Client-Server Programming

Object Model and Namespaces

With most systems, you must create custom logic for your projects and create Tags or Variables for all internal properties. FactoryStudio allows your application to directly access all the objects that you created in your project.

This means that user-created temporary tags are not required to manage the status of PLC network nodes, the total number of alarms in a group, or the number of rows in a dataset. You can now access runtime objects, business objects (representing a network node), or an alarm group or dataset. Then, you can display the required information or take action directly through the object's built-in properties.

The properties and methods used when creating the application are organized in a few divisions or by using the technical names. They are in a **namespace**.

The namespaces in FactoryStudio correspond with the main modules of a project:

Tags	Dataset
Historian	Script
Security	Server
Alarm	Client
Device	Info

The objects inside each area will have a unique name. For instance, you cannot have two tags with “Tank1” for a name, but you can have a TAG and a DISPLAY with “Tank1” as a name as long as they have different namespaces.

For instance, the Tag namespace has all the tags in the application, and each tag has built-in field properties (Quality, TimeStamp, Min, Max, Units, and many others). Examples: Tag.tagname1.bit0, tag.tagname2.timestamp

This concept applies to all namespaces.

Some of those built-in properties (like tags values from the PLCs) are the same in the server and client computers that are connected with that server, but some properties (like the current logged user name) are specific to one client computer.

This brings us the concept of Client-Server Domains, which is explained in the next section.

About Server-Client Namespaces

In FactoryStudio, we have the concept of namespaces that encapsulates a set of real-time variables and methods. Two of those namespaces are @Server and @Client.

The @Server namespace holds all the information about the server computer, while the @client namespace has the properties of each client computer connected with the server. The @Server properties are accessible to all project modules and to all remote computers. The values and methods for the @Client are only accessible locally at that computer.

For Example, @Server.Date can be used in a FactoryStudio module or display, and it will have the same value across all computers using that value. @Client.Username can be used only on a DISPLAY and Scripts that are local to each computer. The value of @Client.Username is local and different to each computer.

When you are running FactoryStudio on only a computer, the concept still applies as there is one Windows process that is the SERVER Process (running the DataAcquisition, Alarms, Historian, and Server side scripts) and the Windows CLIENT process (running the displays and client-side scripts).

Server and Client Tag Domains

When creating distributed applications, FactoryStudio has the concept that a TAG can have a Domain Client or Server. A Tag with a Server Domain, which is the default, will have the same value across all the modules and all client computers (some systems refer to it as a Global or Shared value to the project).

Typically, tags you are using to read from PLCs, Communication protocols, Alarm Management, or a Historian should be defined as Domain Server.

Tags you are using for temporary operator Input, reference variables to dynamic user interface, reports or queries parameters, or local display data should be defined as Domain Client.

Accessing Server or Client Tags from FactoryStudio Modules

FactoryStudio has some modules that always run on the Server, some that always run on the client side, and some that can run on both domains.

The Modules that always are on the Server side are the Alarms, Historian, and Devices. This means that you should use only Server tags when you are defining Alarms, Historian, and Device Communication points.

Graphical Displays always run on the client domain, which means they can access BOTH the Server Tags and the Client Tags. The difference is that the Server tags are automatically synchronized between the Display and the server process (both when reading or writing to the tag). The Client Tags are local to the computer running that display. The Client tags are not written back to the server computer. Since it is connected with the display, the CodeBehind follows the same rule.

The SCRIPT TASKS can run on either the Server side or the Client side according to the user. As expected, the Script-Tasks that are assigned to run on the Server computer will access only the Tags that are the Domain Server. The Script-Tasks that are assigned to run on each client computer will have access to both Server and Local Tags (Global and Local data).

The SCRIPT CLASSES are methods and class libraries. You can also specify the Server or Client domain to the classes to specify which processes can use that library.

Finally, the SCRIPT EXPRESSIONS will automatically run in the Server computer or in the Client computer according the Tags it is uses. If the expression is using only Server Tags, the system will allocate it automatically to run it in the server computer. If the Expression had one more Client tag, it will run at each client process.

Accessing Datasets (SQL tables and Queries) from Server and Client processes

The DATASET module is accessible from both the Client and Server modules. When calling a METHOD (such as the SelectCommand()), the method is executed by who calls the method. Therefore, if the call is from a Display or ScriptTask of type Client, it will be executed and evaluated on the client side. If it is from a Script-Server, it will be evaluated on the server computer.

One important feature of FactoryStudio is that only the Server computer needs to have the connection and security to access the SQL database, even when starting queries on the client computer. When a client side script calls the SelectCommand(), the command is requested by a FactoryStudio service on the server that runs the command, but the results of the query are processed on the client side.

The DATASET module also allows a command to be asynchronously executed on the Server computer. Asynchronously calls are always executed on the server computer. In order to start an asynchronous action, you need to toggle (change) the value of the property, such as Select property.

Here are the two options to run a query compared:

@Dataset.Table.table1.SelectCommand(): Will run synchronously, on the server or client domain, using the tags and results according to where they were called.

@Dataset.Table.table.Select += 1: Will run asynchronously, always on the server computer.

> ⚠ When configuring the Dataset module: You must call the queries from the CLIENT process using the synchronous method to allow the system to properly evaluate the tags. This must be done when you use the mapping feature with tags of type client, or when you embed tag values in the SQL or Where conditions using the {tag.tag1} syntax with tags that are the Client type.

Using Reports from server and client processes

The Report module follows a logic similar do Datasets that are accessible form both the Server and Client process, which resolve the tag values according to where they were activated and have an asynchronous operation that always run on the server side.

When calling a Method, such as the SaveCommand(), the method is executed by who calls the method. Therefore, if the call is from a Display or ScriptTask of the Client type, it will be executed and evaluated on the client side. If it is from a Script-Server, it will be evaluated on the server computer.

Saving files will always save on the server computer, or the filename path will always be resolved on the server side, even if the command to save was started from the client computer.

The Report module also allows a command to be asynchronously executed on the Server computer. Asynchronously calls are always executed on the server computer. In order to start an asynchronous action, you need to toggle (change) the value of the property. Here are the two options compared:

@Report.Report1.SaveCommand(): Will run synchronously, on the server or client domain, using the tags according to where they were called.

@Report.Report1.Save += 1: Will run asynchronously, always on the server computer.

You can start an asynchronous call to save a report using the SaveTrigger column and mapping it to a tag.

> ⚠ When configuring the Report module: If you use local tags or symbols mapping to local tags, you must call the queries from the CLIENT process using the synchronous method so the system will be able to do the proper evaluations of those tags.

FactoryStudio Script Module and Client-Server Domains

FactoryStudio Script, despite being called scripts, are full-featured complied Microsoft .NET Framework Application domains.

We elected to name it script since all of the compiling and linking DLLs and all of the .NET application domain generations are transparently executed to the user in the background while the user is creating the application.

Since it is a complete .NET system, it allowed FactoryStudio to automatically embed many features for performance, distributed processing, multitasking, and security when using scripts. Here are some of those features.

All Scripts marked with the DOMAIN SERVER will run in their own Windows process (.NET AppDomain), which can be located on the same computer the TServer.exe is running, or they be allocated to run on a remote computer. They can also be elected to run as threads in the same TServer.exe process.

The scripts market with the DOMAIN CLIENT will run as independent threads in the Client process (either RichClient, SmartClient or Web).

For Enterprise projects, each Script-TASK will run in its own execution thread. Therefore, a task that requires more time to conclude its execution will not block the execution of other tasks.

The system has a set of built-in protections, such as:

- All of the Scripts created by a user have an added TRY-CATCH protection. Even if code is created by the user, the script engine remains intact.
- The system prevents a Task from reentrancy. If a Task is still running and it is requested to run again, the execution will start only after the previous one finishes.
- The system has advanced configurations to allow creating a queue of tasks for later execution, so you don't miss any critical event.
- The system automatically creates a tread-poll, which means that it automatically controls how many parallel execution threads is optimal on the system, based on how many CPU cores your system has.
- Real-time data synchronization among tasks is performed by the FactoryStudio engine. Each task can use global shared variables, as if it was local to its own execution thread.

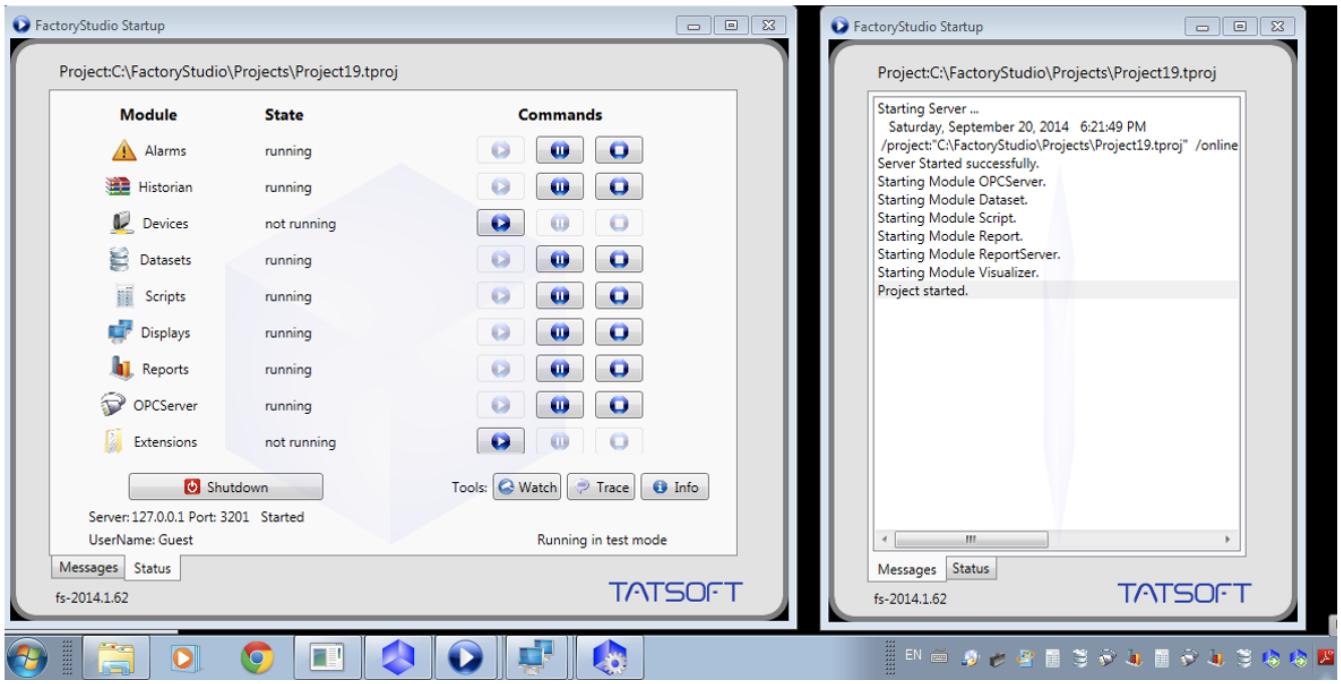
FactoryStudio Execution Processes

Tatsoft FactoryStudio was created to leverage the current and future generations of multi-core CPUs and to add an extra layer of security, having different processes running different parts of the application. Besides providing enhanced performance and security when running all processes in one computer, it also allows easy distribution of the process among many computers.

When you run a project to execute (either in Startup or Test mode), a project runtime loader application, called TStartup.exe, is activated.

The only purpose of this application is to execute the many processes with the right parameters. After the project is running, you can close the TStartup window, or keep it in the background if want to have an easy way to start and stop modules or shutdown the system.

Here is the TStartup.exe application and its user interface. The application icon is the PLAY image.



The main role of the TStartup.exe is to run the TSERVER.exe with the right set of parameters. The TServer application will load the project and the real-time tags. It is the main process that will allow the other modules from the same computer or remote computers to exchange real time data.

You can also start the TServer directly as a batch file or setup it to start as a Windows Service, as described in the product user documentation.

The TServer will allocate a WCF TCP/IP PORT to allow data exchange with other modules. Each runtime project must run in its own unique port. By default, running Startup projects use port 3101 and Test projects use port 3201.

After the TServer is running, the system starts a process called TRunModule.exe to allow the execution of the other modules, such as Alarms, Scripts, Datasets, and so on.

If you open the Windows Task Manager when running a project, you can expect to see a TServer.exe process and many TRunModule.exe.

Finally, a process called TRichClient.exe runs the operators displays (or the equivalent processes for the SmartClient or Browser displays).

> **Hint**
>
> If you need to close the project you are testing and you don't have a "Shutdown icon", you can located the TServer Icon in the window task bar (the blue cube with the play button), and right click to close that process. Closing the TServer will automatically close the child modules process.

FactoryStudio Projects Running as a Windows Service

The TServer.exe can run as a Windows Service as well as the other modules. Refer to the product documentation on how to set it up. The Displays (TRichClient or SmartClient or Web clients) will always run on User Mode, not service, as they are dependent on the user login.