Scripts and .Net Framework

All of the programming in FactoryStudio consists of C# or Visual Basic 100% managed code that is designed to run in the Microsoft .NET framework. These languages are used to create scripts that run periodically or when specified events occur. The following sections describe how to create and work with these scripts:

> When you use the code editor, the system constantly compiles code in the background. You can also build the whole project, as needed. For more information, see Running the Application.

For more information about the runtime objects included in the software platform, see Runtime.

The .NET libraries are available in the software platform, which means, that if you use Microsoft Visual Code, there is a library available to use with our platform. The built-in script has built-in methods you can call using TK.<methodName>. For more details on these methods, see T.Toolkit and TK.Class namespaces.

Configuring Tasks

Tasks are program units, written in VB.Net or C#, that execute either when a trigger event occurs or periodocally at specified intervals. FactoryStudio includes the following built-in tasks:

- ServerStartup—Executed when the project starts running. Runs on the server (computer running TServer.exe).
- ServerShutdown—Executed when the project shuts down. Runs on the server.
- ClientStartup—Executed on each client when the Displays module (TVisualizer.exe) starts running.
- ClientShutdown—Executed on each client when the Displays module closes.

To configure tasks:

- Go to **Edit > Scripts > Tasks**.
- Select a task name, or select the insert row (first row) to create a new task.
- Enter or select information, as needed.

Column	Description
Name	Enter a name for the task. The system lets you know if the name is not valid.
Code	Read-only. This specifies the language used for the code for this task. By default, this is the language you selected when you created the project. From the Code Editor tab, you can change the code language. To change the project default, go to **Info > Settings**.
Trigger	Enter or select the event (tag or other object) that should trigger the task execution, if any. The task executes when the value of the object changes.
Period	Enter the time interval that should occur between executions of the task, if there is no trigger.
Domain	Select where the script executes: • Client—Task executes on each client system. These are tasks that apply locally (on the user's computer). For example, report generation. • Server—Task executes on the server system. These are tasks that apply across the application, that is, globally.
InitialState	Select the task's initial state: • Enabled—Task is ready to run. • Disabled—Task is not ready to run. You may enable the task under specific conditions.
BuildStatus	Read-only. Status of the task code from the continuous compiling process. • Green check mark—Task runs without errors. • Red X—Task has warnings or errors. Double-click to go to the warning or error. Warnings are information only. Errors will prevent the code from running for that specific task. The rest of the application will run.
BuildErrors	Read-only. Displays any errors encountered during the last build.
EditSecurity	Set the security to enable who can edit the tasks.

Description	Enter a description of this task.
[Other columns]	For definitions of other columns that are available in this table, see Common Column Definitions.

- Double-click the row to access the Code Editor tab. You can now enter or edit the code for the task. See "Working with the Code Editor" below.
- Click **Save**.

Configuring Classes

Classes let you create a repository of class libraries, methods, and functions that you can use across the application. You can call them from tasks, other classes, and displays (code behind).

FactoryStudio comes with the following built-in classes:

- ServerMain—Methods library available for all server tasks.
- ClientMain—Methods library available for all clients.

To configure classes:

- Go to **Edit > Scripts > Classes**.
- Select a class name, or select the insert row (first row) to create a new class.
- Enter or select information, as needed.

Column	Description
Name	Enter a name for the class. The system lets you know if the name is not valid.
Code	Read-only. This specifies the language used for the code for this task. By default, this is the language you selected when you created the project. From the Code Editor tab, you can change the code language. To change the project default, go to **Info > Settings**.
Domain	Select where the class executes: • Client—Class executes on each client system. These are classes that apply locally (on the user's computer). For example, report generation. • Server—Class executes on the server system. These are classes that apply across the application, that is, globally.|
Content	Read-only. Show the type of the content in the class (i.e.: method or namespace)
EditSecurity	Set the security to enable who can edit the tasks
BuildOrder	Set the order to build the classes
BuildStatus	Read-only. Status of the class code from the continuous compiling process. • Green check mark—Class runs without errors. • Red X—Class has warnings or errors. Double-click to go to the warning or error. Warnings are information only. Errors will prevent the code from running for that specific class. The rest of the application will run.|
BuildErrors	Read-only. Displays any errors encountered during the last build.
Description	Enter a description of this class.
[Other columns]	For definitions of other columns that are available in this table, see Common Column Definitions.

- Double-click the row to access the Code Editor tab. You can now enter or edit the code for the class. See "Working with the Code Editor" below.
- Apply all namespaces that will be used in your code, e.g.
  - “Using System”
  - “System.IO.StremReader”
- Click Save.

Working with the Code Editor

Editing Code

You can write code in either standard VB.Net or C#. You can also switch between the two. If you change your language selection in the code editor, the system automatically converts existing code to the selected language.

If you need references to your own assemblies, you can use Run > Build > References.

As a fully compliant .NET application, you can find free source code to use, including .NET components, products, and libraries. Plus, you can use your own libraries.

FactoryStudio exposes many .NET libraries to the application, but there are a few methods that are frequently required, such as type conversion, copying tags to DataTables and .NET objects, or dynamically changing the communication settings. Those methods are included in a library toolkit. To use these methods, you need to put TK. in the code editor. IntelliSense will respond with a list of all available methods and provide a summary documentation for any method you select.

Example

```
double x = TK.ConvertTo<double>("123");
```

To edit code:

- Go to **Edit > Scripts > CodeEditor**.
- From the drop-down list in the main toolbar, select the task or class you want to edit. To create a new task or class, see "Configuring Tasks" or "Configuring Classes" earlier on this page.
- If needed, select a different code language from the code editor toolbar.
- You can also format your code to be more readable. To do so, click on Auto Format. A prompt will appear; click "OK".
- Click **Save**.

Using the .NET Debugger

Creating debugger Information

FactoryStudio has an integrated .NET code debugger. In order to use it, the local computer must have a cache with the necessary files to run the debugger. The steps to enable the debugger are:

- Enable the Debug Information on **Run > Build > Messages.**
- As necessary, save the source code you want to debug. This will create the debug information. This step is only necessary the first time you open the project on the computer. Afterwards, the background compiling generate the necessary debugging information to enable the use of breakpoints and step execution.

Attaching the .NET debugger

In order to have a .NET debugging session, the engineering environment must be connected to the runtime, and the .NET debugger must be attached to server or client process. Follow these steps.

- When Running the project, either on **Run > Test** or **Run > Startup** enable the connect check box.
- If the project is already running, go to **Run >Test** or **Run > Startup**, according to the runtime you want to attach, and connect to the runtime system by pressing the connect button on those pages.
- Open any script that has debugging information and press the Attach .NET debugger button. A message on the bottom of the engineering workspace will show that a debugging session is active with the server components or the client components of the running project.
- When the .NET debugger is attached, the system will stop on the defined breakpoints and will stop automatically when any .NET Exception occurs.

Breakpoints, Steps, and Watch

In order to setup a breakpoint, open the desired code, select the line, and press Insert Breakpoint on the toolbar.

When the system stops on a breakpoint, you can perform step by step execution or hit the continue button.

In order to inspect local .NET variables, tags, or project objects, select the text in the script editor. When the execution is stopped on a breakpoint, the toolkit will show the current value of the variable.

You can also add .NET variables or project objects to the Watch window. When adding tags or project objects, you need to use the @ symbols, example @tag.tag1. This will let the system know it is a project object and not a local .NET variable. Keep in mind that the Watch display is only updated when execution is stopped. If you want to have real-time values for tags and objects, open the PropertyWatch diagnostics tool.

Configuring Expressions

Besides the **Edit > Script > Expressions** table, you can use expressions in several other places in FactoryStudio by using the syntax described here. This can be done to determine a value or to configure a condition.

Expressions are plain VB.Net expressions, such as arithmetic expressions or calls to script.class methods from the project. IntelliSense only shows tags and application objects, but the expressions are compiled using the standard VB.Net compiler. Whatever you would write in the code editor should be accepted in expressions as well.

FactoryStudio implements automated replacements, such as == to =. The syntax of an expression is close to that of a C# statement but without a need to add the ";" on the end.

In this way, both VB.Net and C# programmers are able to use the expression fields seamlessly.

In expressions, you do not need to put an @ before tag names. You need the @ in the code editor to differentiate project tags from .NET variables. However, expressions do not have local .NET variables, so you use the project object directly.

For arithmetic operands, use the standard operands as described in the .NET documentation.

To allow single-line evaluations, the .NET language has the IIF command, which currently is used only with VB.Net. The IIF command has three parameters.

> **Example**
>
> IIF (A, B, C)

The first parameter is a condition. This method will return B if condition A is true, and it returns C if the condition is false.

> **Example**
>
> IIF (tag.A = 1, "True", "False") will return the string "True" if the value of tag.A is 1, or "False" if tag.A has a different value.

In this .NET method, all three parameters are evaluated independent of the condition. For instance, if you have IIF (tag.A = 1, script.class.client.Func1(), script.class.client.Func2()), both Func1 and Func2 will always be executed. Only the return value will be different based on the value of tag.A.

The IF or IIF methods need to evaluate the parameters before calling the method. There are many scenarios where you may want to execute only the function according to the value. For these scenarios, FactoryStudio has a method called TIF.

Use the expression:

TIF (tag.A = 1, script.class.client.Func1(), script.class.client.Func2())

Only the Func1() or Func2() will be executed, according the value of Tag.A

The TIF method is defined in the class library that is automatically exposed to expressions that are in the toolkit function library.

For more complex calculations, you can call a class that you create on the Classes tab. See "Configuring Classes" earlier in this chapter.

To configure expressions:

- Go to **Edit > Scripts > Expressions**.
- Select an expression, or select the insert row (first row) to create a new expression.
- Enter or select information, as needed.

Column	Description
Object	Select an existing tag or object.
Expression	Enter the expression. The expression can be a basic mathematical expression, use a class, or be a conditional expression.

Domain	Select where the expression executes: • Client—Expression executes on each client system. These are expressions that apply locally (on the user's computer). For example, report generation. • Server—Expression executes on the server system. These are expressions that apply across the application, that is, globally.|
Execution	Select when the expression executes: • OnChange—The expression executes when the value of any tag in the expression changes. • TriggerOrTimeInterval—The expression executes when the trigger event occurs or when the interval set in the period elapses. • ChangeOrStartup—The expression executes when the value of any tag in the expression changes or at startup. • TriggerOrTimeOfDay - The expression executes when a trigger event occurs or on a specific time of day
Trigger	Enter or select the tag or object that triggers the expression execution. The expression executes when the value of the object changes.
DisableCondition	Enter or select the tag or object that disables the expression execution.
Time	Specify the time when the expression runs.
Label	Set a label to the specified class.
Build Messages	Return the message status after the expression runs
BuildStatus	Read-only. Set after you click **Verify**. • Green check mark—Expression runs without errors. • Red X—Expression has errors.
BuildErrors	Read-only. Displays any errors encountered during the last build.
[Other columns]	For definitions of other columns that are available in this table, see Common Column Definitions.

- Click **Verify** to check the validity of the expression.

The Script Namespace

The namespace **Script** is the entry point for all objects related to the Scripts module.

The **Script**.Task object lists all configured database connections and their runtime properties.

The **Script**.Class object lists all configured tables and its runtime properties.

See Namespaces for the complete programming reference on runtime objects.