Pages and Displays

Displays are components of the user interface of your application. A display can have multiple elements in it, which include controls, data display areas, static areas that always display, and more.

Elements of a display can be static, that is, they always display, such as a menu bar, toolbar, or a status bar. Displays can also change depending on what the user clicks on or selects.

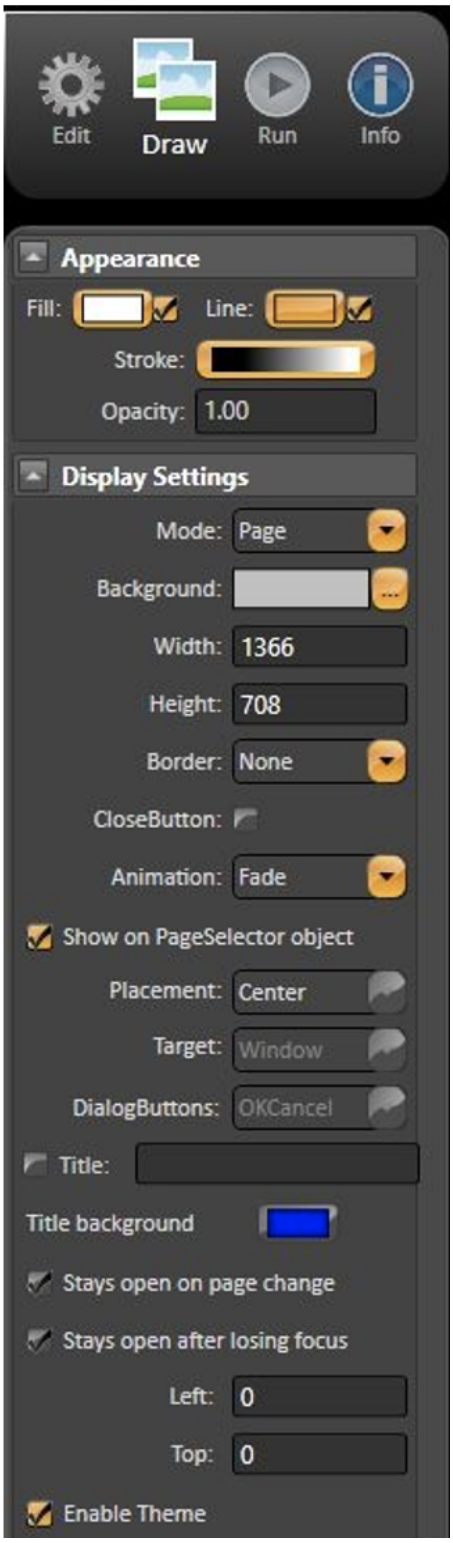
Creating a new Display

To create a new Display, go to **Draw** > **Drawing**. Click Close Display and Create New One. (If you do not see this button in the toolbar, make your window wider.)

The New Display window will open. Enter information, as needed.

Column	Description
Display Name	Enter a name for the display.
Description	Enter a description for the display.
.NET Smart Client	Select this option if the display is designed to be presented natively on the host platform.
HTML5	If you plan to use this display as HTML5 on browsers, select this option when you create the display. **You cannot change it later**. Select this to use this display with any browser. When you select this option, the display settings in other parts of the Draw area only show options that are compatible with the HTML5 feature.
iPad /iPhone iOS Target	If you plan to use this display for iPad and iPhone users, select this option when you create the display. **You cannot change it later**. Select this to be able to use this display with iPads and iPhones. When you select this option, the display settings in other parts of the Draw area only show options that are applicable to both Windows and iPads/iPhones. Select the default view for iPad and iPhone users: Landscape or Portrait.

On the left side of the window in the DisplaySettings, enter or select information, as needed. If you selected the iPad/iPhone iOS Target or HTML5 option when you created the display, not all of the DisplaySettings are available.



The main option in the Display settings is the Display Mode. There are 4 different modes you can choose from:

1. **Page** - Default. Opening a page automatically closes the last page on the current layout and shows last in the list on **Displays > Layouts**. Only pages are shown in this layout.
2. **Popup** - A popup opens on the top of all other displays. When you open a new page, by default, all popup displays are closed. Controls on other displays are are available.
3. **Dialog** - A dialog opens as a modal dialog, which disables the controls on all other open displays until you close the dialog. Clicking OK executes an appropriate method on the display CodeBehind.

4. **PopupWindow** - A window popup opens on the top of all other open displays. When you open a new page, by default, the popup window displays are not closed. Controls on other displays are available.

Display Opening Methods

There are many ways to open a new Display or Popup. For each method, specific characteristics must be considered. Below you will find every method for opening a new Display with some description on their behavior and how to use them.

Display.DisplayName.Open

- Begin Open Display process
- Returns a flag (true or false) indicating success or fail
- Uses the following syntax:

```
@Display.<DisplayName>.Open();
E.g.:
@Display.MainPage.Open();
```

Display.DisplayName.OpenModal

- Open Display as Modal
- Not available for Mono and HTML5
- Returns a flag (true or false) indicating success or fail
- Uses the following syntax:

```
@Display.<DisplayName>.OpenModal();E.g.:@Display.MainPage.OpenModal();
```

Display.DisplayName.NewPopup

- Open a new popup. If the Display is already open, a new instance will open
- Always returns an empty string, independent if it succeeds or fails
- Uses the following syntax:

```
@Display.<DisplayName>.NewPopup(LabelList[label1=tag1;label2=tag2],Left,Top, Width,Height);
E.g.:
@Display.MyPopup.NewPopup("PID=PID1;NAME='PID  1'");
```

Client.OpenDisplay

- Open a Display
- Returns a flag (true or false) indicating success or fail
- Uses the following syntax:

```
@Display.OpenDisplay("DisplayName");@Display.OpenDisplay("About");
```

Client.OpenDisplayAtIndex

- Open Display at x-order list of displays. Current Display is closed

- Returns a flag (true or false) indicating success or fail
- Uses the following syntax:

```
@Client.OpenDisplayAtIndex(string DisplayName,int index);E.g.:@Client.OpenDisplayAtIndex("MainPage",0);
```

Client.NewPopup

- Open a new popup. If the Display is already open, a new instance will open
- Always returns an empty string, independent if it succeeds or fails
- Uses the following syntax:

```
@Client.NewPopup(string DisplayName, LabelList[label1=tag1;label2=tag2],Left, Top,Width,Height);E.g.:
@Client.NewPopup("PopupPID","PID=PID1;NAME='PID 1'");
```

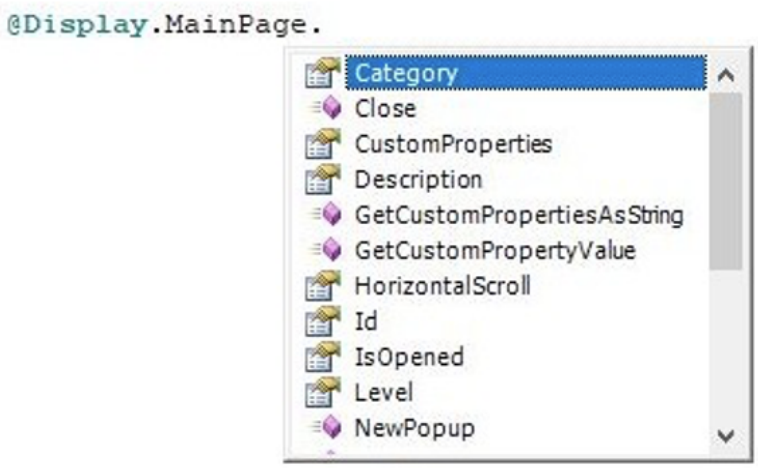
Display Namespace

With most systems, you must create custom logic for your projects and create Tags or Variables for all internal properties. Namespaces allow 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.

When you are building a project's configuration, filling input fields, or creating scripts, the system uses Intellisense auto-completion to guide you to all the existing properties that you are allowed to use according to what you are editing. This feature allows you to easily 'drill down' to a specific property.

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



Below you will find a list of the Display namespace's properties and methods and some properties from the Client namespace.

Display Namespace

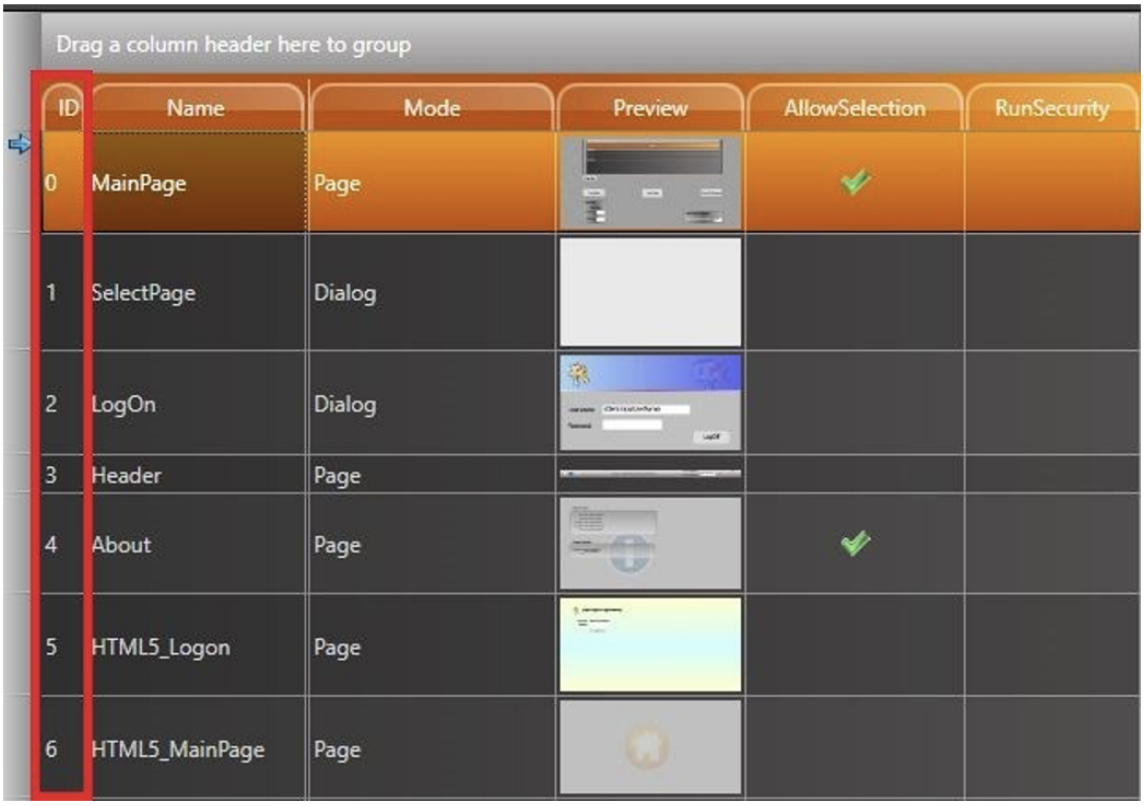
Properties

Category: ReadOnly property with categories that are configured for the Display Object in **Run** > **Dictionaries** > **Categories**. Used for organization purposes.

Description: Description of the display, configured in **Edit** > **Displays** > **Displays**.

HorizontalScroll: Indicates the horizontal scroll value.

Id Object: Object ID (Internal Use). Can be found at **Edit** > **Displays** > **Displays**.



IsOpened: Indicates if the selected display is opened.

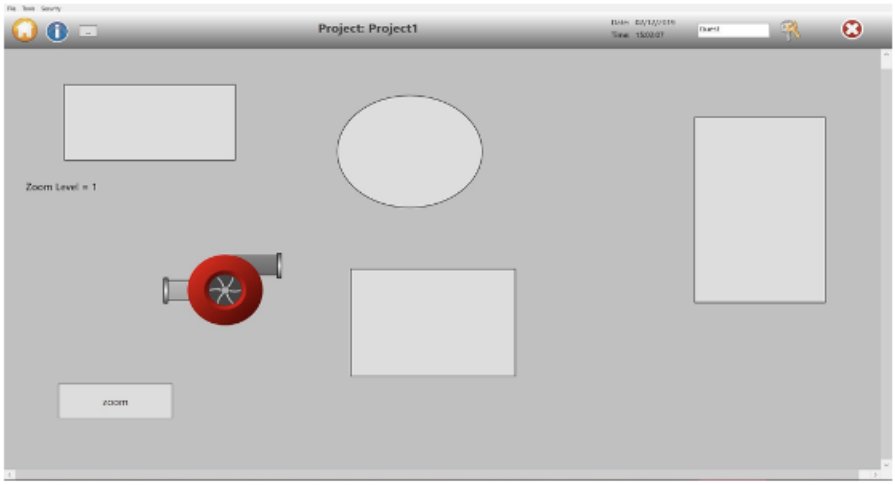
- Returns **false** if is not opened

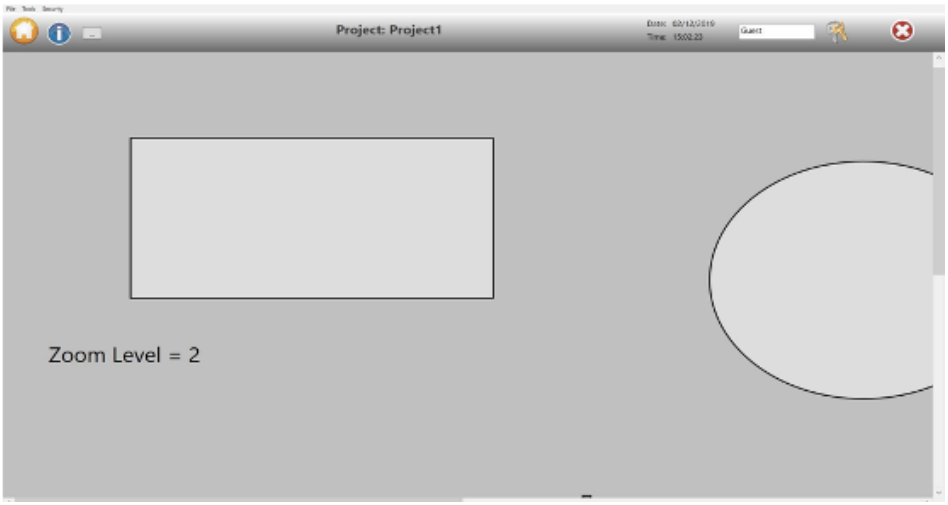
- Returns **true** if is opened

Level: ReadOnly property with levels configured for the Display Object in **Edit** > **Tags** > **Assets**.

VerticalScroll: Indicates the vertical scroll value.

ZoomLevel: Indicates the zoom level of the page. One is the normal zoom level. See images to see a difference between zoom level 1 and 2.





Methods

Close()

- Begin close display
- Returns a flag indicating success (true) or fail (false)

Example

```
bool sts = @Display.About.Close();
```

GetCustomPropertiesAsString(string sep)

- Get all custom properties
- Default Separator is ‘**,** ‘
- Returns a string

Example

```
string msg = @Display.About.GetCustomPropertiesAsString(",");
```

GetCustomPropertyValue(string PropertyName,string defaultValue):

- Get value of custom property name
- **propertyName**: Property Name
- **defaultValue**: Default value if property could not be found
- Returns property value

Example

```
@Display.MainPage.GetCustomPropertyValue("p1","-1")
```

NewPopup(object[] items):

- Opens a new popup. If the Display is already open, a new instance will open
- Always returns an empty string, independent if it succeeds or fails

Example

```
@Display.MyPopup.NewPopup("Max=MaxValue;Min=MinValue");
```

Open()

- Begin open display
- Returns a flag indicating success (true) or fail (false)

Example

```
bool sts = @Display.About.Open();
```

OpenModal()

- Open Display as modal
- Disabled for **Mono** and **HTML5**
- Returns a boolean indicating success (true) or fail (false)

Example

```
bool sts = @Display.MainPage.OpenModal();
```

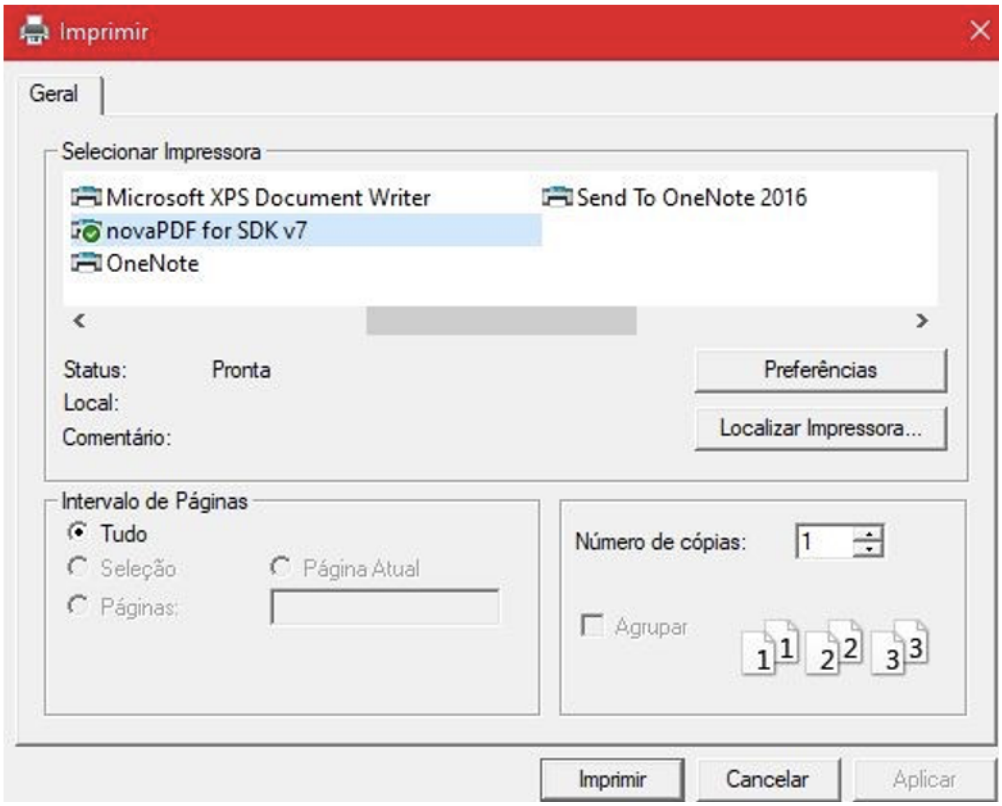
PrintDisplay(bool defaultPrinter)

- Begin print display
- Returns a boolean indicating success (true) or fail (false)
- **defaultPrinter**: flag indicating whether default printer must be used or not

Example

```
bool sts = @Display.MainPage.PrintDisplay(true);
```

- If the default printer is set to **false**, a popup window will appear.



PrintDisplayDefaultPrinter(int PageOrientation) :

- Begin print display using default printer
- **PageOrientation**: 0 for Portrait and 1 for Landscape
- Returns a flag indicating success (true) or fail (false)

Example

```
bool sts = @Display.MainPage.PrintDisplayDefaultPrinter(0);
```

RemoveAllCustomProperties():

- Remove all custom properties
- Returns always Not used

Example

```
@Display.LogOn.RemoveAllCustomProperties();
```

SetCustomProperties(string str, string sep)

- Set custom property. If property does not exist, it is created.
- **propertyName**: Property Name. If the name starts with an '@' and the rest of the string is a valid tag name, the property is a reference to the Tag Value.
- **propertyValue**: Property value
- Returns always Not used

Example

```
@Display.MainPage.SetCustomProperties("p1=10;p2=20",  ";")
```

SetCustomPropertyValue(string propertyName, object propertyValue)

- Set custom property. If the property does not exist, it is created.
- **propertyName**: Property Name. If the name starts with an '@' and the rest of the string is a valid tag name, the property is a reference to the Tag Value.
- **propertyValue**: Property value.
- Returns always Not used.

Example
`@Display.MainPage.SetCustomPropertyValue("p1",  50")`