchronousJobStartup(2000));
      // register notification listener to be called when
      // any updates to jobs submitted by this client occur.
      Client.SetNotificationListener(HandleNotification);
      // Alternatively, you could set up a job-specific
      // notification listener that would only handle
      // notifications for a specific jobId using:
      //   Client.SetNotificationListener(job.JobId,
      //      HandleNotification);
      // NOTE: if setting a job-specific listener, be sure
      // to remove it using
      // RemoveNotificationListener(jobId) when it is no
      // longer needed, to prevent excess memory buildup.
    }
   // method to be called when notification received
   private void HandleNotification(NotificationMessage
      message)
    {
      // if job status has changed, write message to
      // console.
      if (message.Type == eNotificationMessageType.eJobStatusChanged)
      {
         Console.WriteLine("Job " + message.JobId + " is now " + message.JobStatus);
      }
   }
}
```

Java notification listener example

```
public class NotificationExample implements NotificationListener
{
   public NotificationExample() throws NotAuthenticatedException, ApiException,
 ServerCreationException
   {
      ExpressionClient client = ClientFactory.getExpressionClient("http://localhost:8080/
SplusServer");
         // execute command asynchronously (return
         // immediately)
         SplusDataResult job = client.eval("sleep(10)",
           new SynchronousJobStartup(2000));
         // register notification listener to be called when
        // any updates to jobs submitted by this client
        // occur.
         client.setNotificationListener(this);
         // Alternately, you could set up a job-specific
         // notification listener that would only handle
         // notifications for a specific jobId using:
         //   client.setNotificationListener(this,
         // job.getJobId());
         // NOTE: if setting a job-specific listener, be
         // sure to remove it using
         // removeNotificationListener(jobId) once it is no
         // longer needed, to prevent excess memory
         // buildup.
      }
   // method to be called when notification received
   public void handleNotification(NotificationMessage message)
   {
         // if job status has changed, write message to
         // console.
      if (message.getType() ==
         eNotificationMessageType.eJobStatusChanged)
            {
                System.out.println("Job " +
                message.getJobId() + " is now " +
                message.getJobStatus().name());
            }
   }
}
}
```

Write requests using the URL API

You can use the URL API to send administration, expression, or function requests to the server. Use the URL API for testing the health of the server, rather than for creating web-based applications. This section describes sending requests via the URL API.

The URL API is most useful for such tasks as checking server health, retrieving version information, or checking job status. It does not support sending or receiving binary objects.

The URL API accesses five services. The functions you can use to access these services are described in detail, with examples, in the URL API help, available from your server's landing page `http://`*`servername`*`:`*`port`*`/<`*`service_name`*`>`. (Ask your server administrator if you are not sure of the exact URL.)

Important By default, certain Spotfire Statistics Services capabilities are disabled. If you have functions that you created or used in a previous release, you might find that they no longer work as expected. Additionally, any expression that TERR determines to be potentially malicious is disabled. Below is a non-exhaustive example of some of the capabilities disabled by default in Spotfire Statistics Services.

- Sending TERR expressions that perform I/O to the file system or the internet.
- Spawning new OS processes by calling the `system` function.
- Calling into Java using the terrJava package, or using functions in the parallel package.
- Loading new packages, except for those included with TERR.
- Calling `.C` or `.Fortran`.
- Send expressions to the server using the Expression Service.
- Calling `ExtendedServerInfo` or sending other expressions that read from, or write to, your server.
- Sending potentially malicious expressions to the server using the URL API.

Additionally, the Function Service can execute only one function, which allows a connection from Spotfire. If you have functions that you want to have enabled on Spotfire Statistics Services, or if you have additional questions about the configuration settings that control these capabilities, see your server administrator.

Table

Service	Description
Administration service	Contains a subset of functions found in the Java and C# APIs. This service is meant to be used for simple server administration. Includes functions to get information about the server, a list of jobs, or specific job details. Users can also use the URL API to interrupt jobs in the queue or to delete existing jobs.

Service	Description
Expression service	Contains two functions:`eval`, which sends an expression to the server to evaluate; and`getJobDetails`, which retrieves the details about a specific job. (This function is deprecated; use the `jobs` function instead.) By default, the Expression Service is not enabled. See your server administrator for more information.
Function service	Contains two functions: `function`, which sends the call to a function on the server to evaluate; and `getJobDetails`, which retrieves the details about a specific job. (This function is deprecated; use the `jobs` function instead.) The function service can handle only strings and numbers as arguments.
Authentication service	Contains one function, `validateUserCredentials`, for checking whether the current user is authorized to use the server.
Public service	Contains functions for discovering information about the server, including server configuration and version information, and whether it has authentication turned on. You can also use a Public service API to check job status for one or all jobs

Also included in the URL API is the function `jobs`, which, when combined with various resources, can provide information about jobs on the server.

Server information

Notice in the examples that the server version number is required as part of the URL for the Authentication, Administration, Expression, and Function services.

To retrieve the version of the server, use the Public service function `ServerVersion`:

```
http://servername:port/service_name/api/public/ServerVersion
```

This call displays the version in XML, as the following example shows:

```
<value>v8</value>
```

URL authentication

You can use the Public service to check whether authentication is activated on the server, and therefore requires you to supply credentials.

```
http://servername:port/service_name/api/public/authentication
```

this call displays the following XML:

```
<value>false</value>
```

or

```
<value>true</value>
```

Administration tasks

The URL API is most useful for checking server health and performing simple administrative tasks, such as checking for server information, getting the server time and name, interrupting jobs, and deleting jobs if the job list gets too long. For example:

```
http://servername:port/service_name/api/v8/administration/getServerTime
```

This call returns the time on the server, in milliseconds, since epoch.

```
 <value>1265419799119</value>
```

If you need to interrupt a job that is on the server, use the following example construction.

```
http://servername:port/service_name/api/v8/administration/interrupt?jobId=1520
```

If the request was successful, the returned XML appears as follows.

```
 <JobAdminResult responseCode="0">
<JobIdsAffected>
<JobId>1520</JobId>
</JobIdsAffected>
</JobAdminResult>
```

Likewise, if you need to delete a job that is on the server, use the following example construction.

```
http://servername:port/service_name/api/v8/administration/delete?jobId=1520
```

If the request was successful, the returned XML appears as follows.

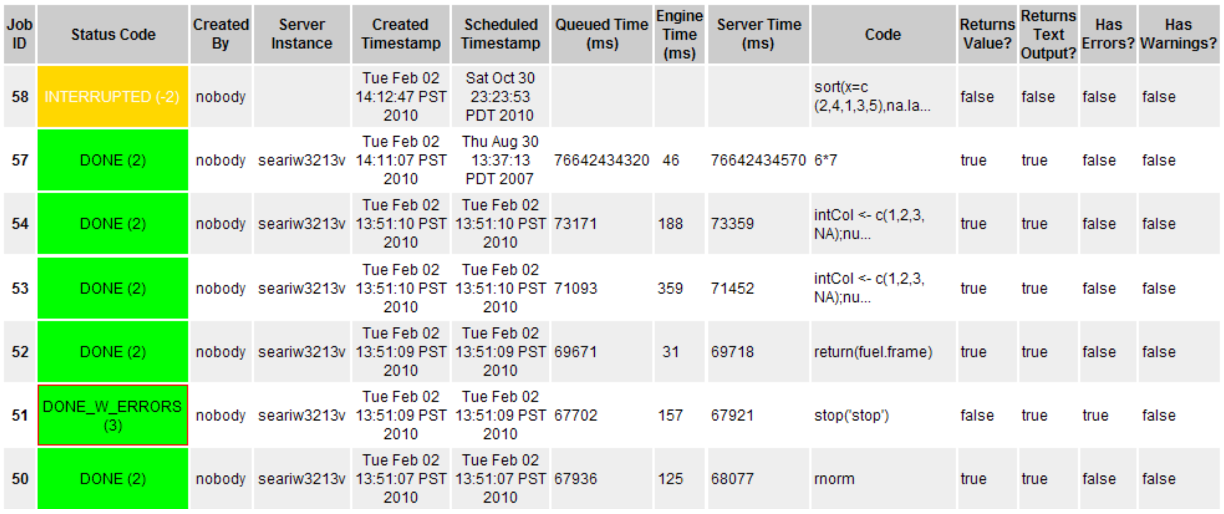
```
 <JobAdminResult responseCode="0">
<JobIdsAffected>
<JobId>1520</JobId>
</JobIdsAffected>
</JobAdminResult>
```

Retrieve the job list

Use the jobs function to get a list of all jobs on the server.

```
http://servername:port/service_name/api/v8/jobs
```

The server administrator can add the style sheet on the server to provide user-friendly formatting to the job list. For example, if you send the above request to a server with a style sheet, your results might look something like these:



Job ID	Status Code	Created By	Server Instance	Created Timestamp	Scheduled Timestamp	Queued Time (ms)	Engine Time (ms)	Server Time (ms)	Code	Returns Value?	Returns Text Output?	Has Errors?	Has Warnings?
58	INTERRUPTED (-2)	nobody		Tue Feb 02 14:12:47 PST 2010	Sat Oct 30 23:23:53 PDT 2010				sort(x=c (2,4,1,3,5),na.la...	false	false	false	false
57	DONE (2)	nobody	seariw3213v	Tue Feb 02 14:11:07 PST 2010	Thu Aug 30 13:37:13 PDT 2007	76642434320	46	76642434570	6*7	true	true	false	false
54	DONE (2)	nobody	seariw3213v	Tue Feb 02 13:51:10 PST 2010	Tue Feb 02 13:51:10 PST 2010	73171	188	73359	intCol <- c(1,2,3, NA);nu...	true	true	false	false
53	DONE (2)	nobody	seariw3213v	Tue Feb 02 13:51:10 PST 2010	Tue Feb 02 13:51:10 PST 2010	71093	359	71452	intCol <- c(1,2,3, NA);nu...	true	true	false	false
52	DONE (2)	nobody	seariw3213v	Tue Feb 02 13:51:09 PST 2010	Tue Feb 02 13:51:09 PST 2010	69671	31	69718	return(fuel.frame)	true	true	false	false
51	DONE_W_ERRORS (3)	nobody	seariw3213v	Tue Feb 02 13:51:09 PST 2010	Tue Feb 02 13:51:09 PST 2010	67702	157	67921	stop('stop')	false	true	true	false
50	DONE (2)	nobody	seariw3213v	Tue Feb 02 13:51:07 PST 2010	Tue Feb 02 13:51:07 PST 2010	67936	125	68077	rnorm	true	true	false	false

Get job details

If you want to know more information about a specific job, you can send a jobs request, specifying `jobID`, `userID`, `status`, or a combination of these resources. To send a request for a specific job, you must have the job identifier (the `jobId`).

For example, the following URL

```
http://servername:port/service_name/api/v8/jobs/120
```

retrieves extensive details about job 120, represented as an XML structure, which includes the job status, its text output, and the code that was run to produce the results.

The following examples demonstrate using other resources, and combining resources to get more refined results. For more information, see the URL API web help, available from the Spotfire Statistics Services landing page.

```
http://servername:port/service_name/api/v8/jobs/users/jdoe
```

```
http://servername:port/service_name/api/v8/jobs/statuses/2
```

```
http://servername:port/service_name/api/v8/jobs/users/jdoe/statuses/2
```

Expression and function requests

Use the Expression and Function services to send requests to the server, and then examine the resulting XML.

An Expression service example:

```
http://servername:port/service_name/api/v8/expression/eval?cmd=1:10&async=true
```

A Function service example:

```
http://servername:port/service_name/api/v8/function/rnorm?n=5&mean=30
```

You can use the Function service for simple requests, such as this example. The Function service can handle only strings and numbers as arguments. If you need to pass a logical or an object, use the Expression service.

The function must be in a library, which is loaded in all engines available on the server. If you need to run a function in a library not loaded in the server `.init` script, talk to your system administrator.

For information about the resulting XML, see The SplusObject.

Interpret and use returned objects

When you send an evaluation request to the server, it provides a generic representation of the value returned as a result of the job execution.

The returned object, a member of the `domain.representation` package in Java and the `Domain.Representation` namespace in C#, is not directly instantiated or modified by your client code. It is constructed by the client API as part of the server response processing.

The returned value representation is a `SpxmlValueRepresentation`, which is automatically converted into an `SplusDataResult` object.

The SplusDataResult object

The `SplusDataResult` object belongs to the domain package (Java) /Domain namespace (C#). Domain contains objects, classes, and data types, enums, and delegates that the Administration, Function, and Expression client interfaces use.

The `SplusDataResult` can be a Domain object (an SPXML representation of an R language object), or it can contain a binary object, depending on the data `SerializationType` property set in the data request, and the engine type that processes the result.

To process a binary object returned by an engine, you must have an engine on the client computer where you can load the object for deserialization.

You can examine the results of an evaluation by calling `getJobDetails` (or, in C# `GetJobDetails`), which returns the job's `SplusDataResult` object.

A completed job's `SplusDataResult` object contains the resulting Domain object, as well as properties describing the job, such as:

- The server XML version.
- Whether the job has a return value, text output, an error, or warnings.
- The user name of the person who created the job (if supplied).
- The date and time the job was created and scheduled or run.
- Elapsed time to run the job, as well as other time-related job and server properties.
- The job identifier.
- The job status (completed, in this case).
- The server instance (if your job ran in a clustered environment).
- A representation of the function or expression that was run (for example, `sqrt(256)`).

An unfinished job immediately returns these properties without a resulting Domain object.

For more information about your particular API and its objects, methods, functions, and properties, see its corresponding help file. A documentation list available on the server landing page (`http://`*`servername`*`:`*`port`*`/`*`service_name`*) can help direct you to the appropriate API help.

The data results XML

A request submitted via the URL API will return an `SplusDataResult` object, represented as an XML structure.

You can specify returning the `SplusDataResult` object as JSON rather than a binary object or SPXML. For more information about this, Return results as JSON.

Example

The following example contains the XML returned from a request to create a Fibonacci sequence, using the URL API:

```
http://servername:8080/SplusServer/api/v8/expression/eval?cmd=x<-filter(rep(0,%2030),%20c(1,
%201),%20method="rec", %20init=c(1,%201))&async=false
```

If you submit your request in a browser via an http:// request, and you do not see the XML displayed in the browser, right-click the resulting window, and then click View Source.

The resulting `SplusDataResult` object, in XML format, looks like the following:

```
<SplusDataResult version="1.0" createdBy="user1" serverTime="3487"
 scheduledMillis="1363216268135" prepTime="0" createdMillis="1363216268135" status="2"
 serverInstance="servername" scheduled="Wed Mar 13 16:11:08 PDT 2013" queueCount="0"
 percent="0.0" jobId="1" hasWarnings="false" hasTextOutput="true" hasReturnValue="true"
 hasError="false" engineTime="3109" created="Wed Feb 4 16:11:08 PDT 2015">
```

Note that the `SplusDataResult` object has the following properties:

Table

Property	Description
`version`	The server XML version.
`createdBy`	The login name of the user who created the request.
`serverTime`	The number of milliseconds the job spent on the server.
`scheduledMillis`	The date and time that the request was scheduled to run, in milliseconds since epoch.
`prepTime`	The number of milliseconds the server takes to prepare to run the job (that is, preparin
`createdMillis`	The date and time that the request was created, in milliseconds since epoch.
`status`	'2' indicates that the job is done. (Other status options include waiting (0), running (1), (-1), and interrupted (-2)).
`serverInstance`	In a clustered environment, the server containing the engine where the job ran. In a sir the name of the server.
`scheduled`	Specifies the date and time the request was scheduled. This synchronous request ran o shortly after the request was submitted.
`queueCount`	Specifies the position in the queue while waiting to run.
`jobID`	The identifier for the job. If a user needs to find it later, the user can pass this jobID to f

`hasWarnings`	`'false'` indicates that the example ran without warnings.
	If `hasWarnings` displays true, they appear in separate section in the `SplusDataResult`
	`<SplusWarnings>`
	`  <![CDATA[`
	`     {"count":1,"warnings":"12345"}`

Property	Description
	`</SplusWarnings>`
`hasTextOutput`	`'true'` indicates that there is console text describing the output. This text is contained `SplusTextOutput`.
`hasReturnValue`	`'true'` indicates that the request produced an `SplusReturnValue`, which contains th results.
`hasError`	`'false'` indicates that the example ran without errors.
`engineTime`	The number of milliseconds the job took to run.
`created`	Specifies the date and time the request was submitted.

The `SplusReturnValue` contains the XML that defines an R language object. The rest of the output contains other details about the returned results. The `SplusTextOutput` contains information that would normally appear in the R Console. You can retrieve the contents of this tag by calling `getTextOutput` (in Java), or by examining the `SplusDataResult` property `TextOutput` (in C#).

The following shows the code, which reflects the expression that the server processed to produce the results. You can retrieve the contents of this tag by calling `getCode` (in Java) or examining the `SplusDataResult Code` property (in C#).

```
<Code>x<-filter(rep(0, 30), c(1, 1), method="rec", init=c(1, 1))</Code>
```

The following shows the `ResultsDir`, which contains the location on the server where the results are stored. You can retrieve the contents of this tag by calling `getResultsDir` (in Java), or by examining the `SplusDataResult` property `ResultsDir` (in C#).

```
<ResultsDir>http://servername:8080/SplusServer/webdav/results/CA98699BC304B54B/</ResultsDir>
</SplusDataResult>
```

The SplusObject

An R object is represented in the Java and C# APIs as an `SplusObject` class. An R object is returned as part of a non-binary object, represented in XML as an SPXML structure. (Supported binary formats, such as SBDF are handled differently.)

See Creating a binary serialization request for R using Java and rJava for more information about working with binary data.)

Spotfire Statistics Services returns the `SplusObject` as an SPXML representation. An `SplusObject` can be one of the following object types:

- Vector
- Data Frame
- List
- Expression Object
- Function
- Call Object

- Generic
- Multidimensional Array

Most of these objects can contain other objects (which can, in turn, contain other objects, including vectors or other `SplusObjects` or generic classes). Within the structure, each object has certain defining characteristics, such as a name, attributes, value, or items, among others.

If you are using the Spotfire Statistics Services Java or C# API to send data through Spotfire Statistics Services, you must limit the data to a size that can be sent using SPXML. If the data exceeds the allowed size of 512,000 bytes, the API throws an exception.

Return results as JSON

If you are building a web application that uses JavaScript or Java, you can set up Spotfire Statistics Services to return results in the JavaScript Object Notation (JSON) format rather than in the XML format (the default).

For more information about JSON, see http://www.json.org/.

To get your results in JSON, specify a standard HTTP accept header value of "application/json".

The JSON object that Spotfire Statistics Services returns is equivalent to the SPXML, except the XML elements become JSON objects (or arrays of objects, if more than one XML element of the same name is included), and XML attributes become object properties.

To use JSON, you must make sure that property names generated are compliant with JSON rules. That is, to return valid properties, check that any hyphen or dash ("-") is replaced by an underscore ("_").

Example

The following XML output example is the results of sending to Spotfire Statistics Services the expression "1:10":

```
<SplusDataResult version="1.0" createdBy="nobody" serverTime="309"
 scheduledMillis="1441915267457" prepTime="0" createdMillis="1441915267457" status="2"
 serverInstance="jdoe-X240" scheduled="Thu Oct 7 13:01:07 PDT 2021" queueCount="0"
 percent="0.0" jobId="38" hasWarnings="false" hasTextOutput="true" hasReturnValue="true"
 hasError="false" engineTime="34" created="Thu Sep 10 13:01:07 PDT 2015">
<Code>1:10</Code>
<SplusReturnValue>
<TERR>
<DisplayOptions decimalPattern="#,##0.00" scientificPattern="0.00E0"/>
<Header>
<Application name="TERR" version="TIBCO Enterprise Runtime for R version 6.0.0 (2021-10-07)"/
>
<Timestamp>Thu Oct 7 13:01:07 2021</Timestamp>
</Header>
<Vector length="10" type="integer">
<Items>
<Item>1</Item>
<Item>2</Item>
<Item>3</Item>
<Item>4</Item>
<Item>5</Item>
<Item>6</Item>
<Item>7</Item>
<Item>8</Item>
<Item>9</Item>
<Item>10</Item>
</Items>
</Vector>
</TERR>
</SplusReturnValue>
<SplusTextOutput>
<![CDATA[ **** Engine execution output **** ]]>
</SplusTextOutput>
<ResultsDir>
```

```
http://jdoe-X240:8080/SplusServer/webdav/results/73FD0479F352EC49/
</ResultsDir>
</SplusDataResult>
```

The JSON output from the same execution, appears as follows:

```
{"SplusDataResult":
    {"SplusTextOutput":"\n\t\t**** Engine execution output ****\n ... Type 'q()' to quit R.\n
\t",
    "scheduledMillis":"1363287836030",
    "ResultsDir":"http://jdoe-X240:8080/SplusServer/webdav/results/DB303112E91DE7/",
    "jobId":"307",
    "queueCount":"0",
    "createdMillis":"1260830585143",
    "status":"2",
    "serverInstance":"jdoe-X240",
    "SplusReturnValue":{"TERR":
        {"DisplayOptions":{"decimalPattern":"#,##0.00","scientificPattern":"0.00E0"},
            "Vector":{"Items":{"Item":
["1","2","3","4","5","6","7","8","9","10"]},"length":"10","type":"integer"}}},
    "engineTime":"23","Code":"1:10",
    "hasTextOutput":"true",
    "hasReturnValue":"true",
    "version":"1.0",
    "scheduled":"Thu Oct 7 13:08:00 PDT 2021”,
    "createdBy":"user1","created":"Thu Oct 7 13:03:12 PDT 2021",
    "hasError":"false",
    "serverTime":"90",
    "hasWarnings":"false"}}
```

JavaScript example (Dojo-specific)

The following JavaScript example demonstrates setting the server to return results as JSON.

This example uses Dojo, a javascript framework for constructing interactive browser applications. Write your code for the framework you use.

```
function loadJobDetails(nodeSelected){
   if (nodeSelected == undefined) return;
   dojo.xhrGet( {
         url:
          "http://localhost:8080/TERRServer/api/v8/jobs/" +
           nodeSelected,
         handleAs: "json",
         headers: {accept: "application/json",
                Authorization:
                 connectionInfo.getBasicAuthentication()},
         preventCache: true,
         load: function(response)
        {
               //handle success
        },
        error: function(response) {
              //handle errors
       }
   }); // end dojo.xhrGet
}
```

Transient and persistent data

For all but the simplest calculations, you must read data from input data files and write results to output data files and graphics files.

Often, it is useful to take an output data file from one calculation and use it as an input to another.Spotfire Statistics Services supports this practice by maintaining "transient" file directories for storing output data files, and "persistent" file directories that can be used for both input and output files.

Additionally, Spotfire Statistics Services provides the class `SplusDataRequest`, a collection of R objects and/or an encapsulated data set, which is passed as an argument to an `eval()` function. See Pass in a data object for more information about using this object.

Access transient and persistent data on the server using WebDAV

You can access both transient and persistent data directories from external client applications using the WebDAV Client API.

Also, you can gather information about the current execution environment using the `spserver.*` functions, a handful of functions that are available on every running TERR engine on the server.

Transient data directories

When a job is run on the server, the engine working directory is set to a new, empty "transient" directory.

TERR or open-source R code can write data to an empty transient directory by specifying non-absolute file paths. For example, executing the expression `cat(1:10, file="foo.txt")` writes a new file `foo.txt` in this directory. Also, you can use this working directory for temporary files.

After job finishes, this "transient" directory is kept around for awhile, and then it is deleted. The time is controlled by the server configuration. The default setting for this property is 24 hours. (See your System Administrator for more information about changing this property.)

Files from the transient data directory can be read using WebDAV or plain HTTP. When a job finishes, the result includes a `ResultDir` containing the URL for accessing this directory. A client application can use this property to construct a complete URL for accessing a particular file in the directory.

The database containing the job information persists, so it can contain references to job results in WebDAV that have been deleted from the transient directory as part of its periodic cleanup.

To retrieve the results directory URL, from the code running on the server, you can use the spserver function `spserver.results.url(file="")`. See The spserver.* functions for more information.

Persistent data directories

Spotfire Statistics Services includes a persistent WebDAV repository that you can use to store input data files or output data files that need to stay around.

The transient results directory is useful for storing output data files that are downloaded immediately by a client application. It is not so useful when the output data are not accessed immediately. Neither is it useful for input data files.

The WebDAV repository includes a persistent directory for each user name. For example, by default:

`http://`*`servername`*`:`*`port`*`/`*`service_name`*`/webdav/users/username/`

In addition, the server includes a common storage directory that all users can access.

`http://`*`servername`*`:`*`port`*`/`*`service_name`*`.webdav/common/`

Use the WebDAV APIs provided with Spotfire Statistics Services to upload and download data files.

Spotfire Statistics Services WebDAV API

The Java Client API and the C# Client API include support for managing files and directories in a WebDAV repository.

Using these APIs, you can perform the following tasks.

- Copy, move or delete files.
- Upload or download files.
- Create a folder.
- Retrieve the contents of a folder.
- Check to see if a file or folder exists.

The C# and Java APIs provide a WebDAV client interface. That is, like the Administration, Function, and Expression clients, the APIs implement a standard Factory design pattern for client instance creation. The ClientFactory class implements a factory and provides a way to obtain a WebDAV client object by calling a relevant method. For example, in C#:

```
string serviceUrl = "http://localhost:8080/SplusServer"; string username
  = "myusername"; string password   = "mypassword"; WebdavApiClient api =
 ClientFactory.GetWebdavClient(serviceUrl, username, password);
```

For more information about the WebDAV API, see its help for the appropriate language, available from the server landing page, by default:

`http://`*`servername`*`:`*`port`*`/`*`service_name`*

The spserver.* functions

When the TERR or open-source R engine is executing on the server, it defines a set of functions that the code can call to access information about the current execution environment.

Table

Function Name	Description
`spserver.file(file="", user.name=spserver.user.name())`	Returns the local file path for permanent storage for the given user. • If `user.name` is not given, the default is to use the current user • If `user.name` is given as "", this specifies the common storage a The file argument is pasted on the end to construct a complete URL
`spserver.home`	Returns the server's home path.

`spserver.on.server()`	A Spotfire Statistics Services job.
	The expression `(exists("spserver.on.server") && spserver.` code that is executed either on the server or the desktop.
	just calling `spserver.on.server()` by itself fails on th
	This function is useful if you are calling a function that requires dif desktop environment and a server environment. For example, if yo generates a graph on the server, you might want the graph to be wr such code on the desktop, you might want to create a window disp
	`if (exists("spserver.on.server") && spserver.on.`
	`     # if running on server: write the`
	`     # graph to a file`
	`     myfile<-spserver.results.file("xx.ps")`
	`     postscript(myfile)`
	`     draw.my.graph()`
	`     dev.off()`
	`     spserver.results.url("xx.ps")`
	`} else {`
	`     # if not running on server, use the default`
	`     draw.my.graph()`

Function Name	Description
	`}`
`spserver.results.url(file="")`	Return the WebDAV URL that can be used for accessing the result f argument is pasted on the end, to construct a complete URL. Illegal are converted to the appropriate URL characters.
`spserver.results.file(file="")`	Returns a file name in the local working directory for storing result is "", this returns the results directory, otherwise the `file` argumen construct a complete file path.
`spserver.service.id`	Returns the server's ID.
`spserver.service.url`	Returns the server's URL.
`spserver.share`	Returns the server's share.
`spserver.url(file="", user.name=spserver.user.name())`	Return the URL for a WebDAV directory providing permanent stor user.name is not given, the default is to use the current user name. this specifies the common storage area. The `file` argument is past a complete URL. Illegal URL characters (like spaces) are converted characters.
`spserver.url.to.file(url="")`	Returns the local file path corresponding to the provided URL in th If the URL is not local to the server's WebDAV root, this function re
`spserver.user.name()`	Returns the user name specified when the current job was created. This might return the empty string, if the server does not require us
`spserver.user.password()`	Returns the password specified when the current job was created. This might return the empty string, if the server does not require us
`spserver.webdav.root()`	Returns the local file path corresponding to the WebDAV root.
`spserver.webdav.url()`	Return the URL for a WebDAV directory providing permanent stor name is not given, the default is to use the current user name. If use specifies the common storage area. The file argument is pasted on t URL. Illegal URL characters (like spaces) are converted to the appro

Example

The most useful of these functions are `spserver.results.url` and `spserver.results.file`, which give simple access to the transient file names. For example, you could evaluate an expression:

```
   {
     myfile<-spserver.results.file("xx.ps")
     postscript(myfile)
     draw.my.graph()
     dev.off()
     spserver.results.url("xx.ps")
 }
```

This job result would be the complete URL for accessing the resulting file.

Troubleshoot SSL certificates

If you are developing a client to work with a server that uses SSL, you must obtain the certificate that the server is using in its SSL configuration.

Using this certificate design, your server and client use a “handshake” mechanism for sending and receiving encrypted data.

We address using SSL certificates with Java only.

For the Java client, if the client and server do not both have the “handshake” mechanism in place, client requests to the server fail.

You can either set a property to bypass checking for the signed certificate, or you can make sure your application validates the server certificate. This section describes both options.

Bypassing the certificate validation

You can bypass checking for a signed SSL certificate with the Java API.

Procedure

- To bypass checking for a signed SSL certificate with the Java API, set to true the property `TIBCO_STATSVCS_SSL_ALLOW_ANY_CERTIFICATE` as follows:

  ```
  System.setProperty(BaseClient.TIBCO_STATSVCS_SSL_ALLOW_ANY_CERTIFICATE, "true");
  ```

  If you set this property to `true`, your Java application does not validate the server certificate.

Installing a certificate

Installing a certificate requires a four-step process.

Procedure

1. Get the server certificate from your server administrator.
2. Create a new truststore.
3. Call Java to run the test and provide the appropriate parameters to identify the truststore.
4. Extract the certificate from the keystore.

Creating the new truststore

To have an SSL “handshake” with the server, you must generate the truststore from the certificate used by the server.

Prerequisites

Make sure your system administrator has followed the directions in the *TIBCO Spotfire Statistics Services Administration Guide* for installing the certificate.

You can perform these steps on the either server and copy the truststore to all machines that will run the client, or you can perform the steps directly on the machines that will run the client.

Procedure

1. At a command prompt, type the following (where drive is the location for your keystore and keystore.cer is the server certificate you got from your server administrator):

   ```
   keytool -import -keystore drive:/truststore -alias tomcat -file drive:/
   keystore.cer
   ```
2. Provide a keystore password. For example:

   mypassword

   Your output should resemble the following:

   ```
   Owner: CN=usercomputername, OU=UserComany, O=UserCompany, L=City, ST=WA, C=US
   Issuer: CN=usercomputername, OU=UserComany, O=UserCompany, L=City, ST=WA, C=US
   Serial number: 78e2951
   Valid from: Tues Feb 3 1:10:16 PDT 2015 until: Wed Sept 14 10:10:26 PST 2015
   Certificate fingerprints:
      MD5: 38:B2:41:FD:4D:3A:34:26:13:0C:34:3C:A6:51:66:EA
      SHA1: CA:3B:A9:9A:87:8B:A1:E6:E5:73:07:89:33:D0:B6:17:20:67:66:8A
   ```
3. Provide the value yes when the system asks if you trust the certificate:

   ```
   Trust this certificate? [no]: yes
   ```

Result

`Certificate is added to the keystore.`

Certificate parameters for testing

Before you run the certificate test for the Jave client, you must make sure you run the JVM with the correct parameters.

Run the JVM with the following two parameters.

```
-Djavax.net.ssl.trustStore=drive:/truststore
-Djavax.net.ssl.trustStorePassword=mypassword
```

where *drive:/truststore* is the truststore generated or copied as described in the steps above, and *mypassword* is the password for that truststore.

Extracting a certificate from the keystore

You must extract the certificate keystore used by the server.

Prerequisites

- Before extracting a certificate from the keystore, when you are running the Java client, you must have set the JVM to run with the correct parameters. See Certificate parameters for testing.
- You must know the password that the keystore was generated with.

Procedure

1. Type the following command.

   ```
   keytool -export -keystore drive:/keystore -alias
   tomcat -file drive:/keystore.cer
   ```

   where *drive:/keystore* is the keystore used by the server.

2. Provide the password of the keystore.

   ```
   Enter the keystore password:  mypassword
   ```

Result

The certificate is stored in `keystore.cer`.

Addressing engine issues

If you encounter problems that you suspect are a result of the engine process, you can examine the file `engine.log`, which is available in the results directory in the WebDAV store. This file contains detailed information about how a specific job was handled by the engine.

Prerequisites

To access this file, you must have the job ID of the affected job.

Procedure

1. In a browser address bar, type the following:
   `http://`*`servername`*`:`*`port`*`/`*`service_name`*`/api/v8/jobs` .
2. In the job list, locate the `jobID` for the affected job.

   the `JobID` appears as a link.
3. Click the `JobID` link.
   The `SplusDataResult` is displayed.
4. In the resulting `SplusDataResult`, locate the `ResultsDir`.
5. Copy the URL, and then paste it into your browser address bar.
   The `WebDAV` directory is displayed.
6. In the resulting `WebDAV` directory, open the file `engine.log`.

Result

The file contents are displayed in your browser. You can review it for further information about the job.

Documentation and support services

For information about the Spotfire products, you can read the documentation, contact TIBCO Support, and join the 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 website is updated frequently and is more current than any other documentation included with the product.

TIBCO Spotfire Documentation

The documentation for all Spotfire products is available on the TIBCO Spotfire® Documentation page. This page takes you directly to the latest version of each document.

To see documents for a specific Spotfire product or version, click the link of the product under 'Other versions', and on the product page, choose your version from the top right selector.

Release Version Support

Some release versions of TIBCO Spotfire products are designated as long-term support (LTS) versions. LTS versions are typically supported for up to 36 months from release. Typically, defect corrections are delivered in a new release version and as hotfixes or service packs to one or more LTS versions. See also https://docs.tibco.com/pub/spotfire/general/LTS/spotfire_LTS_releases.htm.

How to Contact TIBCO Support

You can contact TIBCO Support in the following ways:

- For an overview of TIBCO Support, visit http://www.tibco.com/services/support.
- For accessing the Support Knowledge Base and getting personalized content about products you are interested in, visit the TIBCO Support portal at https://support.tibco.com.
- 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 https://support.tibco.com. If you do not have a user name, you can request one by clicking Register on the website.

System Requirements for Spotfire Products

For information about the system requirements for Spotfire products, visit http://spotfi.re/sr.

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 https://community.tibco.com.

For quick access to TIBCO Spotfire content, see https://community.tibco.com/products/spotfire.

Legal and Third-Party Notices

Index

Special Characters

.NET 6

A

absolute paths 9
administration 28
API 20, 21, 38
ASP.NET 6
Aster database 15
authentication 23, 23

B

binary 22
Binary 6

C

C# 6, 20, 20, 20, 21
certificate 41, 42, 42
certificates 41
certification 41
Cluster 6
Comet 24
Configuration 6
Converter 6
created 31
createdBy 31
createdMillis 31
cross platform 9
custom function 10

D

data 17, 18, 37
database 15
developer 4, 5
domain packages 6

E

engine 21, 44
engineTime 31
errors 17
eval 20
Eval() 7
Exceptions 6
expression API 10
expression interface 10

F

function 15

H

hasError 31
hasReturnValue 31
hasTextOutput 31
hasWarnings 31
help 7, 10, 12

I

IsAuthenticationEnabled 23

J

Java 6, 20, 20, 20, 21, 22, 42, 42
Javascript 35
JDBC 9
Job 6
job details 29
jobID 31
jobs 28, 31
JobStartup 7
JSF 6
JSON 35
JSP 6

L

listener 24
local engine 14

M

MATLAB 16, 17, 18, 19

N

namespace 6
notification 24
notifications 24

O

overview 4

P

package 13

package deployment 7
packages 12
Packages 6
persistent 37
persistent data 37
polling 24
prepTime 31
programming 21
protocol management 6

Q

queueCount 31

R

R code 10
R differences 10
R function 10
relative paths 9
Representation 6
returned objects 31
rJava 22

S

SAS 16, 16, 17, 17
scheduled 31
scheduledMillis 31
server 20, 20, 20
server version 27
serverInstance 31
serverTime 31
SplusDataRequest 7
SplusDataResult 20, 31
SplusDataResult object 31
SplusObject 34
Spotfire configuration 14
spserver.file 39
spserver.home 39
spserver.on.server 39
spserver.results.file 39
spserver.results.url 39
spserver.service.id 39
spserver.service.url 39
spserver.share 39
spserver.url 39
spserver.url.to.file 39
spserver.user.name 39
spserver.user.password 39
spserver.webdav 39
spserver.webdav.root 39
SPXML 34, 35
SpxmlValueRepresentation 31
SSL 41
status 31
system integrator 4, 5

T

transient data 37, 37

U

UDP 24
UdpCallback 24
URL 6, 20
URL API 26, 27, 28

V

validation 13, 41, 42, 42
version 31

W

webDAV 37
WebDAV 6, 37, 38

X

XML 29