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.

FactoryStudio has an underlying .NET object model with a 100% managed code, specifically targeting the development of Real-Time data management applications. The hierarchical object model includes the following top-level objects that correspond to the main modules in FactoryStudio:

Tags	Dataset
Historian	Script
Security	Server
Alarm	Client
Device	Info

The top-level hierarchy is implemented as .NET Namespaces. Each Namespace has .NET classes and objects that are created when building projects. These objects have runtime properties, methods, statuses, and configuration settings.

For instance, the Tag namespace contains every tag that is in an application and each of the tags have built-in field properties, such as Quality, TimeStamp, Min, Max, Units, etc.

Examples:

Tag.tagname1.bit0, tag.tagname2.timestamp
The same concept of tag fields applies to all namespaces, for instance:
Alarm.TotalCount:, Alarm.Group.Warning.Disable:

FactoryStudio has Intellisense auto-completion for when you build a project, fill in input fields, or create scripts. This guides you to any existing properties that are allowed for the object you are editing and allows you to easily "drill down" to a specific property.

When accessing a project's object in the .NET Script Editor, it is necessary to prefix the namespace with an "@" symbol in order to avoid conflict with the names of the .NET local variables.

Examples:

In Script-Tasks and CodeBehind, use:
@Tag.Analog1
@Device.Node.Node1.Status

The @ symbol is not necessary on Grids and Dialogs. Some input fields may require objects of only one type, such as Tag or Display. For these, Intellisense will automatically guide you to the required objects.

These concepts may seem abstract for users that do not have experience in .NET or similar object-oriented systems. However, the power of these concepts will become clear when users learn the engineering configuration tools and the FactoryStudio modules. When users get used to working with object models and Intellisense, the users realize that there is a huge increase in productivity so they no longer want to work with systems that lack these features.

You can see the available Namespaces on Factory Studio here.