

# Tags, Assets, and Templates

## What are Tags and Templates?

### Tags

FactoryStudio uses the term "tag" to refer to real-time variables and their associated historical data. Tags usually map to devices, such as PLC registers or other physical equipment in the production process. A tag can also be connected to entries in SQL databases, external data sources, or an internally calculated value.

All tags have a specific type, such as an integer, text, or date and time. FactoryStudio provides over a dozen predefined types and allows users to define new types. In addition, some tags may be defined as arrays, and some tags may have optional parameters.

Tags are the process variables for your application. Use tags and their properties to set up the data model for your process.

### Templates

Although FactoryStudio provides many predefined types, you may need to define your own type. Templates are used to do this. Templates may be defined with properties similar to those of predefined types. Each template represents a user-defined type that can be used in the same manner as predefined types.

Using templates, you can add to the types of tags available for your project by creating new types to fit your application needs. This could be as machine data, equipment status, vessels, or as the representation of any asset attributes in your plant.

## Built-in Tag Types

FactoryStudio provides 13 built-in tag types, most of which are based directly on .NET datatypes. The 13 built-in tag types are summarized in the following table.

Tag Type	.Net Type	Value Range
Digital	System.Int32	0 through 1
Integer	System.Int32	-2,147,483,648 through 2,147,483,647
Long	System.Int64	-2,147,483,648 through 2,147,483,647
Double	System.Double	-1.79769313486231570E+308 through -4.94065645841246544E-324 for negative values; 4.94065645841246544E-324 through 1.79769313486231570E+308 for positive values
Decimal	System.Decimal	0 through +/-79,228,162,514,264,337,593,543,950,335 with no decimal point; 0 through +/-7.9228162514264337593543950335 with 28 places to the right of the decimal; smallest nonzero number is +/-0.000
Text	System.String	0 to approximately 2 billion Unicode characters
Timer	System.Int32	Same range as Integer, but with built-in parameters to produce a repetitive wave pattern (see notes below)
Date Time	System.DateTimeOffset	from 12:00:00 midnight, January 1, 0001 to 11:59:59 P.M., December 31, 9999
Time Span	TimeSpan	Data Interval in Days, Hours, Minutes, Seconds and Milliseconds, where each of those properties can hold a Double value
Guid	Guid	Standard Microsoft Globally Unique Identifier (GUID)
Data Table	System.Data.DataTable	Holds an in-memory DataTable


Image	System.Byte []	Can hold an Image file or any binary content. The Long value is the maximum size of the content.
-------	-------------------	--

## Decimal

The Decimal type allows calculation with higher precision than the Double type. However, math operations using Decimal can be 40 times slower than using Double. So the Decimal type should only be used when Double precision is not enough.

## Timer

Timer is a built-in integer type that can be used to generate precise timing signals. Timers have the following varieties:

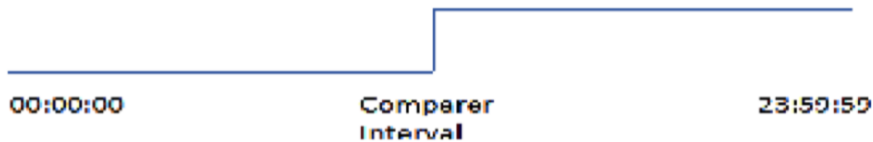
- SquareWave:  The value toggles between 0 and 1.
- Pulse: The tag changes to 0. Then immediately changes to 1.



- DelayOff: The tag behaves as a PLC Timer Off. If you set the tag with a value other than the StartValue during runtime, the tag will hold that value for the period specified in the Interval. The tag goes back to its StartValue after the period of time (Interval).



- Comparer: The tag is set to 1 after the specified comparer Interval, and then goes back to zero at midnight.



For SquareWave, Pulse, and Comparer, the tag toggles between 0 and the StartValue (instead of 0 and 1) if you set the tag StartValue.

## Reference

Reference tags allow dynamic addressing of variables.

The way you use Reference tags in FactoryStudio is closer to how you use references in .NET programming than it is to the old C++ pointers. Reference tags are similar to .NET references because they both have a type. When you create a reference tag, you need to define the type of object the reference will point to. Unlike C++ pointers, a Reference tag cannot be defined to point to invalid memory areas that would cause errors in the application.

The target type for the Reference types is defined in the Parameters columns.

Typed references bring advantages in both the engineering stage and in runtime. In the engineering stage, it allows IntelliSense to directly browse the template members if a reference is pointing to a template. In runtime, it allows string data validation.

All Reference tags have an additional runtime attribute. This is the **link**, and it specifies which tag that the reference will be linked to during execution.

Essentially, the link property is a string property that needs to receive the target tag name before using the reference tag. You can assign a string directly or by using a string expression. The best way to set the link property is to use the method GetName(), which will create the string based on the current tag name. This way, you can rename the tag without having to search the strings. This also shows the tag names linked on the cross-reference utility.

An example project (ReferenceTags) ships with FactoryStudio.



Examples:

```
@Tag.Reference1.Link = @Tag.TagName.GetName() (VB)
@Tag.Reference1.Link = @Tag.TagName.GetName(); (C#)
@Tag.Reference1.Link = "Tag.TagName"; (C#)
@Tag.Reference1.Link = "Tag.TagNa" + "me"; (C#)
```

The reason for using the GetName() method, instead of using strings directly, is that with GetName() you retain the benefits of Cross-Reference and Refactoring.

## Creating and Editing Tags

To create and edit tags:

- Go to **Edit > Tags > Objects**.
- To create a new tag, enter its name and any other applicable properties in the blank top row (Insertion Row). To edit an existing row, select any item in the row and make any desired changes. Here is a complete table of available tag properties. Note that many of these properties are optional and only apply to specific kinds of tags.

Column	Description
Name	Enter a name for the tag. The system lets you know if the name is not valid. <ul style="list-style-type: none"><li>• If you edit the name of an existing tag, the system automatically updates the name throughout the project.</li></ul>
Type	Select the tag type, which may be a built-in type or a user-defined template.
Parameters	Configure any parameters. The parameters vary based on the tag type. Once the type is entered, you can double click on the parameters field to see a dropdown menu with any applicable parameters. <ul style="list-style-type: none"><li>• <b>DeadBand</b> — It is used as a limitation to save new values to the tag. It defines the difference necessary between the old value and the new value. Also is possible to set the DeadBand as an percentage and with decimal numbers as: 10.5; 10%; 300%; 17.8%.  Example 1: If the DeadBand is Set to 5 and initial value of the tag is 0, the next input possible to be saved, need to be <math>\geq 5</math> or <math>\leq -5</math>.  Example 2: If the current value of the tag is 100 and the deadband is set to 10%, the next input possible to be saved, need to be <math>\geq 110</math> or <math>\leq 90</math>.</li></ul>
Array	When this field is blank, the tag is not an array.  When the field contains an integer value of N, an Array is created from position 0 to N.  For example, if the field contains the value 5, the Array is created from Tag[0] to Tag[5]. This means that 6 elements are created.  Two programming styles are accommodated by this method; one that counts elements from 0 to less than five, and one that counts from 1 to 5.

The columns above are visible by default. To add or remove one of the column below, right-click the column heading area and check or uncheck the columns that should be visible.

Column	Description
Units	Enter the engineering units of measure that you want to use as a label for this tag.
StartValue	Enter a starting value for this tag. This is the value the tag will be initialized with when FactoryStudio starts.
Format	Enter a default format for displaying the data. For example: <ul style="list-style-type: none"><li>• N0—Number with no decimal places.</li><li>• N3—Number with three decimal places.</li><li>• X—Hexadecimal (supported only for integral types).</li><li>• C—Currency.</li><li>• When configuring output in the Displays, the format for each output field can be individually defined, but in most scenarios it is easier to attach the default formatting to the tag itself.</li></ul>


Retentive	<p>Select the option to save the value of the tag and its internal properties to the database every time the value changes. This retains the value when the application shuts down and makes the value available when the application next starts.</p> <ul style="list-style-type: none"> <li>• None—Does not retain the value or properties.</li> <li>• ValueOnly—Retains only the value.</li> <li>• Properties—Retains all properties, including the value.</li> <li>• PropertiesOnly—Retains all properties, except the value.</li> </ul>
Min	Enter the minimum value that is valid for the object.
Max	Enter the maximum value that is valid for the object.
Visibility	<p>Select the value visibility on the OPC server for remote projects:</p> <ul style="list-style-type: none"> <li>• Private—Tag is visible only to the local project and redundant pair.</li> <li>• Protected—Read-only tag that is visible on the OPC server to remote projects and OPC clients.</li> <li>• Public—Tag is visible on the OPC server to remote projects and OPC clients.</li> </ul>
Domain	<p>Tag value for the entire project or value specific to each client display.</p> <ul style="list-style-type: none"> <li>• Server—Tag value is consistent across the entire project and all remote client displays.</li> <li>• Most tags in a project should be Server tags.</li> <li>• Client—Tag value is local to each remote computer running a client display (web or visualizer displays).</li> <li>• Use Local tags to denote temporary data specific to individual client computers. The most common use of Local tags is when temporary data is needed to manage the user interface on the displays.</li> <li>• Local tags allow different values on each client computer.</li> </ul>
Comment	Enter any comments about this tag.
ReadSecurity	Select which groups have the right to read the Tag. Tag Security protection can be configured in Display > Client Settings
ScaleMin	Enter the scale min value for communication.
ScaleMax	Enter the scale max value for communication.
Device Point	Read-only. Show which communication point is related to the tag (if related)
WriteSecurity	Select which groups have the ability to write in the Tag. Tag Security protection can be configured in Display > Client Settings
[Other columns]	For definitions of other columns that are available in this table, see <a href="#">Common Column Definitions</a> .

Continue adding as many tags as you need.

## Notes

You can create a tag from anywhere in FactoryStudio by clicking **New Tag** in the toolbar.

Like any other configuration table, you can import CSV files or copy/paste content directly from an Excel spreadsheet or from other applications.

With FactoryStudio, you can replace names at any time. An easy way to create a tag is to click on the Name column of the insertion row, then press space and enter. Each time you do this, the system will create the same type of tag that was last created. In the insertion row, select the Type, then click on the header or any other part of that grid. This will create a tag with a default name. Also, you can configure more than one row at a time by selecting the rows with the Shift button, then right-click and select "Edit Combined Rows". A new popup will open with the information for the rows. The settings changed in this window will change all of the selected rows. If a column has , it shows that this column has more than one configuration.

## Tag Formats

The format property defines the display format of tag values. These formats follow the specifications provided in Microsoft .NET. For valid numeric formats, refer to [Standard Numeric Format Strings](#). For example: N1 (number with 1 decimal place).

For valid date and time formats, refer to [Standard Date and Time Format Strings](#). For example: d (short date).

For a more in-depth discussion of format strings, refer to [Formatting Types](#).

<i>Numeric format examples</i>	
Specifier	Description
N0	Number with no decimal places
N3	Number with 3 decimal places
X	Hexadecimal (supported only for integral types)
C	Currency

<i>Date/time format examples</i>	
Specifier	Description
T (only)	Long time pattern (equivalent to HH:mm:ss).
d (only)	Short date pattern (equivalent to M/d/yyyy (month/day/year) for en-us).
dd	Show the day of the month as a number from 01 through 31.
ddd	Show the abbreviated name of the day of the week.
dddd	Show the full name of the day of the week.
MM	Show the month as a number from 01 through 12.
MMM	Show the abbreviated name of the month.
yy	Show the year as a two-digit number.
yyyy	Show the year as a four-digit number.
hh	Show the hour as a number from 01 through 12.
HH	Show the hour as a number from 00 through 23.
mm	Show the minute as a number from 00 through 59.
ss	Show the second as a number from 00 through 59.
fff	Show the millisecond as a number from 000 through 999.
tt	Show the A.M./P.M. designator.

---

## Creating Templates

Templates let you create new tag types based on existing built-in types.

### To create a tag template:

- Go to Edit > Tags > Templates.
  - Click **New**.
  - The "Create New Tag Template" dialog shows.
  - In the "New Type Name" field, enter a name for the tag type. In the Description, enter a description of the tag. Click **OK**. The Templates tab displays with the name of the new template at the top of the tab.
  - Click the insertion row to create a new attribute for this tag template.
  - Enter or select information, as needed. The properties are the same ones for Tags. See Creating and Editing Tags above.
  - To delete a template, select it from the User Custom Type drop-down list, then click **Del**.
  - On the **Objects** tab, you can now use this new template in the Type column.
- 

## Assets and Categories

### Assets

Assets let you configure additional metadata for your project when you have the Enterprise version of FactoryStudio. For example, you can organize the objects in your project into a hierarchy. This lets you group tags that are related to each other. The hierarchy may reflect such things as the area of your manufacturing floor or the location of your machinery.

To create assets:

- Go to **Edit > Tags > Assets**.
- Right-click "Elements" and select **New Level**.
- Enter a name for the level.
- Right-click the new level and select **Insert Asset**.
- The Select Object window displays, with all the objects organized by type on the left side.
- Select the object type on the left side and the object you want on the right side.
- Click **OK**.
- The object becomes a child of the selected level.
- Continue adding child or sibling levels and inserting assets, as needed.
- Right-click to rename or delete a level, or right-click an asset to delete it.
- In the **Objects** tab, select the new level in the Level column.

## Categories

If you have the Enterprise version of FactoryStudio, you can create user-defined categories of data that you can use as tag metadata. Categories are useful for filtering, both when creating the project and during runtime.

To create categories:

- Go to **Run > Dictionaries > Categories**.
- Enter or edit the name and description of the category.
- Continue adding as many categories as you need.
- On the **Tag > Objects** tab, select the new category in the Category column. Other project elements can use categories for project organization.

---

## The Tag Namespace

All project tags are available in the runtime modules as .NET objects for the Tag Namespace.

All of the built-in tag types share a common set of properties and methods which are defined in the base class **TagObj**. The tags created from user defined templates are implemented by the base class **UserType**.

Class Type	Description.
TagObj	Base classes to all Tag objects.
Digital	Runtime properties for tags of Type Digital.
Analog	Runtime properties for all Analog Tag Types.
AnalogInt	Runtime properties for tags of Type Integer.
AnalogLong	Runtime properties for tags of Type Long.
AnalogDecimal	Runtime properties for tags of Type Decimal.
AnalogDouble	Runtime properties for tags of Type Double.
Text	Runtime properties for tags of Type Text.
TDateTime	Runtime properties for tags of Type DateTime.
Timer	Runtime properties for tags of Type Timer.
TimeSpan	Runtime properties for tags of Type TimeSpan.
Reference	Runtime properties for tags of Type Reference.
TDataTable	Runtime properties for tags of Type DataTable.
UserType	Runtime properties for tags from Templates.

See [Namespaces](#) for the complete programming reference on runtime objects.