myPROJECT Designer 6

User Manual

Version May 2015

Table of content

1. About myPROJECT Designer 4
2. Start with myPROJECT Designer 6
2.1 Creating Project 8
2.2 Managing Workspace 12
2.3 Checking Project 14
3. GUI/HMI Editor 17
3.1 Creating Views 17
3.2 Selecting Objects 22
3.3 Drawing Primitives 22
3.4 Creating Text Elements 23
3.5 Inserting Images 24
3.6 Poly-lines 26
3.7 Moving Objects 28
3.8 Resizing Objects 28
3.9 Rotating Objects 28
3.10 Skewing Objects 29
3.11 Corner Filleting 29
3.12 Combining Objects 30
3.13 Fill and Stroke 32
3.14 Rulers and Guides 39
3.15 Layers 42
3.16 Copying and Pasting Elements 43
3.17 Objects Order 45
3.18 Grouping 46
3.19 Repeated Actions Mode 49
3.20 Visual Scripts 49
3.21 Used Tags 50
3.22 Live View 50
3.23 Zoom on Zone 50
3.24 Undo and Redo 51
3.25 Adding Components and Icons 51
3.26 View Properties 53
3.27 Error Box Properties 54
4. Layout Views 56
4.1 Page Layout 56
4.2 Adding Layout View 57
4.3 Creating New Layout 57
4.4 Using Layouts in Views 58
5. Entering Tags and Expressions 60
5.1 Entering Tags 60
5.2 Entering Mathematical Expressions 61
6. Tag Database 66
7. Linking Views with PLC 67
7.1 Animations 68
7.2 Effects 82
7.3 Time Sequence 87
8. Open/Set command 90
8.1 Open Command 90
9. Set Command 94

9.1 Writing Values to PLC ... 94

10. Parametric Views ... 100
10.1 Connection Change in Parametric Views ... 101
10.2 Use of $number$ in texts ... 101

11. Visual Scripting ... 102
11.1 Using Script in Views ... 102
11.2 Declaring Variables ... 103
11.3 Writing Script ... 104

12. Documents ... 107

13. Creating iOS Trends ... 108

14. Creating Advanced Trends ... 110

15. iOS Alarms ... 114

16. CAS Alarms ... 117
16.1 Digital Alarms ... 117
16.2 Analog Alarms ... 117
16.3 Alarm Window ... 121
16.4 Alarm History ... 122

17. Data Logging ... 124
17.1 Data Logs ... 124
17.2 Continuous Data Logging ... 125
17.3 Triggered Data Logs ... 126
17.4 Tabular Views ... 126

18. Connections ... 128

19. User Access Levels ... 130
19.1 Access Levels ... 130
19.2 Specify User Accounts ... 130

20. Server-side Scripts ... 134
20.1 Introduction ... 134
20.2 Server-side Scripts folder ... 135
20.3 Scripts Toolbar ... 135
20.4 Server-side Scripts Folder Structure ... 135
20.5 Script Editor Functions ... 136
20.6 Event-driven Asynchronous Callbacks ... 137
20.7 Organizing Applications into Modules ... 138
20.8 Creating Reports ... 140
20.9 Scripting Functions ... 145
20.10 Debugging ... 165
20.11 Script Status ... 168
20.12 Ser2Net ... 170
20.13 Script Samples ... 171

21. Creating and Testing Components ... 172
21.1 Creating Components ... 172
21.2 Component Variables ... 174
21.3 Replacements ... 178
21.4 Testing Component ... 185
21.5 Entering Advanced Functions (Equations) ... 186

22. Devices ... 189

23. Communications 191
23.1 EtherNet/IP (ControLogix) 191
23.2 MicroLogix and SLC 193
23.3 Modbus 194
23.4 OPC UA 202
23.5 MELSEC-Q 204

24. Download/Upload from/to Device 208
24.1 Download to Device 208
24.2 Upload from Device 209

1. About myPROJECT Designer

myPROJECT Designer, part of the *mySCADA* bundle, is an integrated development environment used for configuring, developing and managing HMI/SCADA applications. In this manual you will find everything you need to create a full-featured SCADA (Supervisory Control and Data Acquisition) project visualization. With this tool you can create and manage *mySCADA* projects, configure connections with devices, enter tags, alarms, and trends. It also allows you to design advanced mimic graphics with specific animations, corresponding with PLC tag values.

A simple to use interface allows for easy manipulation of the project's configuration and data processing. The project data are stored in a single directory for easy backup and restoration.

myPROJECT Designer has an integrated GUI (Graphical User Interface) visualization editor for easy creation of professionally looking mimic graphics. The graphics are based on the Scalable Vector Graphic format, which means you will always have a sharp look of your controlled technology.

Key features

Free for personal and business use

Simple to use

GUI design in Scalable Vector Graphics (SVG)

Animations and effects based on tag values can be added to any shape or object

Support for background images (JPEG, GIF, PNG)

Ability to attach PDF-documents to the project

Ability to attach MP3-sounds to the project

Built-in script editor

Available for Mac OS X, Windows and Linux

Drawing possibilities

Shape tools: rectangles, circles, ellipses, paths, texts, images

Path tools: Bezier curves, conversion to a path, union, subtraction, intersection, merge

Group editing

Advanced text support

Images import (jpg, png)

Transformations: resize, rotate, skew, align, distribute

Properties manager

Resource manager: gradients, patterns and markers

Object viewer

Project management

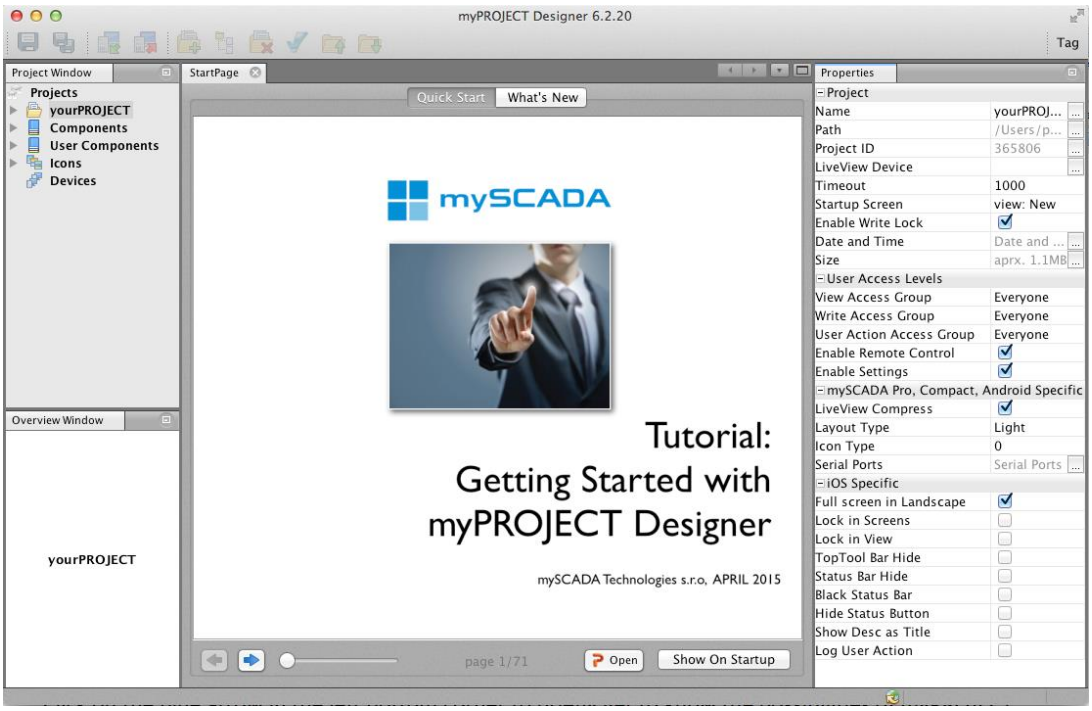
Key components of *mySCADA* projects are the visualization screens - views. This is where the schematic visualizations of controlled devices are displayed. There single objects or groups of objects can be created and defined especially for communication with connected PLCs. These specified objects could be then animated – changing visual appearance, based on the PLC tag values.

The graphic screens are internally represented as SVG files, defined project connections as configuration text files together with alarm states and trends also stored in separate text files. The text files can be edited directly in *myPROJECT Designer*, so you do not have to edit them individually. The scripts are also stored into external files and can be edited in your favorite text processor. Complete *mySCADA* projects consist of configuration texts, SVG files, scripts, documents and sounds, and are saved directly to your hard drive.

In the following sections you will learn how to create a project, add, configure or delete connections, define alarms and trends and other project settings.

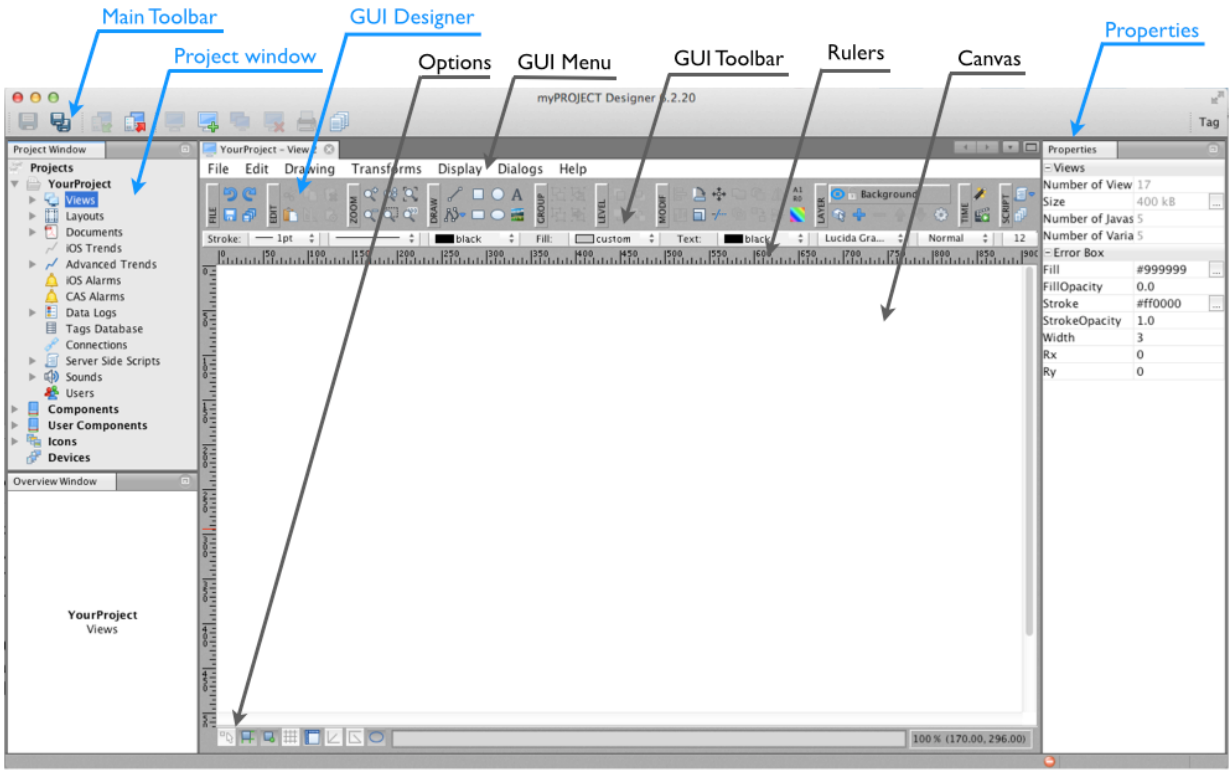
2. Start with myPROJECT Designer

The first screen you will see after opening the application is the *Start Page*.



Click on the blue arrow in the left bottom corner to briefly get to know the possibilities of *myPROJECT Designer*. The *Start Page* opens up always upon the start of the application - if you wish to prevent this, click on the **Show On Startup** button at the bottom. You can close the *Start Page* by clicking on the cross in the header tab. If you wish to display the *Start Page* again, go to the main menu, section *Help* and select *Start Page*.

Look at the picture below to briefly get to know the *myPROJECT Designer* interface:



Main menu

File – manipulation with project files and printing

Project – has the same structure as the Project Window

View – work with views

Documents – load, copy and deletion of attached PDF-documents

iOS Trends – opening up and closing trend for iOS Devices

Advanced Trends – opening up and closing trend

iOS Alarms – opening up and closing simple alarms for iOS Devices

CAS Alarms – opening up and closing complex alarms

Data logs – define data logging

Tag Database – manage your tags

Connections – connections to your PLCs

Server-side Scripts - managing scripting in *mySCADA*

Sound – load and deletion of MP3-files

Devices – Equipment with installed *mySCADA* application, accessible from editor

Library – work with Library

Window – direct access to different editor tabs and window control

Help – opens the help menu

Main toolbar

The basic toolbar consists of the following functions:

Save – saves your work

Save all files – saves all opened files

Upload from Device – downloads a project from the selected *mySCADA* device. Be aware that all files of your current project will be overwritten!

Download to Device – uploads a project to the selected *mySCADA* device. Be aware that all project data on the selected device will be overwritten!

New Project – creates a new project

Open Project – opens an existing project

Close Project – closes a selected project

Check Project – checks the selected project

Project Up – moves up the project in the list

Project Down – moves down the project in the list

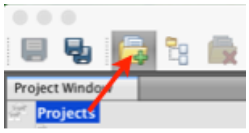
The main toolbar content may change, depending on a selected project element (i.e. project, view, sound, script, document or alarm).

The set of described icons is attributed to the project elements only; other possible sets are described in the chapters devoted to the relevant elements.

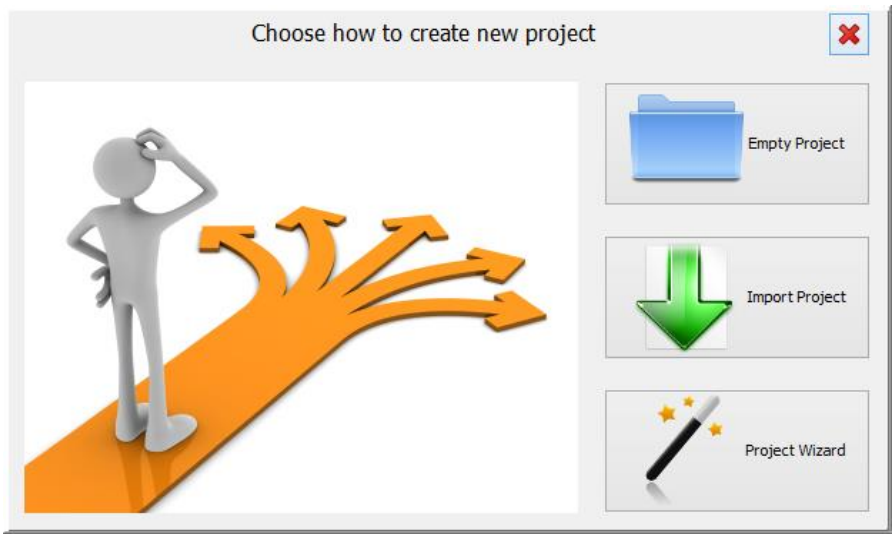
2.1 Creating Project

Start using *myPROJECT Designer* with creating a new project.

1) Click on the **New Project** icon in the main toolbar or use the command *New Project* from the main menu *Project-> Projects.*



2) A new dialog window will show up with 3 options:



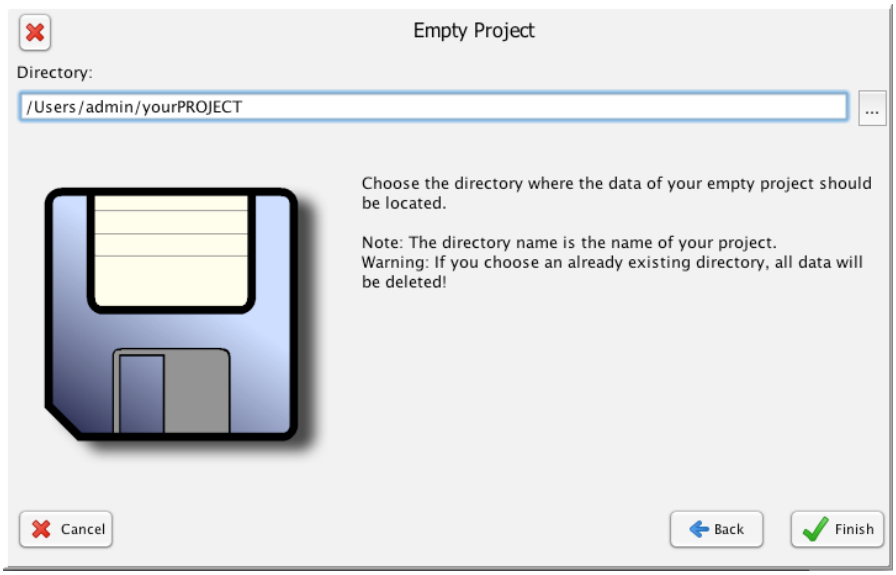
Empty Project

With this option you create a new empty project.

1) Choose the directory where your empty project will be located.

2) If the selected directory is not empty all its files will be deleted.

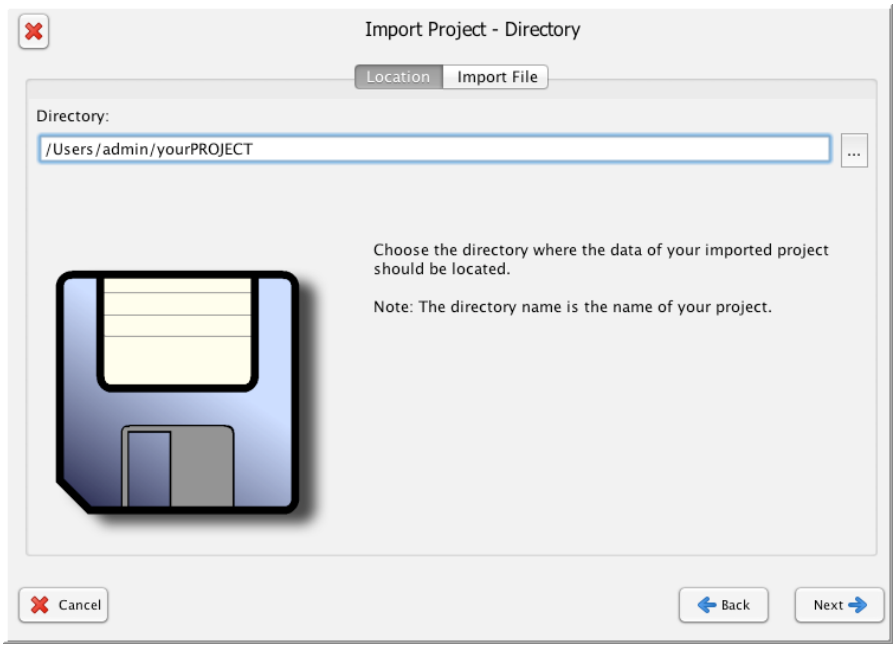
3) Click on the **Finish** button to create an empty project.



Import Project

This option imports another project from a MEP file (all exported projects of *mySCADA Designer* use this suffix).

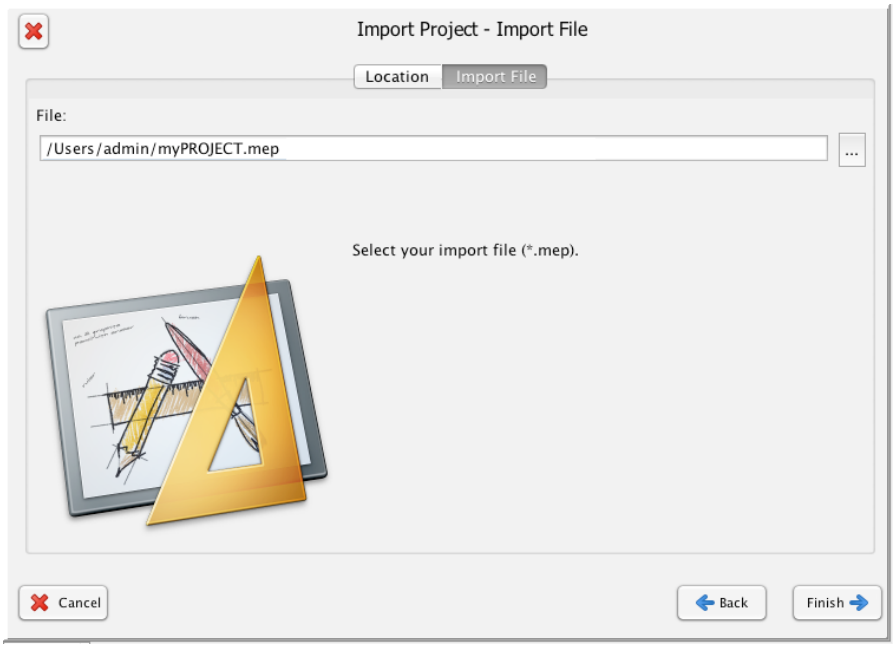
1) Choose the directory where data of an imported project should be located and click on **Next**.



2) Click on **Import File** (*.mep) and then on the **Finish** button - an imported project will be created into the selected directory.

3) If the directory is not empty all its files will be deleted.

4) Click on **Finish** to finalize importing.



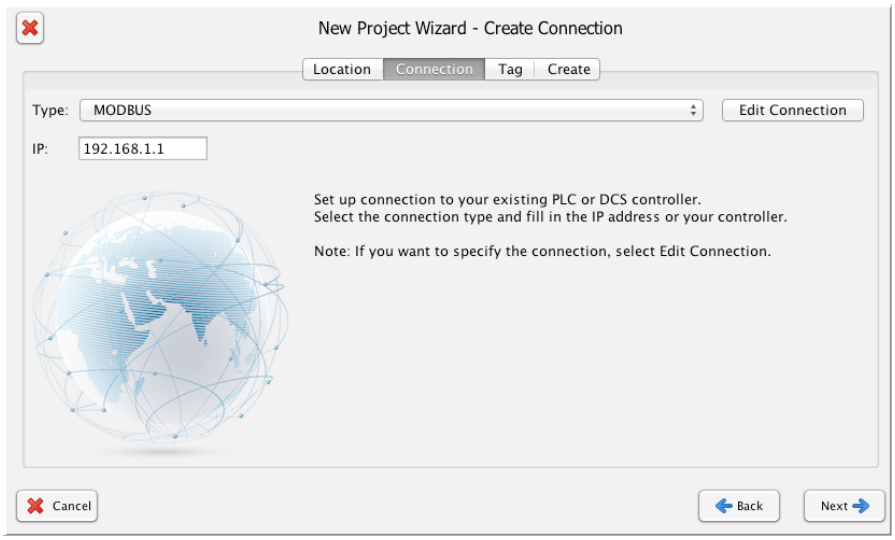
Project Wizard

This feature helps you create a functional base of your new project step-by-step. It will create a connection to the PLC, set up a simple screen with animations and pre-configure alarms and data logging.

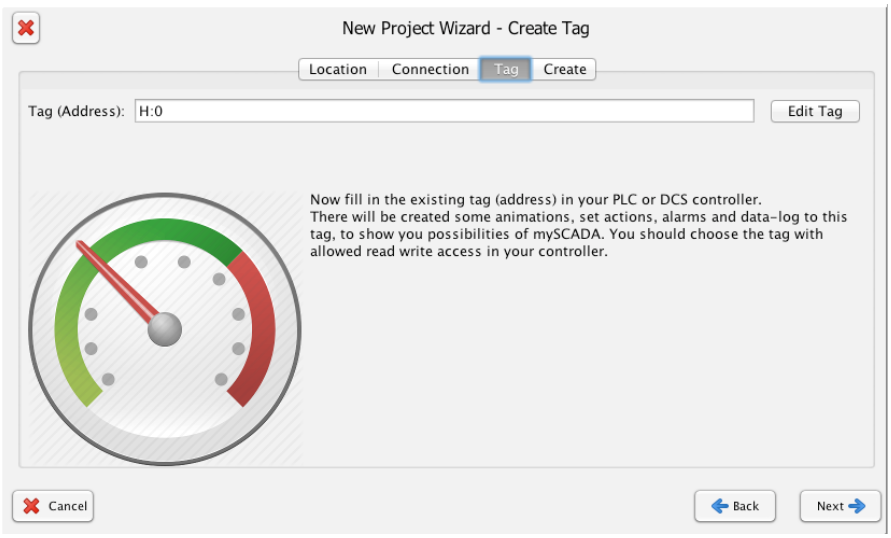
1) Select the directory where you want locate your new project and then click on **Next**.

   *Note: You have to click on **Clear Directory** first if the selected directory is not empty.*

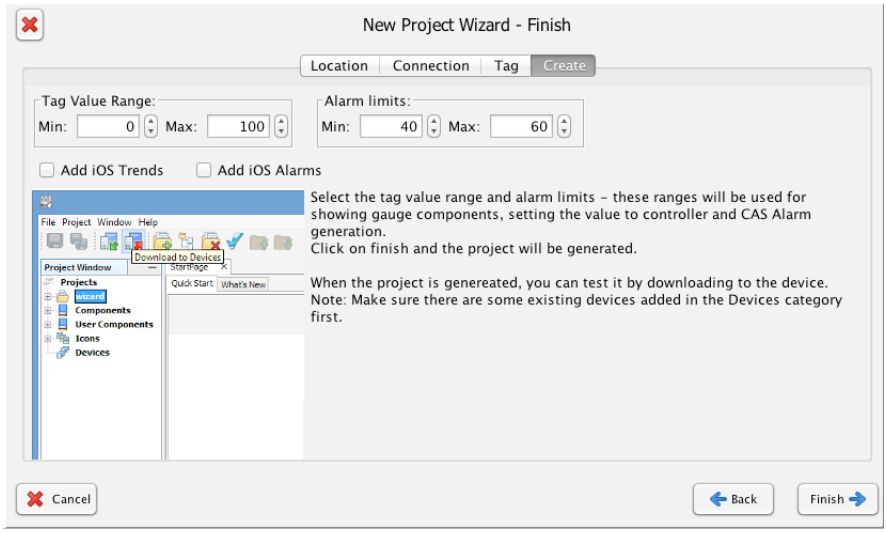
2) Set the connection to your device and click on **Next**.



3) Set the Tag (Address) and click on **Next**.



4) You can now set the Tag Value Range, Alarm limits, add iOS Trends and iOS Alarms.



5) Click on **Finish** to close the *Project Wizard*.

2.2 Managing Workspace

All *myPROJECT Designer* windows are organized into panes. You can move the windows arbitrarily as the designer remembers position of both automatically and manually closed windows until the next time opening.

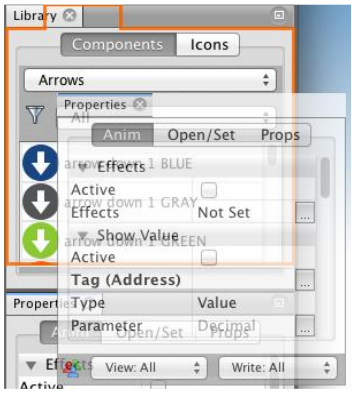
Each window can be dragged away from the workspace and stay undocked until you dock them back by using *Alt+Shift+D* key combination. You may resize the windows, as well.

If you wish to put all the windows into their default state, use the *Reset Windows* command from the menu *Window –> Reset Windows.*

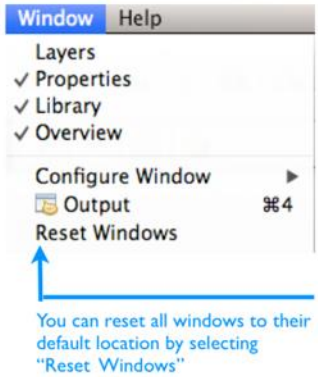
Moving windows

Click on the window header and drag it to the desired position,

Note: The red preview box indicates where the window will appear once you drop it.



Automatically appearing windows



Some windows appear only when you are performing a task to which they are related. For example, the *Library* window appears only when you are designing a screen view. In the menu *Window* you can select which windows will be appearing automatically.

Important shortcuts

Shift + Escape	Maximizes a currently opened window
Ctrl + Shift + W	Closes all open documents in the *Source Editor*.
Alt + Shift + D	Pins a detached window to the Main window.

Project Window

It displays a tree structure of opened projects, available libraries and the list of connected devices.

Project (Your project name)

Views – all HMI screens for given project

Layouts – all defined graphical layouts for given project

Layout Views – views (HMI screens) that can be used in the layout

Documents – PDF-documents, attached to the project

iOS Trends – configured trends used in iOS devices

Advanced Trends – advanced trend, use with data-logs to retrieve online and historical data

iOS Alarms – configured alarms for iOS devices

CAS Alarms – here you can configure Complex alarm system

Data Logs – configured data logs

Tags Database – managing your tags

Connections – PLC connections of a given project

Server-side Scripts – user-defined scripts

Sounds – associated sounds in MP3 files

Users – defined project users

Components - library of dynamic HMI/GUI components sorted into categories

User Components - library of editable dynamic HMI/GUI components sorted into categories

Icons - library of icons sorted into categories for HMI/GUI development

Devices – list of available devices to load a project from/to. Here you can find all *iPods, iPhones, iPad Touches, Android devices, Raspberry PI devices, mySCADA Boxes*, and *mySCADA Pro* and Server PCs you can connect to. You can also specify the device manually to reach remote network devices. Note that it may take a few minutes to find all your devices available in the local network.

Overview window

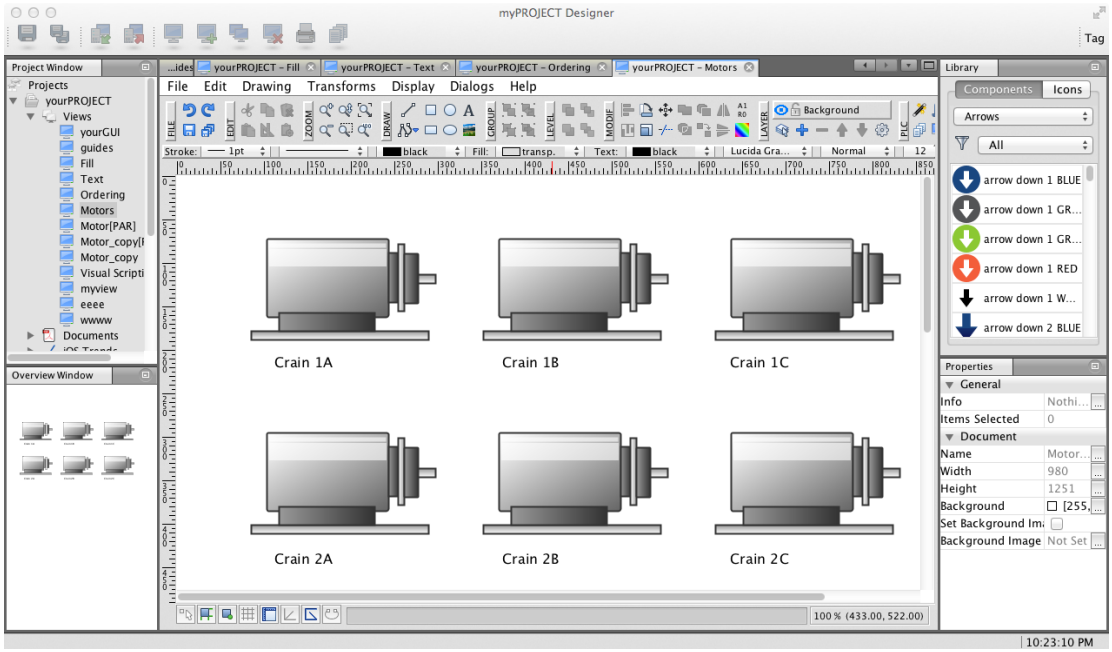
This window shows a preview of selected views otherwise it is blank.

Properties window

This window creates access to object and project properties. The properties of specific objects are described in the relevant chapters.

Main window

This window is your main working area where the content of all project elements opens up. It gives you the ability to draw graphic visualizations, change connection details and alter the alarm or trend details. The scripting editor also uses this window.



This button expands the list of all opened projects and select one of them. The arrow next to the project name will aim to the currently displayed project.

This button maximizes the main window by docking the rest of all opened windows.

These buttons are used for switching different opened tabs, in case the tabs do not fit into the main window.

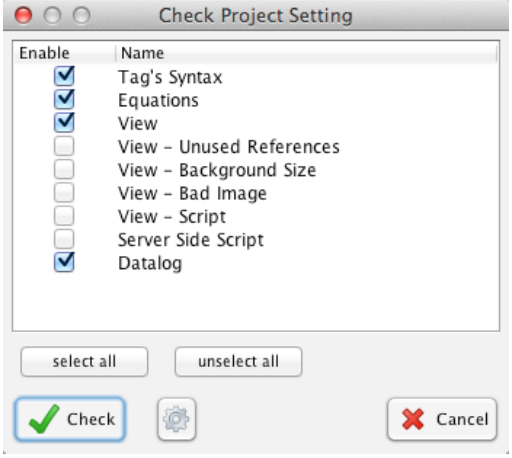
2.3 Checking Project

If you want to make sure your *mySCADA* applications run properly you have to comply with certain rules when creating a project. For this purpose you can use the *Check Project* function which checks if your project complies with these rules.

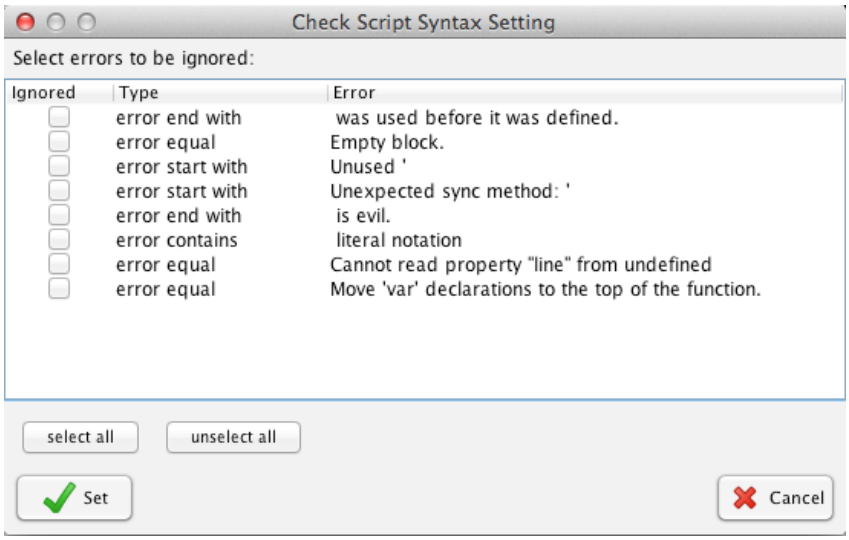
Use the command *Check Project* from *Project->Projects* or click on the **Check Project** icon in the main toolbar.

Example:

1) Select parts of your project that require checking:

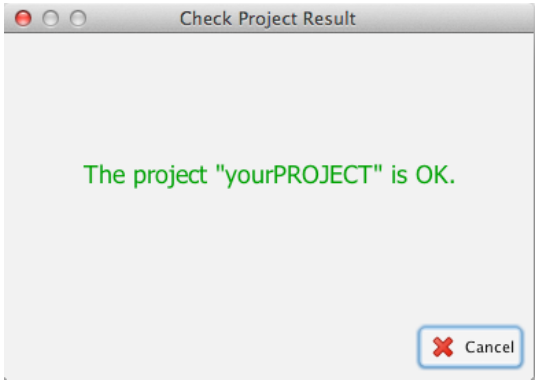


2) The **Syntax Settings** button sets which errors will be ignored:

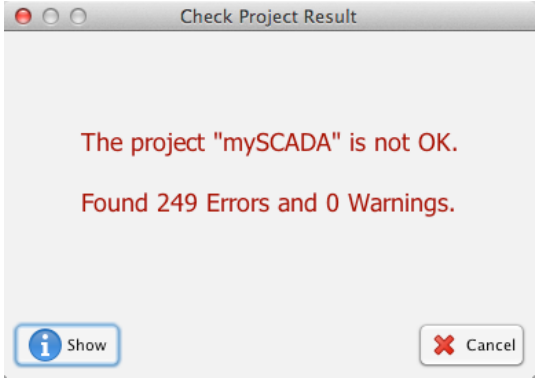


3) The **Check** button starts checking the project syntax.

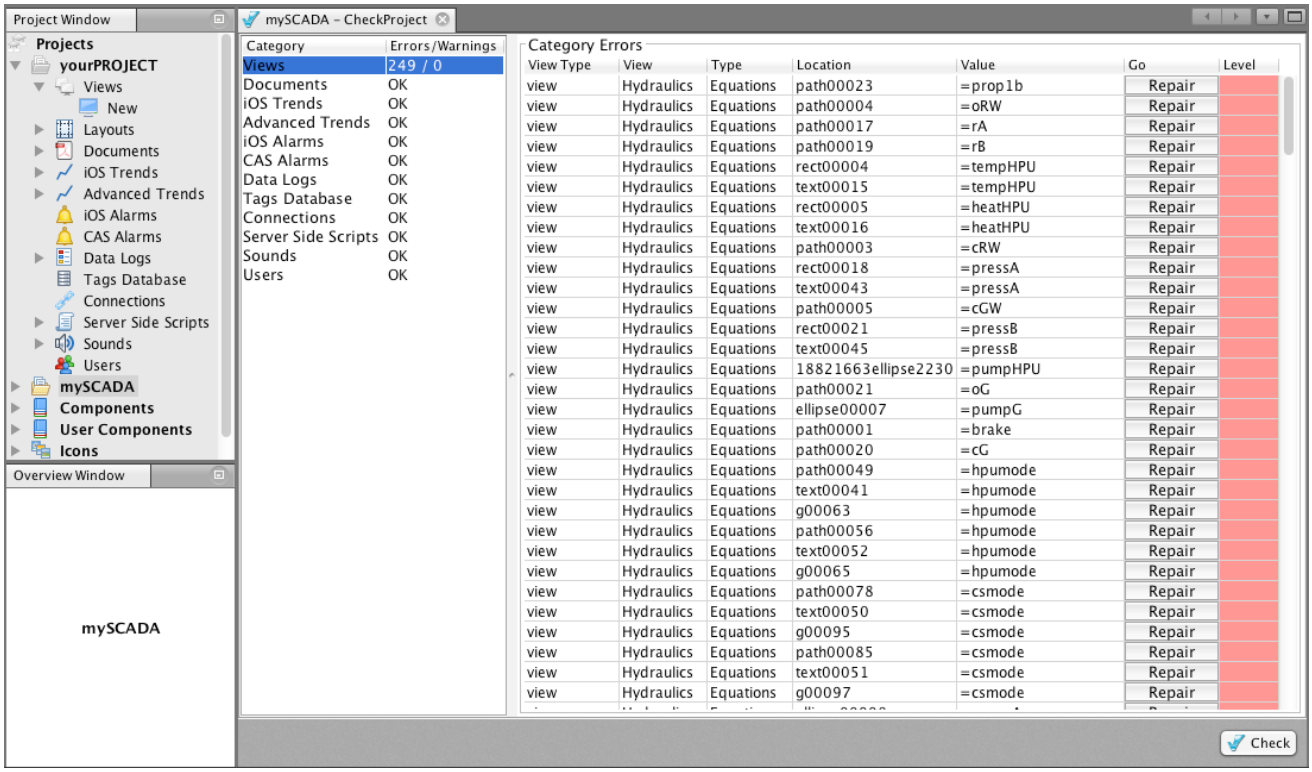
If the check is successful a confirming dialog will show up:



If the check finds errors an error dialog will show up:



4) Click on the **Show** button to display all errors.



*Note: All errors are divided into categories on the left; a selected error category will be displayed on the right side. If you wish to correct all the found errors click on **Repair**.*

3. GUI/HMI Editor

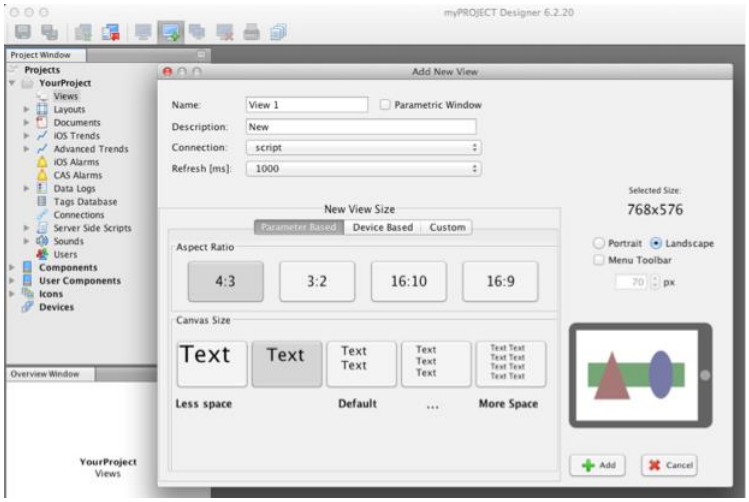
The GUI editor is a drawing interface based on Scalable Vector Graphics of XML technology.

3.1 Creating Views

To create views for the process visualization open the ***Views*** folder in the *Project Window*, the main toolbar will show you these options: *Open View, Add View, Duplicate View, Delete View, Print View and Show Used Tags*.

1) Click on the **Add View** icon located above the main screen or use the command from the main menu *Project->Views->New View.*

   A new dialog window opens to let you set the visualization parameters:



Name: view name

Description: additional comment for visualization

Parametric window: option to create a parametric view - use parametric views when you need more similar views different only in a data source; such source is specified by the received index when a parametric view is prompted (see *Sets – Open command* and *Parametric window*).

Connection: list of PLC connections, defined in the current project. This is a default connection - if you enter the tag and do not specify the alias, this connection will be assumed as *default*.

Refresh: refresh rate of the visualization, time in milliseconds

Page orientation:

- *Portrait* predefined vertical display
- *Landscape* predefined horizontal display

Note: If your device uses a pop-up menu tool bar (iPhone, Smartphones, etc..) you can set up its width in pixels which will be automatically subtracted from the final screen size.

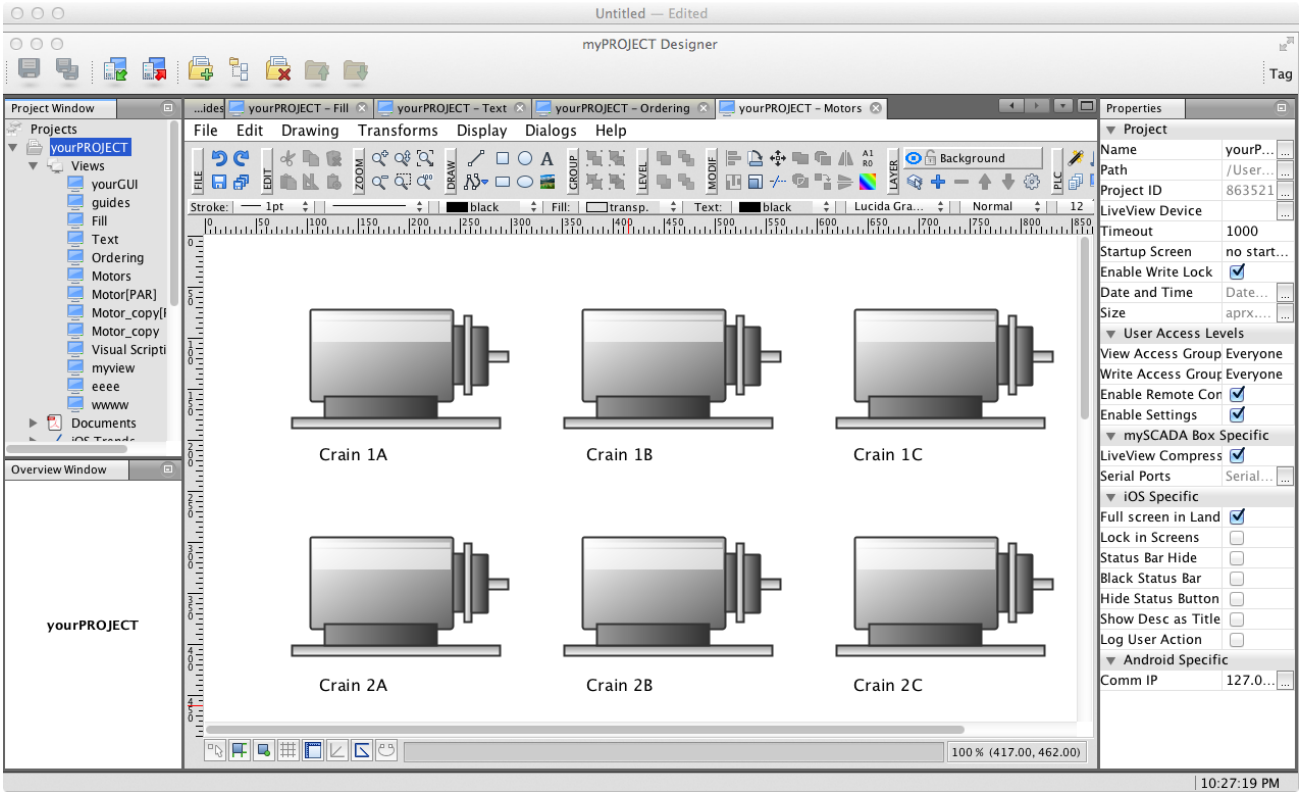
New view size:

- Parameter Based – here you can set the aspect ratio and the canvas size
- Device Based – variation of pre-defined settings for many types of devices
- Custom – allows you to define your own view settings

Note: You can change the screen dimensions any time later if you right-click on the canvas and select 'Resize canvas' from the pop-up menu.

2) Click on **Add** to create a view with selected parameters or press **Cancel** to exit without creating or saving the view.

In the *Project Window* you can see all the views listed in the ***Views*** folder. In the *Properties window* you can see the parameters of a currently selected view. To open an existing view you can click on the **Open View** icon or select *Open View* from the *Project->Views* menu. You may also use the other options such as: *copy, delete, import a view etc.*



Menu

File

This menu allows you to manipulate with an opened view:

Export – exports the current view into other formats: *JPG, PNG, BMP and PDF*

Print – prints the current file (shortcut **Ctrl+P**).

Check – performs the check of your view, use it if you experience a strange behavior - it will check out possible IDs duplicity, misplaced objects out of the canvas etc.

Close – closes the current file (shortcut **Ctrl+W**).

Edit

This menu provides basic operations with objects, like: *undo, redo, copy, cut, delete, group, lock, select, etc.*

Undo – erases the last change made in the program (shortcut **Ctrl+Z**)

Redo – reverts the change of the last undo (shortcut **Ctrl+Y**)

Copy – creates a copy of a selected object or area into the clipboard (shortcut **Ctrl+C**)

Paste – pastes the object or area from the clipboard (shortcut **Ctrl+V**)

Paste on same location – pastes the copied or cut object to the same location as the source.

Cut – cuts a selected object and places it in the clipboard (shortcut **Ctrl+X**)

Delete – deletes a selected object (shortcut **Ctrl +D**)

Paste dimensions – provides transfer of certain parameters of a selected object:

Paste size – modifies selected objects, setting to the size of the file being copied

Paste width – modifies selected objects, setting to the size of the file being copied

Paste height – modifies selected objects, setting to the size of the file being copied

Paste size separately – resizes selected objects

Paste width separately - changes the width of selected objects

Paste height separately – changes the height of selected objects

Group – groups multiple objects together (shortcut **Ctrl+G**)

Ungroup – splits the grouped objects

Enter group – allows access to individual objects inside an object group (shortcut **Ctrl+E**)

Exit group – returns to a normal selection mode, i.e. the whole group will be selected at once (shortcut **Ctrl+Shift+E**)

Select all – selects all objects on the canvas (shortcut **Ctrl+A**)

Deselect all – deselects all objects on the canvas

Drawing

In this menu you can create figures, lines and curves, add pictures, texts, etc.

It contains the same features as the main *toolbar*, for detailed information see the *toolbar* section of this chapter.

Transforms

In this menu you can align, center, distribute, order, flip, rotate objects and resize the canvas.

Display

In this menu you can enable grid and rulers, different zoom options are available too.

Dialogs

This menu includes two features: *Graphics Object Inspector* and *Memory Monitor*.

Graphics Object Inspector - allows tree-like access to any element of a currently opened view

Memory Monitor - shows an amount of memory being used by *myPROJECT Designer*

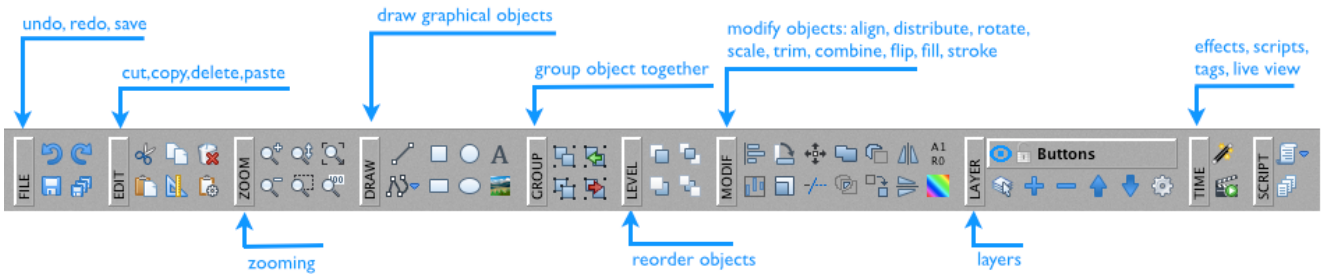
Help

This menu offers additional information about *myPROJECT Designer*.

The most frequently used functions from these menus are also available in the main toolbar or the right-click menu.

GUI toolbar

In the upper part of the view editor window there is a secondary GUI toolbar where can find all the necessary functions for designing and animating the views.



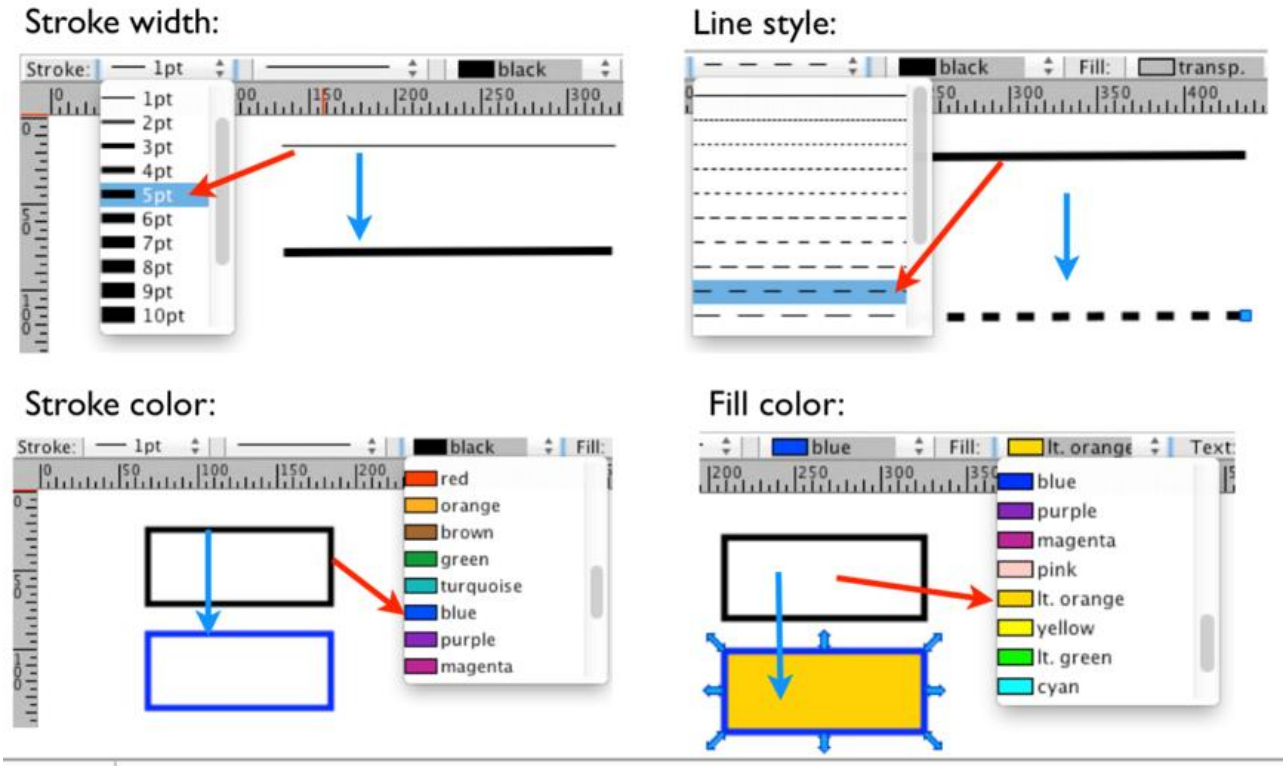
Properties bar

It is located right below the GUI toolbar. It allows you to view, change the properties of selected objects or set default drawing properties such as: *fill and stroke color; line type and line thickness; font family, type and size.*

If there is no object selected, you can specify the default properties for new objects to be created. If you click on an existing object you will see its current properties that you can modify.

In the following example all the new objects will have transparent fill and black solid line stroke:

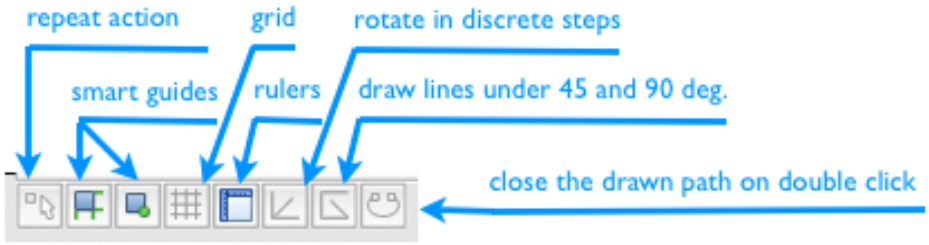
With the *Properties bar* you can easily change properties of multiple objects at once.



Working with the Properties bar: change stroke width, stroke style and color and fill color. The properties bar lets you change also the font family and size.

Options bar

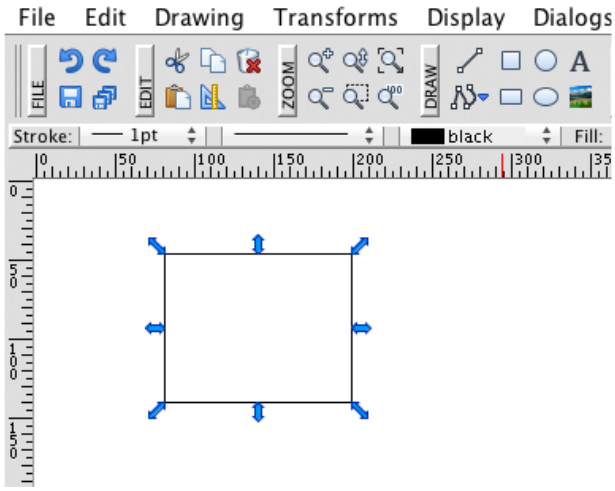
This bar is located in the left lower part of the *Main window*.



Here you can activate the *repeat action mode*; turn on/off the *smart guidelines*, *snap-to-points* and *square mode* function; enable the *grid*; set the *discrete rotation step*; force horizontal/vertical line draw; and automatically *close the drawn paths*.

3.2 Selecting Objects

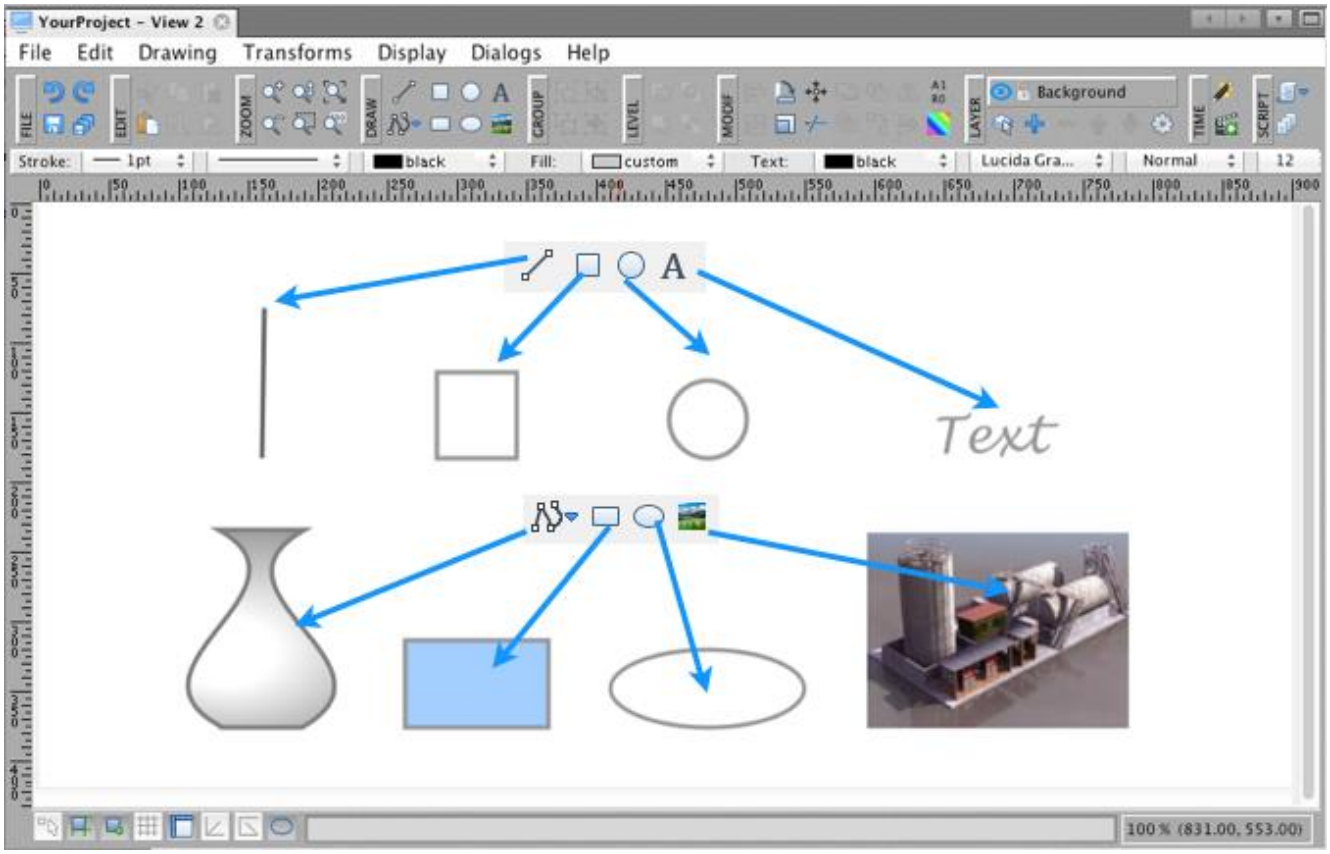
Click on the canvas and drag the mouse to select object(s). If the objects are selected correctly the selection border shows up and the object properties will be displayed in the *Properties window*. To add objects to your current selection press-and-hold the **Shift** key and click on more objects.



Note: If you drag the mouse from the left to the right, only the objects fully enclosed in the selection window will be selected. If you drag the mouse from the right to the left all the objects that interact with the selection window will be selected.

3.3 Drawing Primitives

In the DRAW section of the GUI toolbar you can choose a type of object you want to draw: *line, poly-line, square, rectangle, circle, ellipse, text* or load a picture from file.



Rectangles and Squares

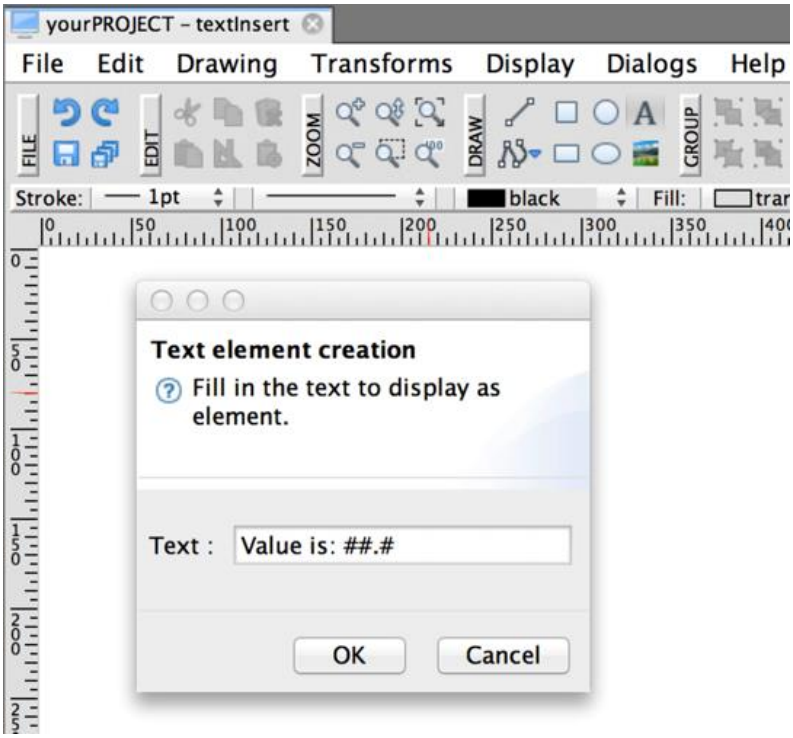
To create a rectangles or squares, click on the **Create Rectangle** icon in the GUI toolbar (you can also find it in the menu under *Drawing*). Click on the canvas, hold and drag the mouse to get the desired shape and size.

Ellipses and Circles

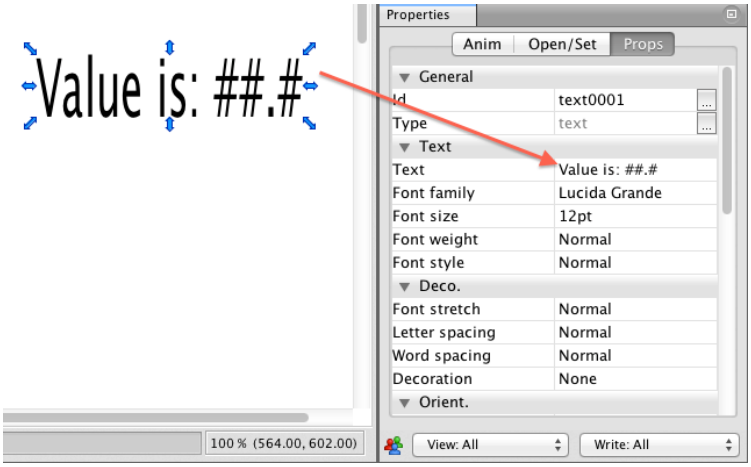
To create ellipses or circles, click on the **Create Ellipse** icon in the GUI toolbar (you can also find it in the menu under *Drawing*). Click on the canvas, hold and drag the mouse to get a desired size and shape of the object.

3.4 Creating Text Elements

This feature allows you to add text on the drawing canvas. Click on the **Text** tool icon in the main toolbar and then click on the canvas to the desired point. This will open a new window for the text input.



Type in the text to be displayed and click on **OK**. The text is now placed on the canvas and can be moved, resized, rotated, etc., like any other object.



Note: The text in the element can be changed any time later - select Properties on the right side of the window and look at the cell Text.

3.5 Inserting Images

With this function you can insert images in *PNG, JPG, JPEG* format.

Click on the **Insert Image** icon in the main toolbar, then click on the canvas and drag the mouse to set the picture size. This will open a new window for the picture selection. Navigate to the picture you would like to add and click on **Open**.

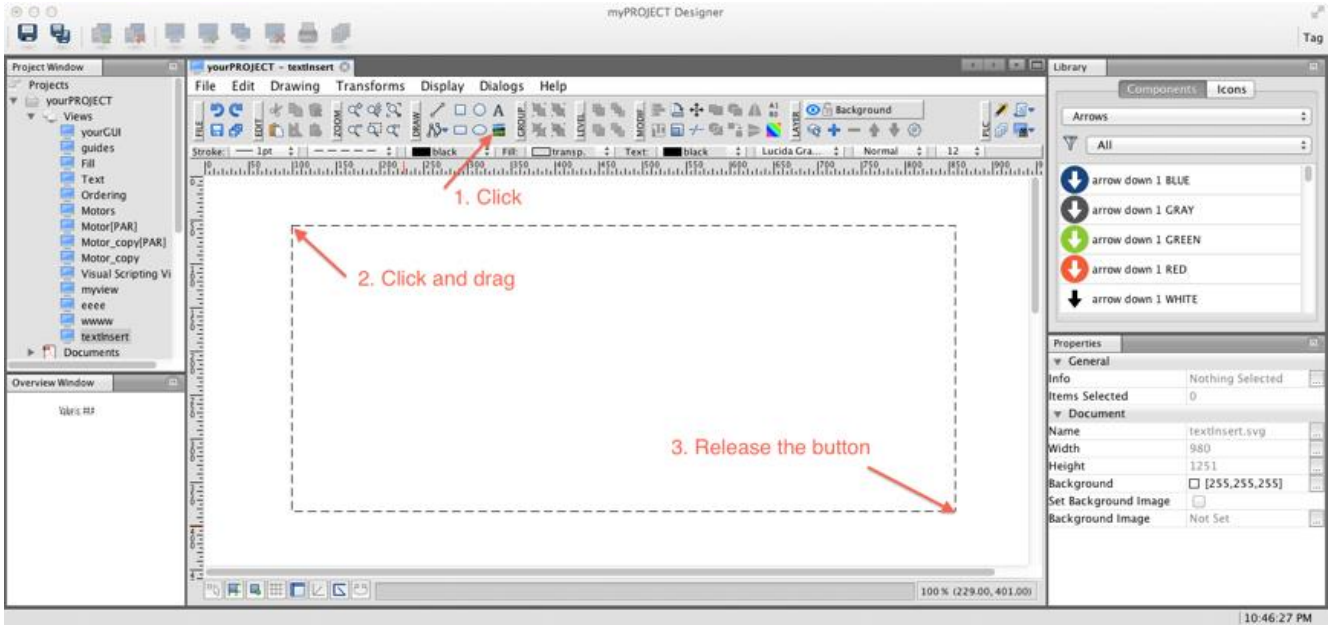


Figure 1

First, click on the picture icon in the toolbar then click and drag the picture to the desired area and release the mouse button.

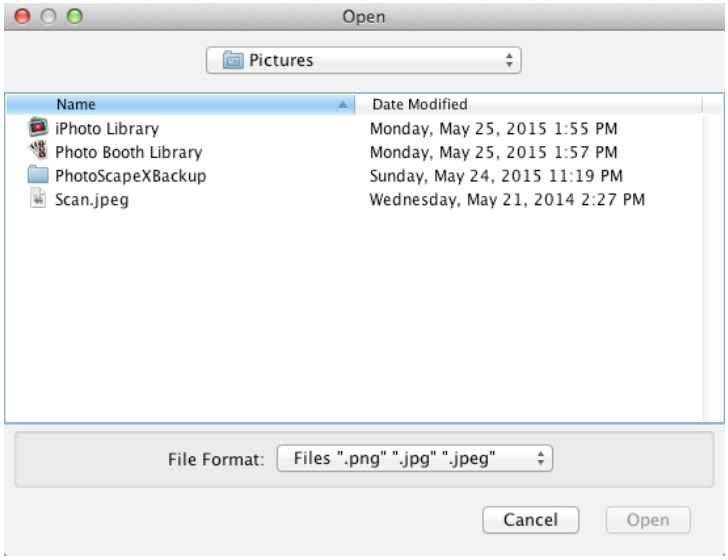


Figure 2

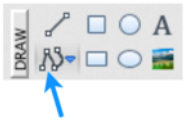
Select a picture to import

Figure 3

The picture is now on the canvas and can be moved, resized, rotated, etc. like any other object.

Note: You can import images with their original size by a single-click on the canvas.

3.6 Poly-lines



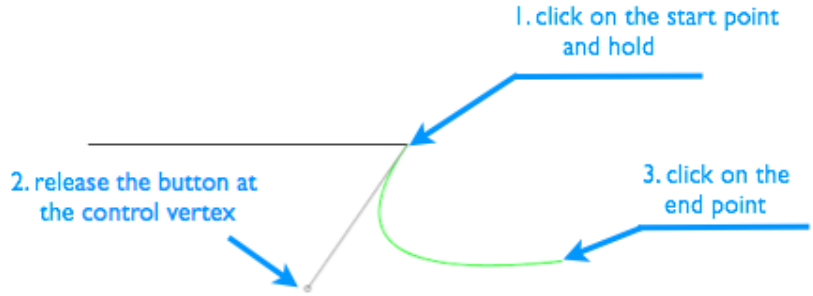
With this function you can create independent and continuous lines, arcs, splines or any of their combinations.

Line

Click to set the start point, move the mouse to the end point of the line then click again.

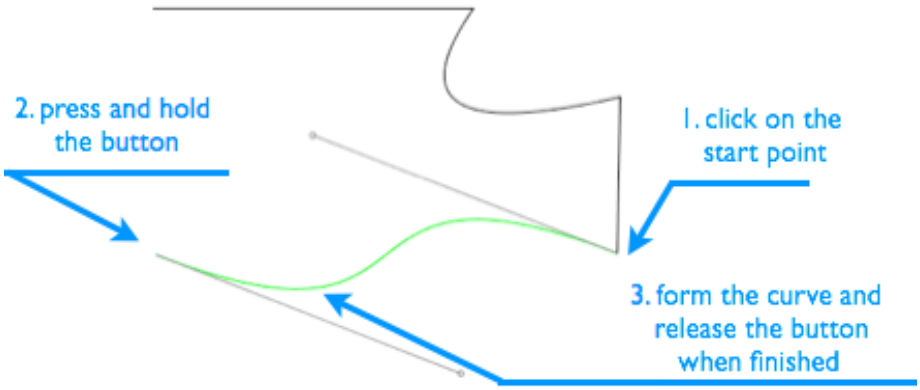
Arc

Click and keep the mouse button pressed from the start point, move the mouse and release the button at the control vertex point, then click again to set the end point.

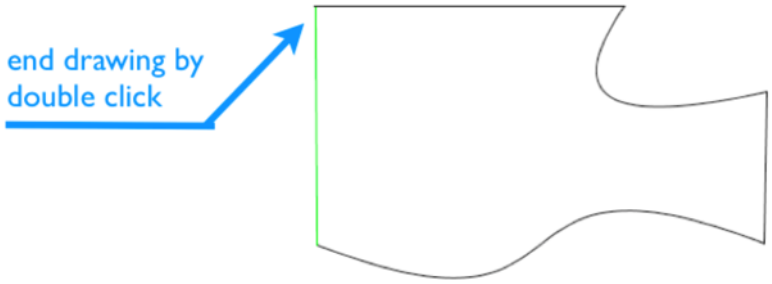


Spline

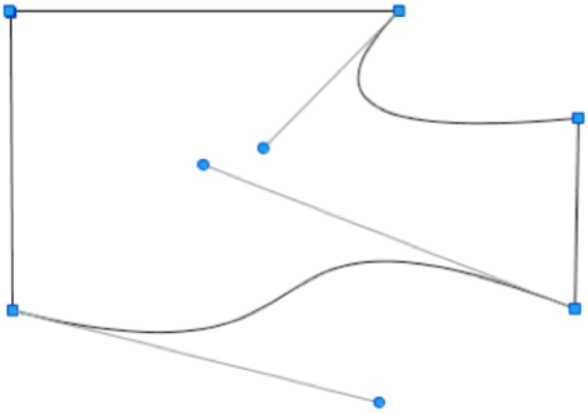
Click on the start point and move the mouse to the endpoint, then click again and keep the mouse button pressed to form the curve, release the mouse button to finish drawing.



Double-click or press the ESC key to finish poly-line drawing.

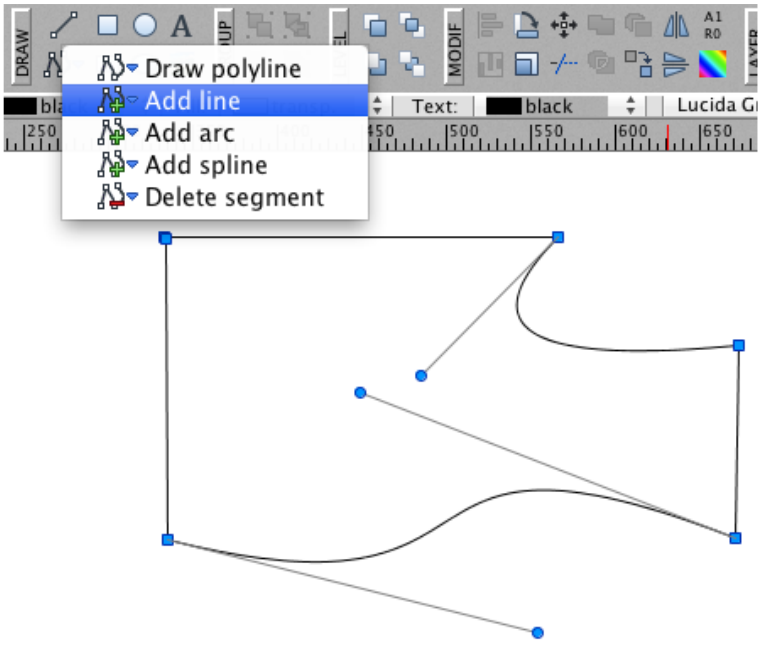


When you finish object drawing you will see all control, end and vertex points that you can modify.



Segments editing

You can add, remove or delete segments from a created spline. Right-click on the **Create Spline / Bezier Curves** icon to display the *Options menu*.



Add segment

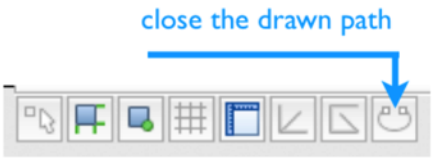
Select a segment you would like to add from the options menu. Click on the control point of the poly-line where you want to add the new segment.

Delete segment

Select *Delete segment* from the options menu. Click on the end control point of the segment you want to delete

Close path

If this option is active all the poly-lines being drawn will close-up automatically.



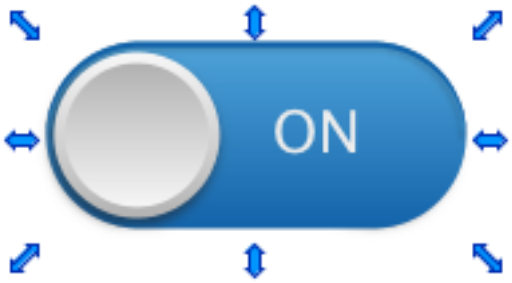
3.7 Moving Objects

Select object(s) you want to move. Move the mouse inside the selection border and drag the object to the desired position.

3.8 Resizing Objects

You can resize object(s) by dragging the selection border.

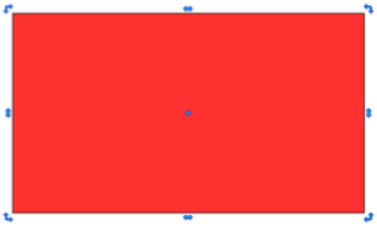
1) Select an object you want to resize – you will see the blue arrows selection border.



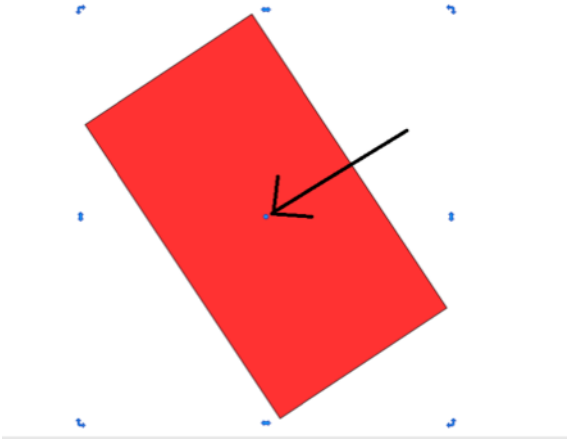
2) Drag the arrows to change size of the object.

3.9 Rotating Objects

1) Double-click on an object you want to rotate, the blue arrows will appear around the selected object.



2) Drag the corner arrows until you get the desired rotation angle.

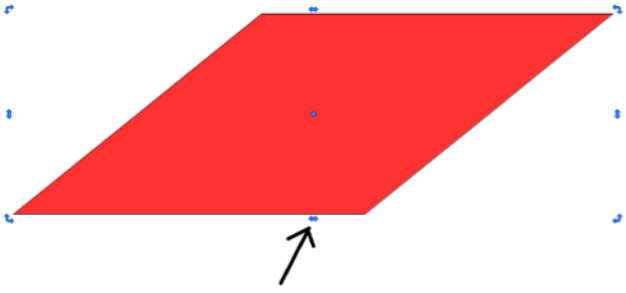


Note: You can also change the rotation center of the selected object.

3.10 Skewing Objects

1) Double-click on an object you want to skew

2) Drag the black arrow in the middle to get the desired skew.

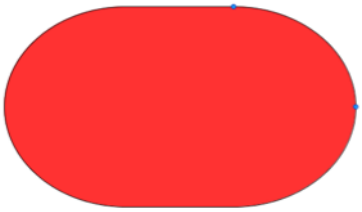


3.11 Corner Filleti

The rectangles and squares can be corner filleted.

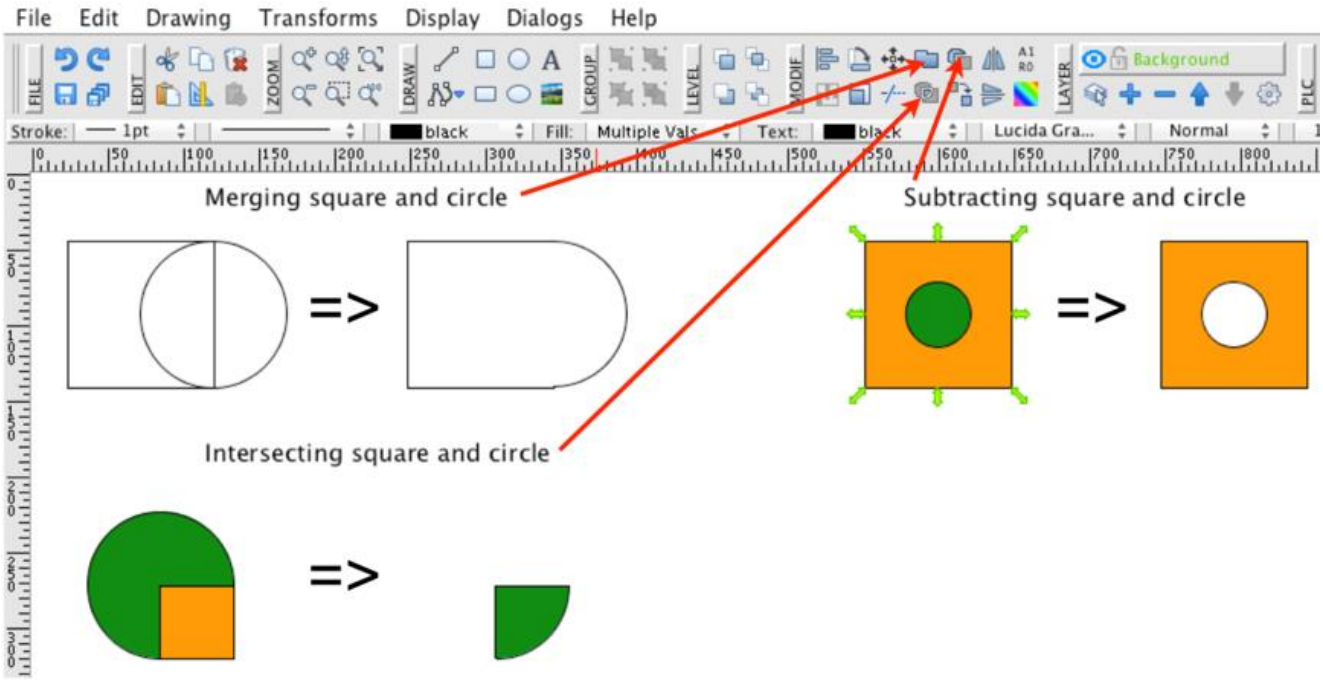
1) Triple-click on an object to show the blue point in its corner.

2) Click on the blue corner point, hold and drag the mouse to get the desired shape.



3.12 Combining Objects

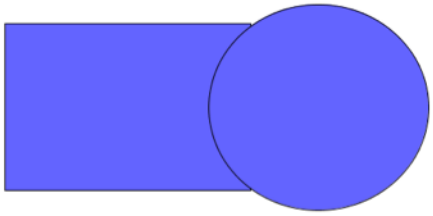
You can combine multiple simple objects covering each other and together with *merge, subtract, intersect* and *turn-to-path functions* you can create one complex object of arbitrary shape and color.



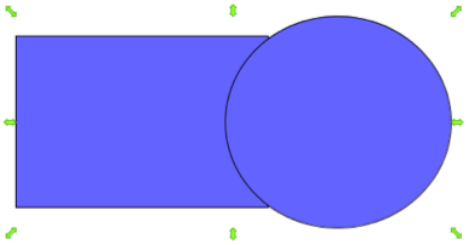
Merge

This feature can merge two objects into one including the contours:

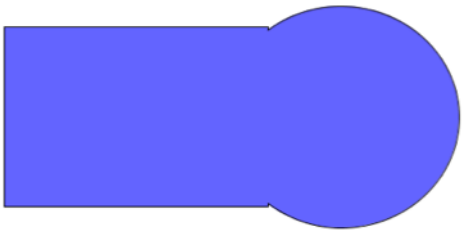
1) Create objects you want to compose the final object from.



2) Select both objects by mouse.



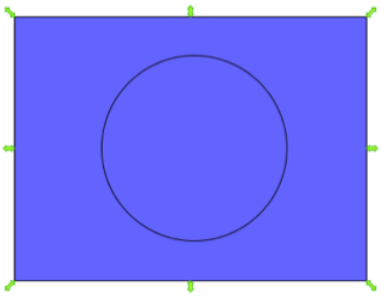
3) Go to the menu *Drawing –> Path operations* and select *Merge selected objects together* - the two objects will merge into one.



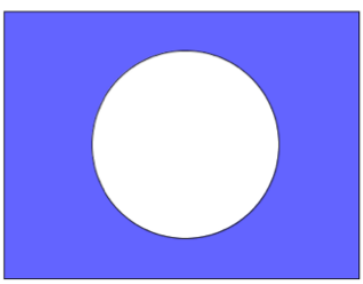
Subtract

This feature is opposite to the *Merge* function. You can subtract portions of a bigger object to create a new object of the desired shape.

1) Select objects and make sure the outline is in the desired shape of the final object. Then select the inner object that you want to remove.



2) Go to the menu *Drawing –> Path operations* and select *Subtract objects*.

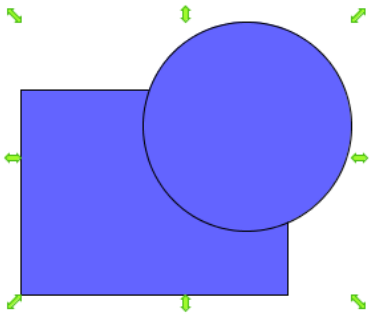


Now you have got a new cutting that can be formed – use the same steps as described earlier for objects shaping.

Intersect

This feature creates new objects out of an overlapped area of other two different objects.

1) Arrange two arbitrary objects so their overlapping pieces make a desired shape, then select the objects.



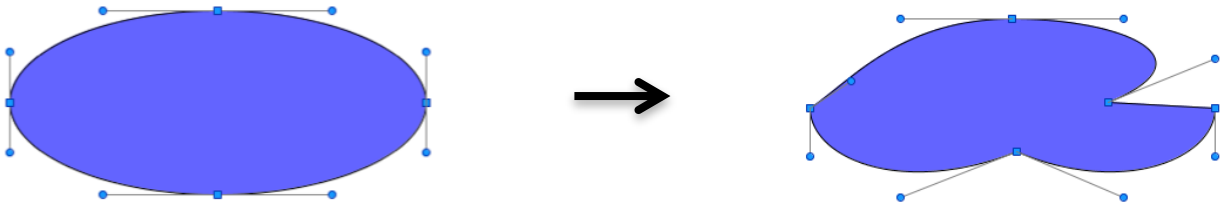
2) Go into the menu *Drawing –>Path operations* and select *Intersect.*

Now you have a new cutting ready to be formed into a desired shape by clicking and dragging the surrounding dots.

Turn-to-Path

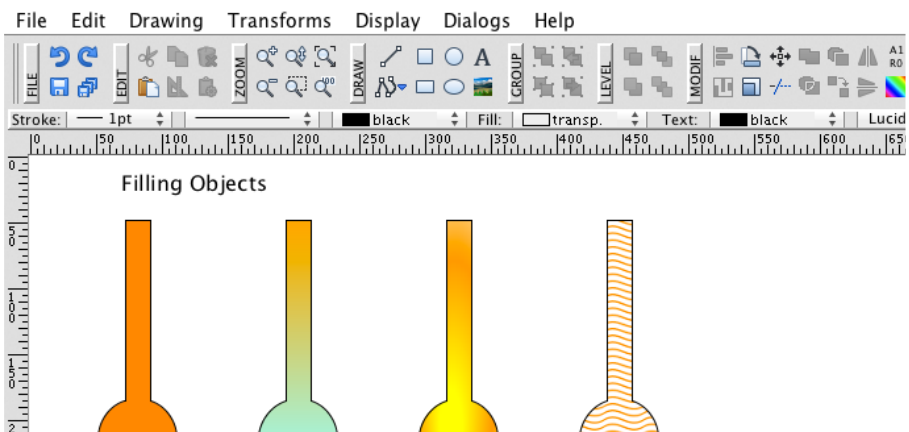
This feature allows objects deforming. Go to *Drawing->Path operations->Convert Selected Objects to General Path Objects*. You will see a series of points appear around the object.

Simply drag the surrounding points to achieve the desired shape.



3.13 Fill and Stroke

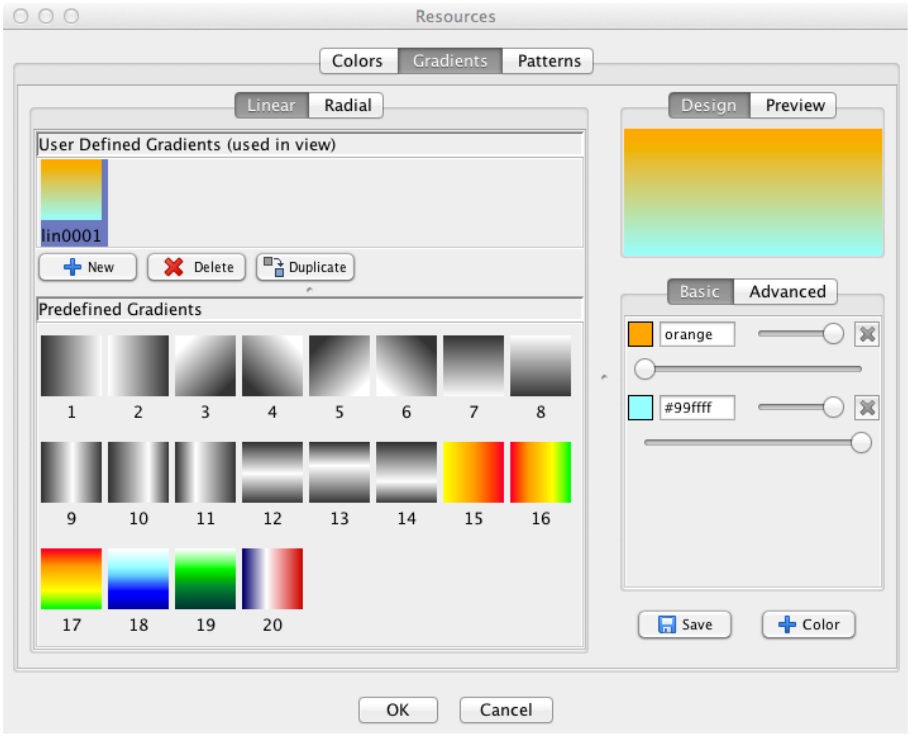
Graphic objects can be filled (or stroked) with a solid color, linear gradient, radial gradient or complex pattern.



Change a fill or stroke:

1) Select object(s) whose color you want to change.

2) Click on the **Color, Gradients and Patterns** button. 



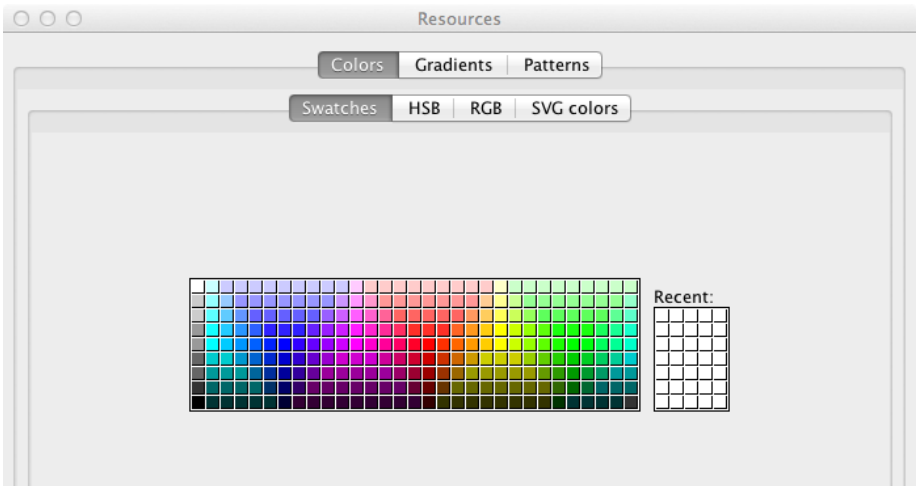
3) Choose a solid color, gradient or pattern.

4) Confirm the changes clicking on the **OK** button.

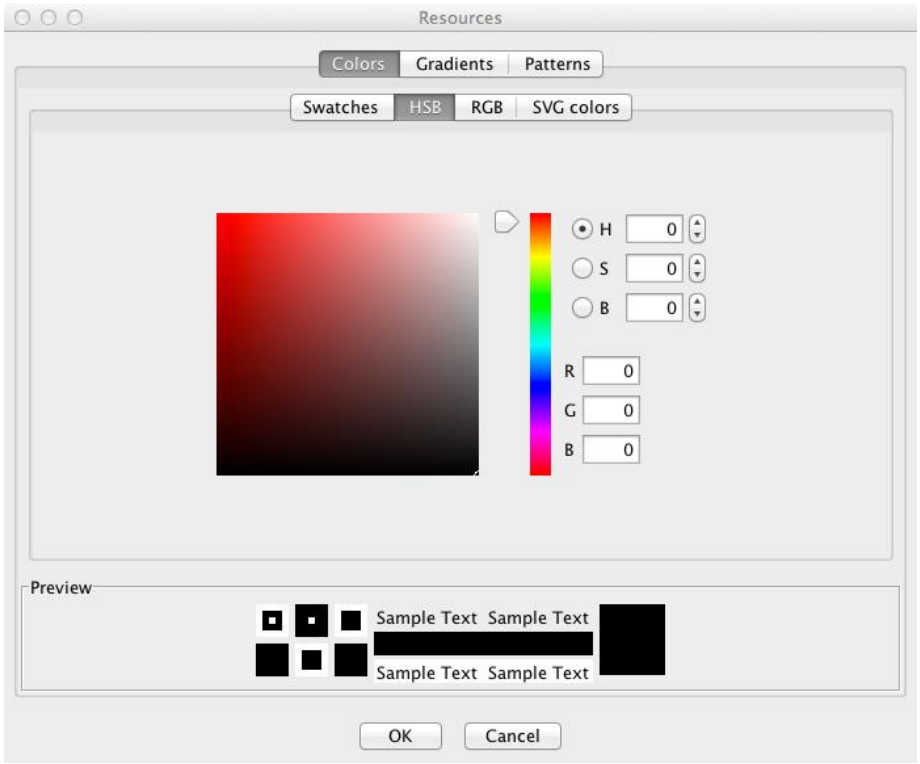
Solid color

Navigate to the **Colors** tab in the **Resources** tab; there are a few options how to create a solid color:

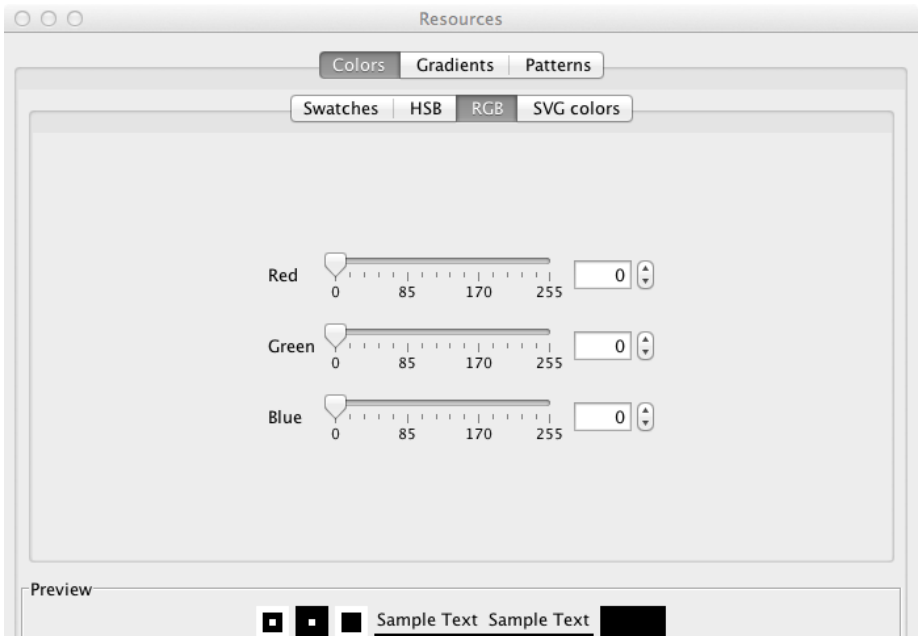
- **Swatches** - select the color from the pre-defined color palette



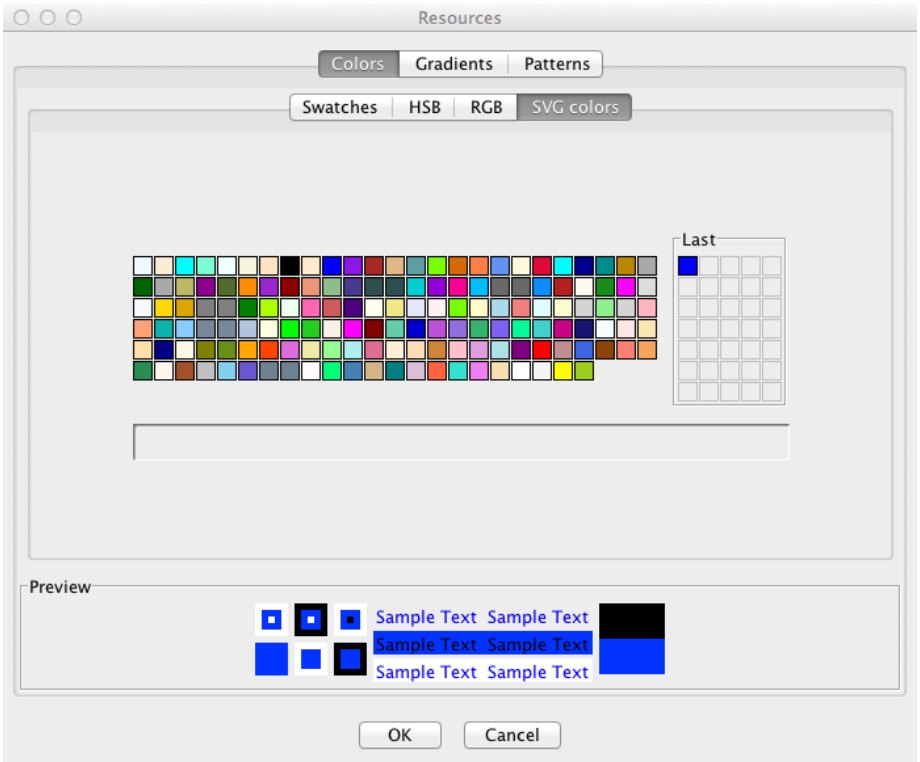
- **HSB** - freely mix your desired color; you can determine the color by HSB or RGB values



- **RGB** - use this pane if you know the color code in Red Green Blue values



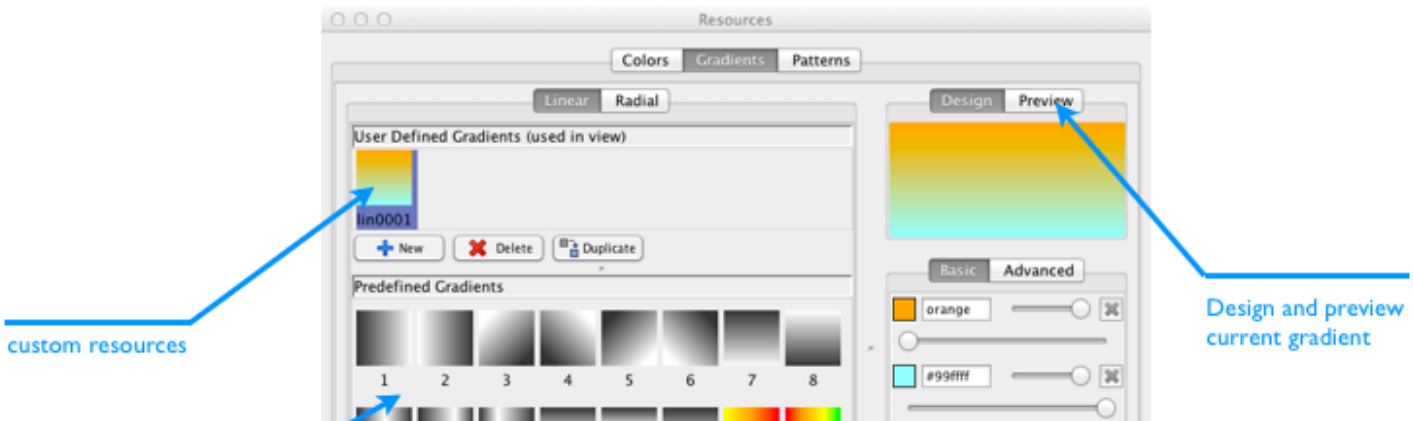
- **SVG Colors** - select your color from the pre-defined SVG standard colors palette



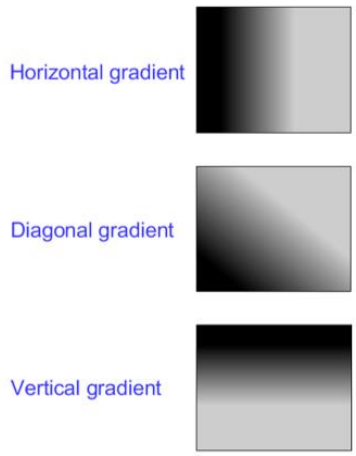
Gradients

Using the gradients enable you to create your custom color blends and give your objects a plastic look. You can create smooth color gradations over one or more objects and save them for later use on other objects.

The **Resources** tab contains pre-defined linear and radial gradients and patterns. You may create new gradients and patterns or modify the existing ones.

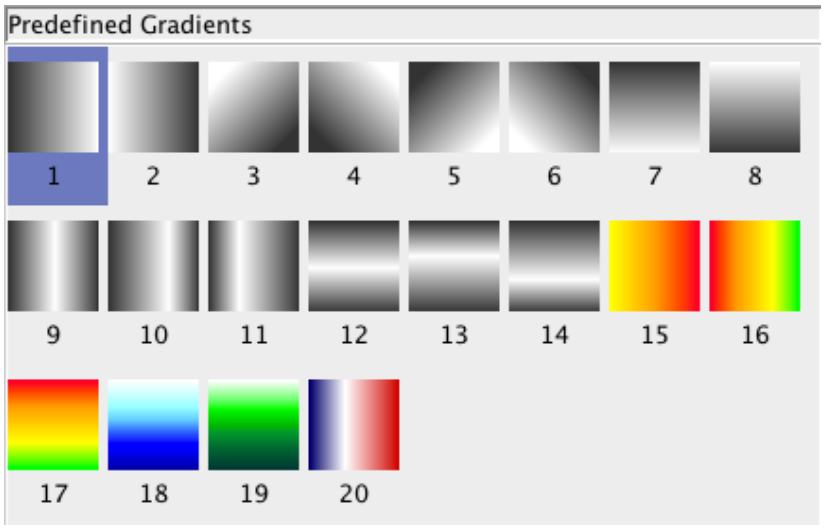


Linear gradient



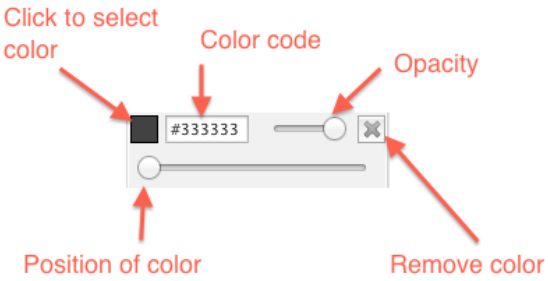
The linear gradient function enables you to create horizontal, vertical and diagonal gradient fills.

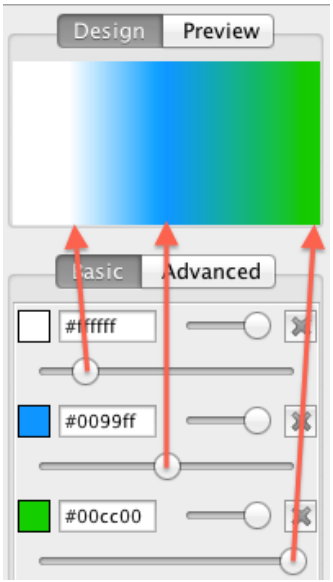
You can choose from pre-defined linear gradients in the **Resources** tab:

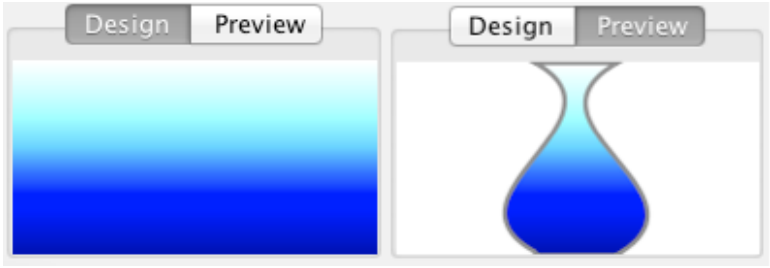


When you click on a pre-defined gradient you will see its properties in the right side panel.

You can see the gradient preview in the small top window and if you click on the tab **Preview** you will see how the object will look like.







1 Position of color and its mapping to gradient definition

The bottom part of the panel is divided into two tabs, **Basic** and **Advanced**:

Basic - you will find all defined colors of a selected gradient

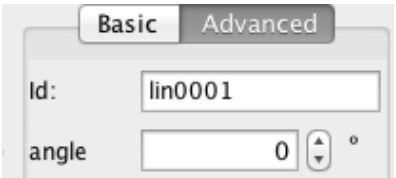
Each color has following properties:

Color - sets the color and corresponding color code

Opacity - specifies the alpha channel (transparency)

Position - specifies the starting point of this color

Advanced - you can fine-tune your custom gradient; define id, angle and the fill options.



Id - here you can name the newly created gradient

Angle - you can change the angle of the gradient from horizontal to a custom angle

x1,y1,x2,y2 - linear gradients are defined around the bounding box of an object they fill, x1 and y1 specify the initial point of the gradient of the bounding box, x2 and y2 represent the end point of the gradient

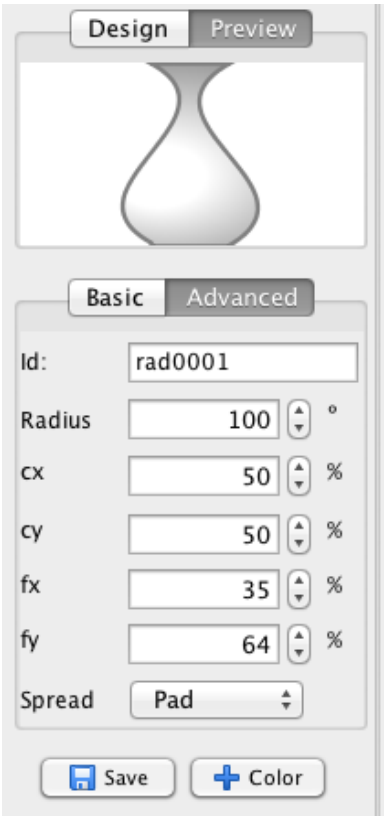
Spread method - Pad (basic fill option – no reflection or repeating)

Repeat (repeats the shading)

Reflect (reflects the shading)

Radial gradients

This function operates with circular gradient fills – same principal as for the linear gradients, only the **Advanced** section is slightly different:



Id - here you name the newly created gradient

Radius - sets the gradient radius

cx, cy and *r* define the outermost circle of the radial gradient

fx and *fy* define the focal point of the radial gradient

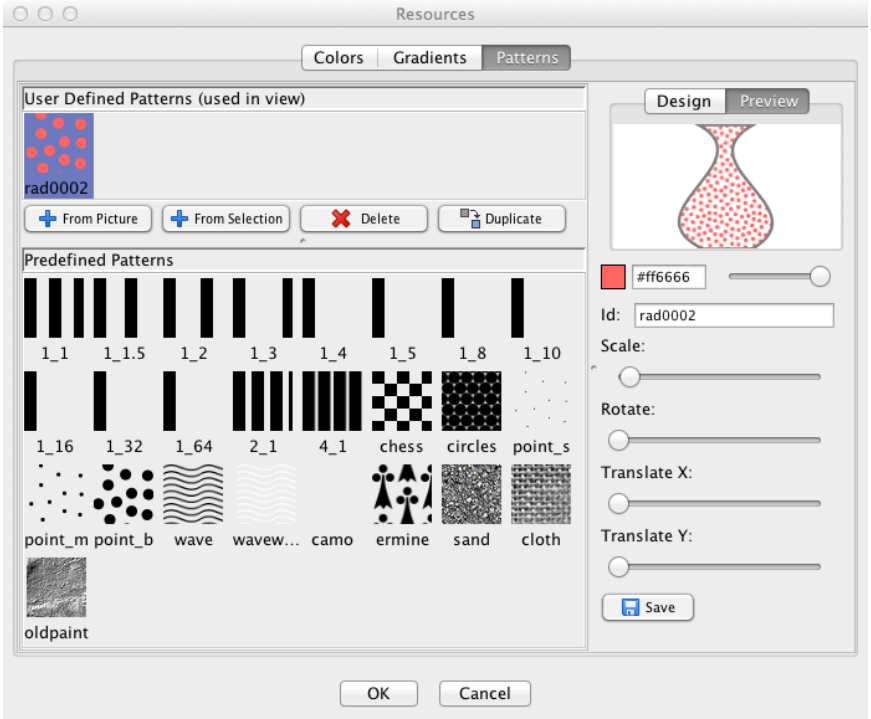
Spread method - Pad (basic fill option – no reflection or repeating)

Repeat (repeats the shading)

Reflect (reflects the shading)

Patterns

Different objects or images can be used as a padding of other objects. myPROJECT Designer lets you choose from pre-defined useful patterns or create new ones, either by modification of the existing ones or importing from a file.



3.14 Rulers and Guides

Rulers

Rulers help you accurately place and measure objects on the canvas. The common ‘0’ point of both horizontal and vertical rulers is the root.

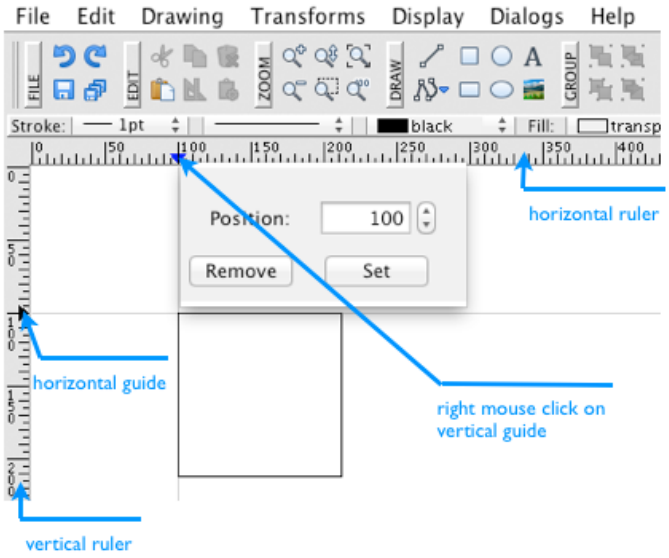


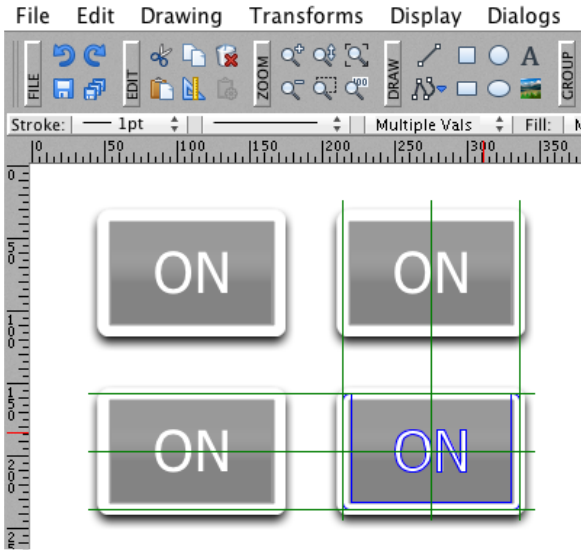
Figure 4 Creating Guides

Guides

Guides help you align text and graphic objects. You can create ruler guides (straight vertical or horizontal lines) to align the objects. To create a new guide, click on any point on the ruler, the created guide can be later moved. You can set its exact location or delete it by the right-click.

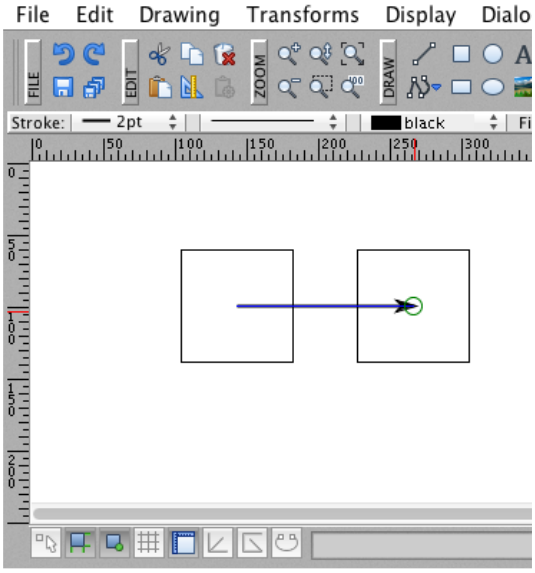
Smart Guides 

Smart guides are temporary orthogonal snap-to-point guides that appear when you are creating or manipulating with objects. They help you align, edit, and transform objects in respect to other objects.



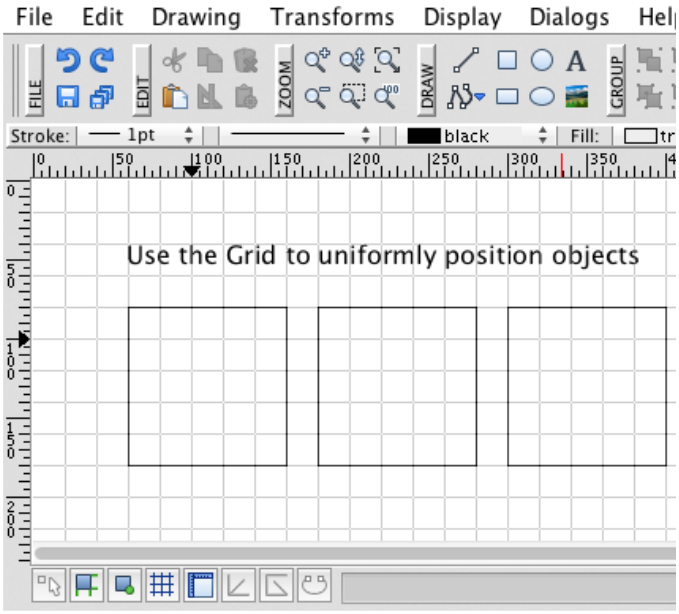
Snap-to-Point 

Snap-to-point guides are temporary bullet points that appear when you are creating or manipulating with objects. They help you to snap your object to other objects.



Grid

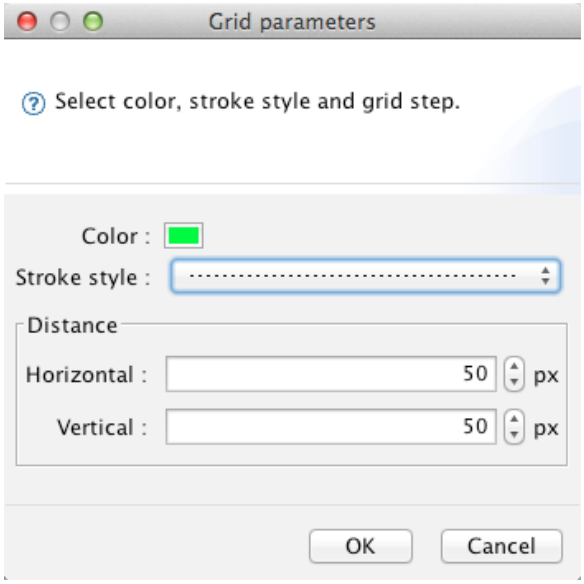
The grid displays behind your graphics on the canvas but does not print to the project visualization – objects align automatically when the grid is active. You can activate or deactivate it from the *Display* menu or the *Options bar*.



Grid settings 

Here you can set the grid parameters like color, stroke style and the grid step.

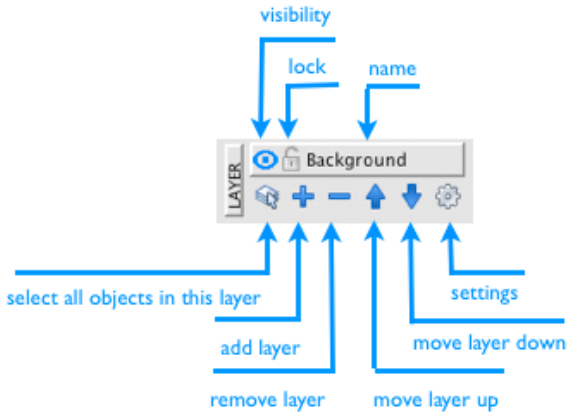
Go to the menu *Display –> Grid settings*.



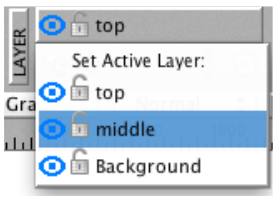
3.15 Layers

When creating complex visualizations it might get challenging to keep track of all your items on the canvas. The smaller items might get hidden under the larger ones and working with all objects becomes difficult. Using the layers can help you manage all the items of your artwork.

Think of the layers as of transparent planes each containing graphic items that are glued together. If you change the layers order you change the position (visibility) order of the items on the canvas too. You can move items within one layer or you can group together multiple items spread throughout several layers. If you copy objects, then a new object is copied into the layer of the original one.

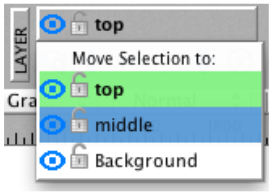


If there is no object selected the name of active layer is displayed.



Clicking on the layer name will show the list of the layers to be set as active.

If you select an object, the layer containing this object will be displayed and highlighted with green color on the layer list.



If you want to move objects to a different layer, select the objects on the canvas and select the desired layer from the list.

3.16 Copying and Pasting Elements

Paste to same location

This function is used for pasting objects being copied into the same location on the canvas - it is best utilized for specifying more parameters for one visible object, such as a button.

Begin with creating an object and click on *Edit -> Copy*. Now paste the object being copied by clicking on *Edit -> Paste to same location* (you can also right-click on the object and select this function). This operation can be repeated as many times as you need.

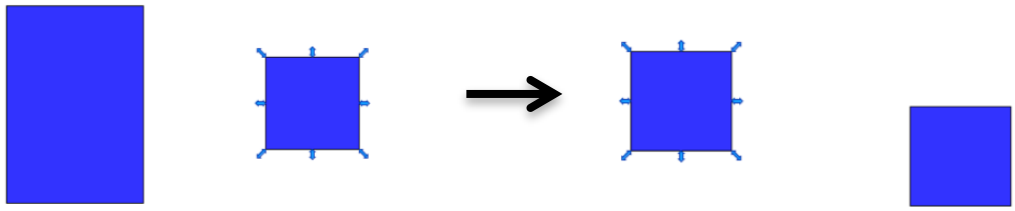
Note: You can verify that an object has been pasted successfully by moving it aside and then moving back by clicking on Edit -> Undo.

Paste dimensions

This feature allows you to paste specific dimensions of currently selected objects. Select an object whose dimensions you want to copy. Then select an object you want to apply the dimensions to by clicking on *Edit –> Paste dimensions*.

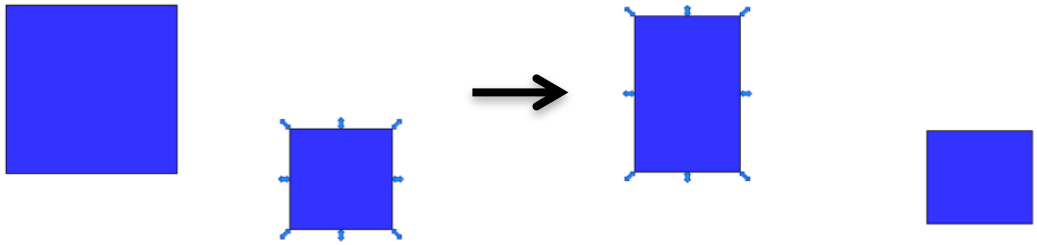
Paste size

This function will paste all dimensions.



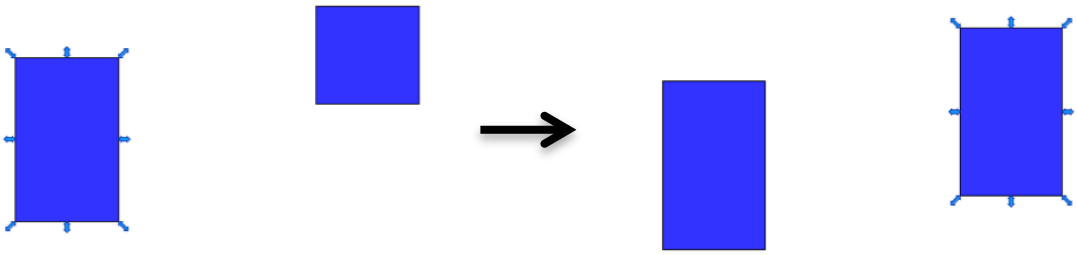
Paste width

This function will paste only the width.

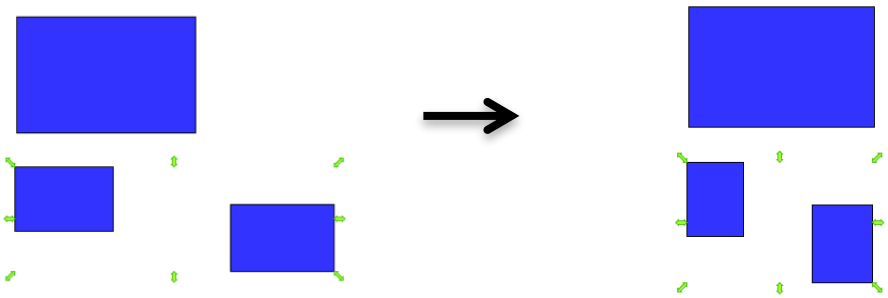


Paste height

This function will paste only the height.



When the function ***Past dimensions*** is being applied to multiply selected objects, the dimensions are transferred to the selection bounding box and the space among the objects is accordingly scaled.

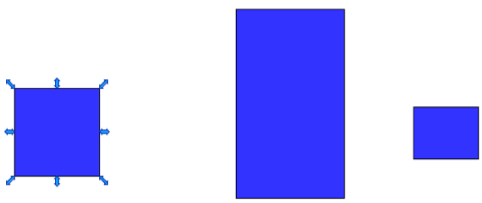


Paste size separately

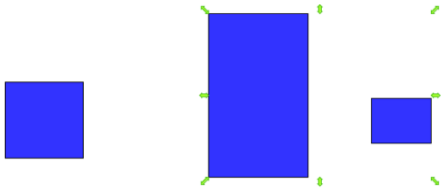
This function allows you to apply dimension to multiple objects at one time.

In the following example we will copy the left square and then resize the remaining rectangles:

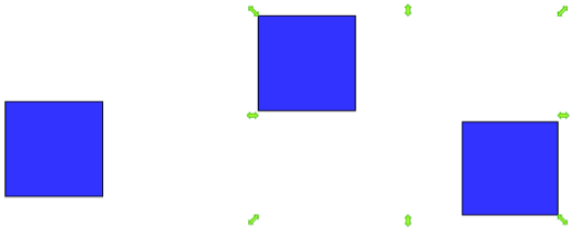
1) Create multiple objects and then copy the object of the size you want to apply to the remaining objects.



2) Select the remaining objects.



3) Open the *Edit* menu; select *Paste dimensions - Paste size separately* (you can also right-click on the object and select this function). All objects are now of the same size.



Paste width separately

This function allows you to change the width of multiple objects at once as described in the previous example.

Paste height separately

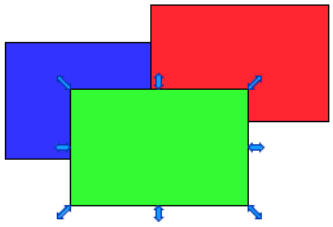
This function allows you to change the height of multiple objects at once as described in the previous example.

3.17 Objects Order

Graphic objects use a hierarchical order. If two objects overlap, the one located in the higher layer overlaps the other one located in the lower layer. You can change the object order by raising them up to the top or lowering down into the background.

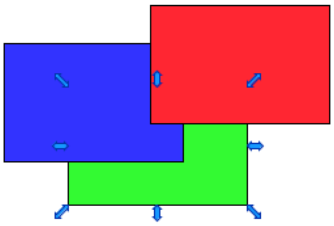
Example:

1) Create a new object and use *Edit -> Copy* and *Edit -> Paste* on same location twice. Now you have three rectangles of the same size. For easier visualization fill each object with a different color (function Properties – Fill – Color) and drag the objects so they partially overlap.



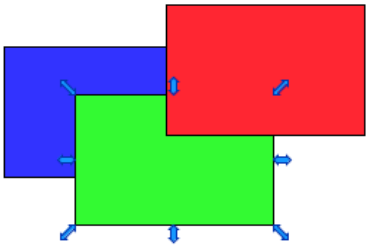
2) Now click on *Transforms -> Order* to change the objects order and use one of the functions below:

Lower to the background - moves a selected object into the background so that all other objects are on the top



Raise to the top - moves a selected object to the top so that all other objects are beneath it

Raise - moves a selected object one level up

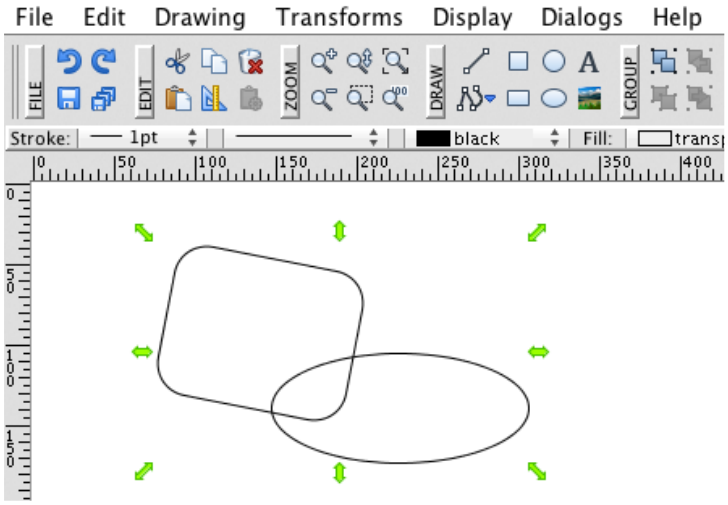


Lower - moves an object down one level down

3.18 Grouping

Grouping elements 

You can compose your graphics from multiple elements. Moving or copying these graphics might be difficult, but to simplify these operations you can group multiple objects together so you can use them as a single unit.



	group selected objects together
	un-group selected object
	enter group - changes objects inside the group without un-grouping
	exit group - end enter group mode

Select the objects to be grouped and click on the icon  in the toolbar (or select from the menu *Edit*). The objects are now grouped and behave as one object – the group can be moved, resized, rotated, skewed etc.

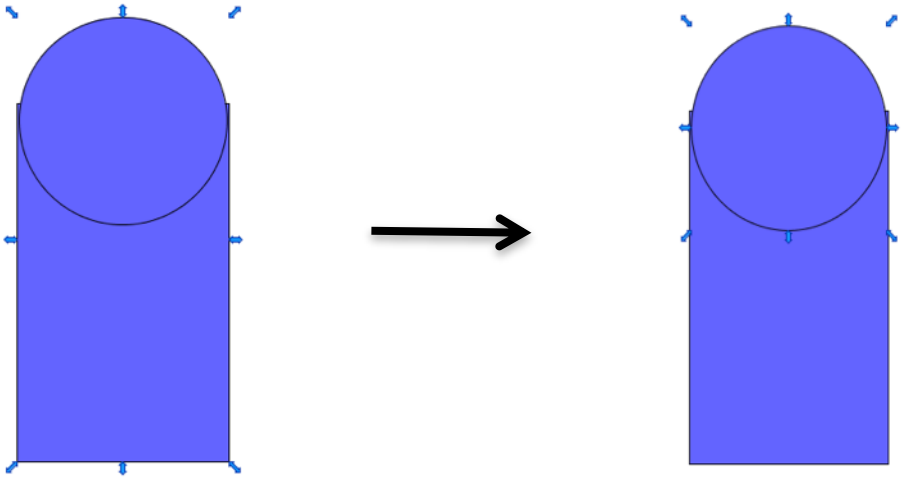
Selected objects: Group elements:

Ungrouping elements 

This function is used for dividing a grouped object back into elements.

Select the grouped object and click on the icon  in the toolbar (or select from the menu *Edit*) and the object will divide into its elements.

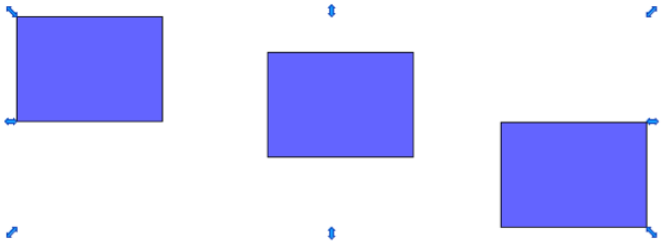
Grouped elements: Ungroup elements:



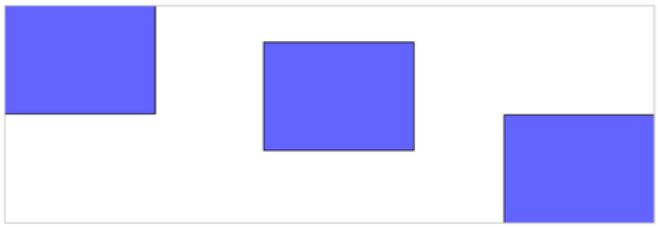
Enter group 

With this function you can work with individual objects that are grouped. You can work with each object individually without ungrouping.

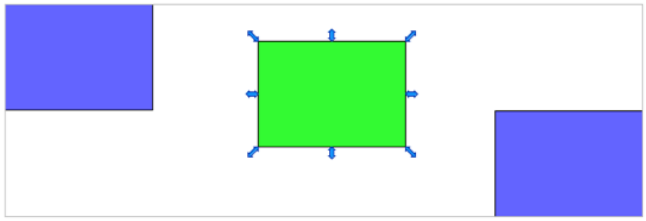
1) Select a group of objects.



2) Click on *Edit* (shortcut **Ctrl+E**) and select *Enter group*.



3) Now you can work with any object in the group individually.



Exit group

Once you have finished working with the objects in the group you can leave the group by clicking on *Edit -> Exit group* (or shortcut **Ctrl+Shift+E**).

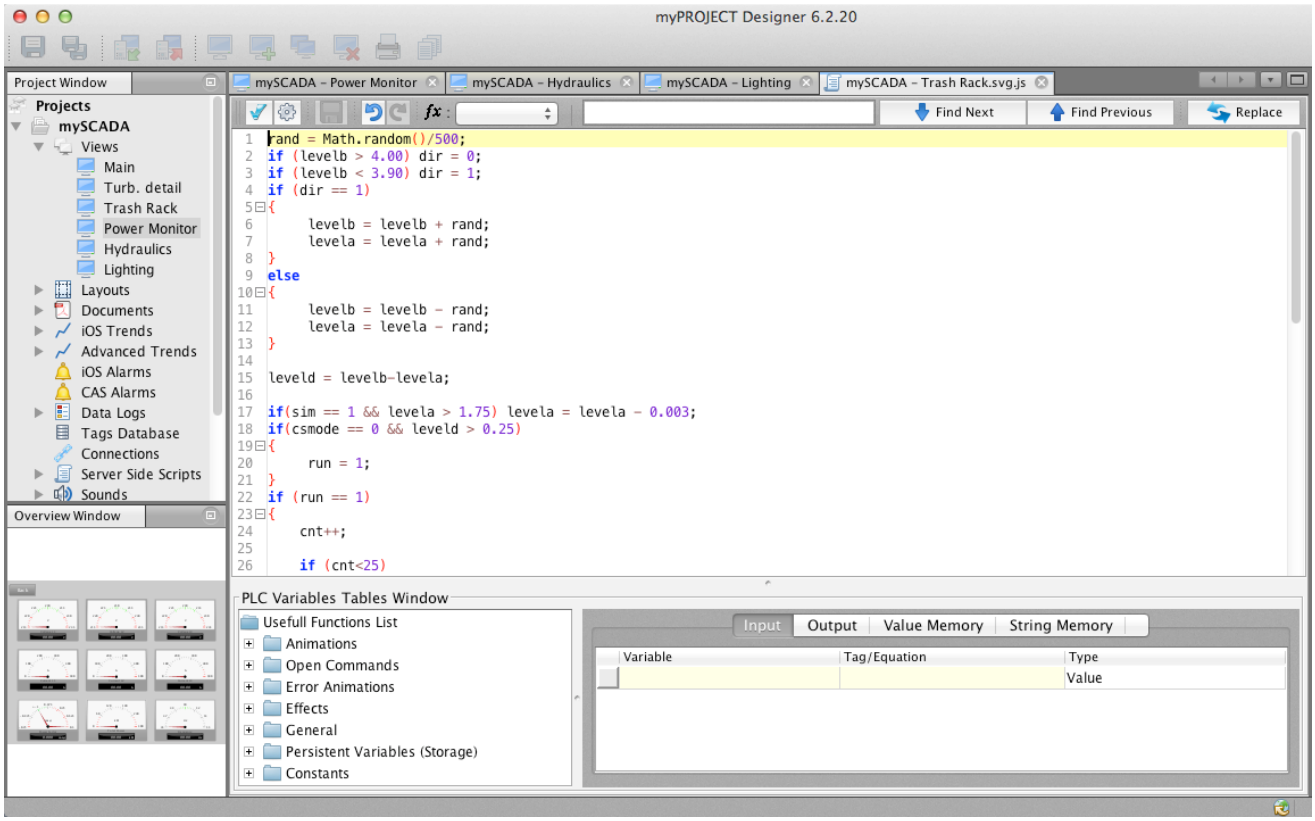
3.19 Repeated Actions Mode

This mode allows you to continue drawing the same type of objects without having to re-select the object type or shape each time you draw it.

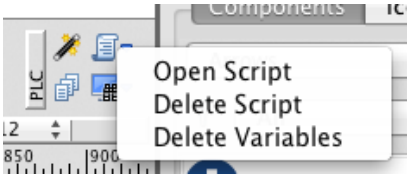
Click on the **Repeated actions mode** icon in the lower options bar and select an object to draw.

3.20 Visual Scripts

Click on the **Scripts** icon in the toolbar to open a new tab with scripting editor. For more details please see the chapter *Server-side Scripts* of this manual.

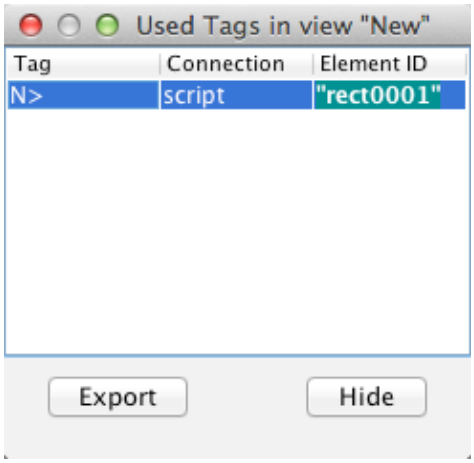


The **Script** icon has a context menu with which you can open a current view script in the editor, delete an existing script or clear all defined script variables. Use the right-click to activate the menu.



3.21 Used Tags

This function lists all tags used in an active view.



3.22 Live View

With the function ***Live View*** you can load only an active view into your device. This function is serves as a quick test / preview of active views without loading the whole project.

Click on the arrow next to the **Live View** icon and open the list of all available devices. Select one of them to set it device as a ***Live View*** recipient. Clicking on the icon sends a current view to a previously selected device. This device will remain selected as the *Live View* recipient until another device is selected. Remember that selected devices need to be connected to your local network/VPN, to be able to receive the ***Live View***. This function is currently available to iOS devices only.

3.23 Zoom on Zone

This function zooms a selected area into a whole workspace. First click on this icon in the toolbar, then click and drag the mouse on an area you want zoom.

3.24 Undo and Redo

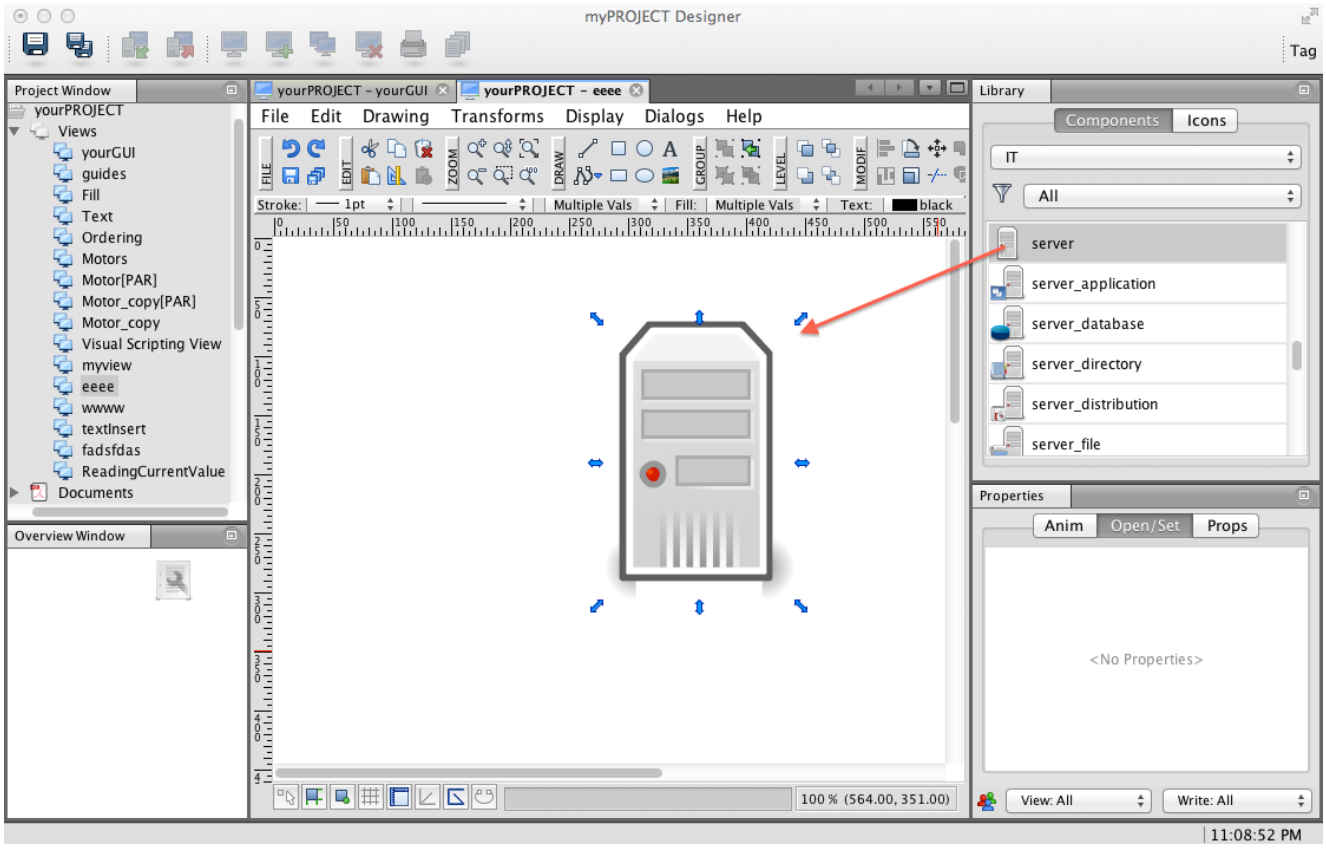
The **Undo** button allows you to revert your action by one step back. You can find it in the toolbar or in the menu *Edit*. You can also right-click on an object and select this function.

The **Redo** button allows you to move one step forward after using the *Undo* function, you can find it in the toolbar or in the menu *Edit*. You can also right-click on an object and select this function.

Note: If these buttons are grayed out it is not possible to use them for that moment.

3.25 Adding Components and Icons

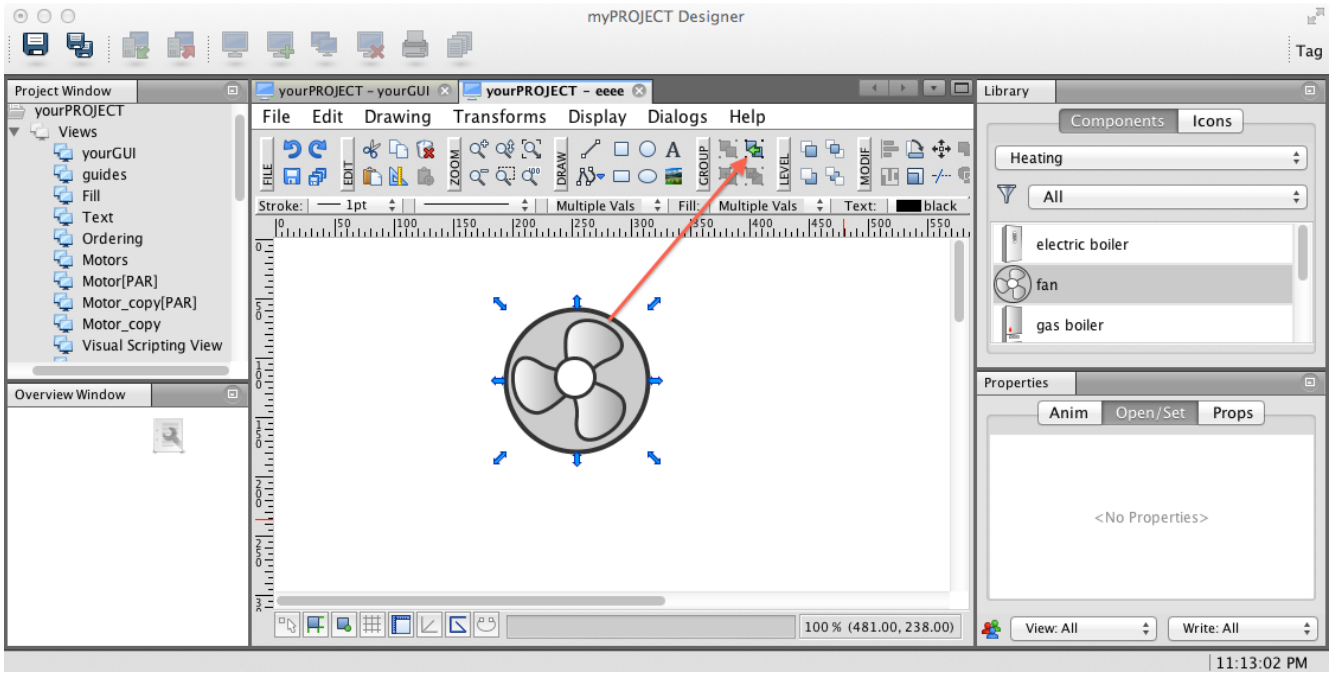
On the right side of the workspace there is a panel, called *Library* with *Components* and *Icons*. All pre-loaded library items can be accessed with selectors in this tab.



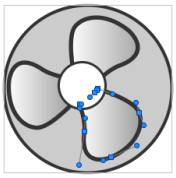
From the context menu you can click on the **Components** tab and the scroll preview list will show up. Each specific icon or a component can be dragged from this list to the canvas and used for visualizations.

Editing components

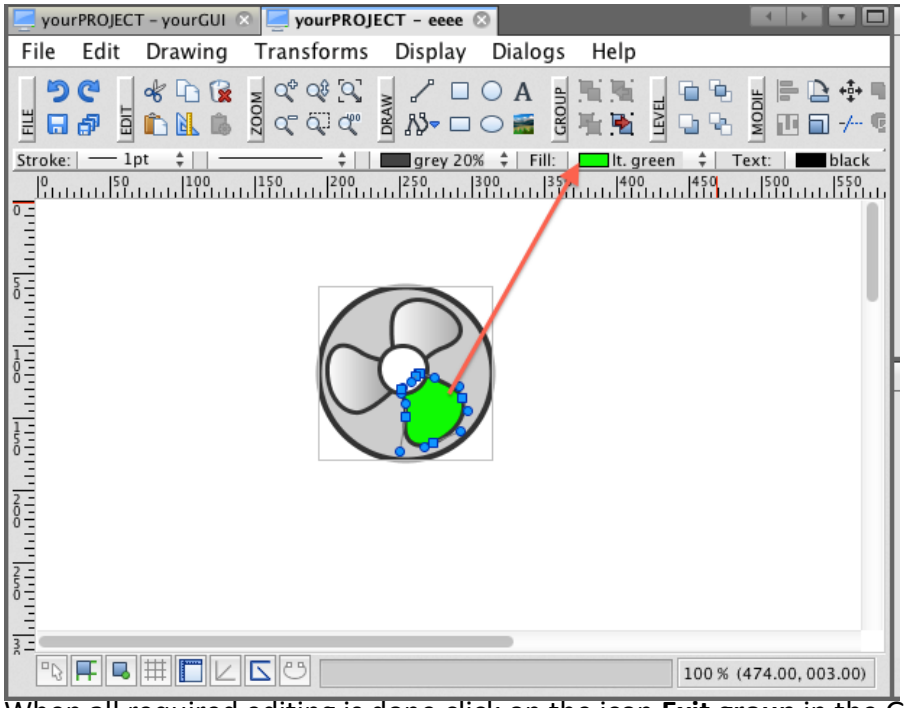
If you need to edit any component, e.g. change its center or a color of some parts, select this component and click on the **Enter group** button in the GUI toolbar (see the picture below).



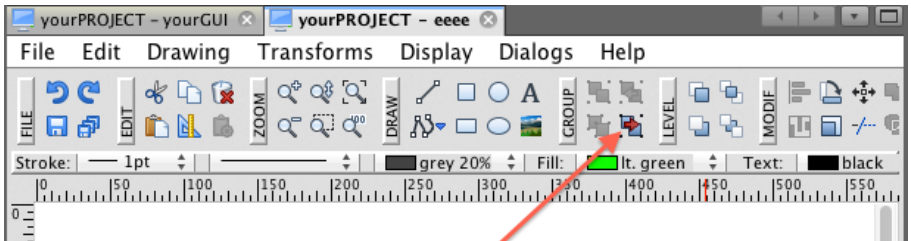
Now, the elements of a selected component are accessible. If the desired element is grouped with others, use the same procedure as described above to enter the required sub-group.



To change colors of the selected element, use the properties toolbar.

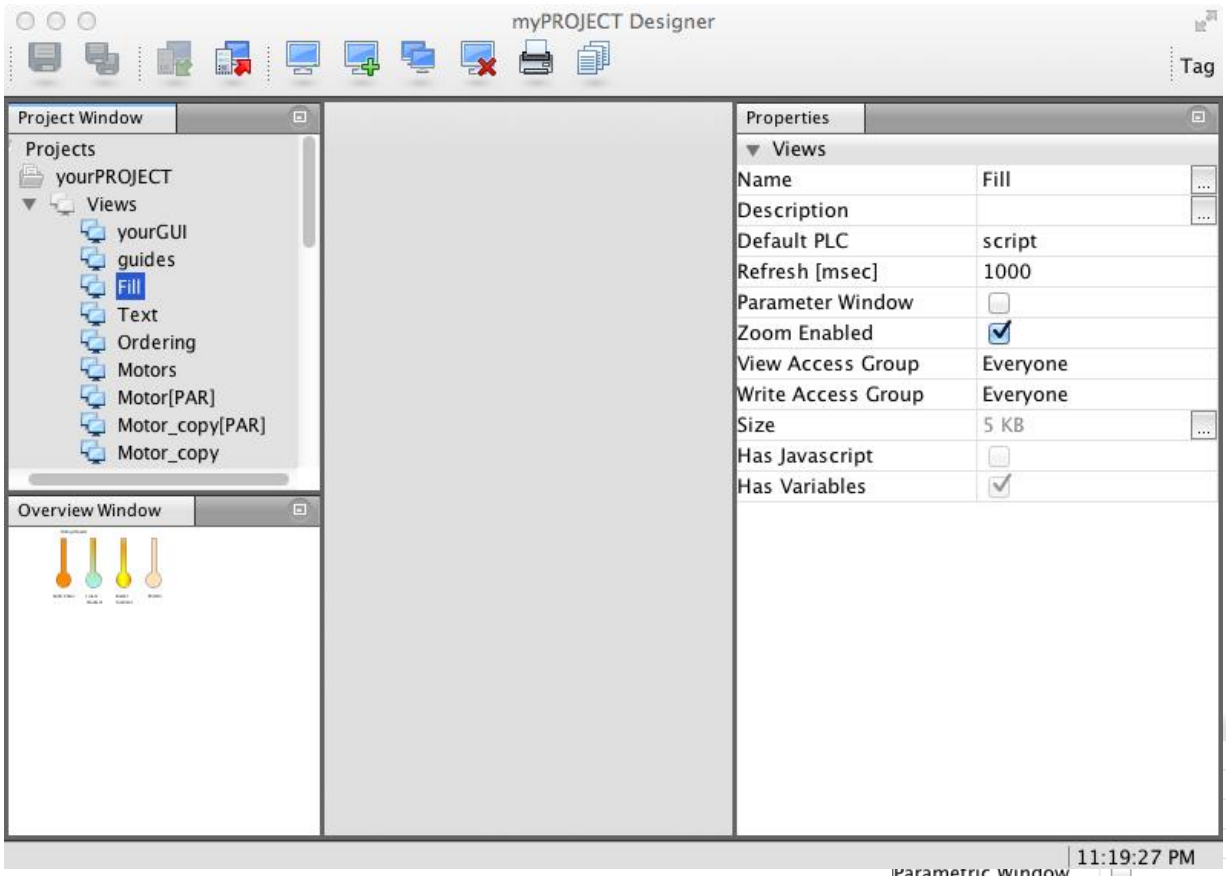


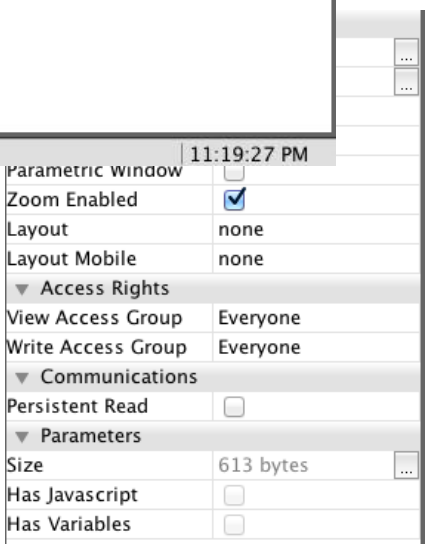
When all required editing is done click on the icon **Exit group** in the GUI toolbar.



3.26 View Properties

If you select a view from the project tree its properties will show up in the *Properties window*.





Following options are available:

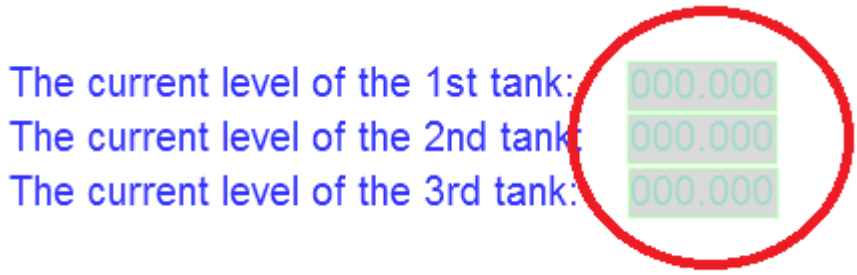
- *change view name*
- *change view description*
- *select another default connection for the view*
- *change refresh rate*
- *change "Parametric view" property*
- *Enable/disable runtime zoom for the view*
- *Set Access Rights*

  View Access Group: Sets the minimal level of access for viewing.
  Write Access Group: Sets the minimal level of access for writing to PLC.

- *Persistent read* – enables persistent *read–to–read* all the data from the PLCs even if a view is not being viewed by anyone, this option is useful for a fast view refresh when loaded. However, heavy usage can lead to a slower reaction of the system as more data is read from the PLCs continuously.

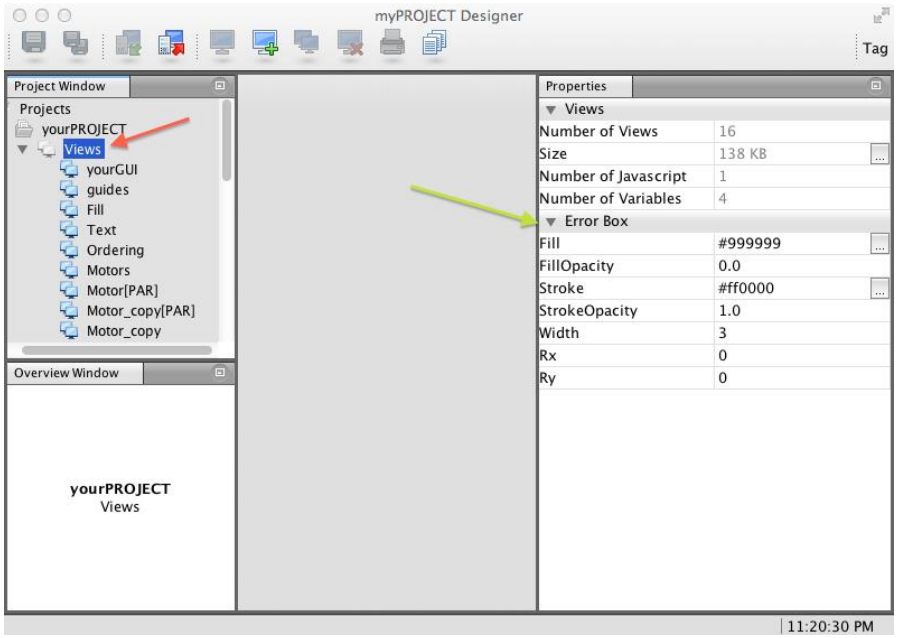
3.27 Error Box Properties

You can define a graphic representation of an "offline" error for objects tied with the tags for animation. When such tags cannot be read from the PLCs, *mySCADA* will display an error box around such objects.



In the *Error Box* properties you can define the look error boxes in your project.

Select the ***Views*** folder and its properties will appear in the *Properties window*:

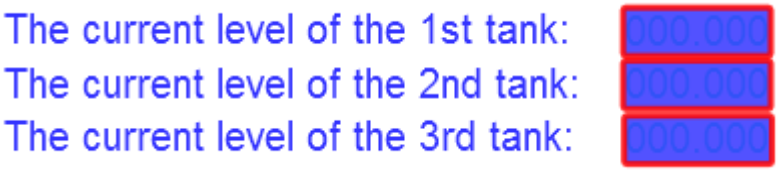


Set the fill color, fill opacity, frame color, frame thickness and opacity of the *Error Box*. With the parameters *Rx* and *Ry* you can change softness of the *Error Box* corners.

Error Box	
Fill	#999999
FillOpacity	0.75
Stroke	#ff9900
StrokeOpacity	0.75
Width	3
Rx	0
Ry	0

For example, if you fill the *Error Box* properties with the parameters below the communication error will be displayed as follows:

Error Box	
Fill	#3333FF
FillOpacity	0.85
Stroke	#FF0000
StrokeOpacity	0.85
Width	6
Rx	3
Ry	3

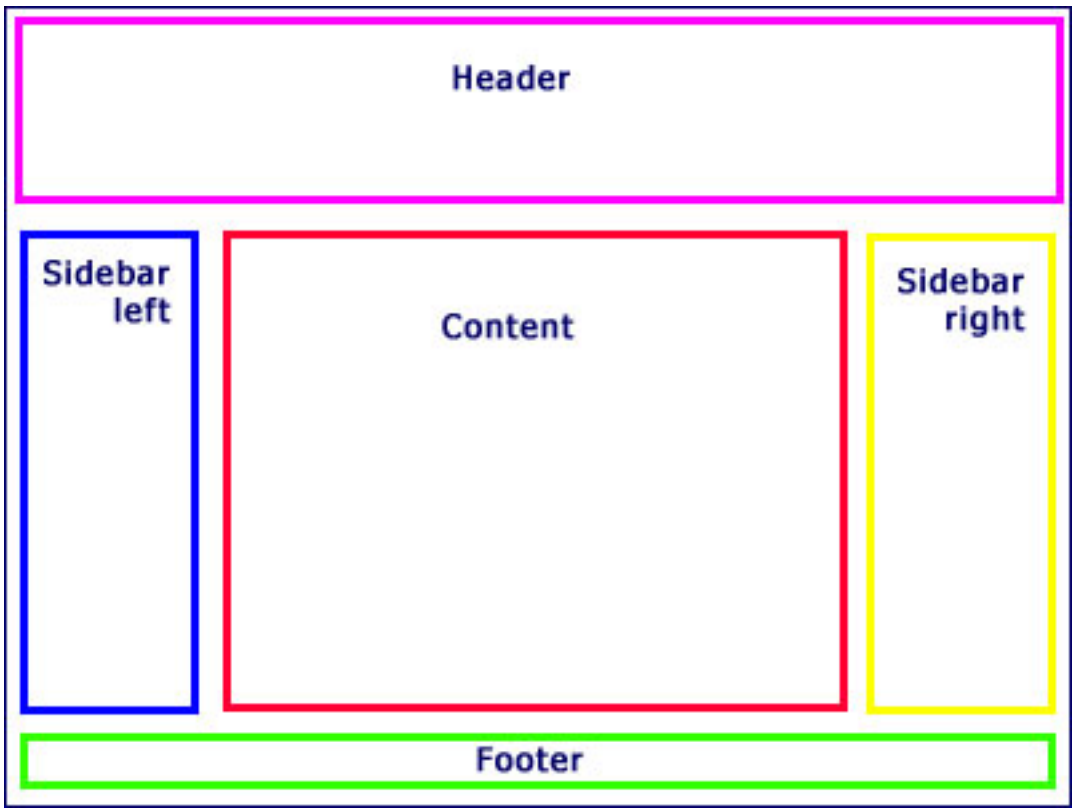


4. Layout Views

4.1 Page Layout

The *Layout* page defines the arrangement and style of the page content. In *myPROJECT Designer* you can create multiple, user-defined layouts. Each view you create can use different layouts.

Note: You can create new layouts at any time during the project creation.



Header - top section used for displaying a logo, name, logged user, main menu etc..

Main Content - section displaying your views, it sits prominently in the middle of the page

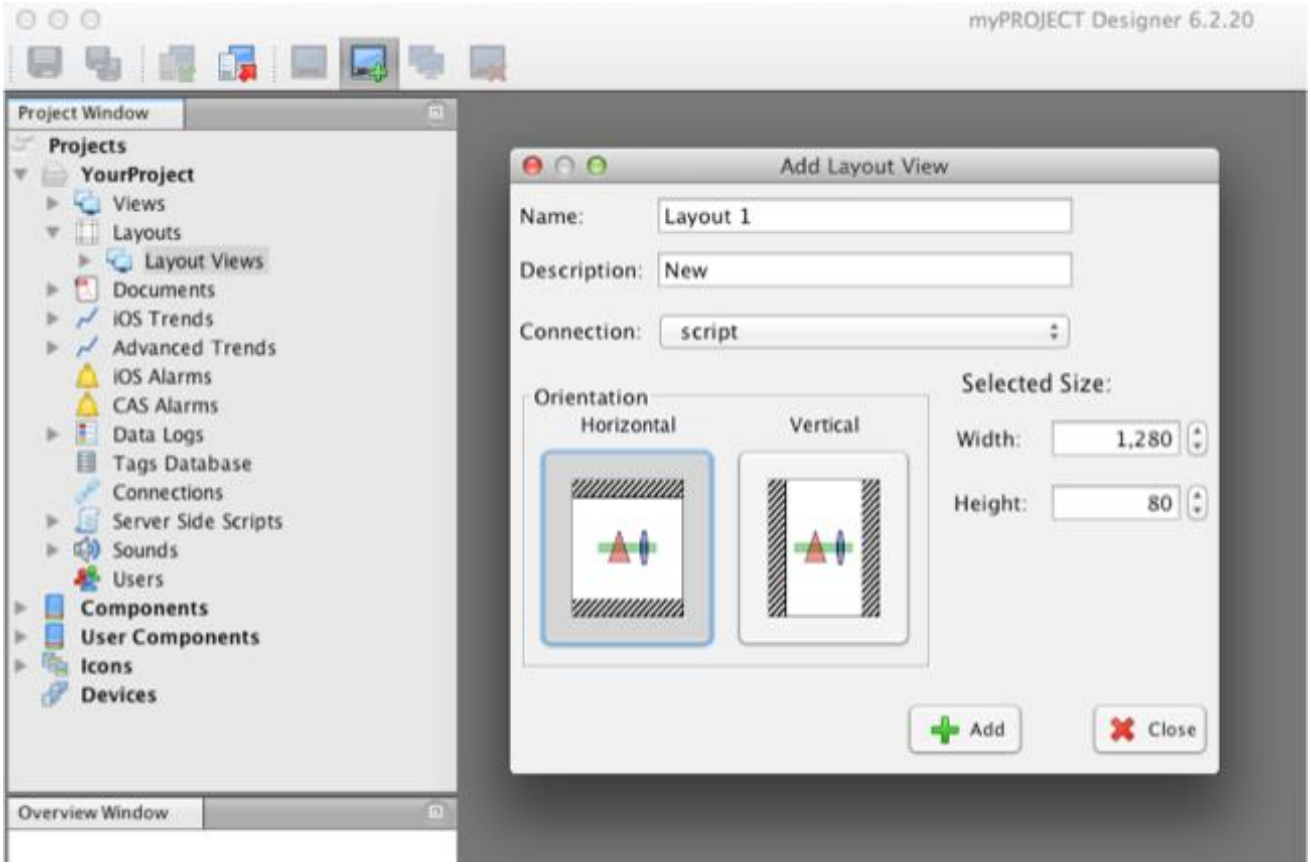
Sidebar Left/Right - columns on both sides of the main content section used for displaying additional menus, pictures etc., or can be used for control buttons and gauges

Footer - bar spanning across the page bottom

Note: To use the Layouts, first you have to create Layout Views. The Layout Views behave exactly the same as regular views therefore you can apply any functionality there, such as animations, effects and visual scripts.

4.2 Adding Layout View

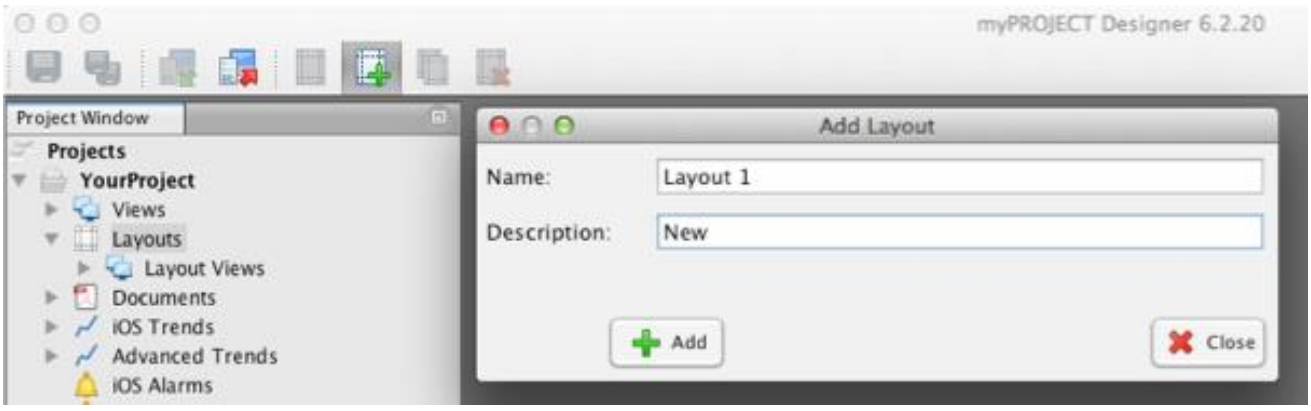
1) Select *Layouts -> Layout Views*
2) Click on the **Add Layout View** icon in the main toolbar
3) Enter the name and description
4) Define the orientation and select the layout size



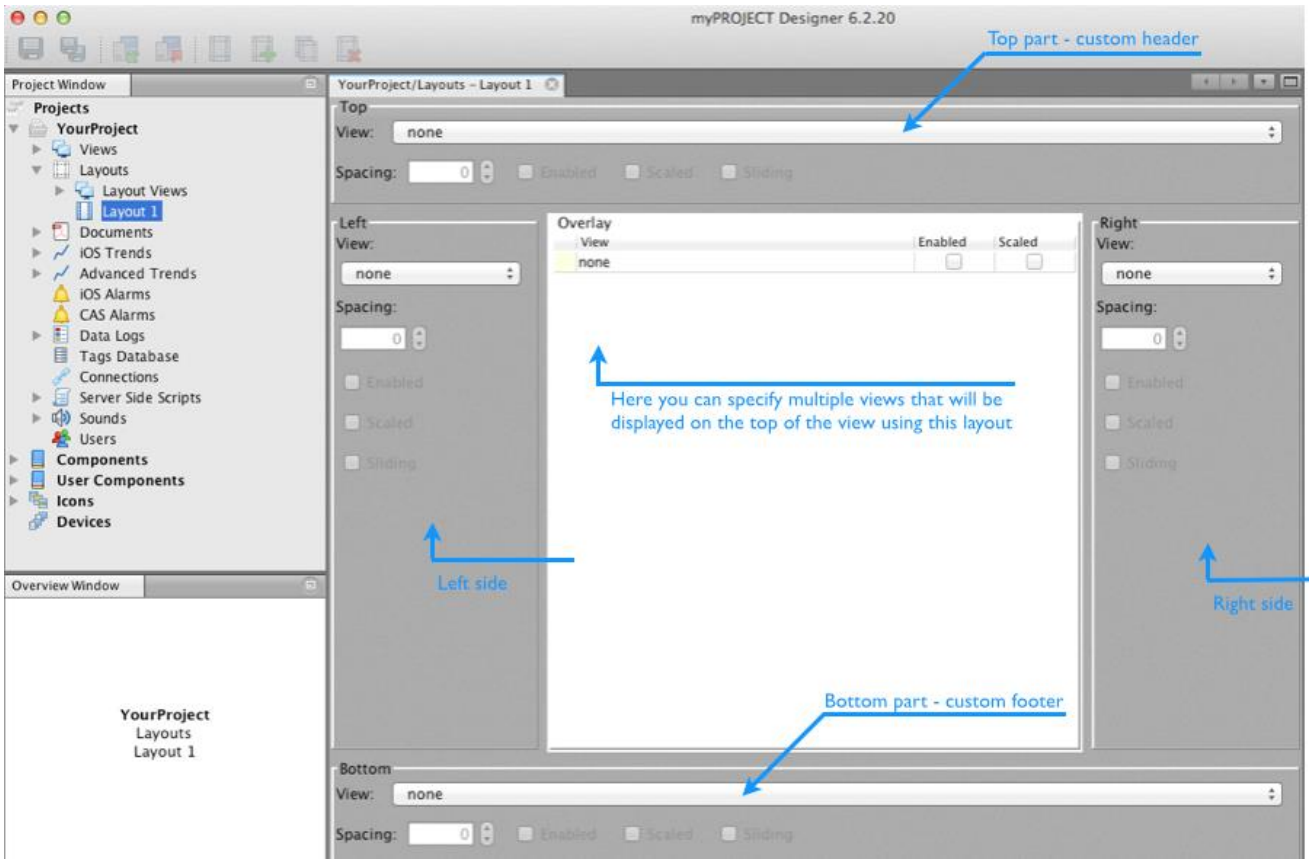
4.3 Creating New Layout

After you have created views you can use them for the layouts:

1) Select Layouts in the *Project Window*
2) Click on the **Add Layout** icon
3) Enter the name and description and click on **Add**



After creating a new layout, double-click on it to fill in its properties:



For each section you can select a corresponding *Layout View*. After you have selected a required view, do not forget to tick off the *Enabled* box to activate it.

Options for each section:

- *Spacing* - creates spaces among the sections, defined in pixels
- *Enabled* – tick box to enable particular sections
- *Scaled* - makes the section scaled according to the rest
- *Sliding* - makes the section hiding; in the runtime user can open it in by clicking on the tab at the section center

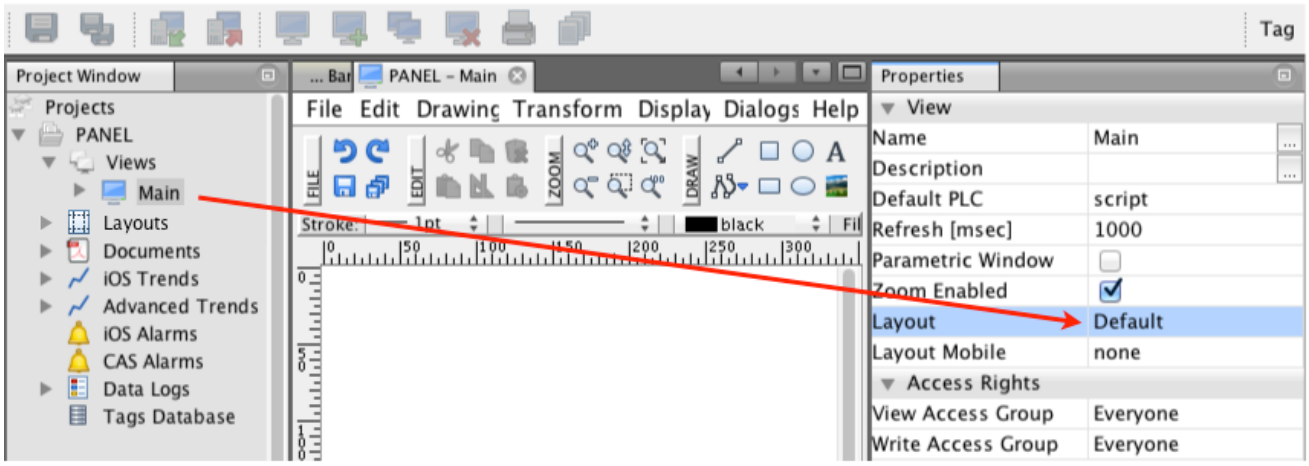
Overlay section

In this section you can put one or more layout views above existing ones. This can be useful for showing messages or images tied to a visibility animation – instead of copying these messages or images into all views you just add them to a layout view and use as an overlay for all the views.

4.4 Using Layouts in Views

1) Select the view you want to apply a layout to

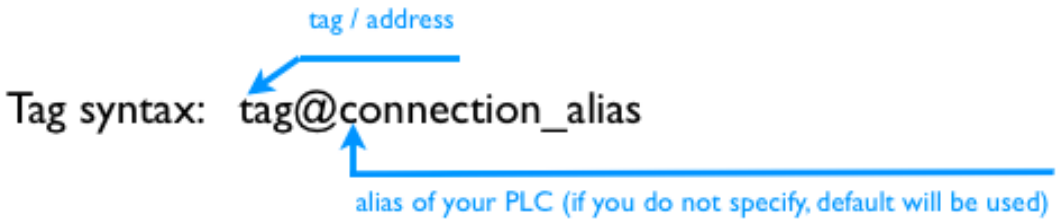
2) In the *Properties window* select *Layout* from the drop-down menu



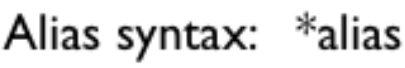
5. Entering Tags and Expressions

5.1 Entering Tags

Before linking your visualizations with PLCs, first you have to enter the name or address of the tags you wish to read/write data from. The tag syntax depends on the PLC type, which you want to access.

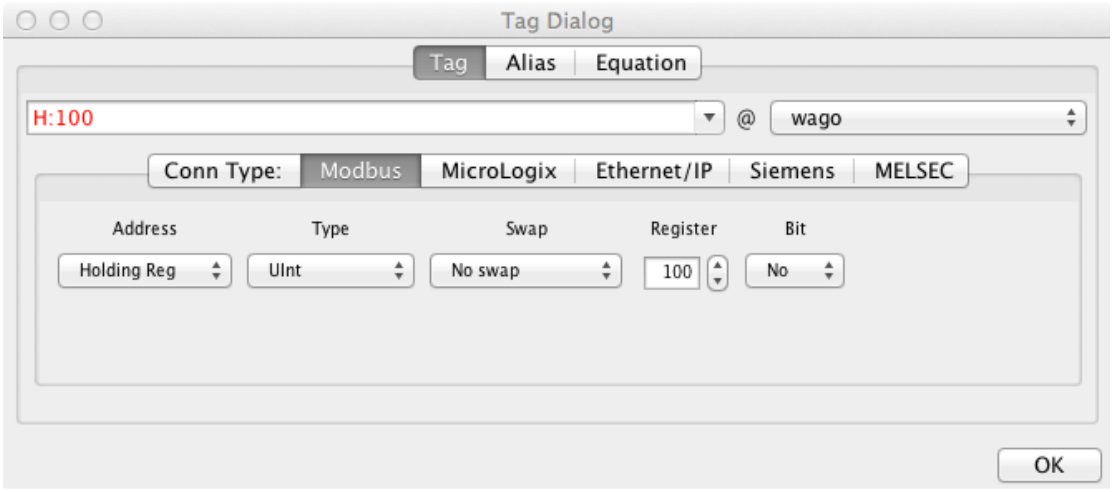


Note: You need not enter full tag syntaxes all the time. Instead, you can use a symbolic simplified link to your tag, called Alias. The Aliases can be defined in the Tag Database.



You can type the tag directly in the tag edit field:

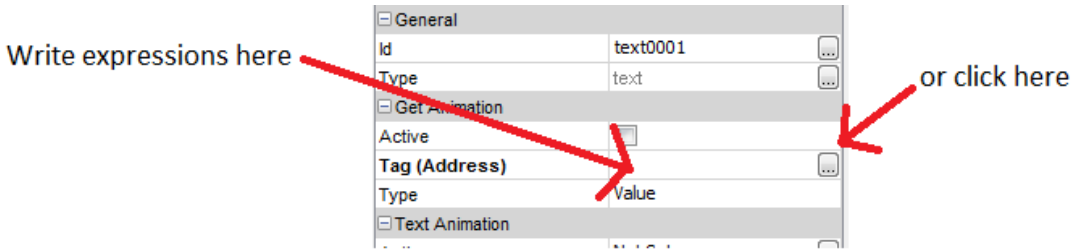
You can call the tag editor by clicking on the **...** button on the right side of the tag edit field - a new dialog window will show up:

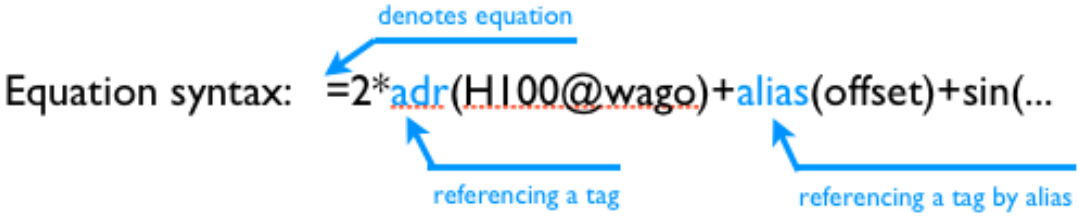


The *Tag Dialog* will guide you through tag entering and will check if the syntax is correct.

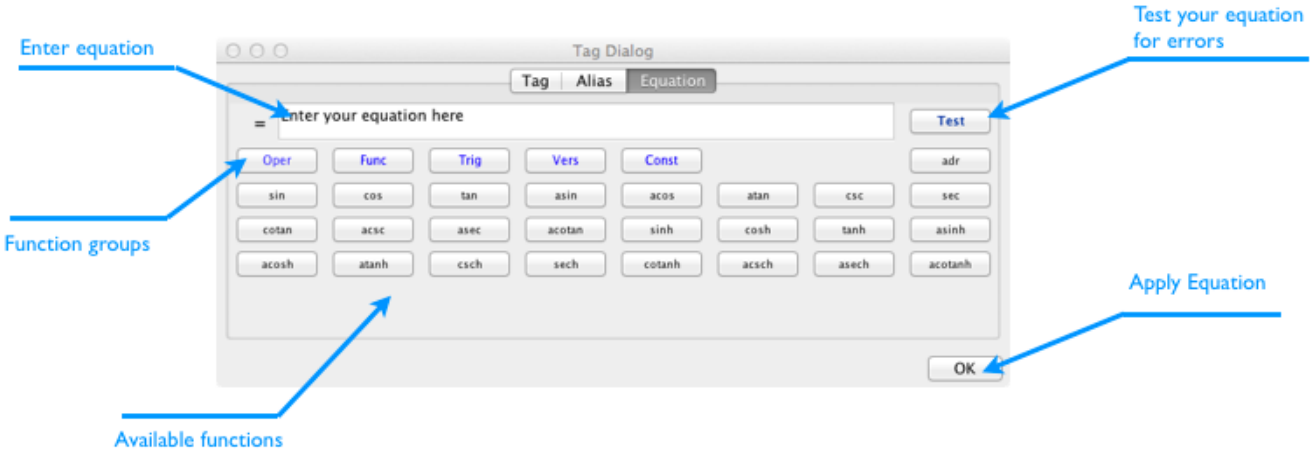
5.2 Entering Mathematical Expressions

Instead of writing the tag name, you may enter mathematical expressions. This way you can scale and offset the values read from the PLCs or create more complex data processing. These expressions can be entered either directly or through the dialog window.





In the equations you can use operators + - * / and common mathematical functions such as *sin, cos, exp,* etc*...* Also you can do binary comparison and much more. To get the complete list of options call the *Tag Dialog* and click on the tab **Equation**.



Formatting & Limits

- Tag name is entered by *adr()* function
- Tag name can be also entered as alias using the *alias()* function
- You can use only supported functions and operators

The supported functions and operators are listed under the input box into five groups: **Oper, Func, Trig, Vers** and **Const** – they will insert into the box above if selected

You can use multiple tags in an expression.

Example:

1) We read a value from Modbus H:0
2) Let us scale this value by 10 and offset by 0.5
3) Formula to enter is *"=10*adr(H:0)+0.5"*

Supported functions and operators:

Operator	Function	Description
Standard Operators		
+	add	addition and unary positive
- r −	subtract and negative	subtractionand negation
* o ×	multiply	multiplication
r÷	divide	division
	percnt	modulus andpercentage of a value
!	factorial	factorial
**	pow	exponential
°	deg	converts values to radians
Bitwise Operators		
&	and	bitwise and
|	or	bitwise or
^	xor	bitwise xor
~	not	bitwise not
<<	lshift	bitwise left shift
>>	rshift	bitwise right shift
Comparison Operators		
=	l_eq	equal
!=	l_neq	not equal
<	l_lt	less than
>	l_gt	greater than
<= or ≤	l_ltoe	greater tha orequal
>= or ≥	l_gtoe	greater than or equal
Logical Operators		
&& or ∧	l_and	logical and
|| or ∨	l_or	logical or
! or ¬	l_not	logical not

Degree operator °

The degree operator (°) is very useful when converting a user input. Because all of the trigonometric functions require their parameters to be passed in radians, the degree operator will convert its operand into radians. Thus, 45° is equivalent to *dtor(45)*.

Percentage sign %

When the percent sign is interpreted as the *modulo*, then:

10 % 3

... evaluates to 1 (the remainder after 10 is divided by 3) however, if you flip the switch to make the % sign to stand for percentage, then you do:

250 + 10%

By default, % is usually shorthand for "/100". In other words, 42% becomes 42/100, or 0.42.

However, if the % term is the right-hand sign of *subtraction* or *addition* (such as in "250 + 10%"), then the percent is evaluated as percentage of the left-hand side (i.e. "250 plus 10% of 250").

If you choose to interpret the percent sign as the modulo operator, you can still request a percentage by using the function name directly:

(10 % 3) + percent (50) = 1.5

Factorial and Logical 'Not'

Differentiating between factorial (!) and a logical not (!) is more difficult. A "!" is interpreted as a logical not if:

it is the first token

it is preceded by a binary operator

it is preceded by a right associative unary operator

Otherwise it is treated as a factorial. A ¬ token is always treated as a logical not (for obvious reasons).

Supported functions

Functions using more than 1 parameter:

- **sum()** - returns a sum of the passed parameters
- **count()** - returns the number of passed parameters
- **min()** - returns the minimum of the passed parameters
- **max()** - returns the maximum of the passed parameters
- **median()** - returns the median of the passed parameters
- **stddev()** - returns the standard deviation of the passed parameters
- **average()** - returns the average of the passed parameters

- **random()** - returns a random integer. Can take 0, 1, or 2 parameters. The first parameter (if given) is the lower bound of the random integer. The second parameter (if given) is the upper bound of the random integer.
- **nthroot()** - returns the n^{th} root of a number, for example, *nthroot(27,3)* returns the cube root of 27, or 3.

Functions using 1 parameter:

- **sqrt()** - returns the square root of the passed parameter
- **log()** - returns the base 10 log of the passed parameter
- **ln()** - returns the base e log of the passed parameter
- **log2()** - returns the base 2 log of the passed parameter
- **exp()** - returns e raised to the power of the passed parameter
- **ceil()** - returns the passed parameter rounded up
- **floor()** - returns the passed parameter rounded down

Trigonometric functions:

- **sin(), cos(), tan()**
- their inverses (**asin, acos, atan**)
- their reciprocals (**csc, sec, cotan**)
- reciprocals of the inverses (**acsc, asec, acotan**)
- hyperbolic variations of all the functions above (**sinh, cosh, tanh, asinh, acosh, atanh, csch, sech, cotanh, acsch, asech, acotanh**)
- versine functions (**versin, vercosin, coversin, covercosin, haversin, havercosin, hacoversin, hacovercosin, exsec, excsc, crd**)
- **dtor()** - converts the passed parameter from degrees to radians
- **rtod()** - converts the passed parameter from radians to degrees

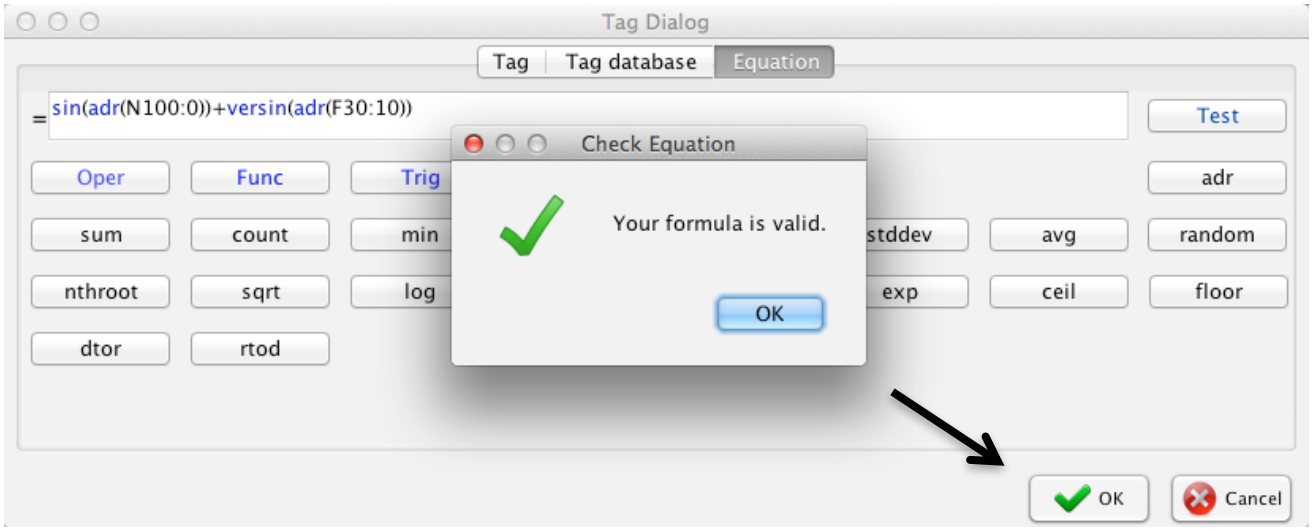
Functions using no parameters ("constant functions"):

- **phi()** - returns the value of ϕ (the Golden Ratio), also recognized as *ϕ()*
- **pi()** - returns the value of π. Also recognized as *π()*
- **pi_2()** - returns the value of π/2
- **pi_4()** - returns the value of π/4
- **tau()** - returns the value of τ. Also recognized as *τ()*
- **sqrt2()** - returns the value of the square root of 2
- **e()** - returns the value of e

- **log2e()** - returns the value of the log base 2 of e
- **log10e()** - returns the value of the log base 10 of e
- **ln2()** - returns the value of the log base e of 2
- **ln10()** - returns the value of the log base e of 10

The parentheses are used for grouping sub-expressions and setting the order of execution, and can be nested to any depth. All computation is carried out with a double precision floating point. In case of error the editor shows a warning and the error expressions will not be evaluated.

You can always check validity of entered expression Clicking on the **Test** button.



Click on **OK** after completion to see the expression in the **Tag(Address)** field.

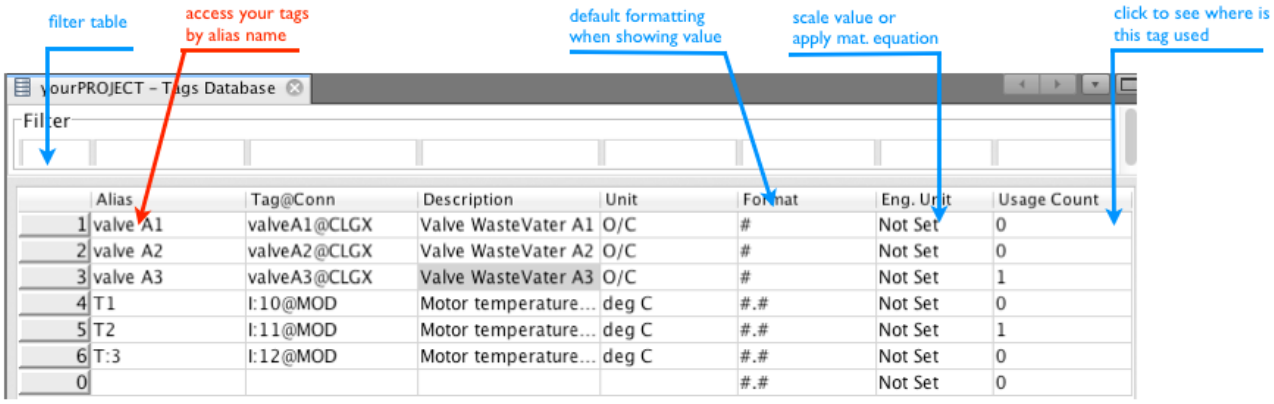
Examples of valid expressions:

- *adr(N100:0)*adr(F30:10)*
- *sin(adr(N100:0))+cos(adr(F30:10))*
- *median(adr(N100:0),adr(F30:10),adr(N20:5),adr(F10:10))* – multiple arguments should be separated with commas.

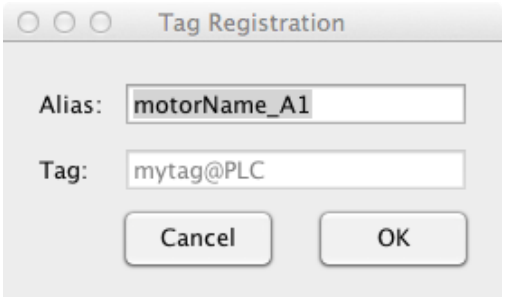
Note: 2>=1 logical operations always return binary result (0 or 1).

6. Tag Database

This feature is very useful for management of all tags and connections. You can start creating your project with creating the tag database and design the visualizations afterwards.



If you write a new tag anywhere in *myPROJECT Designer*, a dialog box will show up for adding this tag to the *Tag Database*:



Note: Instead of writing 'mytag@PLC' you can use the alias 'motorName_A1' anytime.

Click on **OK** to confirm adding the tag to the database. When the tag is created you can use its *Alias* as a reference.

7. Linking Views with PLC

There are two options to link your visualization with the PLC:

Animations

With animations you link the visual appearance of graphic objects with real values read from the PLC. The visual change is reflected immediately, e.g. you can show PLC tag/variable values in a text element; change the fill and stroke color of an object etc.

Effects

With effects you add dynamics to graphic objects. An effect is the visual appearance of an object in a specified time sequence and can be triggered either by a tag/variable from the PLC or user's action, i.e. *finger tap* or *mouse click*.

Example:

Imagine you want to display a blinking motor on your visualization if the tag/variable read from the PLC is equal to *1*.

1) Set the *Blinking* effect on the motor by specifying the blinking speed and infinite repeat. This way the motor will be blinking as long as the tag read from the PLC is equal to *1*.

2) If you use the *Color* animation instead, the motor will change its color if the value read from in the PLC equals *1* and returns back to normal when the value is *0*.

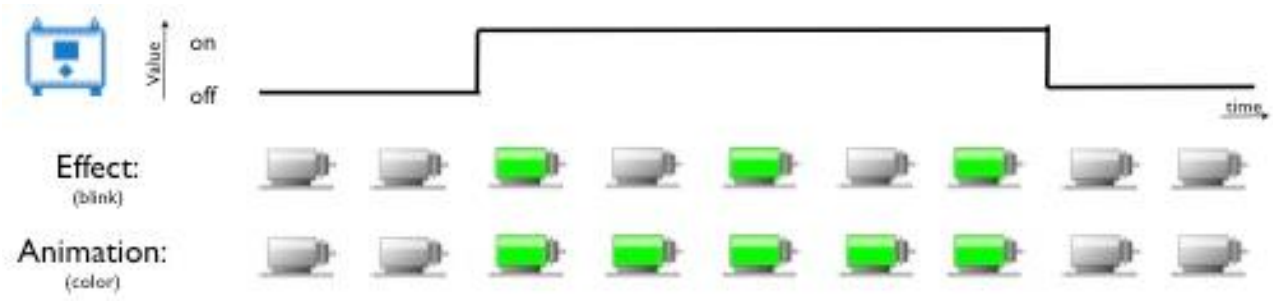
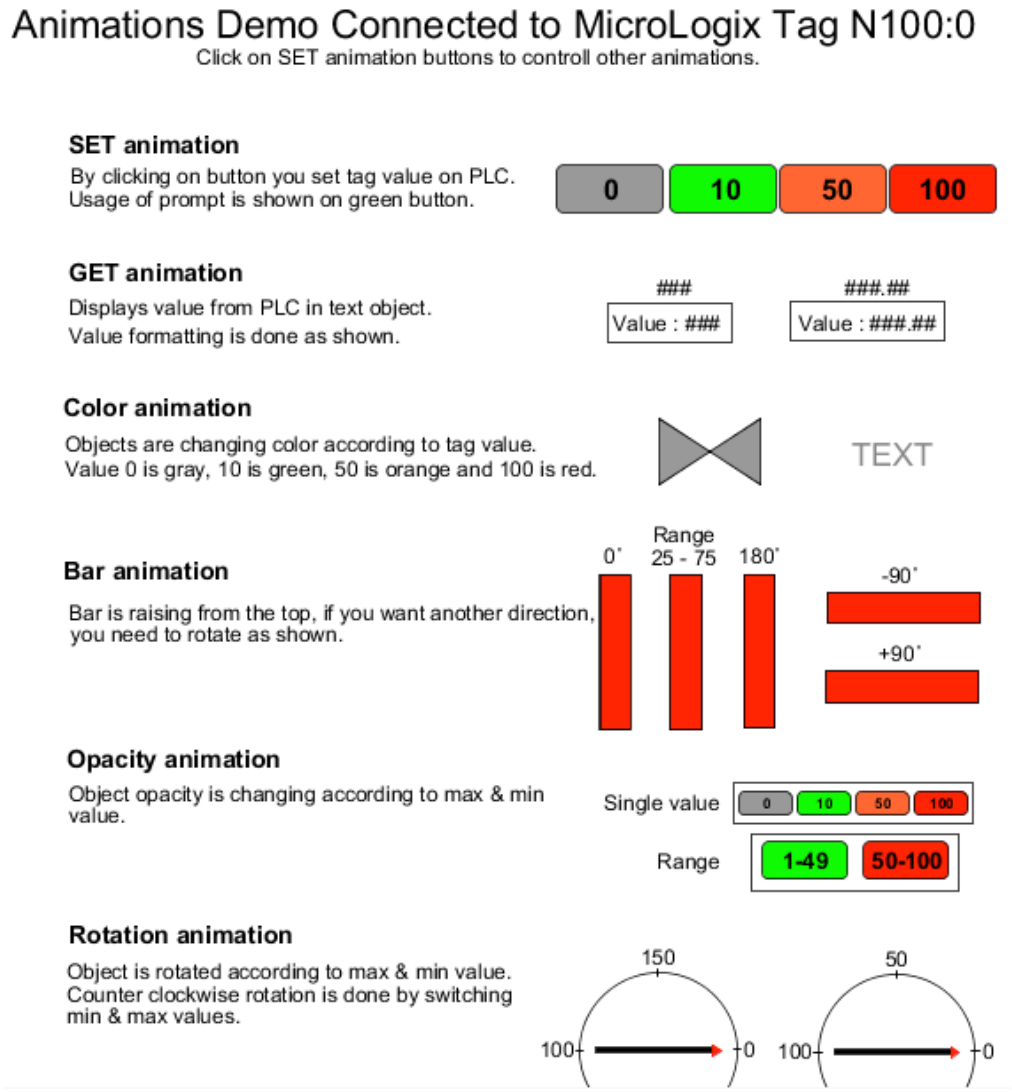


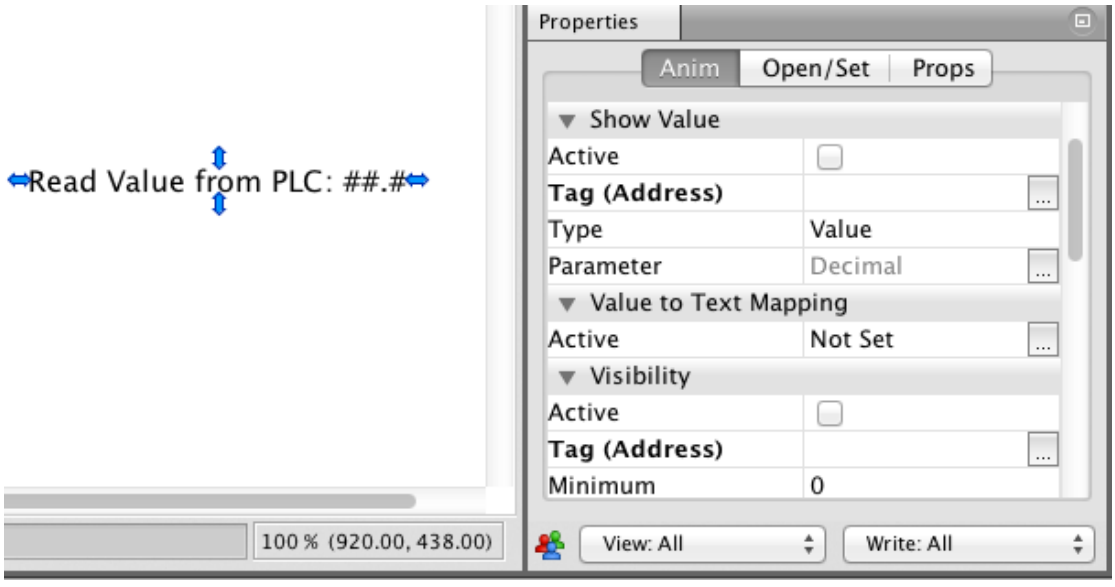
Figure 5 Difference between effect and animation

7.1 Animations



Getting started with Animations

In the picture above you can see examples of data reading from the PLC and its application to get color, scale, opacity, rotation and text animation. Create and select an object you want to apply animation to:



After selecting an object you will see **Anim** section in the *Properties window* where you can find available animations for the selected object.

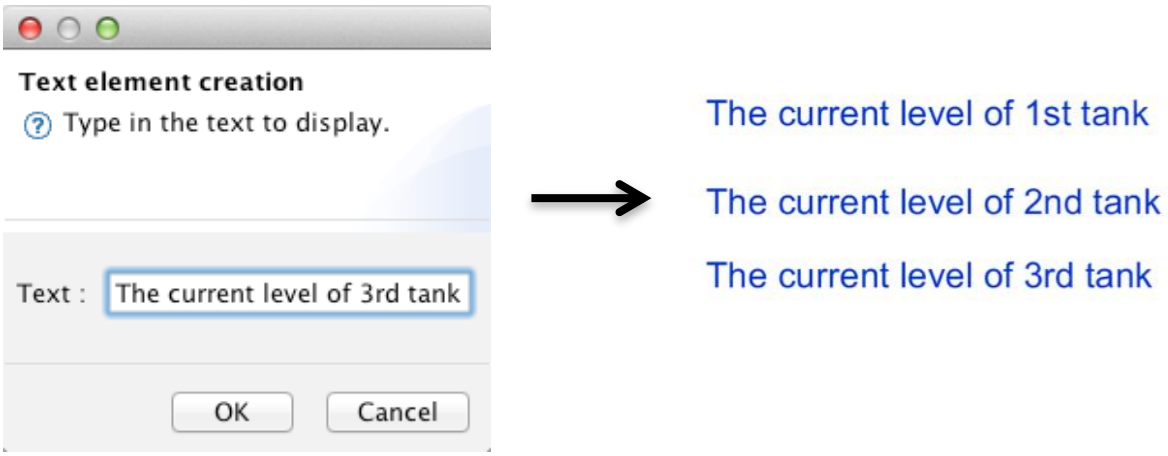
Show value

This animation can display tag values in a text element.

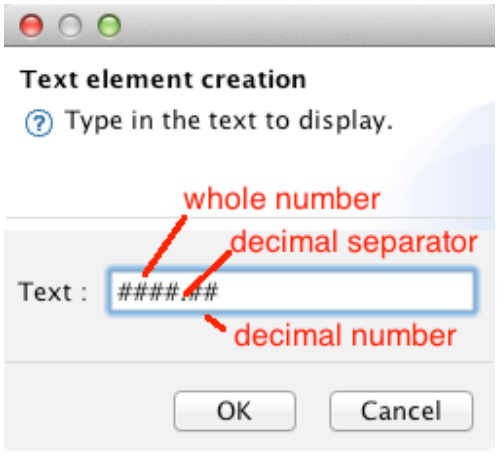
Example:

1) Let us assume there are 3 tags in *ControlLogix* PLC, representing the water levels – *level1*, *level2* and *level3*.

2) Create the text elements for these three water levels using the *Create Text Element* tool A.



3) Continue with adding the text elements, responsible for reading values from the PLC. This process is exactly the same as creating of the text elements. The text written to the text element specifies how the value read from the PLC will be formatted on the screen. The format specification is described in the following figure:

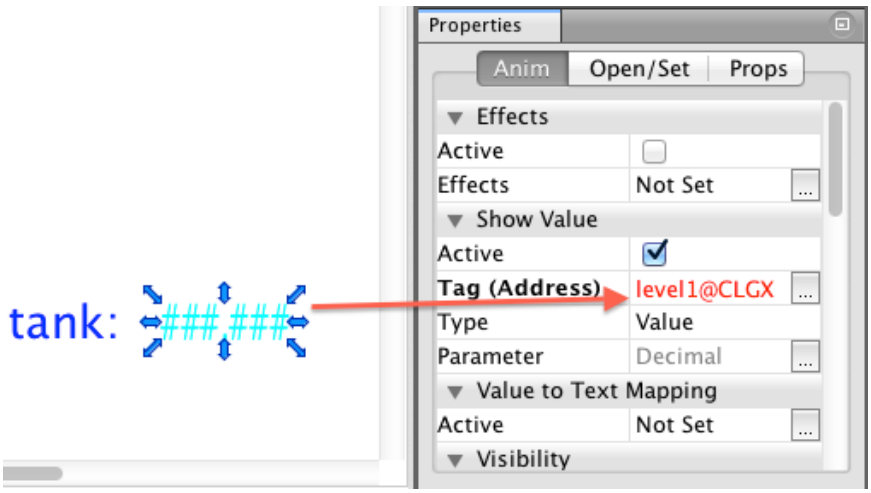


Note: Any value received from the PLC that does not follow the format specification will be automatically transformed to a set format. For example, if the PLC returns a value of 3.47 and the visualization expects only one decimal place with the format (##.#), then a displayed number will be rounded to one decimal place showing the value of 3.5.

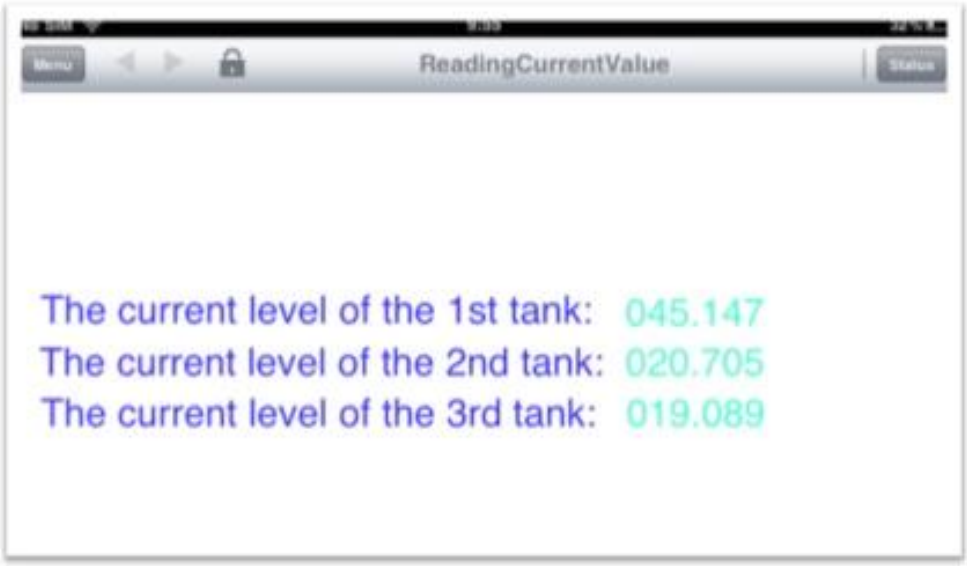
4) Connect the text elements to specific PLC tags.

Click on the text element that should be animated, according the data received from the PLC. You will see the animations properties in the *Properties window*. Select the **Anim** tab, in the ***Show Value*** animation section; enter the specific tag address that the text element will read from.

Note: If the entered tag address is invalid, the text will be marked as red.



5) Upload the view to your supported device - the screen should look as follows:

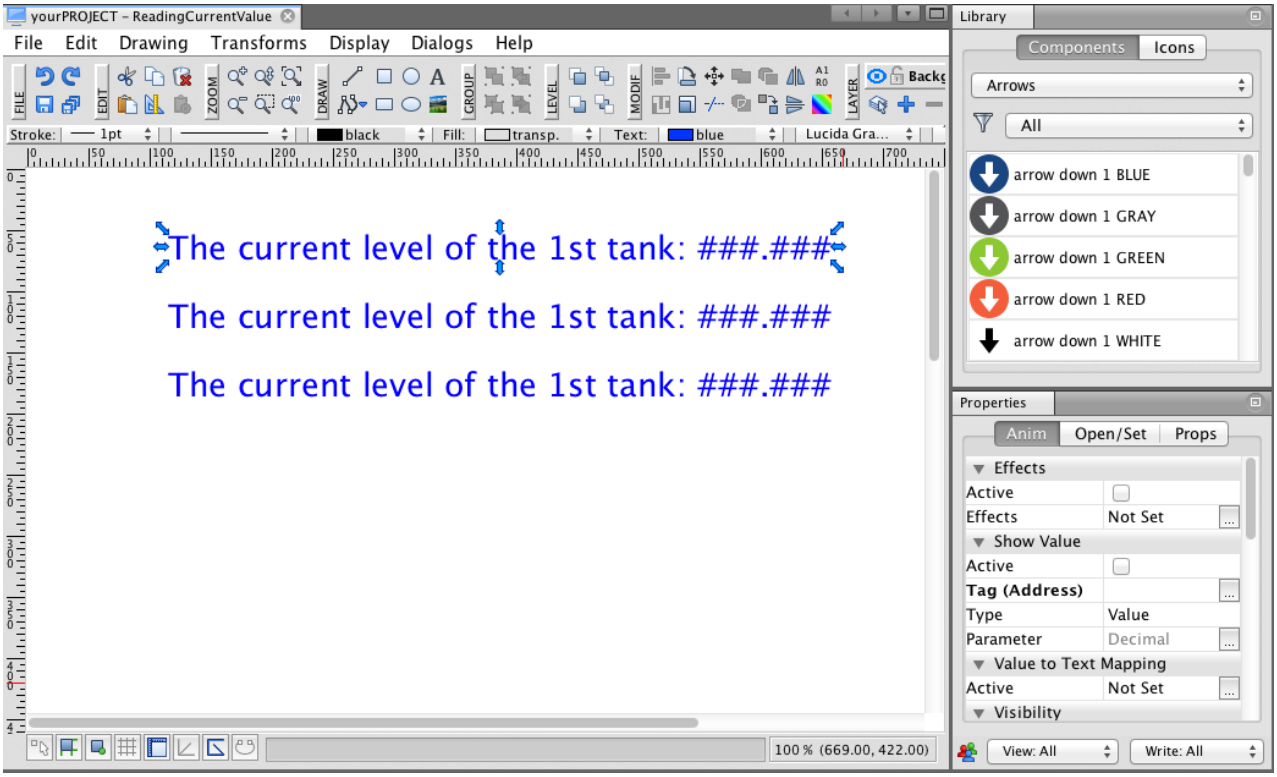


In the previous example we used the tags from *ControlLogix*. You can generally use tags for all supported PLCs, as long as you use a proper syntax (for a brief description of proper tag syntax see the chapter *Basic Tag Syntax*).

You can also combine a static text element with PLC values, read through the function *Get Animation.*

Example:

1) Let us create a text element *"The current level of the 1st tank: ###.###"* and duplicate into other two elements.



2) Now, after configuring *Get animation* in the way as described above, you will see the same visualization as for the text elements and values.

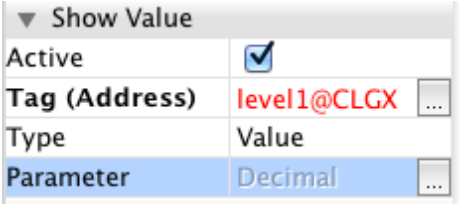
General

Id – each newly created component has an automatically assigned id, a unique identifier that can be overridden (renamed), you are forced to give each component a unique meaningful name

Type – indicates the component type, cannot be edited

You can combine multiple animations on the same object.

Options

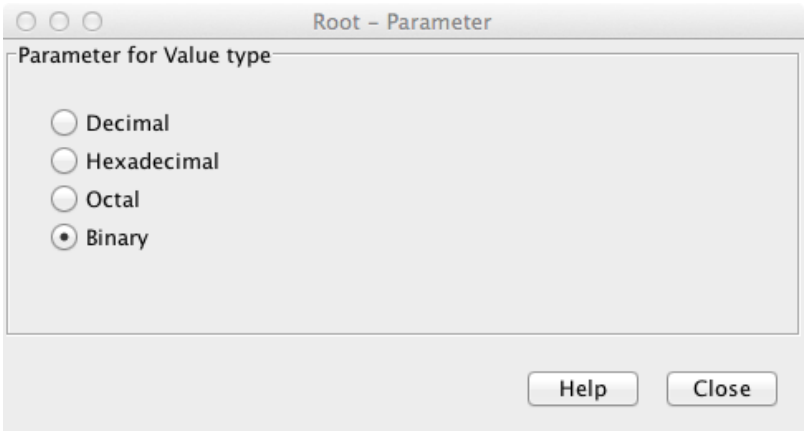


Type

With this option you can display data in 3 ways:

- **Value** - shows numerical values from the PLC

- Decimal
- Hexadecimal
- Octal
- Binary

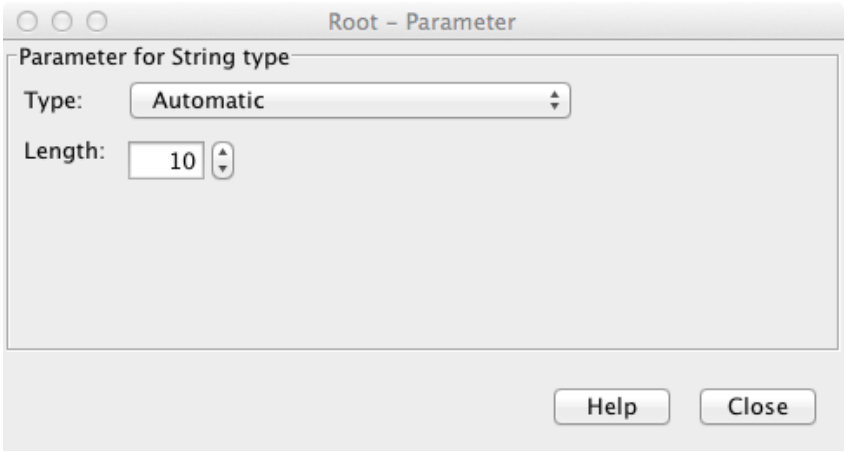


- **String** - shows string/text values from the PLC

  This enables you to read array data from the PLC and show as a string.

Automatic

The PLC type used in the tag definition selects the string type automatically



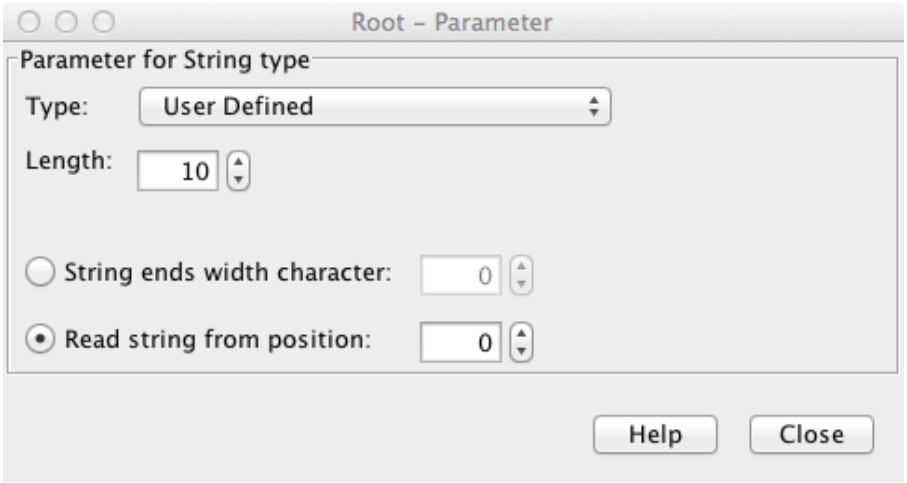
Rockwell: Rockwell family string, the first element in an array corresponds to the string length, the rest are ASCII encoded characters.

Siemens: Siemens family string

Modbus: Modbus family string, every 16-bit register corresponds to one character (encoding UTF-8), the string should end with 0.

User defined

You have a complete control of converting a PLC data array to a string. First of all you should specify the maximum length of the string. Then you can specify which character terminates the string or specify at which position the length of the string is encoded.



- **Date** - This enables you to read data array from the PLC and convert to the current date.

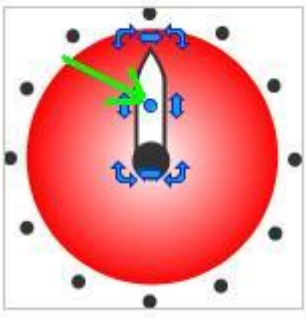
Rotate animation

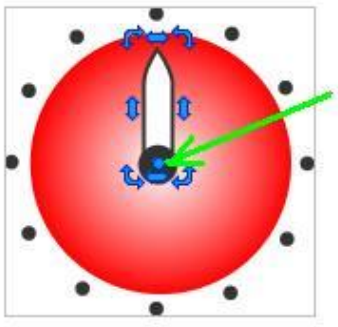
The rotate animation is a feature that allows you to rotate an object around the set axis, according to percentage of maximum and minimum values from 0 to 360 degrees (can be redefined by the *Range* parameter).

Enter the tag name and the minimum tag value, which correlates to the 0^{th} degree; and the maximum tag value, which correlates to the 360^{th} degree.

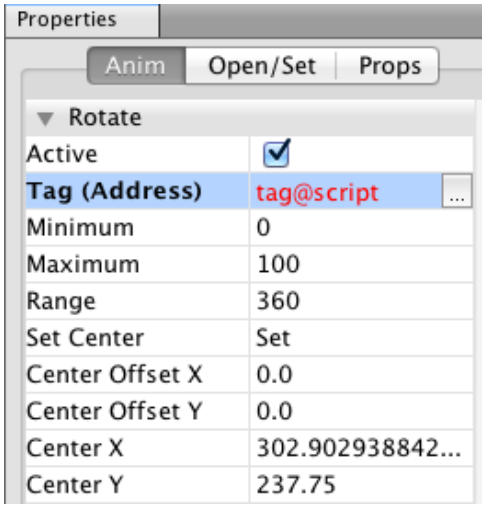
Example:

By default the rotation axis is in the geometric center of an object or group. You can change the rotation axis double-clicking on the object and set its axis, which is marked as a blue dot. Move the selected point to the desired position, as shown in the picture below:





With the set rotation axis you can create an animation to get arrow of the circular gauge revolve according to the input tag value. Go to the **Rotate** section of the **Anim** tab in the *Properties window* and fill in the **Tag (Address)**, **Minimum** and **Maximum** values.



Set Center property can be set to

Clear Center – clears the custom rotation center and sets to the default (geometrical) center

Cancel – closes the menu without any changes

Center Offset X (Y) fields show the difference between the custom and geometric rotation center

Center X (Y) fields show the absolute coordinates of the selected rotation center

Range property defines the total revolution angle, which is 360 degrees by default

Visibility animation

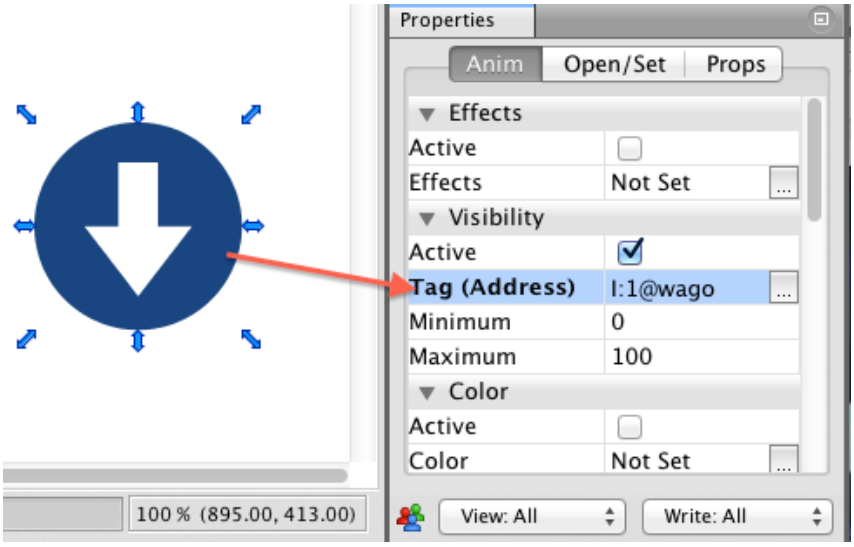
This allows you to control visibility of an object. There are 3 inputs required for this animation: **Tag**, **Min** and **Max**. When the value is within the min and max values range the object will be visible, otherwise it will be hidden.

To animate visibility, based on the discrete (*Boolean*) value set **Min** and **Max** to 1. This way an object will be visible when tag value equals 1 (TRUE).

Example:

1) Click on an object, which you want to animate – this will prompt the properties in the **Anim** tab of the *Properties window*. From here navigate to the visibility section.

2) Set the **Tag (Address)** and the **Minimum** and **Maximum** variables – if the tag value is within this range, the object will be visible otherwise it will be hidden.



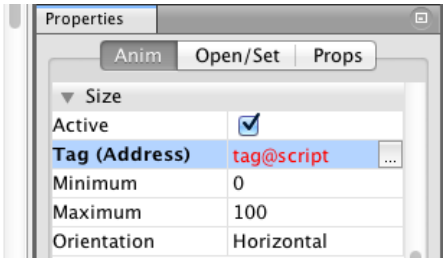
Size animation

This animation type is used for changing the size of an object, in correspondence with the entered value.

Example:

1) Select an object you want to animate. You can control the direction in which re-sizing will take effect (left to right, right to left, top to bottom, bottom to top).

2) Go to the **Size** section of the **Anim** tab in the *Properties window* and fill in the **Tag (Address)**, **Minimum** and **Maximum** properties. The **Maximum** property will correlate to the original height of the object. If an actual tag value overcomes the set **Maximum** value the animation stops at 100% the object's size, the **Minimum** correlates to the height equal to 0, if the tag value goes below the set value the size will stop at 0%. The orientation parameter controls if the change in size is in horizontal (width), vertical (height), or in both directions.



Horizontal direction

The base point of an object is located in the left edge and its width increases in the *left-to-right* direction. If you want to reverse the direction from *left-to-right* to *right-to-left* rotate the object by 180 degrees.

Vertical direction

The base point is located on the top edge and its height increases in the top-down direction. If you want to change the direction from *top-down* to *bottom-up* rotate the object by 180 degrees.

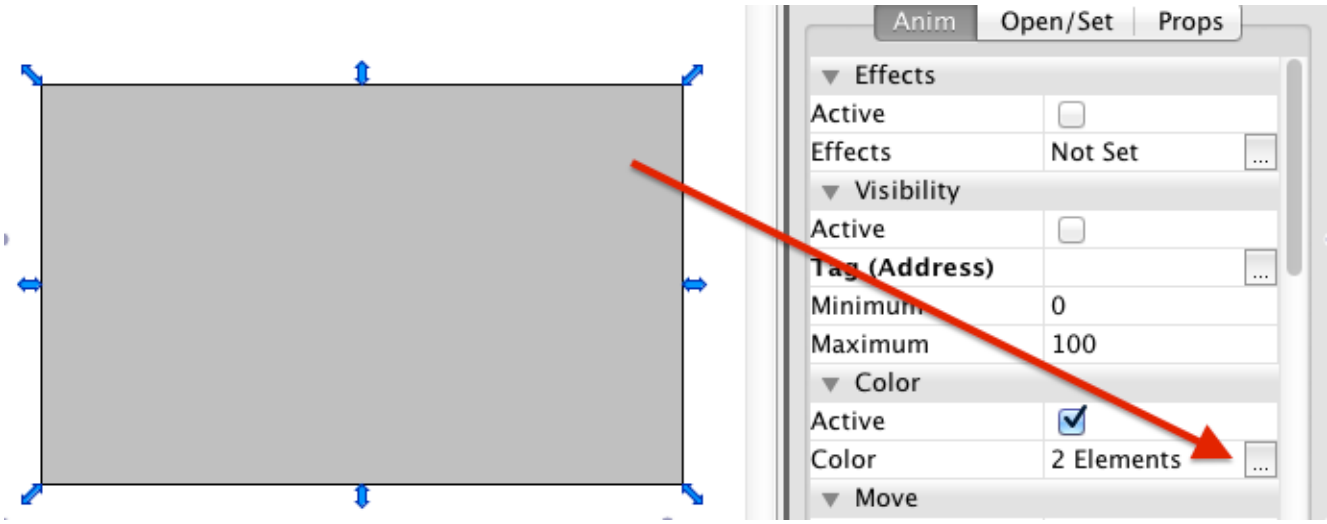
Color animation

This animation displays a certain color when the associated tag is in an acceptable value range. You can enter multiple conditions to achieve multiple color changes with one animation. The conditions are evaluated from top to bottom; therefore the bottom condition has a higher priority.

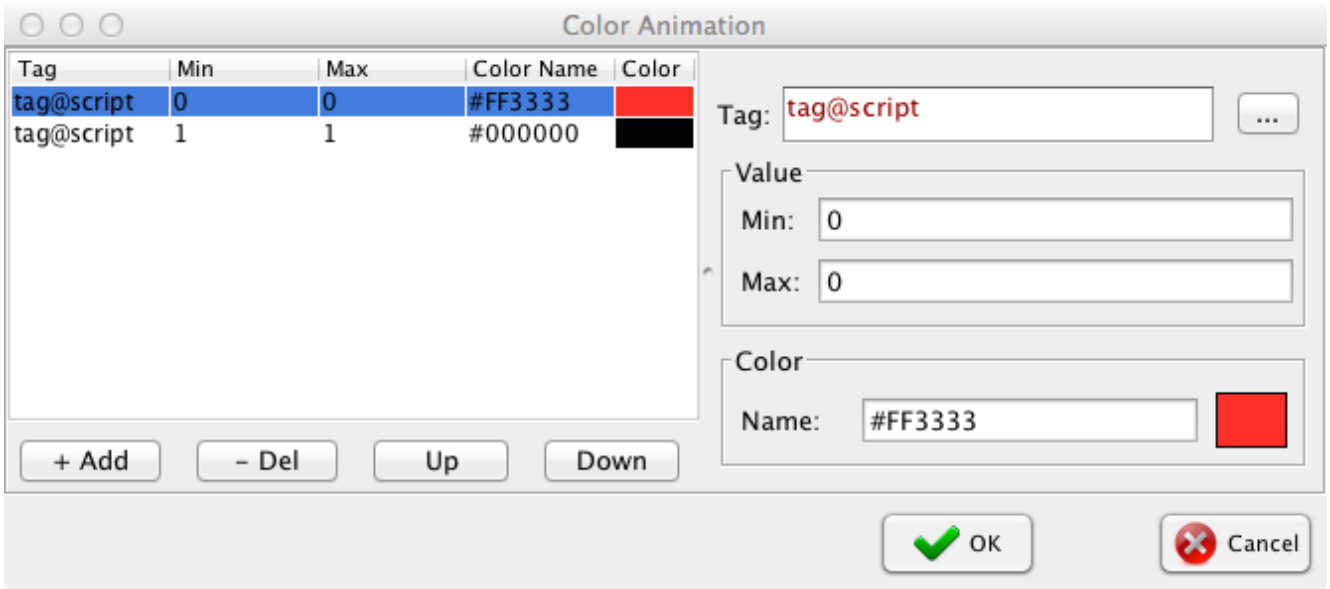
To animate colors based on a discrete (Boolean) value set **Min** and **Max** to 1. This way, an object will be visible when the linked tag value equals 1 (TRUE).

Example:

1) Select an object you want to animate, click on the **Color** tab in the **Anim** section in the *Properties window* and then click on the **…** button.

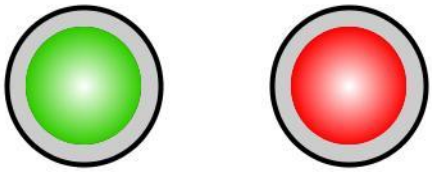


2) You will see a new dialog window, click on **Add** and fill in the **Tag**, **Min** and **Max** values. You can have multiple conditions. If the tag value is *0* the object will be colored red, if the value is *1* it will be black. Click on **OK** to confirm.

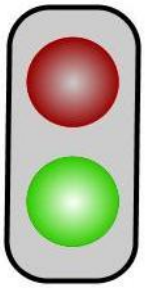


Examples of color animations

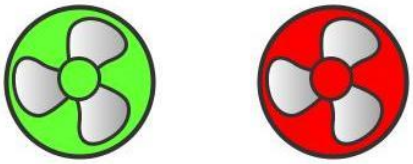
Buttons



Traffic lights



Fans



Valves

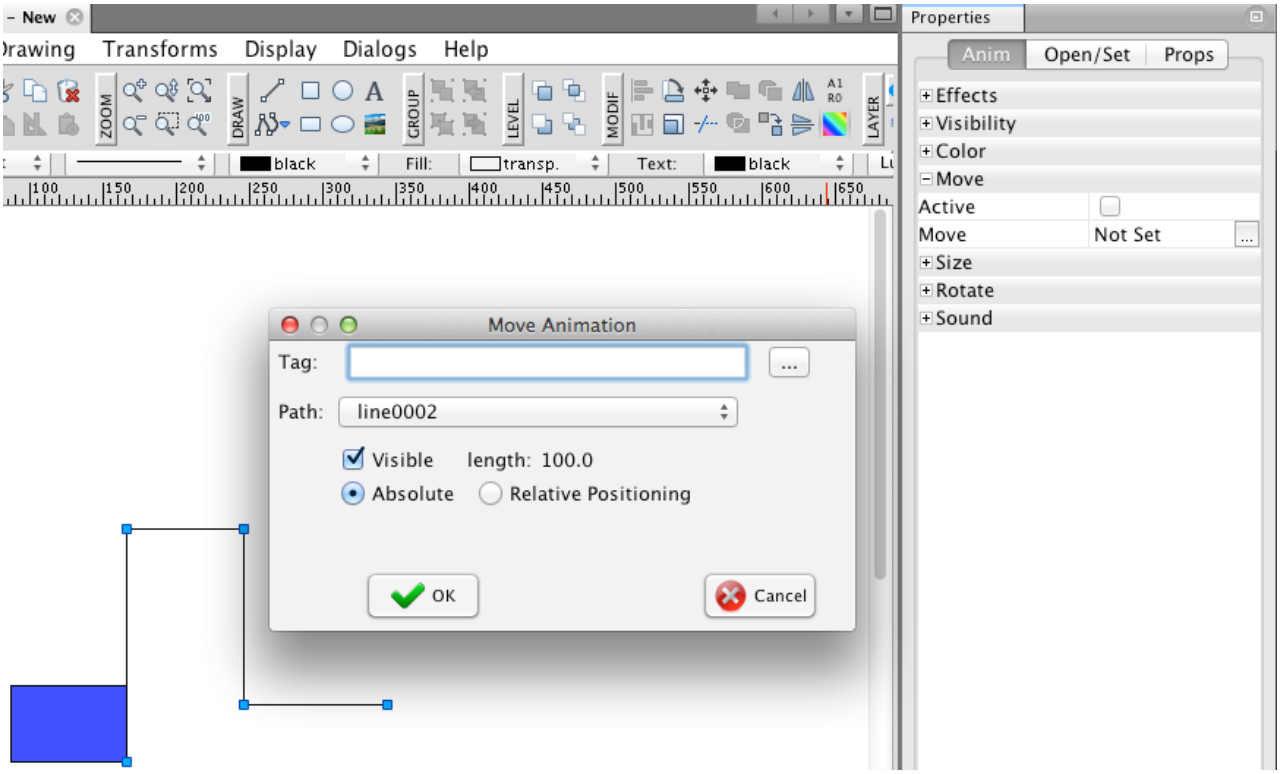
Motors

Move animation

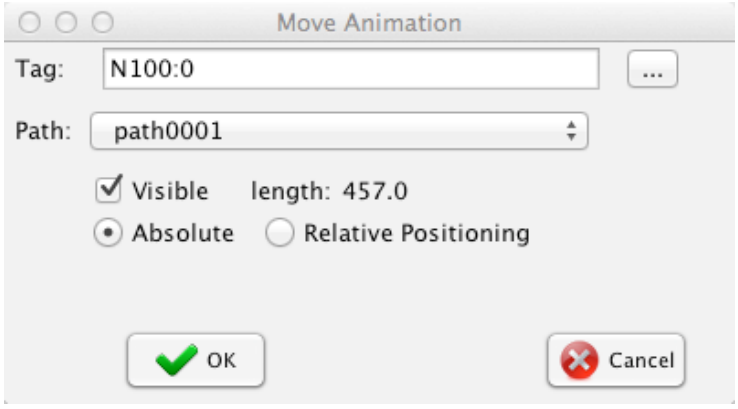
This animation is a programmed motion of an object along a defined path, based on the tag value.

Example:

1) Create an object and a path to move this object along. To add the *Move* animation select the object and click on the **Active** button in the **Move** animation tab of the *Properties window*, as indicated in the picture below:



2) Fill the tag address in the dialog window and select one of the paths available. Be aware that only UNCLOSED paths can be used for the *Move* animation.

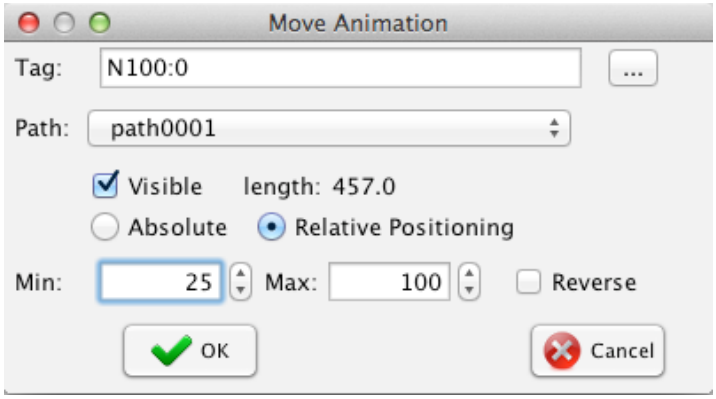


The other available options are:

Visible – move path remains visible if this option is enabled

Length – represents the length of a selected path, filled automatically and can't be edited

Absolute – this option tells mySCADA engine to map the whole path length for the *Move* animation up to the tag value of 1:1



Relative Positioning – this option allows using only a certain tag value threshold. The **Min** value represents the starting animation point (the object moves only when the tag value reaches this point), the **Max** value represents the ending animation point (the object moves to the the end of the path when the tag value reaches this point). The

Reverse - reverts the movement from *start-end* to *end-start*

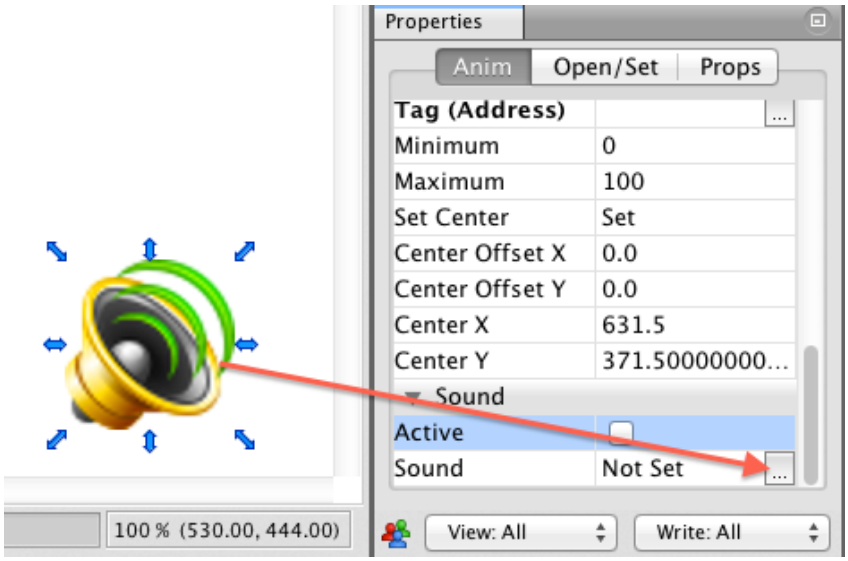
Note: A typical use of the Move animation might be moving objects on a conveyer belt etc..

Sound animation

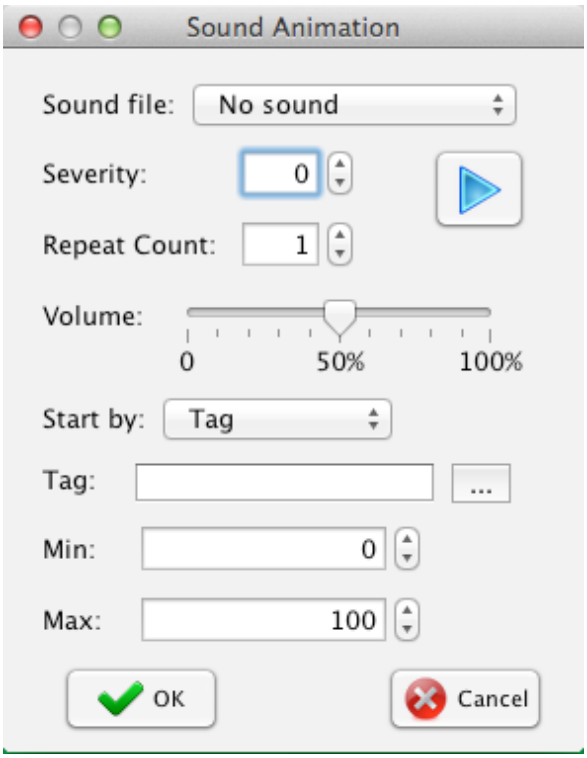
Thus animation can be used for notification of non-standard situations or voice announcements. You need to import sound files before you start to create the *Sound* animation, read the *Sound* chapter to find more details about the importing process and limitations.

Example:

1) Create a new element of any kind and click on the **Anim** *tab* in the *Properties window*.

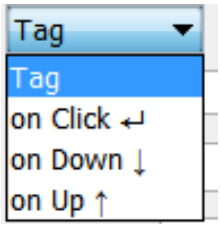


2) Select the **Sound** animation.



In the dialog window you can set:

- Sound files to be loaded
- Severity (priority), in case of simultaneous occurrence of multiple sound animations sound with numerically the highest severity will be played
- Repeat Count – number of repetitions of selected sound during animation
- Volume – defines relative volume (from system maximum) of a played sound
- Start by – sets the trigger of the sound animation:



Tag – sound will play once the selected tag value reaches the set range

on Click sound will play if you click on the object

on Down – sound will play if the object is pressed (click-and-hold)

on Up – sound will play if object is released after click or press

- Minimum and Maximum fields set range for tag value, when animation will be activated.

3) Click **OK** once you have set all the required parameters.

With illustrated settings sound will be played when tag value will be in range 150-333, with 3 repetitions of selected sound.

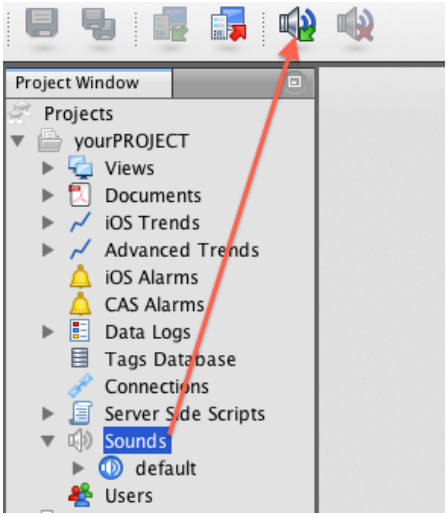
Sounds

You can create a sound animation, which will play loaded MP3 files on defined occasions.

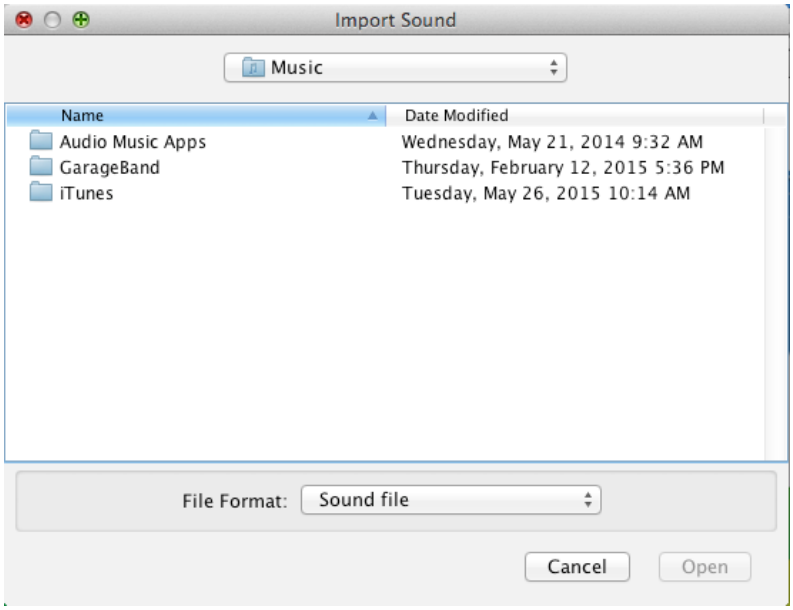
Note: Maximum size of the MP3 file is 3.5 MB.

Sound import

1) Select the ***Sounds*** folder in the project tree and click on **Import** in the main toolbar or right – click on the folder and select *Import* from the context menu.



2) The *Import Sound* dialog will appear.



3) Select the MP3 file from available folders.

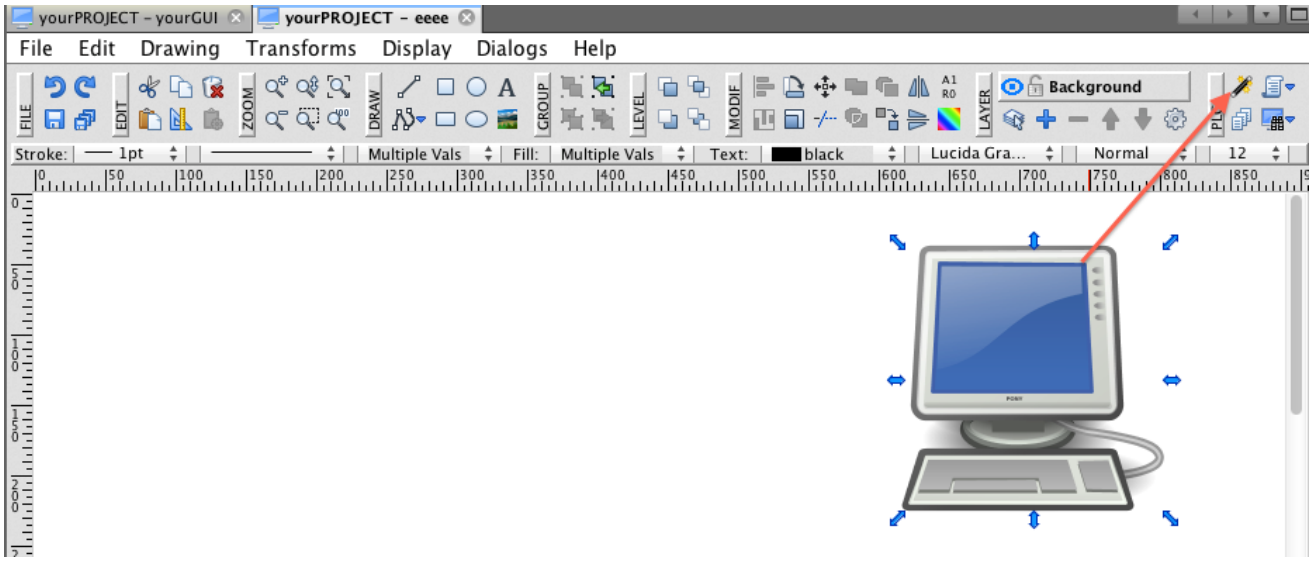
4) The sound file will be loaded into the ***Sounds*** folder.

*Note: If you wish to delete a sound file, select it from the **Sound** folder and click on the **Delete Sound** icon in the toolbar or select the Delete command from the right-click menu.*

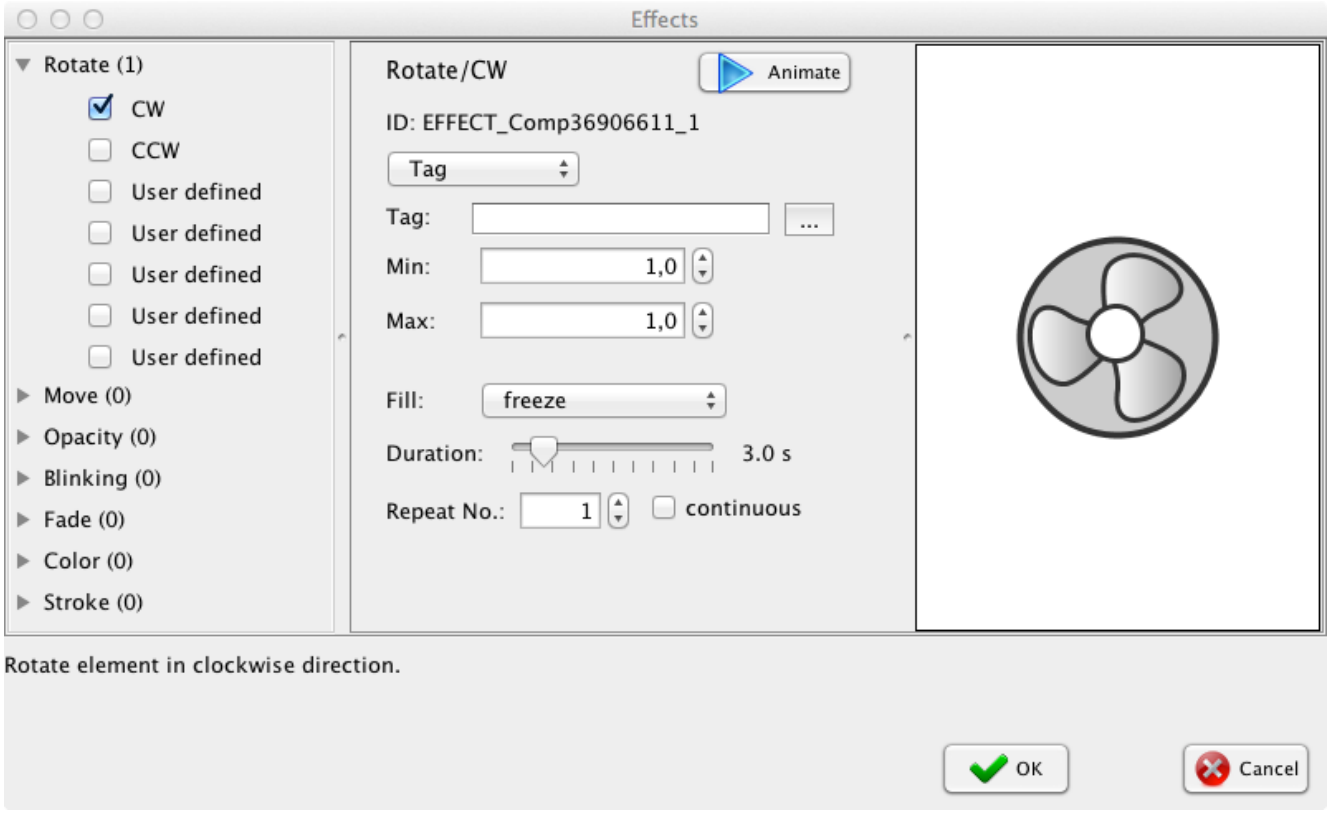
7.2 Effects

With effects you can add dynamics to your graphic objects, as described in the previous chapter (*see difference between Animations and Effects*).

1) Select an object you want to set effects on and click on the **Effects** button in the GUI toolbar.



2) A new dialog window will show up.



Note: If you do not select any object (or select more than one), list of all objects with effects will be shown.

The effects make the graphic visualization less static and their function is similar to animations, however they do not reflect the real state of a technological process. We advise using the animations for visualization of a precious technology state, and use the effects for visualizing the real-time change of objects.

The states of objects during an *animation* are based on actual values obtained from the PLC, for example: the motor rotation visualized through *Animations* behaves according to the values obtained from PLC, but the rotation visualized with the *Effects* is based on the values set by yourself, and will use the "hard" data only as a trigger.

The effects are grouped into 7 predefined categories with custom, user-defined settings.

Rotate – different rotation effects

Move – dynamic movement along the x and y axis

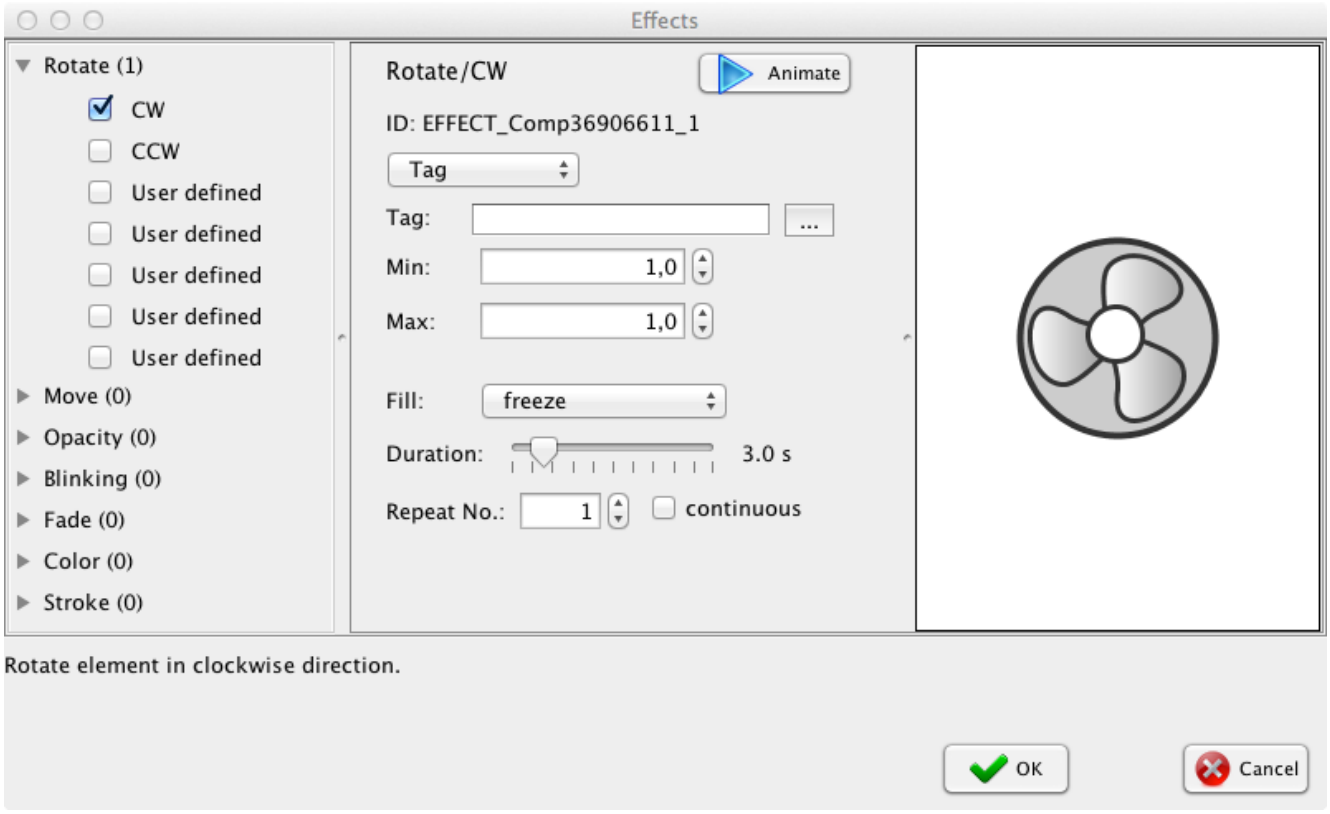
Opacity – dynamic visibility animation

Blinking – special visibility effects that allows visualizing blinking of object

Fade – fade object visibility in and out

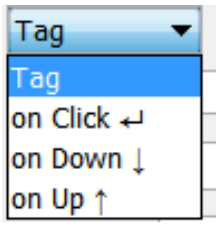
Color – change of an object color by morphing from the initial to the final

Stroke – change of an object stroke, improving perception of a selected object.



Animate button – allows you to see a preview of the selected effect.

In the context menu you can set the effect trigger:



Tag – effect will be activated when selected tag value will reach given range

on Click – effect will be activated when object is clicked

on Down – effect will be activated when object is pressed (click and hold)

on Up – effect will be activated when object is released after click or press

If you select ***Tag*** from the trigger menu, set the tag address and the range of the **Min** and **Max** values.

Note: If you select the User action trigger the fields mentioned above will not be available.

The **Fill** option lets you specify an event upon the end of an effect:

- *Freeze* – leaves an object without changes
- *Remove* – returns appearance of an object to its initial state (before animating)

Duration slider – sets duration of one effect cycle.

Rotate

You can choose from two preset effects:

- *CW (Clock Wise)*
- *CCW (Counter-Clock Wise)*

Additional options can be set for *User-defined* effects, where you can set duration of a single rotation cycle, number of repeated cycles (or infinite rotation) or different angular velocities for different parts of the rotation cycle.

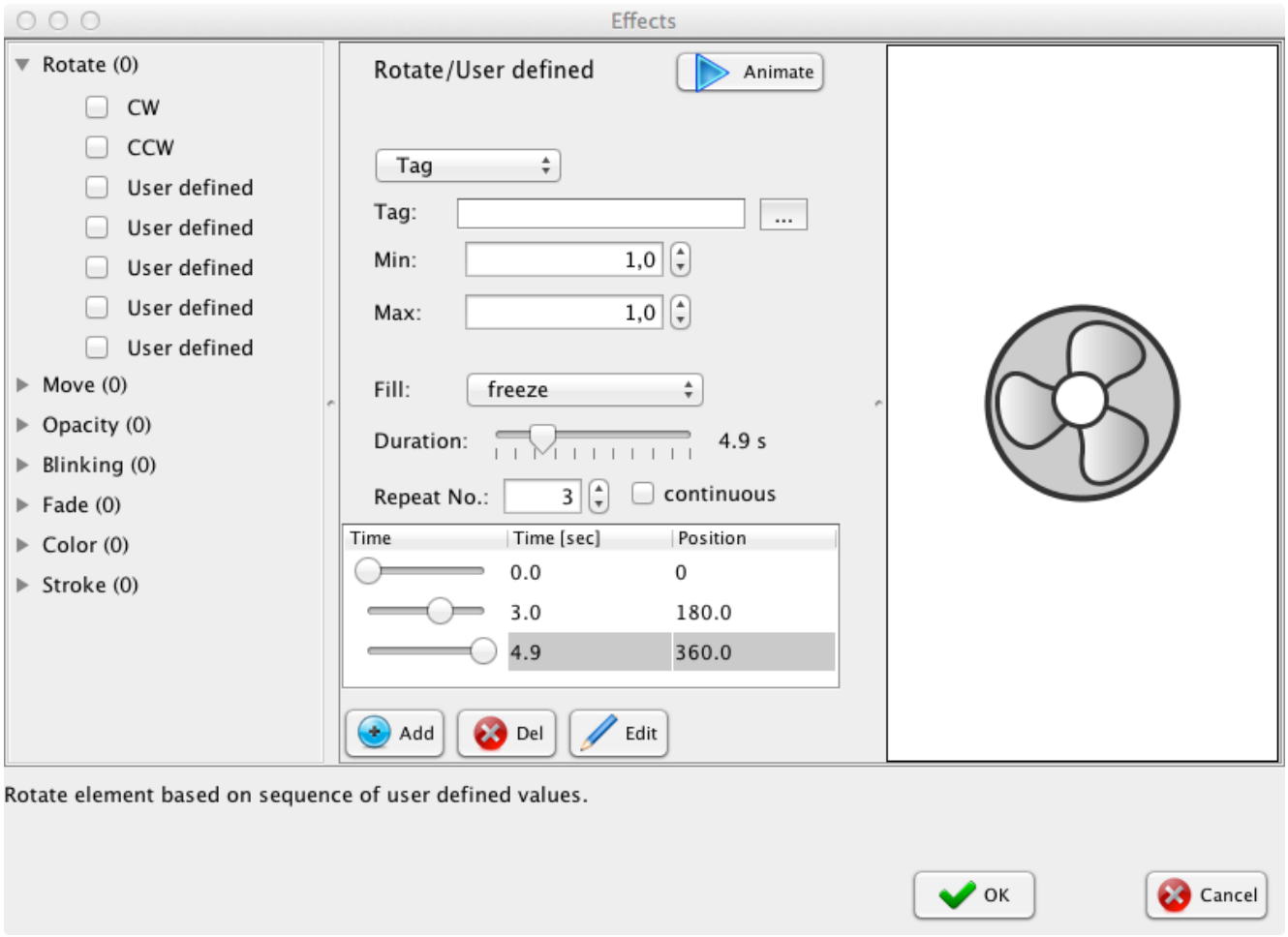
Repeat No. – allows setting number of rotation cycle repeating

Continuous – sets effect to an infinite loop

In the next block you can set duration for each cycle segment, you can add, remove and edit the fragments.

Example:

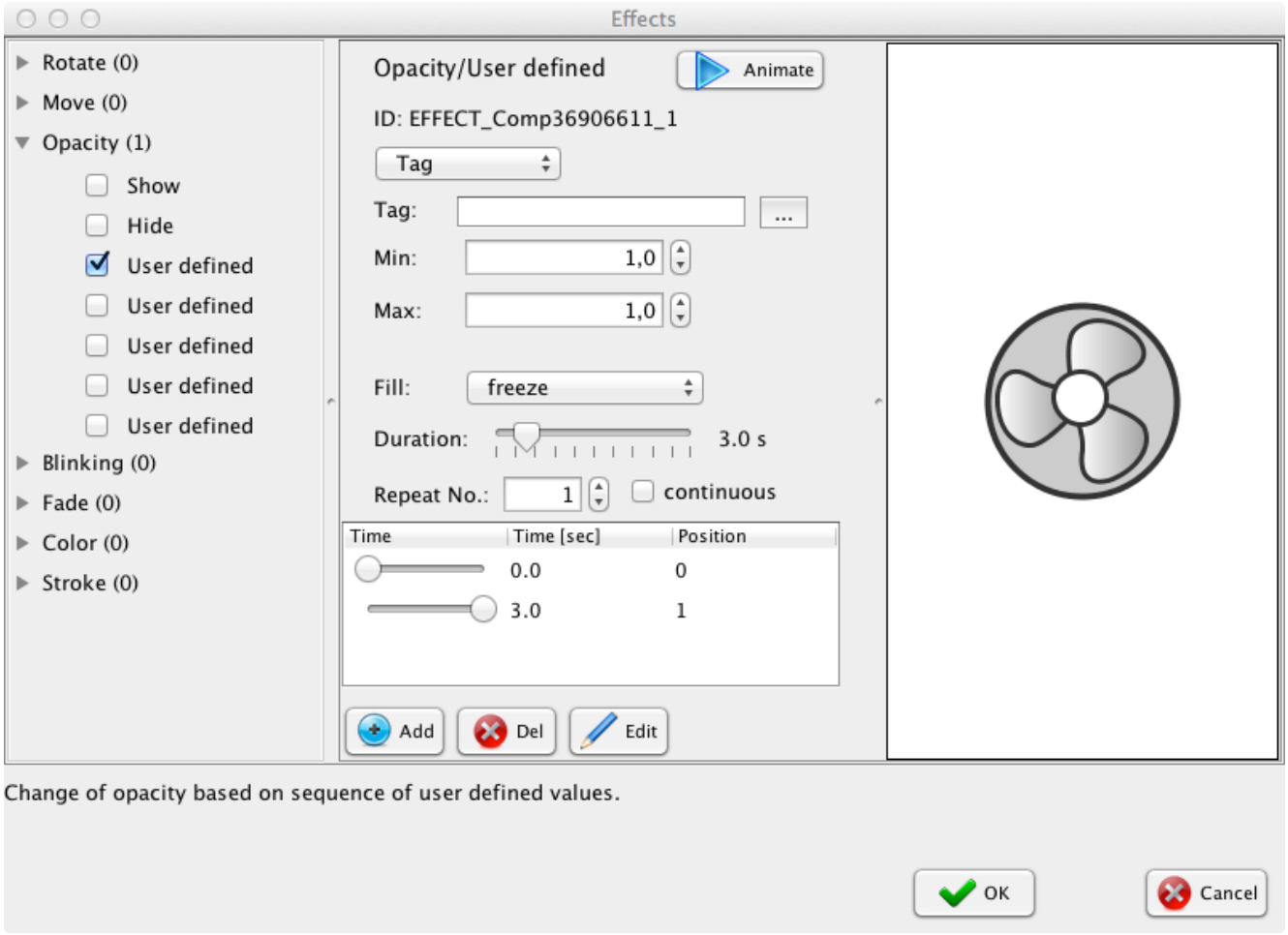
The following illustration shows the rotation effect, triggered by clicking on an object. It will last for 4.9 seconds and repeat three times. Rotation from 0 to 180 degrees will take 60% of the total time, resulting in a jump of the rotation speed.



Opacity

This effect changes visibility of an object. It has the same properties as the *Rotate* effect, which are described in previous paragraph. Transition settings are similar to that of the *Rotate* effect, but instead of the rotation speed it sets the speed of visibility change. For each change point the **Position** column should be filled with a value in the range of [0 , 1], where 0 means *transparent* and 1 means *fully visible*. The **Time** column allows to set duration of the transition between two change points. You can add, remove and edit fragments with the button right below the fragments tab.

There are two preset effect: ***Show*** and ***Hide*** that make a smooth visibility transition of an object in the set time. There are also additional slots for *User-defined* effects, where you can set duration of the effect cycle, number of the cycle repetitions and the transition speed between visibility states.



Blinking

This effect periodically changes visibility of objects. There are three preset effect: ***Slow, Medium*** and ***Fast*** that allow to set frequency of blinking. There are also additional slots for *User-defined* effects where you can set duration of the effect cycle.

Color

This effect changes the color of objects. There is one preset effect: ***Fill***, which makes a smooth change of the fill color in the set time. There are also additional slots for *User-defined* effect, where you can set duration of the effect cycle and the transition speed between colors.

For a predefined ***Fill*** effect you can set the target color to which the fill will change from the initial color set in the *Properties.* In the *User-defined* presets you can set different colors to be used.

Stroke

Those effects are very similar to ***Fill*** effect, they also change color but of object stroke instead of fill. Settings are same, as in predefined **Fill** effect. Please note that object needs to have stoke-width set higher than 0 for effect to take place.

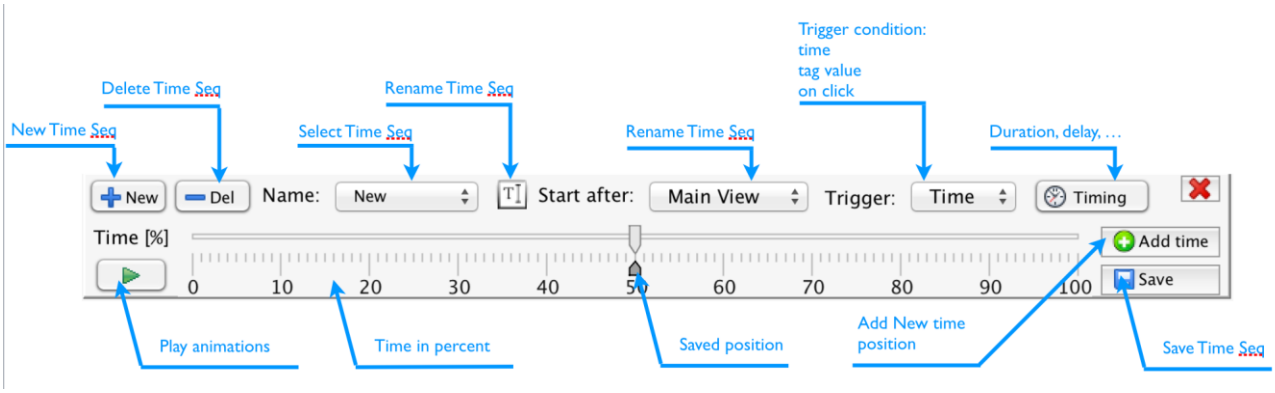
Note: You can combine multiple effects on the same object, but please be aware of overlapping behavior in such case.

7.3 Time Sequence

Time Sequence is a simple to use and extremely powerful function that adds timed animations to the views. The *Time Sequence* editor is implemented in the GUI editor and allows you change graphic properties of your objects in specified time intervals. *myPROJECT Designer* automatically computes the transitions between the time intervals.

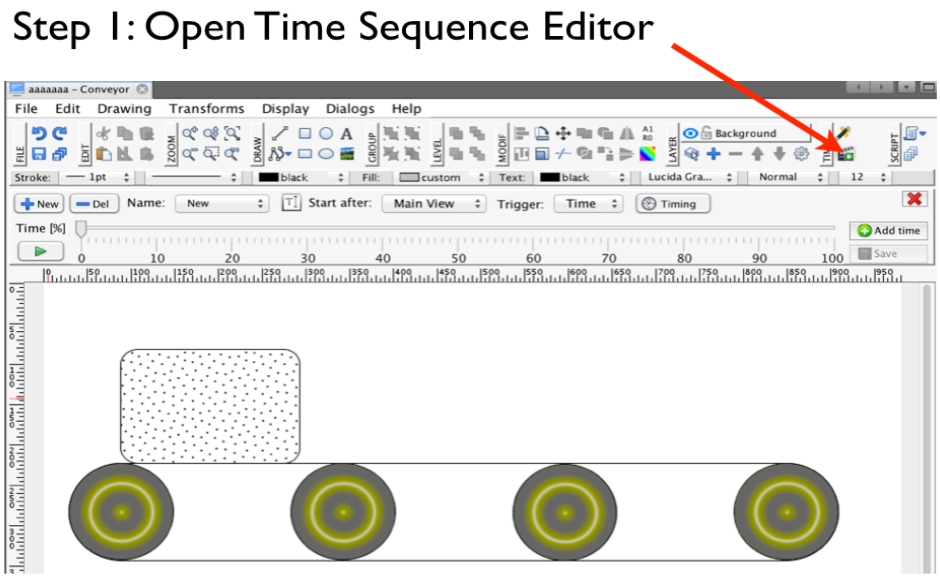
You can open the time sequence editor by clicking on the **Time Sequence** icon, located in the toolbar.

The GUI editor automatically adds the *Time Sequence* toolbar to the *Main Window*.

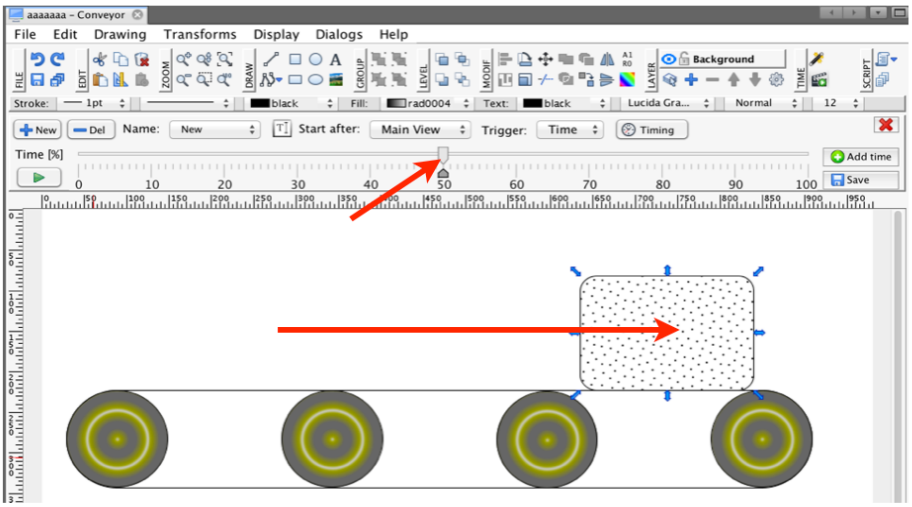


Creating a time-sequenced animation is very easy. You just move the time slider and for each time position modify the graphic objects as you want them to appear at these time positions. The smart algorithm inside the *myPROEJCT Designer* runtime will compute the animation so it is time-fluent.

Example:



Step 2: Move a Time Slider than modify objects



Allowed:

- position - move
- size change
- color change
- visibility

Prohibited:

- new element creation (please use visibility)
- rotation

Now if you play the animation the object on the conveyor will move.

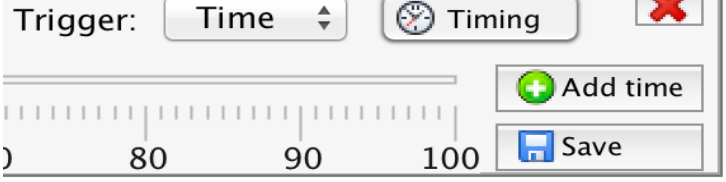
Additional options

Trigger:

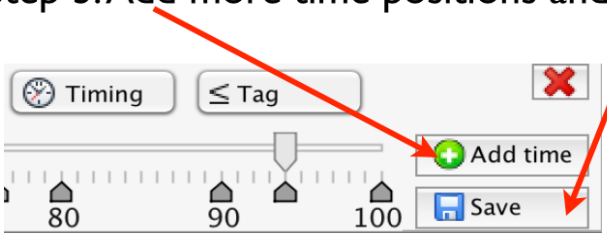
- *Time* the view is shown at
- *Tag* read from the PLC or a virtual (script)
- *Click* on any graphical object

Timing:

- Overall duration of sequence
- Time offset of the sequence start
- Number of repeats (or choose continuous)



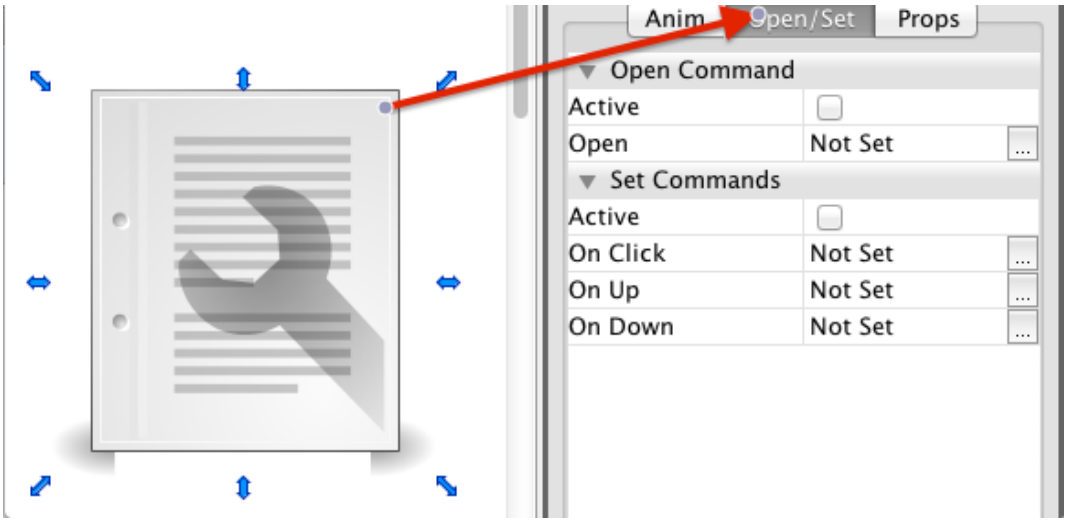
Step 3: Add more time positions and save



8. Open/Set command

The **Open/Set** commands group combines the *Open command* that opens *views*, *Documents* and other objects with the *Set command*, which is used for writing values to the PLC.

Select any object in the view, than open **Sets** tab in the *Properties window*.



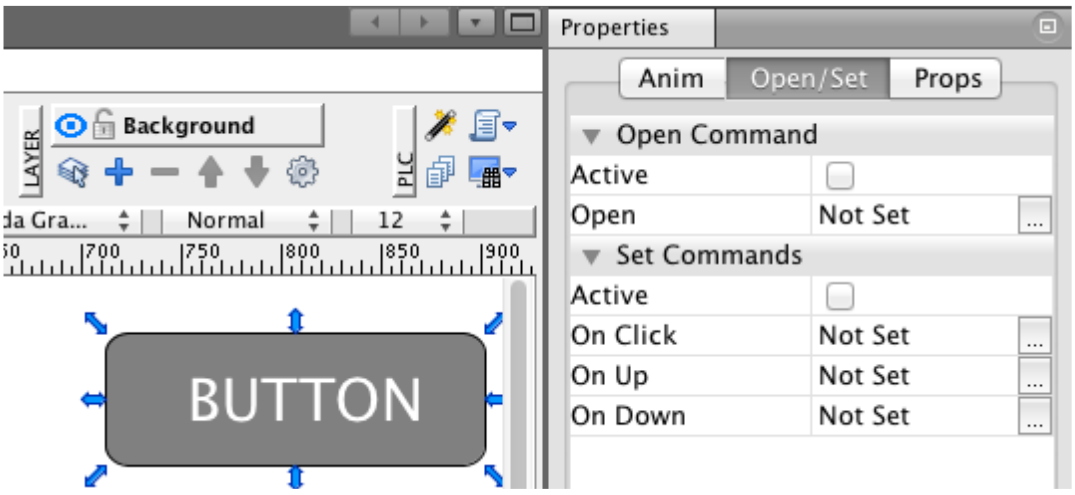
8.1 Open Command

This command is used for navigation among the HMI screens, e.g. you can open other project views from currently open ones.

The *Open command* prompts opening of other *views* and can be applied on any object. It can be used for attaching a detailed visualization to the whole process visualization, i.e. you can switch to detailed views of certain parts of a technology group.

You can configure objects to open a new screen by selecting them and proceeding to the **Open/Set** tab in the *Properties window*.

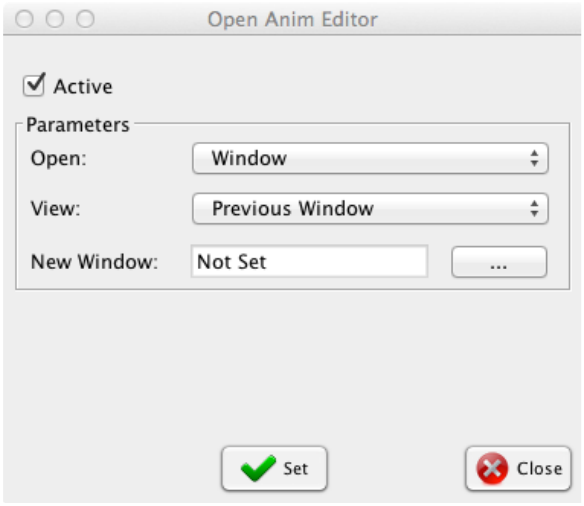
Note: It is possible to open the previous windows or any other created views of the project.



The *Open command* section properties are:

Active – check this box to activate the *Open command*.

Open – set or not set, press the **…** button to show the *Open Dialog* to set the parameters



– select the view type that will be opened by the command, available settings are:

Window – another view will be opened

Parametric Window – parametric window will be opened

Trend – trend will be opened (iOS only).

Advanced Trend – advanced trend will be opened

Document – opens one of the project documents, index sets on a page document will be opened.

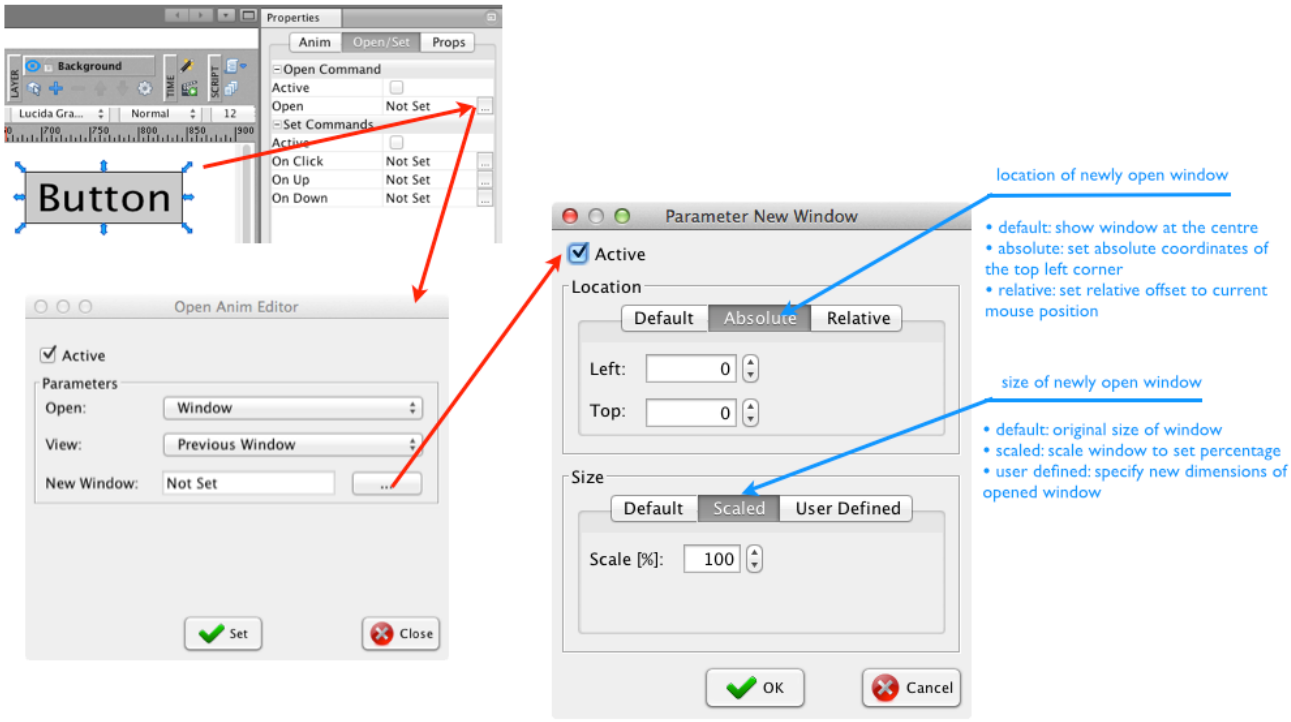
View – select a view that should be opened by the command, choice of options depends on the view range, already defined in a current project and the selected *Open* property, when **Open** is set to *Window* only the views will be available for selection, on *Open* set to *Trend* you will be choosing from available trends only etc.

Page – this property is relevant to the *Document* command where you set the page number, which your document should be opened on

Open in New Window (Faceplate)

1) To open in a new window, click on the **...** button next to the **New Window** button

2) A new dialog window will show up - here you can specify additional parameters:



Location – sets the location of a newly open window

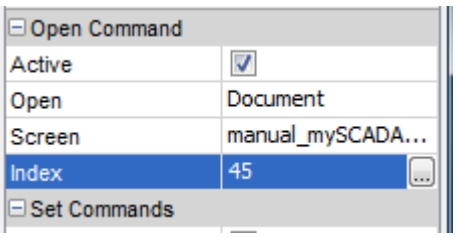
- Default: show window at the center
- Absolute: set absolute coordinates of top left corner
- Relative: set relative offset to current mouse position

Size – size of a newly open window

- Default: original size of window
- Scaled: scale window to set percentage
- User Defined: specify new dimensions of opened window

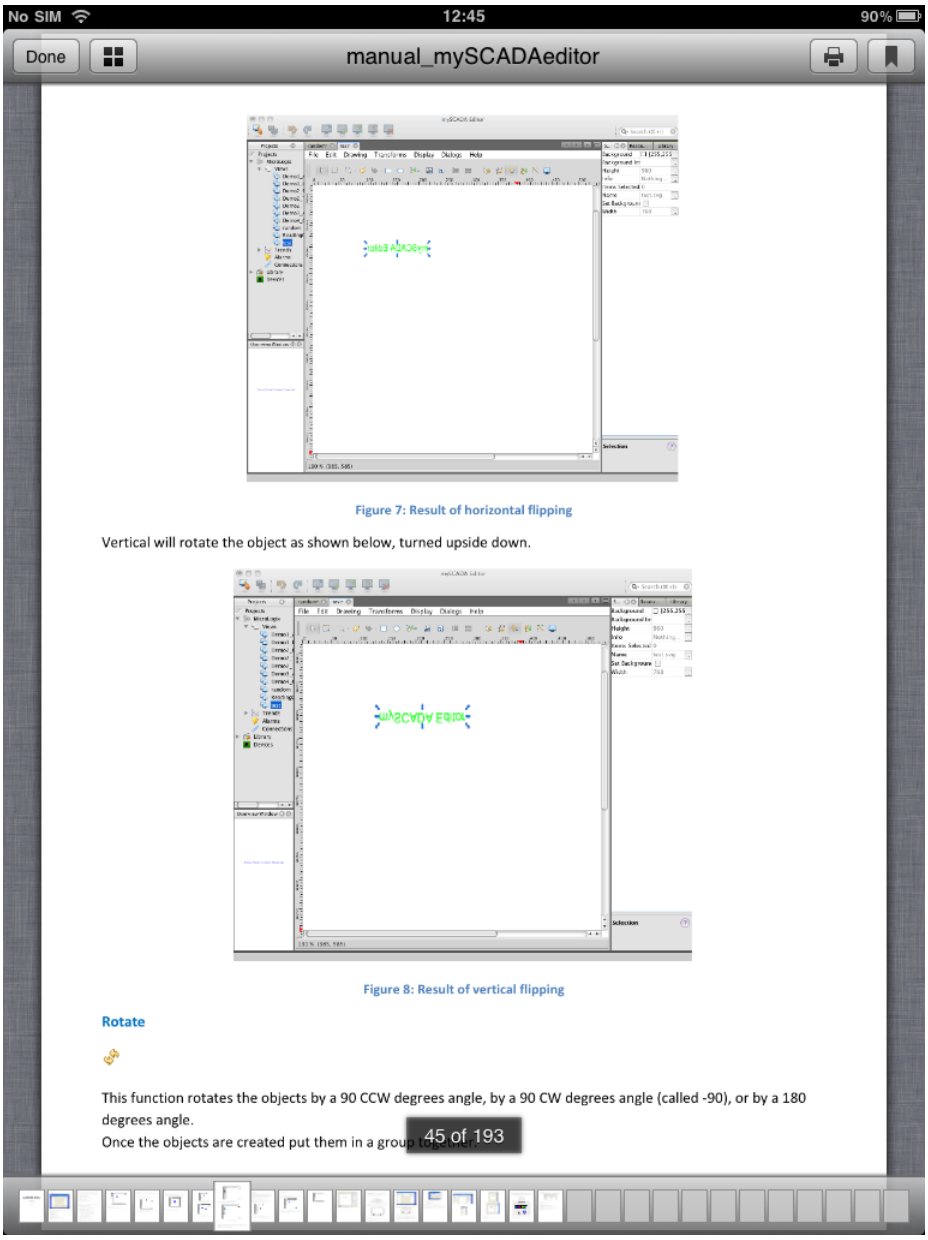
Open document

This function is useful when you need to link some part of your visualization with some document. For example you can attach maintenance manual to controlled device visualization, so operating person would be able to access required information in a much convenient and faster way.



Example:

The following attached document will open on the page 45.



9. Set Command

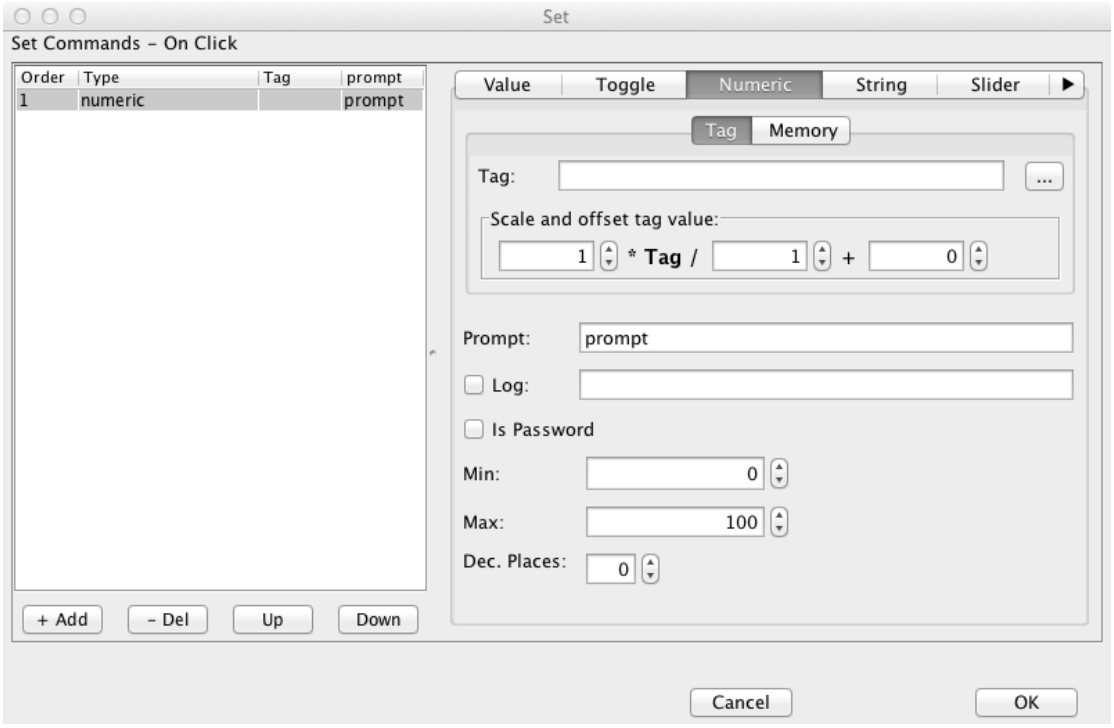
9.1 Writing Values to PLC

It is possible to configure some objects to actually write new values to the PLC. The ability to toggle a specified bit, write a static predefined value, write any value given by the user or write any value given from multiple choices from a user-created list are all possible.

There are three types of triggers available for each object and each of them can have its own dataset to write. The data can be written on **on Click**, **on Up** and **on Down** actions. **On Up** and **on Down** triggers are independent, so you can write two different sets of data with one button by applying different actions to it.

- ***on Click*** – Set command will be activated when an object is clicked on
- ***on Down*** – Set command will be activated when an object is pressed (click and hold)
- ***on Up*** – Set command will be activated when an object is released

All three triggers have the same properties. See the ***on Click*** command description below:



You should set the tag address first, with the parameters **Scale** and **Offset** you can change user input before sending to the PLC.

You can add any message you want to prompt on using the *Set command*. If you want no confirmation prompt upon *write* – leave this field blank.

If you want the *write* command to be logged tick off the **Log** checkbox and fill the message field.

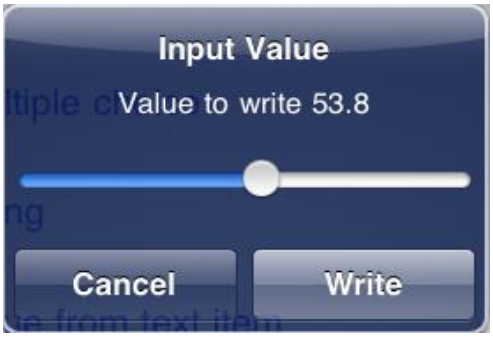
Values for the *Set command* can be obtained in several different ways:

- ***Value*** – static set value will be send to the PLC

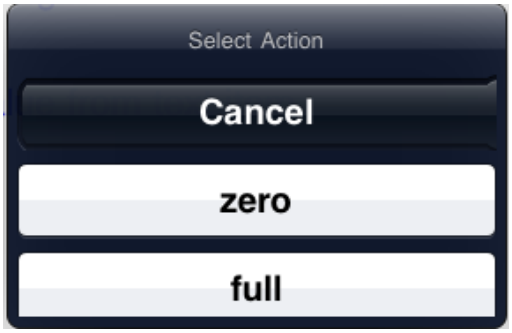
- ***Toggle*** – binary switch, set 0 to 1 and 1 to 0, should NOT be applied to non-binary variables
- ***Numeric*** – prompts a dialog in the visualization, where the user can enter any numeric value, there are options to treat such dialogs as password input (hiding input values), and to set allowed range of inputs with **Min** and **Max** limits and a number with decimal places

- ***String*** **–** prompts a dialog in visualization, where user can enter any string value, there are options to treat such dialogs as password input (hiding input values), and to set allowed number of letters with **Min** and **Max** limits, predefined text can be set also with **Predef. Text** option

- ***Slider*** – prompts a dialog in visualization, where user can select any numeric value with a slider, additional options allow setting **Min** and **Max** limits of the slider value, the decimal places allow precision between two slider states



- ***Multiple choice*** – prompts a dialog in visualization, where user can select one of the predefined values, which should be entered as tag value/text pairs during the animation setting



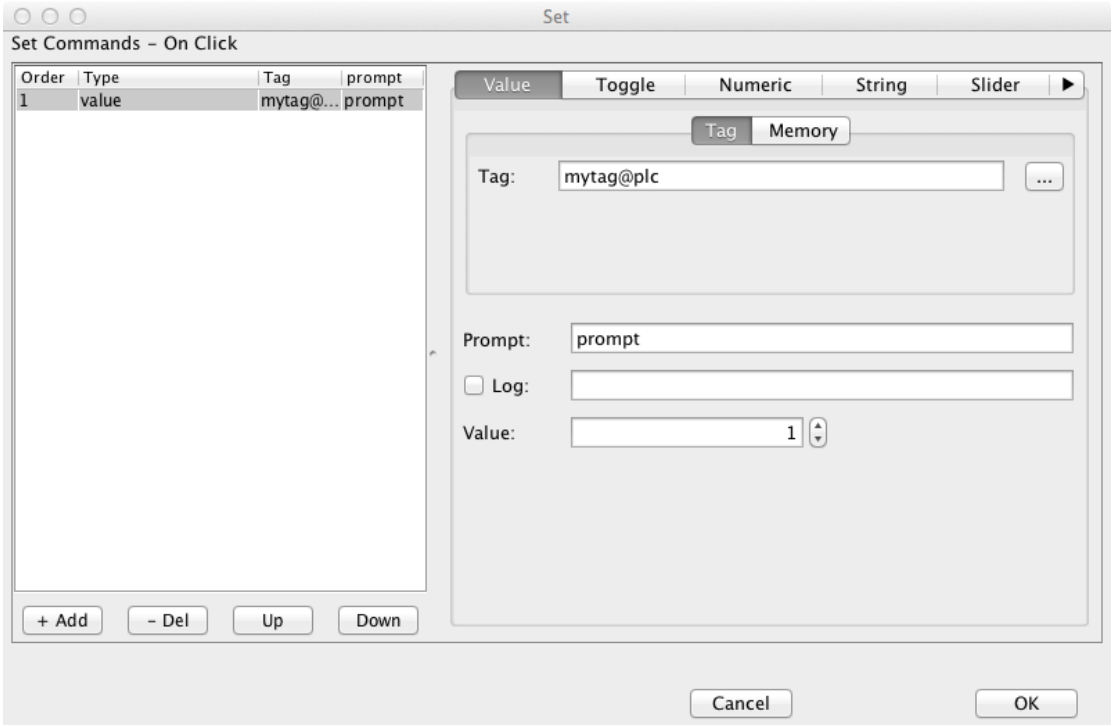
- ***Value from text item*** **–** writes a value to the PLC/variable from the text element on the screen

Writing predefined value

Objects can be configured to write a predefined value to the PLC when they are clicked on.

1) Select an object and click on **Open/Set** tab in the *Properties window* and in the *Set command* section select the ***On Click*** tab - the ***On Click*** dialog window will appear.

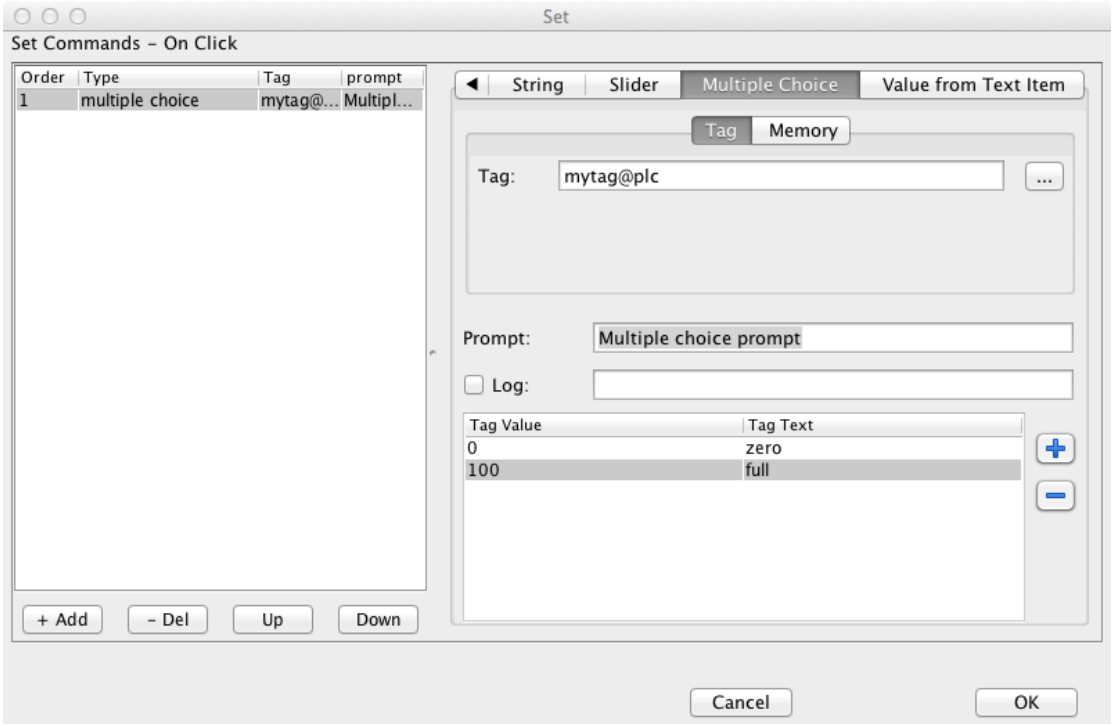
2) Click on the **Value** tab, enter the tag name/address and, if desired the *Prompt* message. Finally enter the value you wish to send to the PLC and click on **OK**.



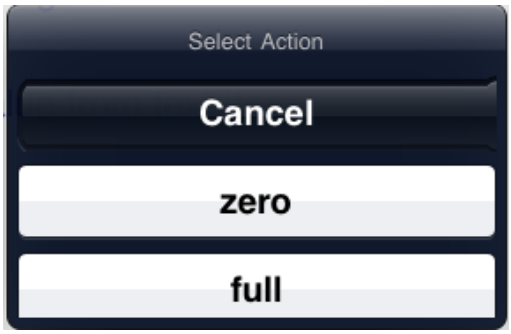
Writing value from predefined list

This configuration can be selected when selecting the **Multiple choice** tab and filling the table with possible values along with corresponding texts that will be displayed in a dialog during visualization.

Example:

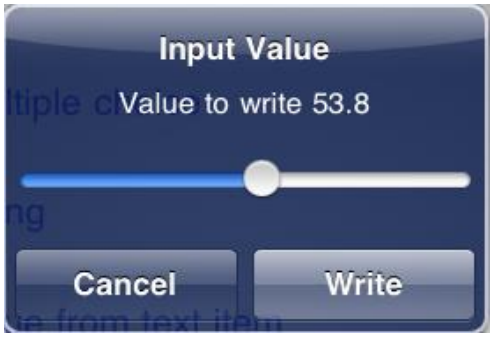


Clicking on **zero** sends 0 to the PLC, while the **full** button sends 100.



Writing numeric value with slider

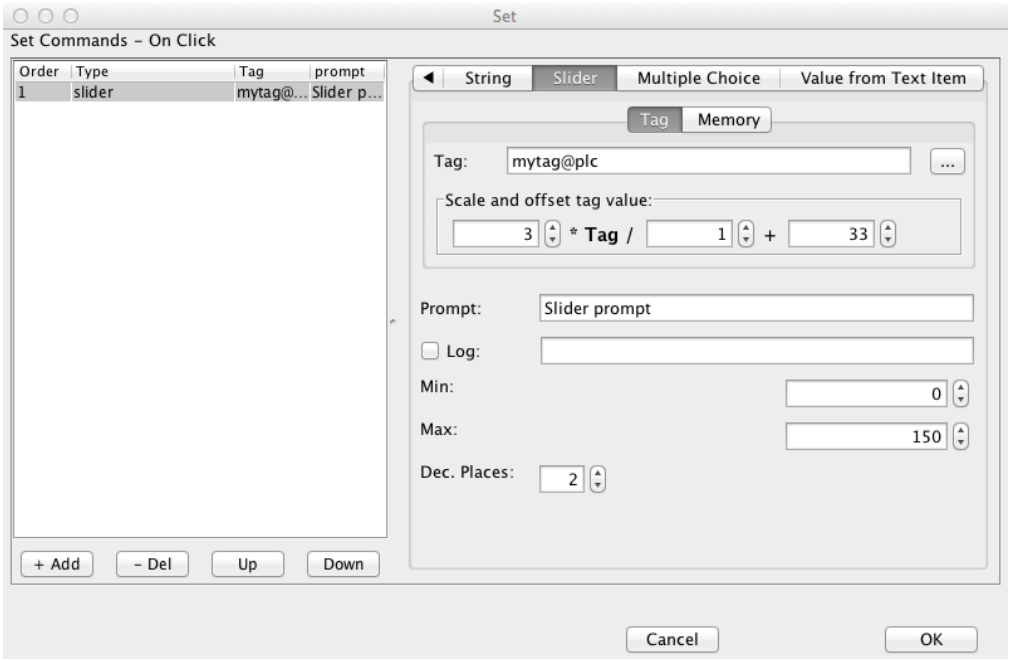
The next possibility is to write an arbitrary value to the PLC. To activate this configuration setting select one of the *Set command* options and activate the **Slider** tab. Enter the range of **Min** and **Max** the user will select a value from.



Example:

The picture below shows the configuration setup so that the ***on Click*** command prompts the *write* dialog where the user can write a value in the range of 0 to 150.

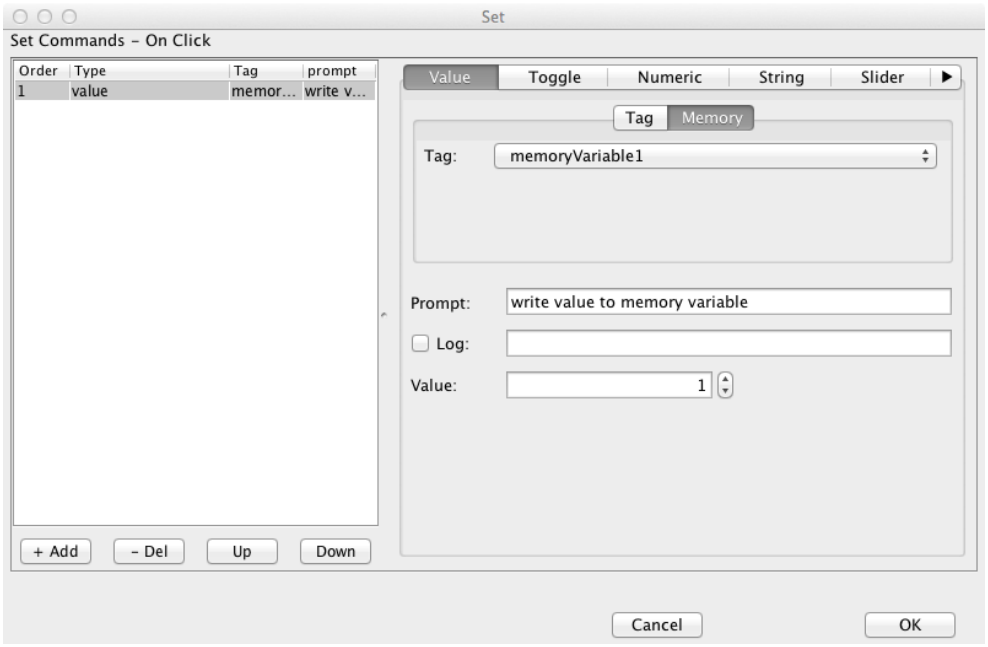
The resulting value will be multiplied by 3 and increased by 33 before writing to the PLC as a result of the **Scale/Offset** addition.



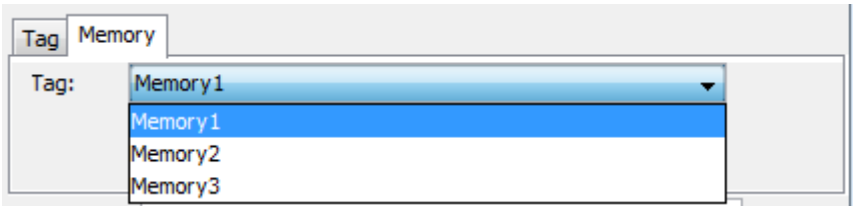
Writing to value memory

With the *Set command* you can also write variables to the value memory of the script. The memory variable needs to be defined before you add the *Set command*.

1) Click on the **Memory** tab to add 'write value to memory variable'.



2) Select one of the defined memory variables from the list.



The rest of the parameters are the same as for the regular tags.

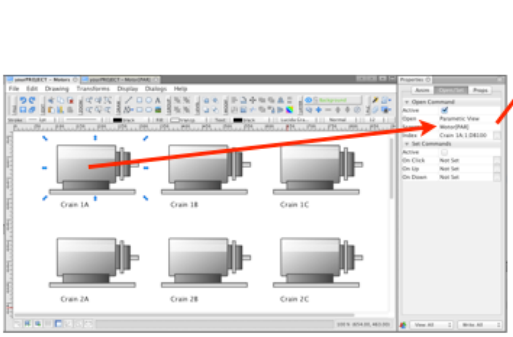
3) Click on **OK** to create the *Set command* or **Cancel** to leave without change.

10. Parametric Views

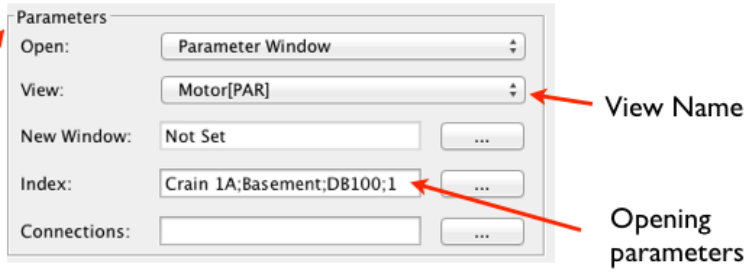
Instead of designing multiple similar visualizations of multiple field technology that differs only in its operational parameters and not appearance, you can use one master view, common for all. You can set and use the opening parameters to link the views with specific pieces of technology.

Example:

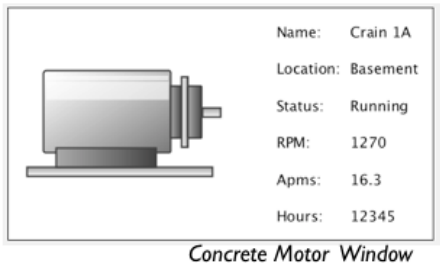
Imagine you visualize a plant with 200 similar motors and instead of designing 200 identical screens with different PLC links you only create one view. The visualization user can select a specific motor by entering the desired parameters when opening the requested view.



Overview Window created by the user - every motor has different opening parameters



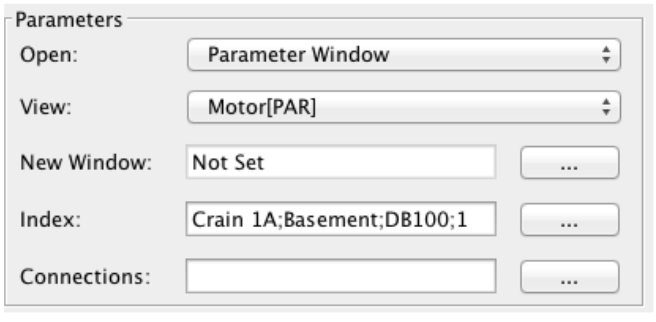
Parametric Views are denoted by [PAR] after its name. You can specify parametric views when designing a new view.

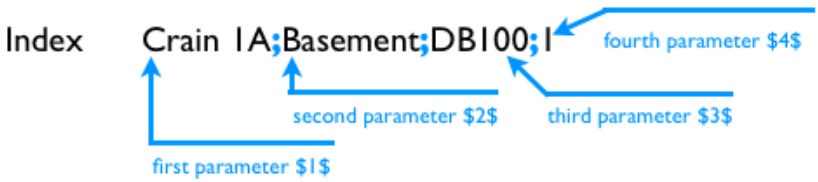


Concrete Motor Window

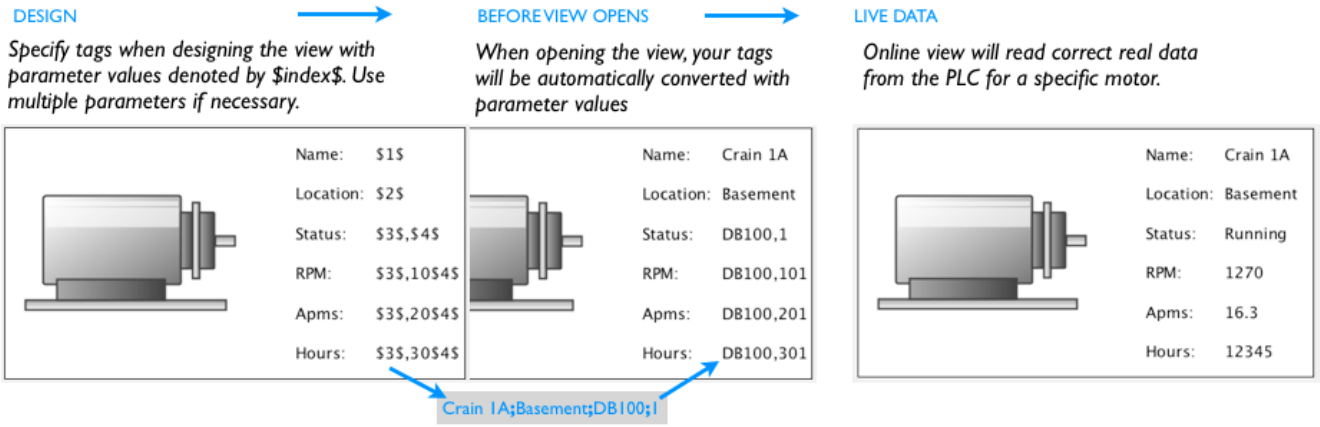
Specifying opening parameters

You can set the opening parameters in the *Index* field, the parameter can be a number or string. The parameters are separated by a semicolon ; .

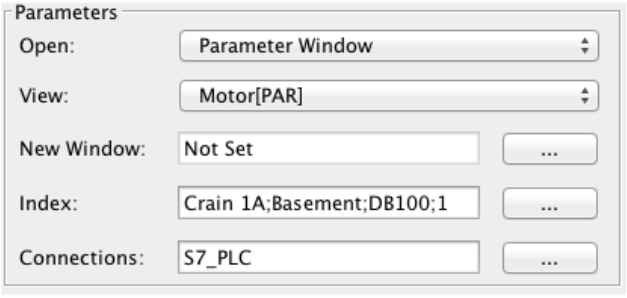




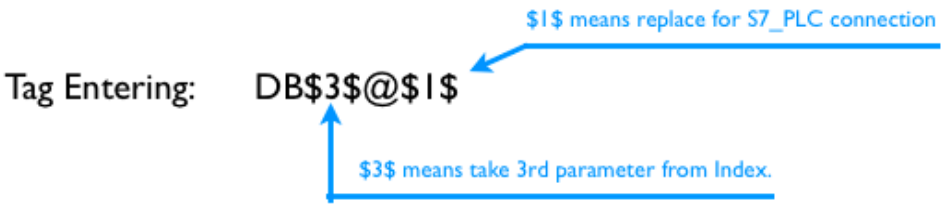
You can access the parameter value by writing the *$Number$* notation, where the number stands for the parameter position. The first parameter in the example below will be accessed as *1*. You can set the parameter values when specifying the tags.



10.1 Connection Change in Parametric Views



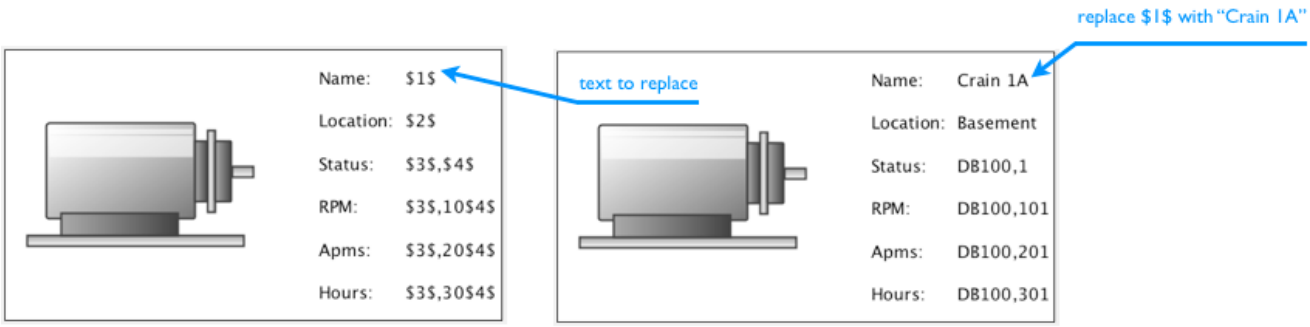
Specify connection for an opened view: you can specify to what PLC your view will link to in the ***Connections*** section:



Again, as with the *Index* parameter, you can define multiple replacements for your connections. If you want to set more than one connection, use a semicolon for separation.

10.2 Use of $number$ in texts

You can also use the *Index* parameter to include any parameter in the text elements:



11. Visual Scripting

myPROJECT Designer offers many tools to perform the most common data acquisition, display, animation, effects and all of this without coding. For maximum flexibility *mySCADA* also includes a complete scripting language based on *JavaScript,* which allows you to interact programmatically with most of *mySCADA* functions with a high level scripting language.

Easy to learn

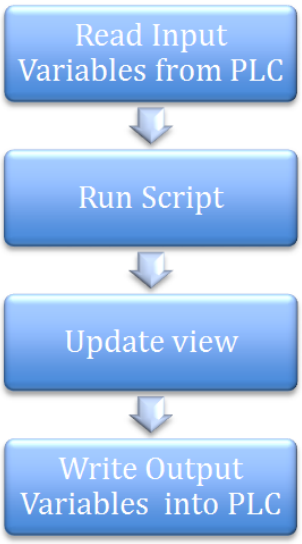
Scripting can be used for all sorts of tasks. The possibilities are endless as it is designed to be easy to use, extending the functions of *mySCADA*. You do not need to be an experienced programmer to be able to use scripting. Using *JavaScript* as a scripting language means you do not have to worry about memory allocations, leaks or complex programming issues. Using is very simple and straightforward.

Features

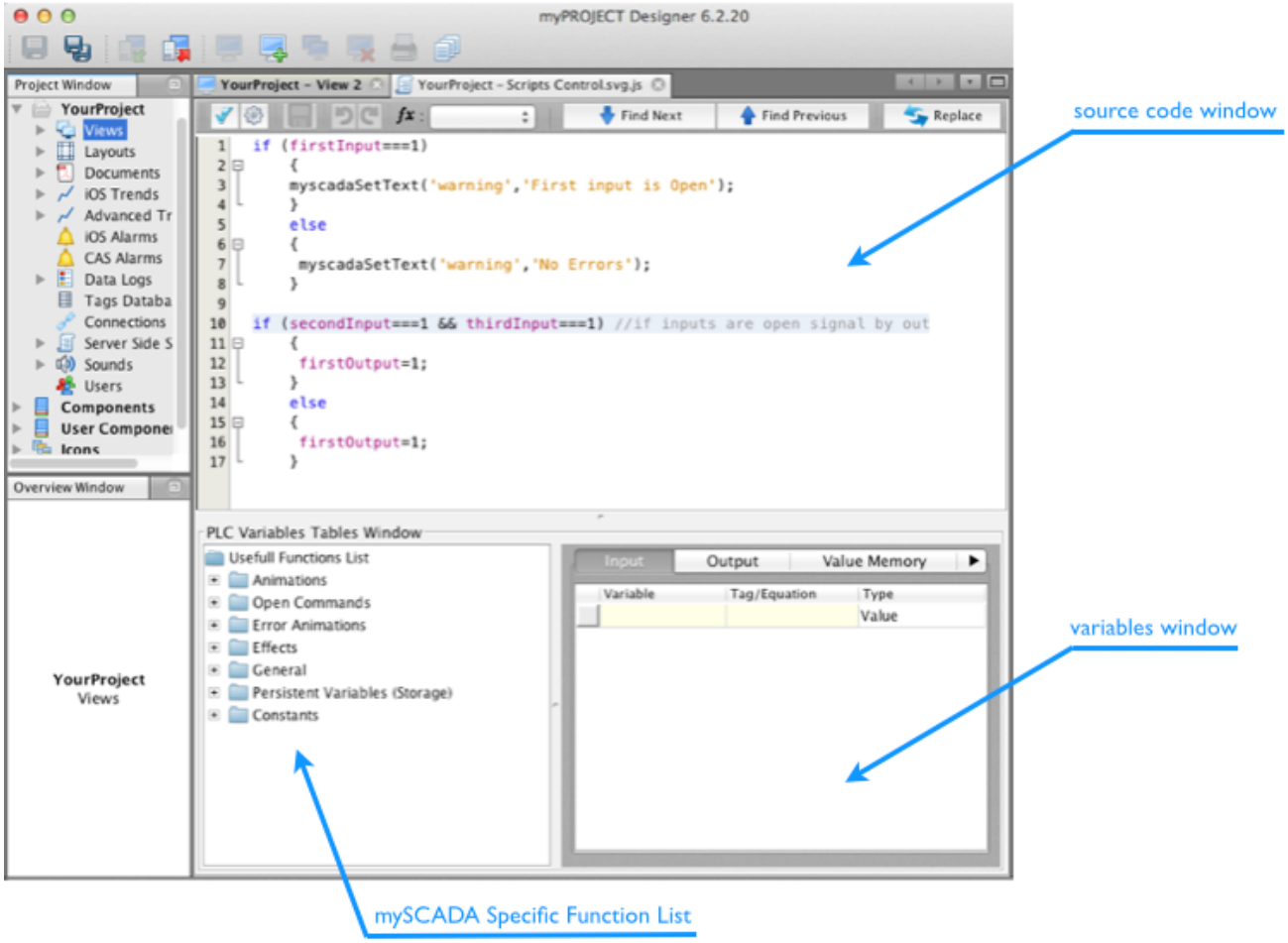
mySCADA has many built-in features that can be extended with scripting. If you are starting out, you can use all the built-in features to acquire data, create graphics, and make dynamic animations, and other tasks without writing any code. You can then pick one area that needs a little extra flexibility and write some simple script while still using all other functions. As you get more experienced, you can take further advantage of powerful scripting.

11.1 Using Script in Views

For each view you can create your own script, which is evaluated every time an active screen is refreshed. The refresh logic works as follows:



First, if you want to add a user-defined script, open your view and click on the **Script** button in the main toolbar. Then the script window will show up where you can insert your user-defined script:

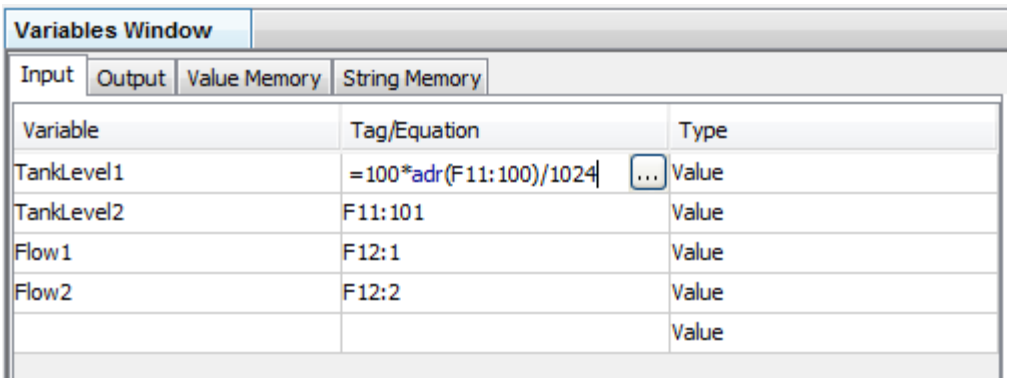


The script editor is divided into 3 main windows:

- *Script Window*: this is a window where you put the user-defined script
- *Variables Window*: here you define your view variables
- *Useful Functions list window*: in this window you can find specific functions, which help you control your components in the corresponding window.

11.2 Declaring Variables

You can declare your variables in the *Variables Window*:



There are 4 types of variables to use:

- *Input variables*: use these variables for reading data from the PLC, each time the script is evaluated it firstly reads the PLC tags and stores into the input variables
- *Output variables*: after the script evaluation, v the output variables values are written into the PLC, you can set if to write the value each time the script is evaluated or only on change - this is controlled via the *Update* field (*Always, On Change*)
- *Value memory*: here you can declare your persistent variables, when a view is loaded memory variable has its default value, you can change this value in the script and it will prevail until you switch to a different view or close the application
- *String memory*: string memory variables are the same as value memory variables but are of string type

11.3 Writing Script

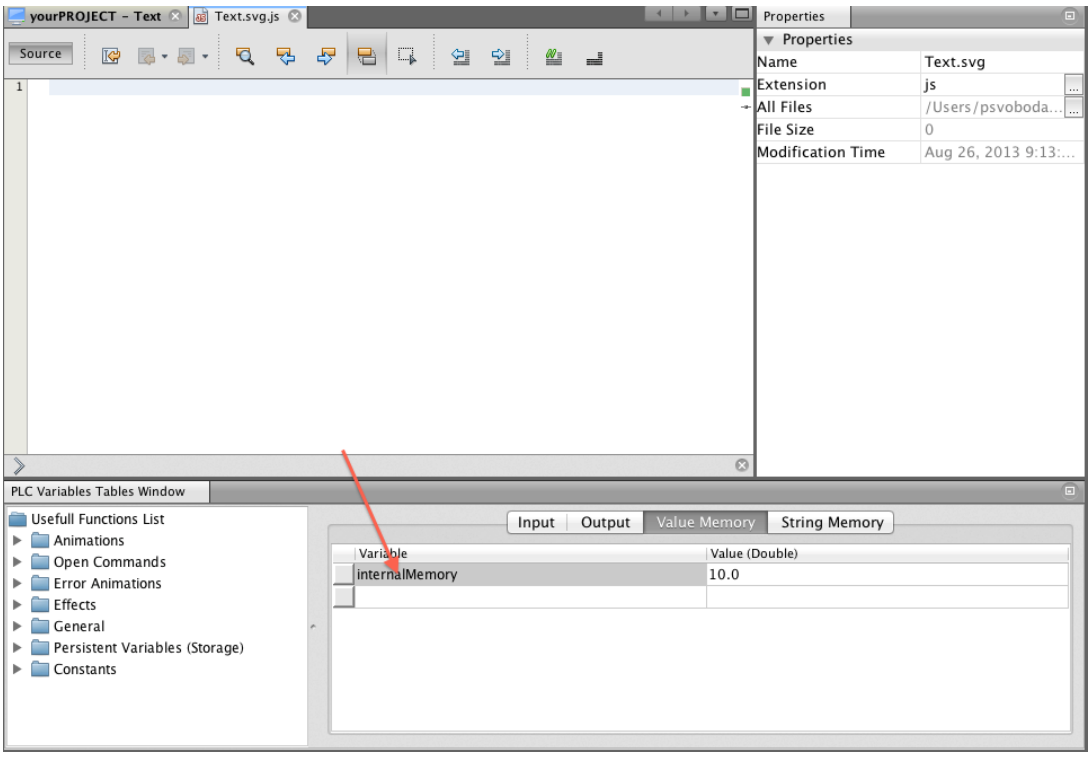
You can write the script in the *Script window*. You can use any function or expression from *JavaScript* - the editor automatically highlights your syntax and makes an error check.

Using variables in controlling animations

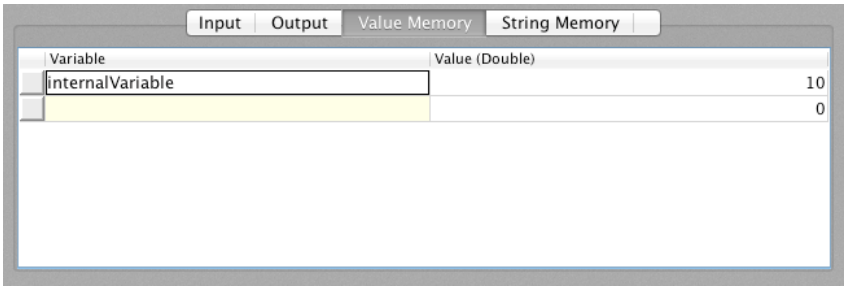
You can use declared view variables in the animations, for linking the view variable with an animation use the equation editor as with regular PLC tags.

Example:

1) Let us show the value of the *Value Memory Variable* in a text element in your *view*.
2) Open the *Script window*.

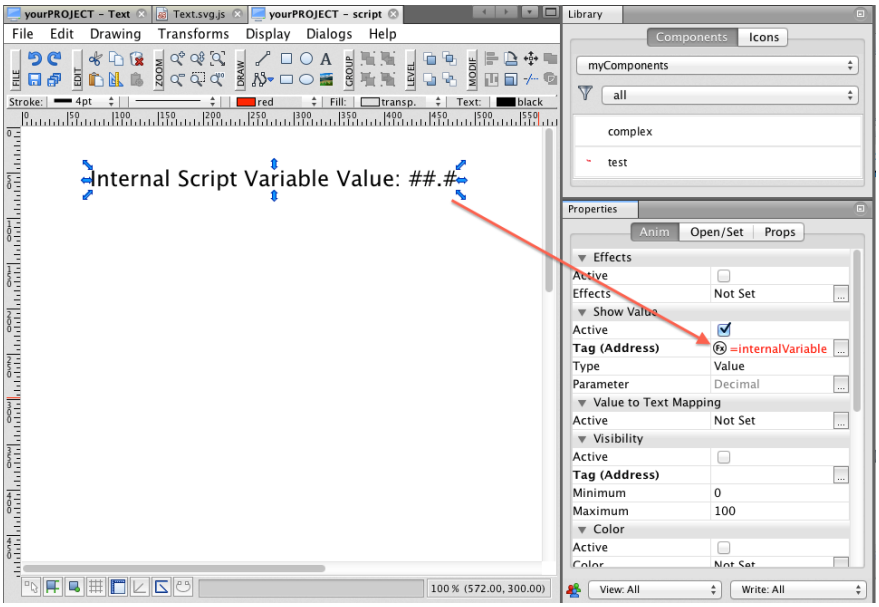


3) In the **Value Memory** tab of the *Variables window* create '*InternalVariable*' for example, with a default value of 10.0.



4) Switch back to the *view* you are editing.

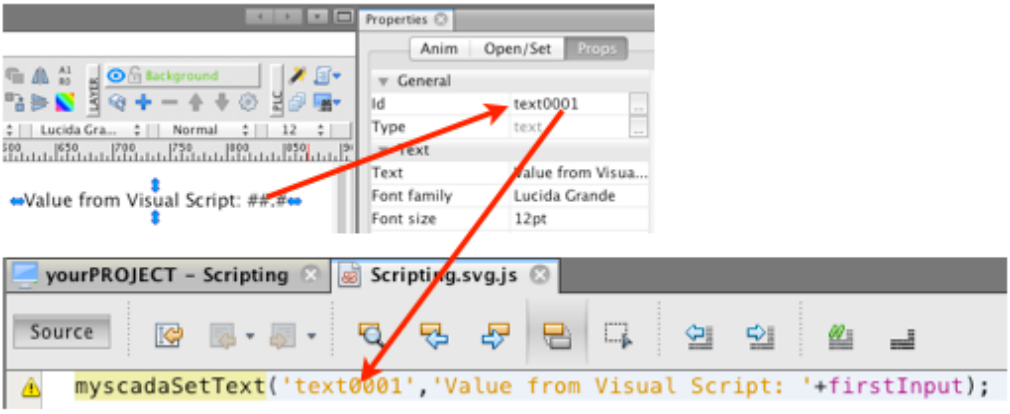
5) Create a text element in your *view*, click on it and create an animation with a memory variable typing *"=InternalVariable*" or selecting memory variable with the equation editor.



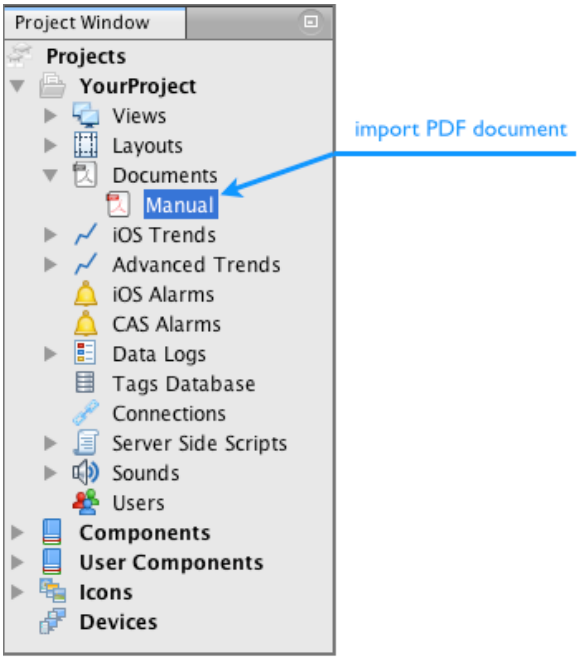
Also you can control animations of your graphic object directly in the script with specific *mySCADA* functions. You can find the complete list in the *Useful Functions window*.

To set the value of the element directly in the script use the following function:

myscadaSetText('text0001',internalMemory);



12. Documents



With this powerful function of *myPROJECT Designer* you can link documents with your project. For example, you can have user manuals linked with the components on your HMI screen, or you can even attach a schema or picture to your project. As a result you get a complete project in one package consisting of HMI screens and the linked documentation files.

You can attach any documents directly to your project by selecting the function *Import Document.*

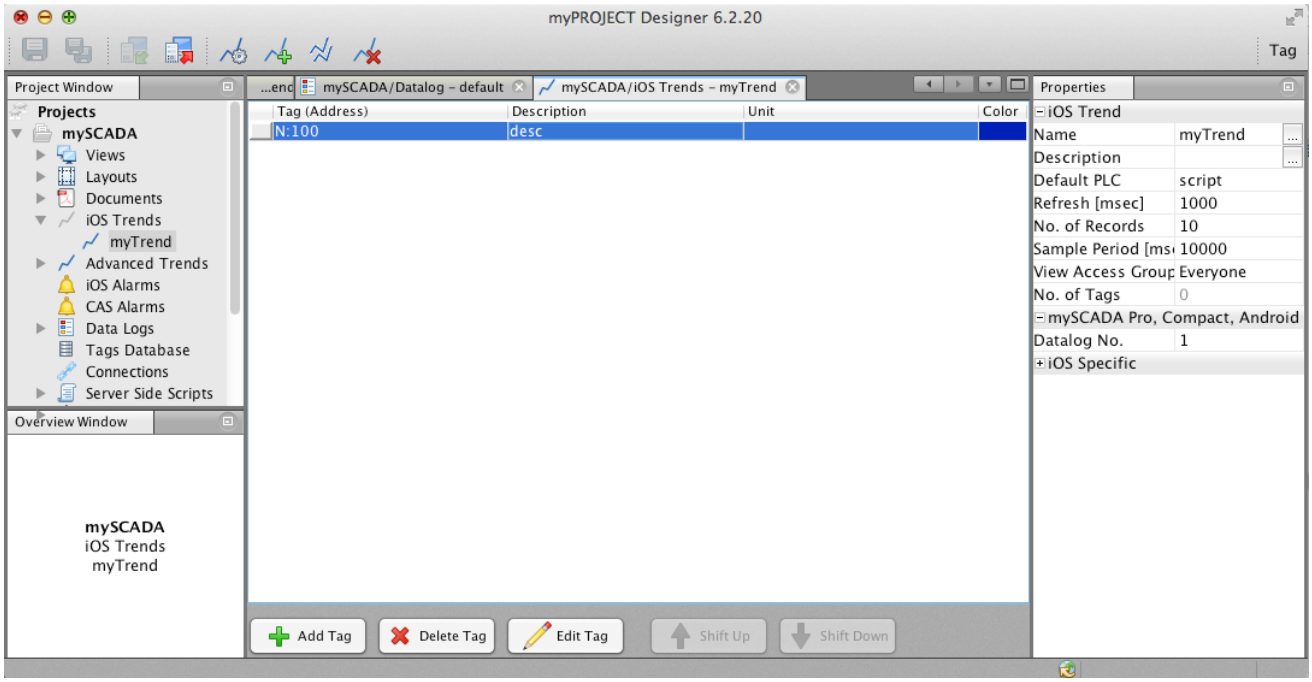
Note that all attached documents must be in a **PDF format!**

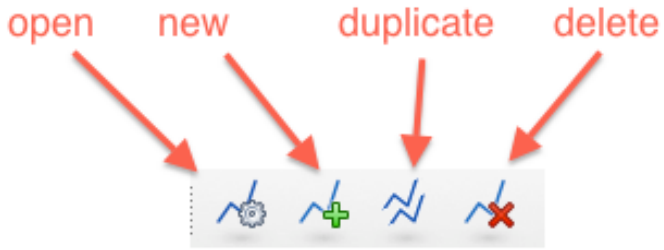
Click on the **Import document** icon in the main menu or right-click on the *Documents* in the *Project Window* menu and then select *Import.*

Once your documents are imported into your *mySCADA* project, you can link them with any object on HMI screens. This can be done with *Open command* function.

13. Creating iOS Trends

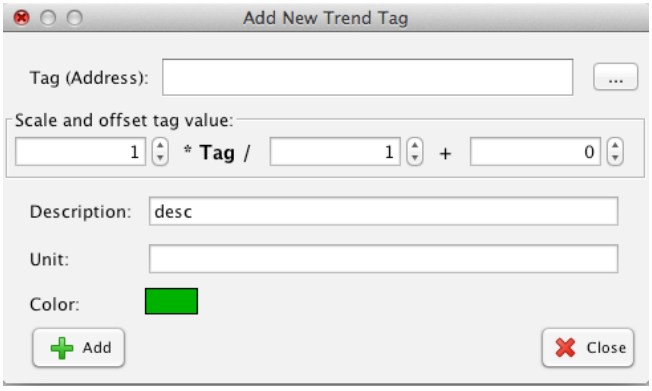
Visualization of trends is another important part of the *myPROJECT Designer*. The same as alarms in iOS, values used for the trends should be represented as part of an array in the PLC. *mySCADA* stores every read value into the built-in database, to create a value archive for each read tag. The archived values can be later compared with the current values of the same tag. The *Trend* charts for each of the watched tags can be also created. It is possible to monitor multiple trends at once as well as edit, add, and delete trends from the *Main window*. To see the current trends in the project navigate to the ***iOS Trends*** folder in the *Project Window*.



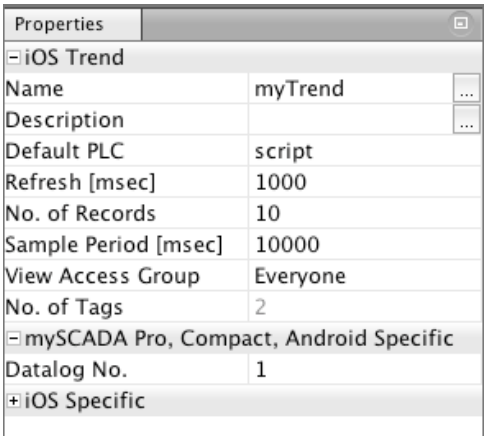


To add a new trend to your project select *iOS Trends* in the *Project window* and then click on the **Add Trend** icon or use the command *Trend –> Add New Trend* from the right-click menu. Click on the **Add Tag** button at the bottom of the main window to add a tag for monitoring, enter the tag address along with the trend description.

Instead of a raw tag value you can use arbitrary mathematical transformation of the tag value by choosing the *Equation* as a form of the tag representation in the *Add Tag* dialog. You can find more details about the *Equations* in the relevant chapter of this manual. You can specify a physical/mathematical data unite, open or deleting an existing trend etc.



You can edit the trend properties by selecting them in the ***iOS Trends*** folder in the *Project window*.

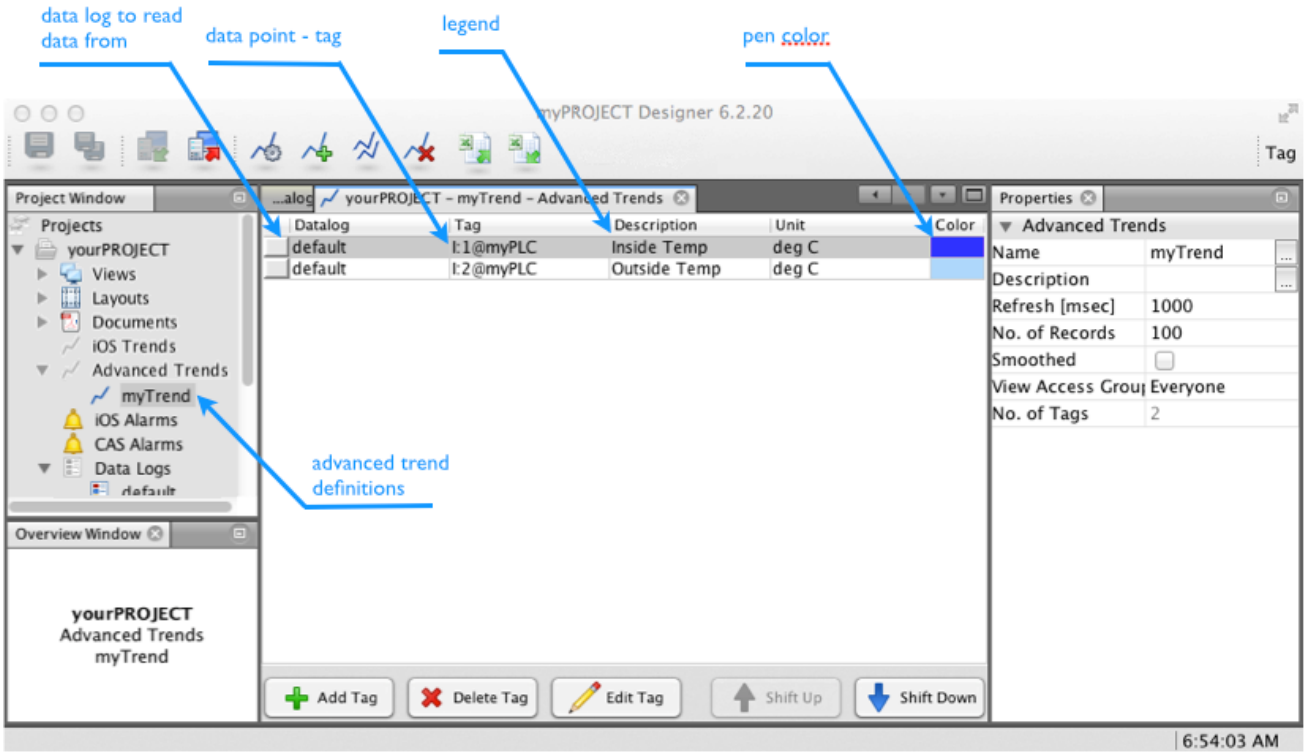


You can set the following properties:

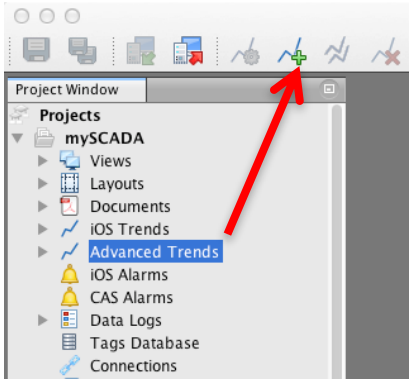
- ***Trend name***
- ***Trend description***
- ***PLC connection,*** used for gathering data
- ***Refresh rate*** for the trend (same as for the view)
- ***No. Of Records*** – number of elements *mySCADA* will read from PLC field for this trend (for Online trends only)
- ***Sample Period*** – timescale between samples
- ***Reverse*** – inverse order of reading of samples
- ***View Access*** – user access level
- ***Online*** – check this settings for iOS application trending, otherwise *mySCADA Box* format will be used
- ***mySCADA Box Specific*** - ***Datalog No***.: defines from which data log the values should be read

14. Creating Advanced Trends

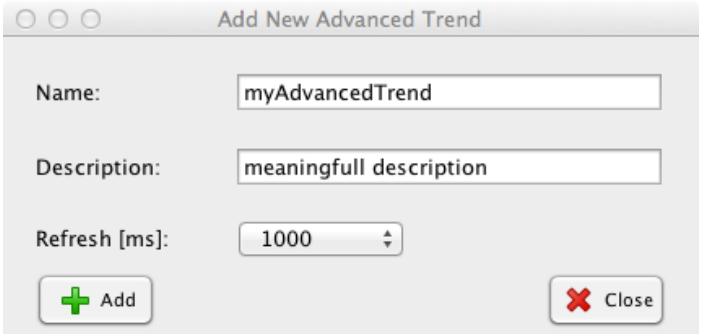
You can easily present your historical data in the form of a historical trend. To do so, firstly look at the chapter *Data Logs* to set your data log to retrieve data from.



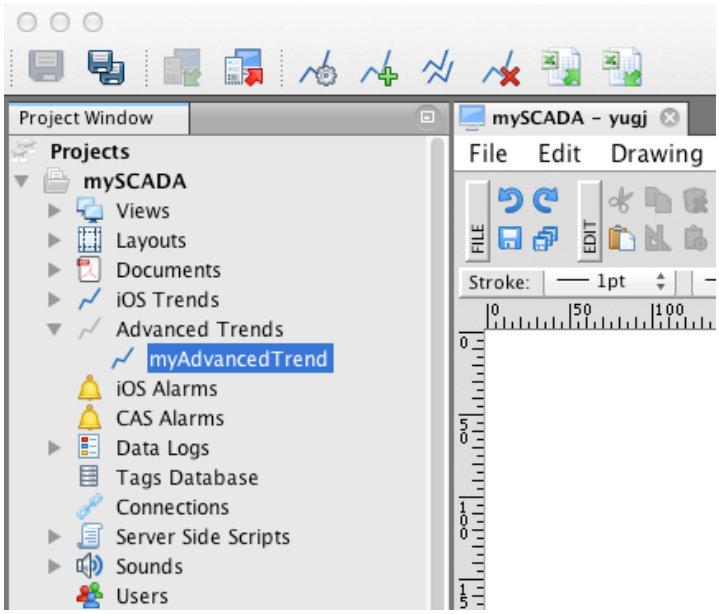
1) If you have your *Data log* ready, create a new historical trend by going to the ***Advanced Trends*** folder and selecting a new trend as shown in the following picture:



2) A new dialog window will show up where can fill in the trend name and description. Then click on **OK**.

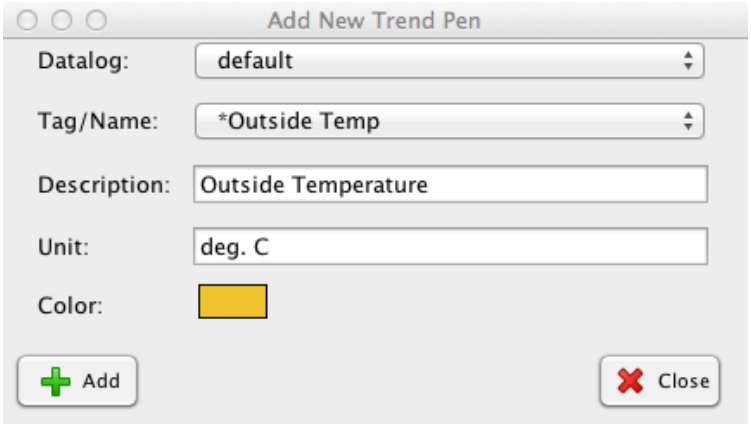


3) Now the new trend *'myAdvancedTrend'* is created and you can find it in the ***Advanced Trends*** folder of the *Project window*.

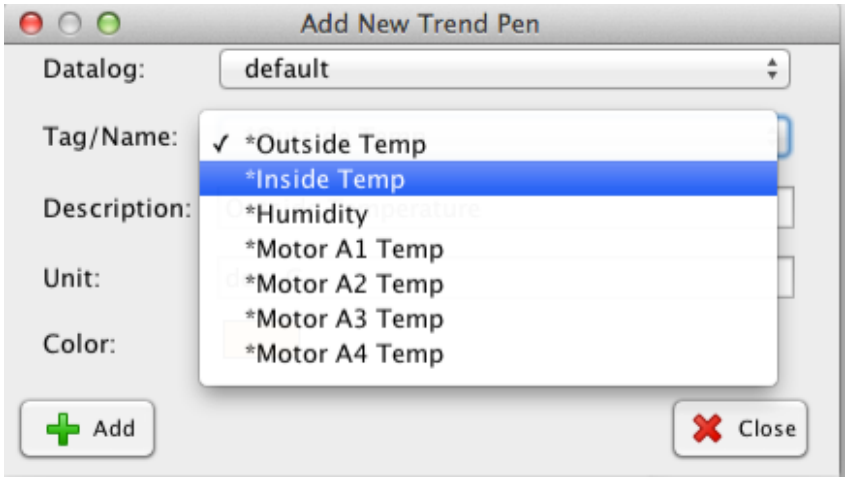


4) Double click on the newly created trend to open it.

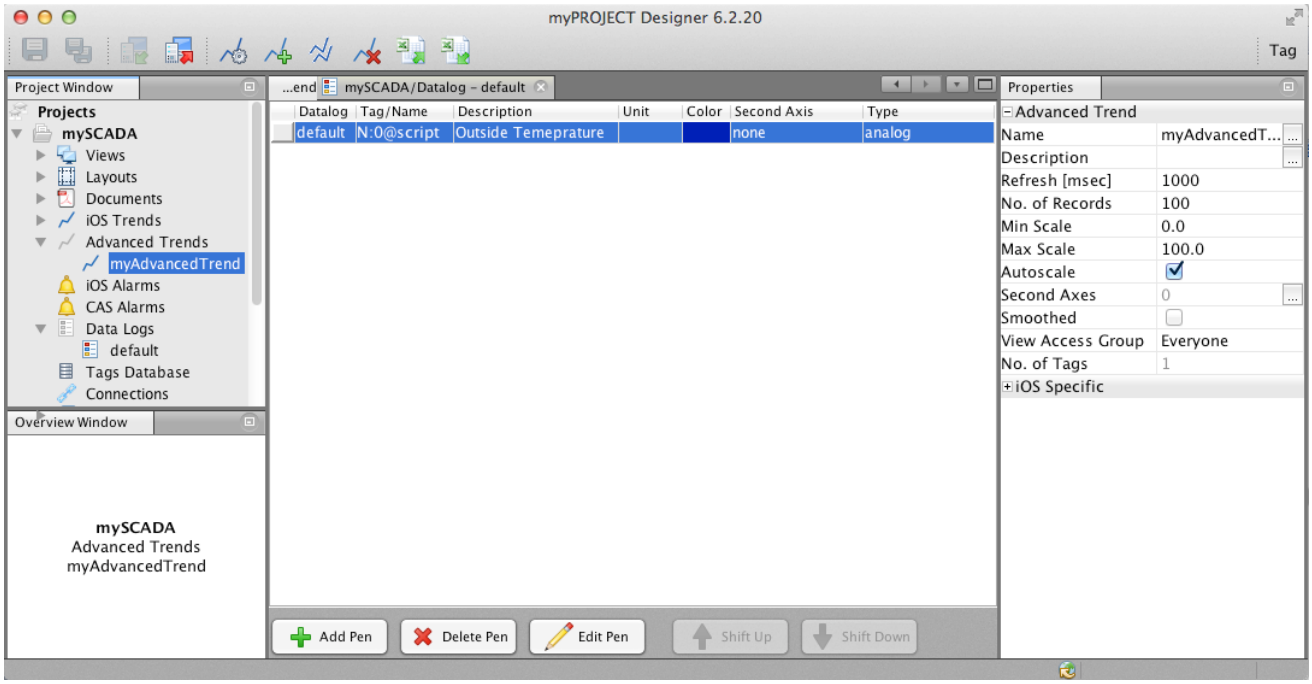
5) Click on the **Add Pen** button in the lower part of the *Advanced Trend Definition window*:



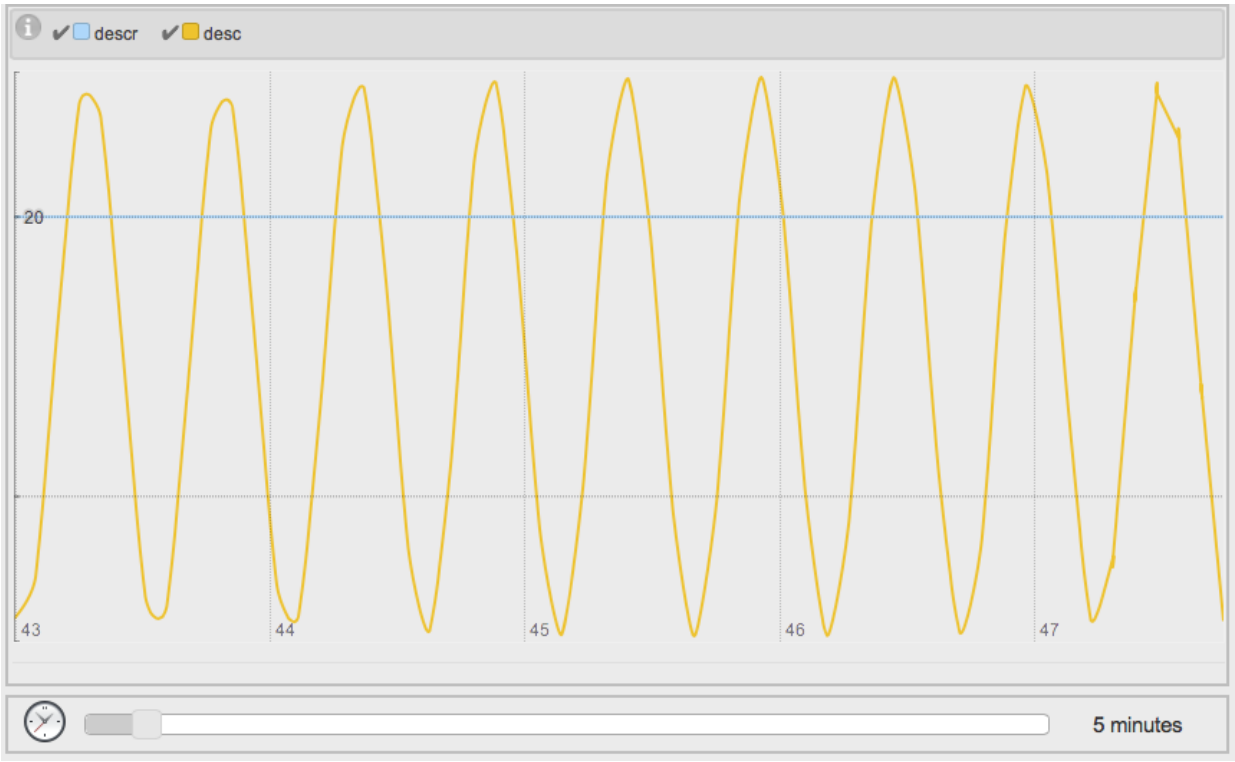
6) It the pop-up dialog box, you can define your new pen to show. First you should select a data log from which to show data, then select your data point to show (only the tags from the selected data log are available):



7) Finally fill in the *Description* and *Unit*. You can also change the pen color and confirm by pressing the **Add** button. The pen is now listed in the table below:



Note: You can freely add as many pens as you wish, they can read data from multiple data-logs.


descr
desc
20
43
44
45
46
47
5 minutes

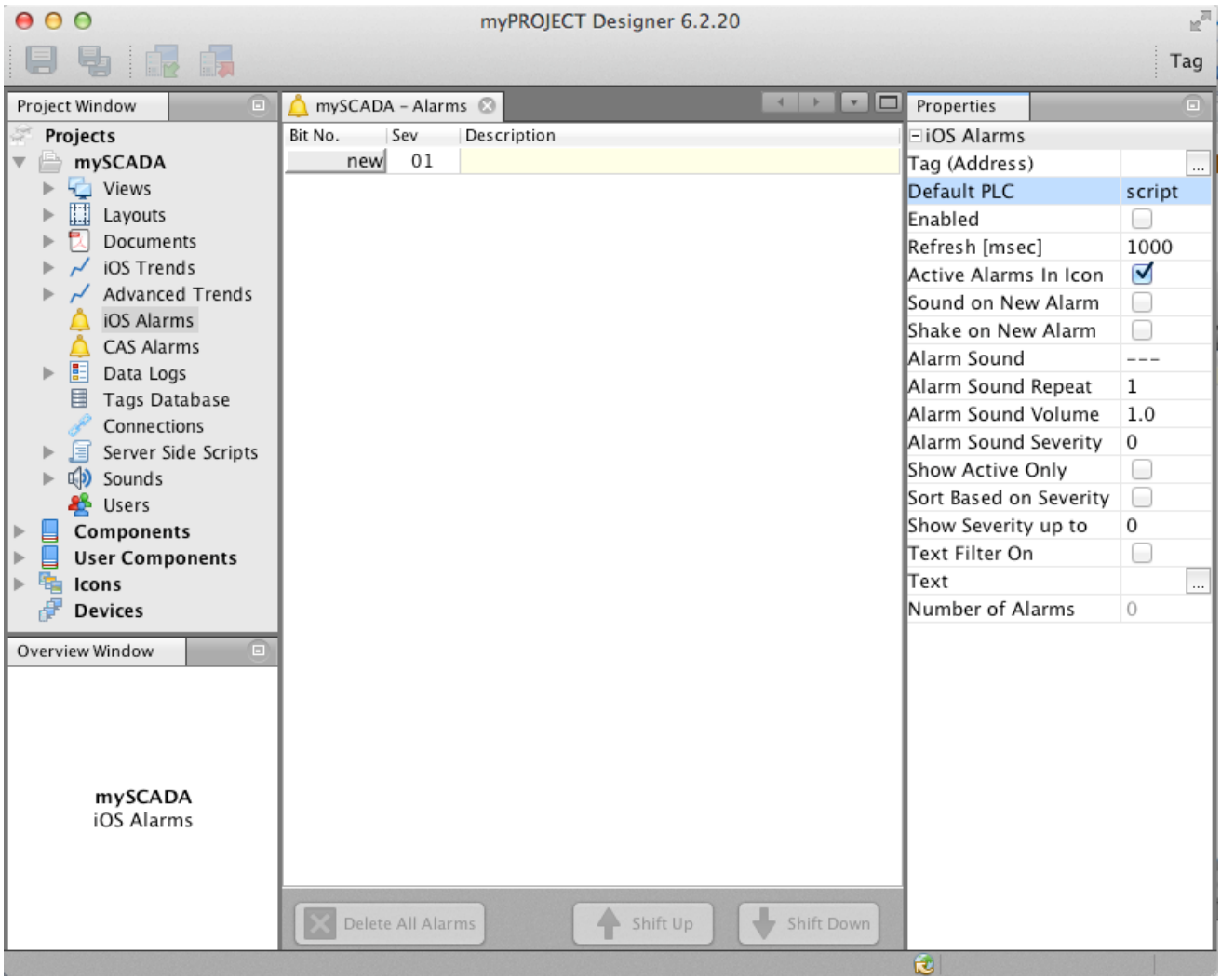
15. iOS Alarms

A very important part of *mySCADA* is the ability to instantly signal any information about dangerous or other important events with *Alarms*.

In *myPROJECT Designer* you can create alarms list specifically for iOS application. For the alarms on other *mySCADA* products, you can use the *CAS Alarms*.

The alarms must be stored as an array data because *mySCADA* for iOS applications only interprets array values as the alarms, so the user needs to set up an acceptable range for each alarm.

1) Select ***iOS Alarms*** folder in the *Project window*. First you have to set the *Connection* and choose which PLC connections, defined in the project, will be used. Then enter the triggering tag for the alarm list and enable it by ticking off the appropriate checkbox.



2) You can alter other alarm properties, such as:

Active Alarms in Icon – displays number of active alarms on the *mySCADA app* icon if the application is running in the background

Sound on New Alarm – plays system sound when one of alarms became active

Shake on New Alarm – vibrates the device on a new alarm (as long as *mySCADA* is running on a device with such function)

Show Active Only – disables displaying of inactive alarms on the alarms list

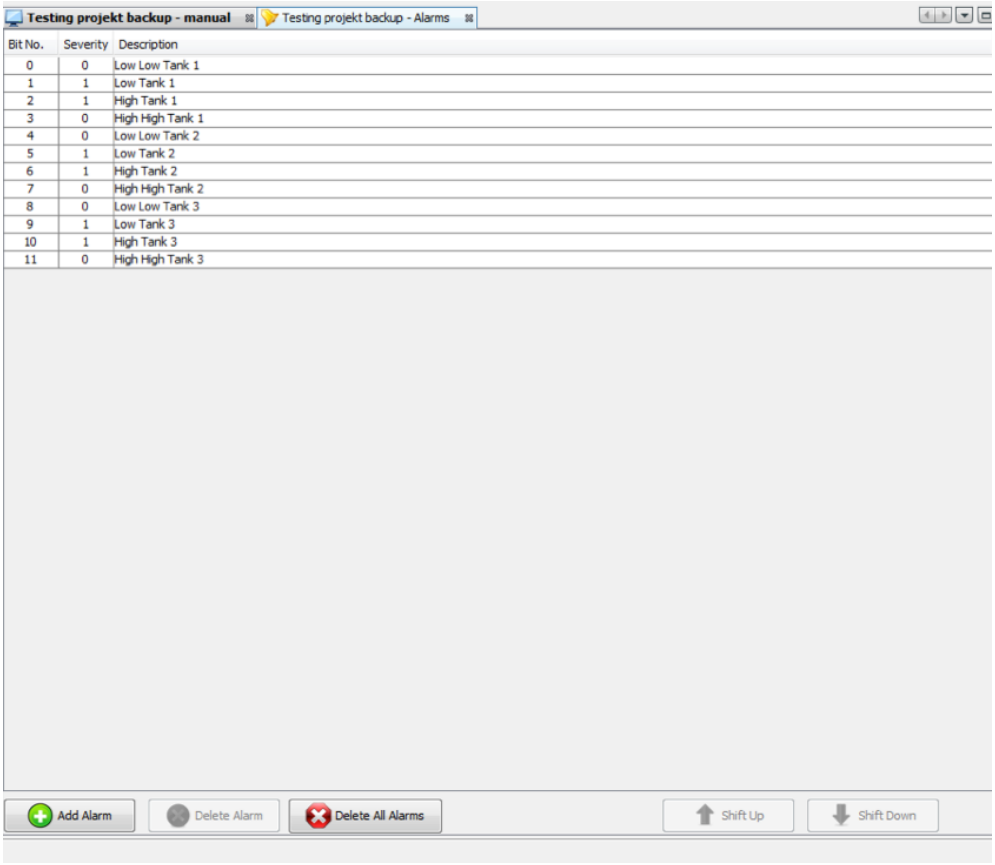
Sort Based on Severity – sorts the alarms on the list according to their severity

Show Severity up to – filters out the alarms with severity lower than set

Text Filter On – filters out all alarms, except those containing text, filled in the ***Text*** property, in their description

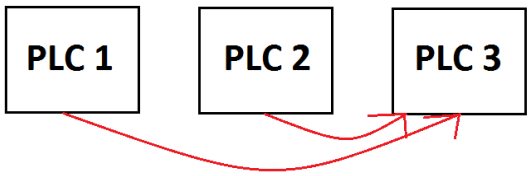
The list of alarms is shown in the ***Alarms*** folder of the *Project Window*. New alarms can be created by clicking on the **Add** button located at the bottom of the *Alarms window*. The alarms can deleted, shifted down or up on the priority list.

After clicking on **Add** the window displayed in the picture below will appear. The *Bit No.* is automatically filled with index of the bit representing the current alarm in the array of alarm variables. The alarm *Severity* level and *Description* must be filled also (put 0 for the maximum severity).



Example:

In the following example there are 3 PLCs, where the alarms are read from the PLC 3. The PLC 1 controls the temperature within the range of 0°C to 120°C. To create an alarm on this value you need to evaluate the temperature readings from the PLC 1 and send the *Boolean* value to the prepared array in the PLC3.



The desired temperature lies within the range of 50°C to 80°C, so you create two alarms - for the high and low temperatures. The first alarm gets the 2nd position in the bit array of the PLC 3 and will be activated when the temperature falls into the 0°C - 50°C range. The second alarm gets the 3rd position in the array with the range of 80°C – 120°C. Then create an alarm list with appropriate descriptions for these alarms.

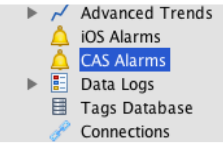
Bit No.	Severity	Description
0	0	Low Tank
1	0	High Tank
2	0	Low Temperature
3	0	High Temperature

If the temperature reading gets to 40°C, for example, PLC 1 sends the value 1 to the bit array element no. 2, the alarm in will pop-up in the *mySCADA* application.

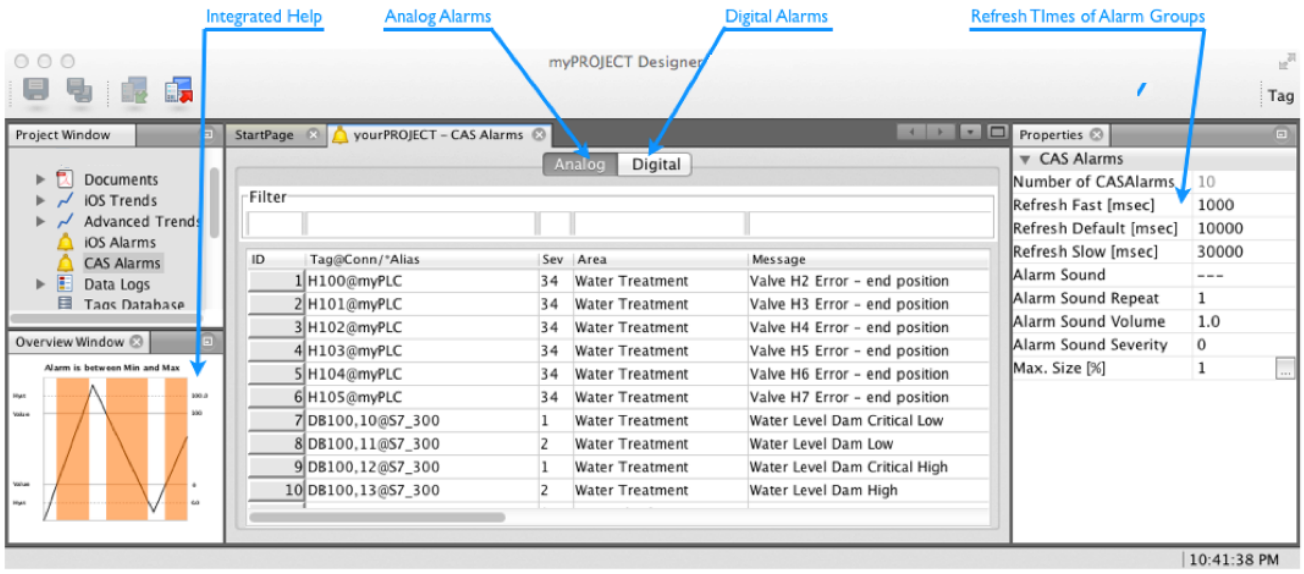
16. CAS Alarms

A very important feature of *mySCADA* is the ability to instantly signal any information about dangerous or other important events with *Alarms*. They are important part of most control applications as they alert operators if something goes wrong.

Alarms can signal that a device or process has ceased operating within acceptable, predefined limits or can indicate breakdown, wear or process malfunction. Often it is important to keep a record of the alarms to know if they have been acknowledged.



To define alarms for your project, double-click on the *CAS Alarms* in the *Project window*. For your convenience the alarm definitions are split into two tables, one for **Analog** and second for **Digital** alarms:

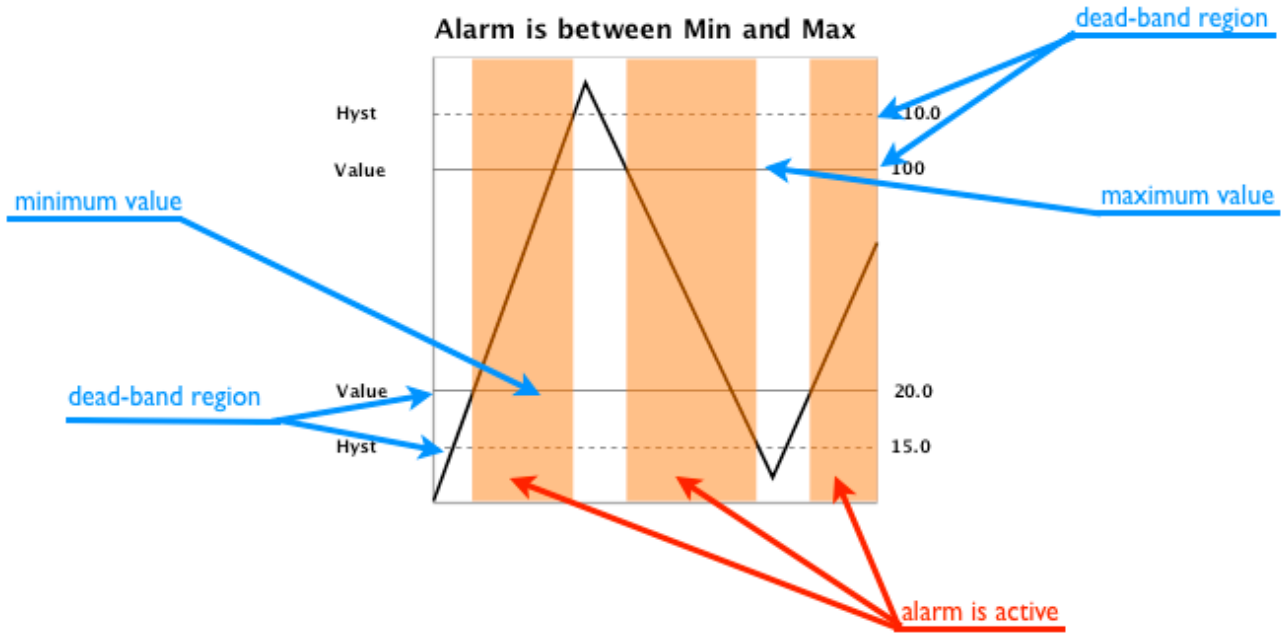


16.1 Digital Alarms

Digital alarms are tied to digital values read from the PLC. A digital alarm is either ON (1) or OFF (0). Instead of the thresholds, digital tags have alarm states 0 and 1.

16.2 Analog Alarms

Analog alarms are tied to analog values read from the PLC. Along with common parameters, which are the same for both *digital* and *analog* alarms, you specify the minimum and maximum values for your tag or equation. You can also specify the *Dead-band* region for any value to eliminate false alarms. The number of alarms tied to one tag is not limited. Also you can define complex conditions, including multiple tags or mathematical conditions for a single alarm.



Description of Alarm table fields:

Unique ID	Automatically generated ID, needed if accessing alarms from the *Server-side Script*
Tag@Conn / *alias	Tag (Address) or *Equation* specifying data read from PLCs
Severity	Unsigned integer value specifying importance of given alarm, lower the number - higher the priority
Area	You can divide alarms by geographical or virtual area they belong to, area is a string value which you can use for filtering the alarms
Message	Message of your alarm
Device	Name or description of a device the alarm belongs to, one device can have multiple alarms defined
Inv (Inverse)	Inverts an alarm (if a digital alarm is inverted it will be active at 0, for analog alarm activation area is reversed)
Hide	Enables hiding the alarm from the user, this is useful when you are using alarm as a condition in the triggered data-log, hidden alarm is not shown in the *Alarm window* and is not logged into the database.
Delay	Specifies the delay in milliseconds of how long the condition must be active for to activate the alarm, this is the time hysteresis function
Refresh	Specifies how often your alarm will be refreshed, you can use: *default, fast, slow*. You can change the refresh values for each group in the *Properties window.*

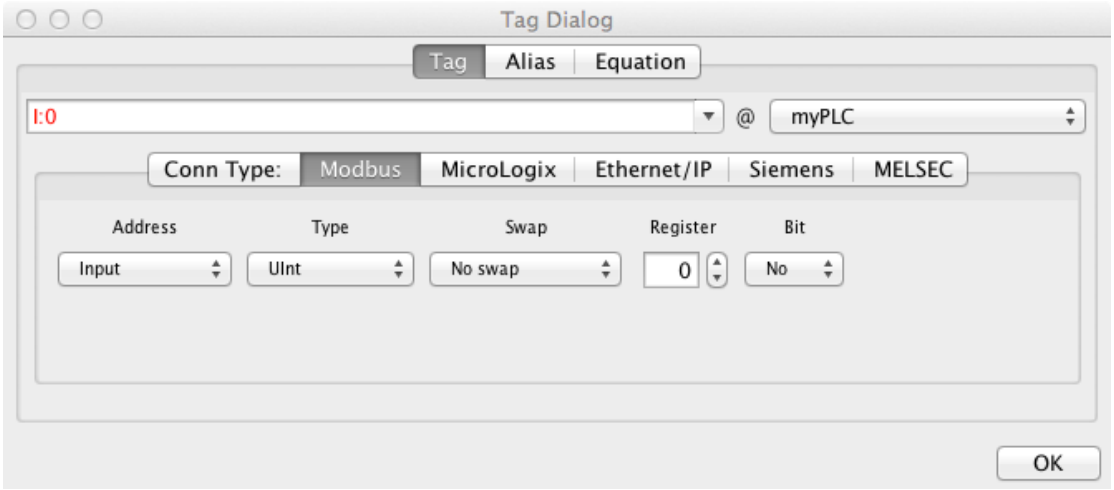
Format	Defines how the value will be shown in the alarm table, use the '##.#' format for specification
e-mail	Sends an email upon each alarm activation or deactivation
SMS Act	Sends a message upon each alarm activation
SMS Deact	Sends a message upon each alarm deactivation
G0 - G9	Checks an appropriate user group to receive alarms by email or SMS

Unique ID

Each alarm has a *Unique ID,* which is created when you define a new alarm and its value is fixed once the project is put into the runtime mode for the first time. This value is unique for all alarms, meaning that each alarm has its own *Unique ID*, which is saved in the alarms' definition file. Once this ID has been created it will remain until removing the alarm from the definition table.

*Tag@Conn / *alias*

Each alarm must be connected to a tag or equation. The *Tag* is the value read from PLC (or your computed value from script), with *Equation* you can tie your alarm to multiple tags or evaluate complex formulas.



You can create multiple alarms tied to one tag, as long as the alarm descriptions are different so that alarms operate correctly.

Severity

Alarms can range in the severity from 0 (most severe) up to 4 byte unsigned integer value (least severe), to indicate different levels of importance.

For example, an alarm with severity 10 might warn that a tank is half full of liquid, while another alarm with severity 5 indicates that the tank is about to overflow. Both alarms monitor the same tag but have different severity levels.

When you set up the alarm severity, specify what the severity levels mean and what actions they will trigger. Severity determines the order in which alarms are displayed on the alarm list.

Alarm areas

The alarms can be grouped in different areas so that they can be displayed in the alarm window based on the area they belong to. This may be helpful to enable you to divide the alarms according to the different plant zones they come from.

Message

Alarm messages report information about alarms.

Device

You can define multiple alarms for a single device. In the live alarm view or during a browsing of alarm history, you can filter your data based on device value.

Minimum and Maximum values

Minimum and maximum values are available in analog alarms only (for digital alarms minimum and maximum is equal to 1). By default, you specify the region when will be alarm active. So if you would like to activate alarm when the level in tank is equal to 90 and stop the alarm at level 100 your minimum value will be 90 and maximum value will be 100.

Inv (Inverse)

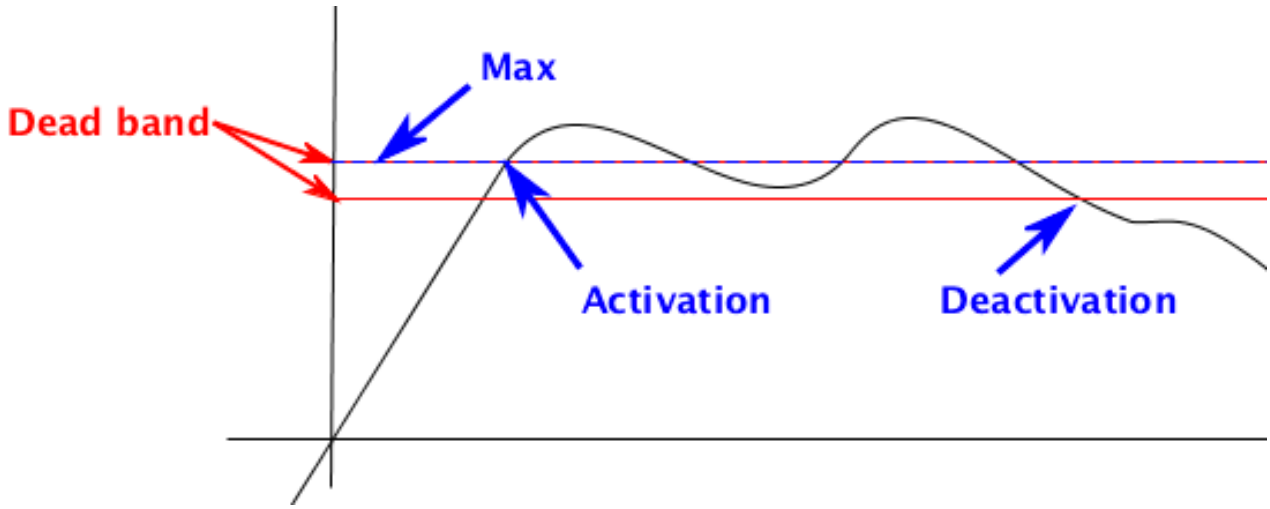
This parameter lets you invert your alarm definition. For a digital alarm, the alarm is activated when value is equal to 0 and deactivated when equal to 1. For analog alarms, when inverse is active, you specify by minimum and maximum values the region when the alarm is **NOT** active.

Recipients (G0 up to G9)

Through these properties you can select the recipient user group to which the message, SMS, E-mail etc., is to be sent. The user profile, which is defined through the 'Users' settings, must contain a telephone number or E-Mail, needed for sending messages.

Dead-band

With some measured values, such as line pressure, tag values can fluctuate rapidly above and below a critical threshold. Where such conditions exist, you can create a dead band as a buffer to prevent the fluctuations from triggering unnecessary alarms.



Specifying frequency of alarm checks

The system does not check for alarms more frequently than the *Refresh* update rate specified in the alarm definition.

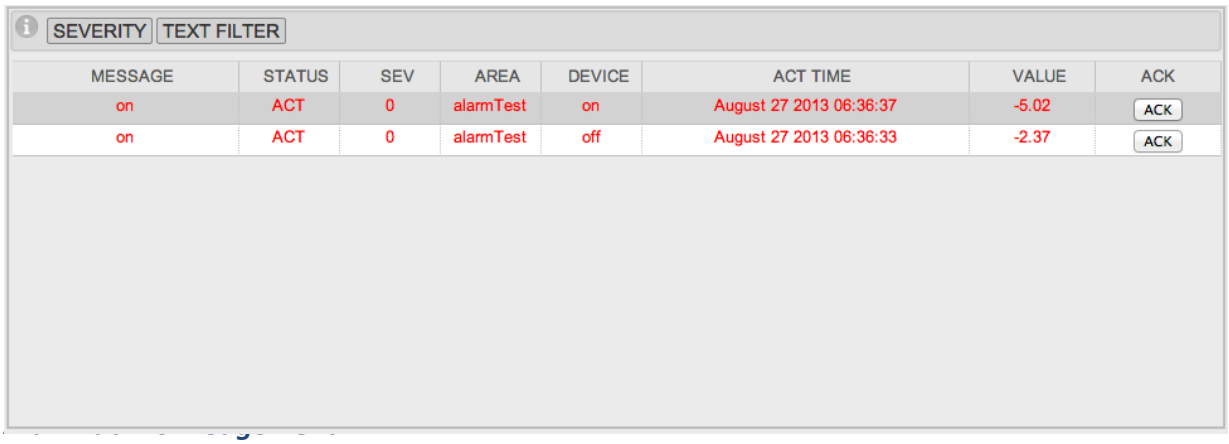
Match the maximum update rate to the rate at which you expect tag values to change. For example, if you are monitoring temperatures that fluctuate slowly, check alarms less frequently than with the manufacturing processes that change rapidly.

You can specify one of tree possible refresh rates: *default, slow, fast*. Each refresh rate group can be changed in the *Properties window*.

16.3 Alarm Window

The alarm window allows the operator to perform complete management of the technology alarms. This window allows you to visualize the alarms present in the technology or in a restricted area of the technology.

The alarm window can display all the technology alarms or a set of alarms, arranged by the user-defined areas. If necessary, the user can click on the filter button and fill in the area name.



The operator can acknowledge the alarms displayed in the alarms summary. This does not correct the alarm triggering condition, but indicates that the operator is aware of it.

Alarm suppression

You can suppress alarm monitoring for one or multiple alarms. This is useful for testing, repairing or maintaining a piece of equipment. Click on the **Suppress** button to suppress alarm monitoring. To view the list of the tags not being monitored, use the *Suppressed* list. You can also turn monitoring back on from this list.

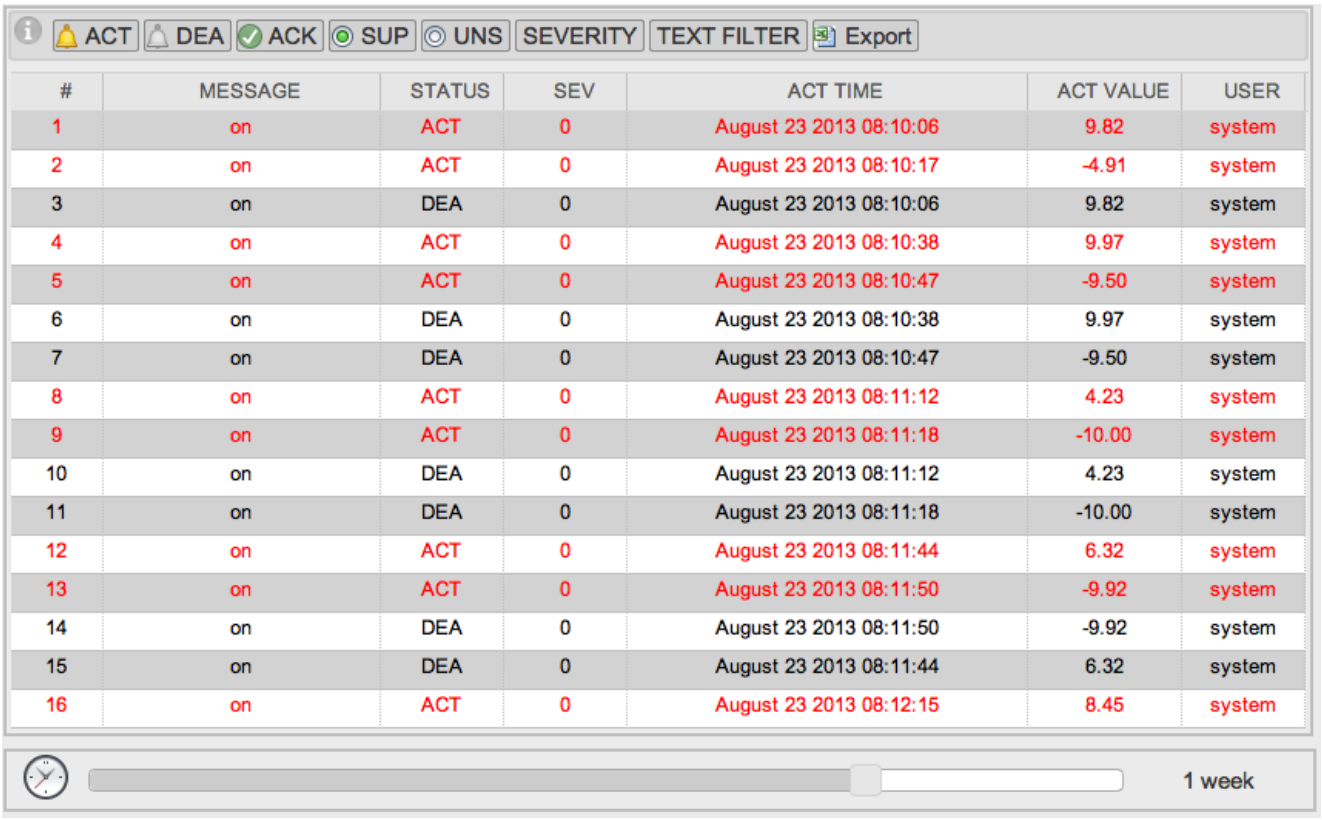
Sorting and filtering in run-time

By default, alarm information in the alarm summary is sorted firstly by date and time, severity and then by area name.

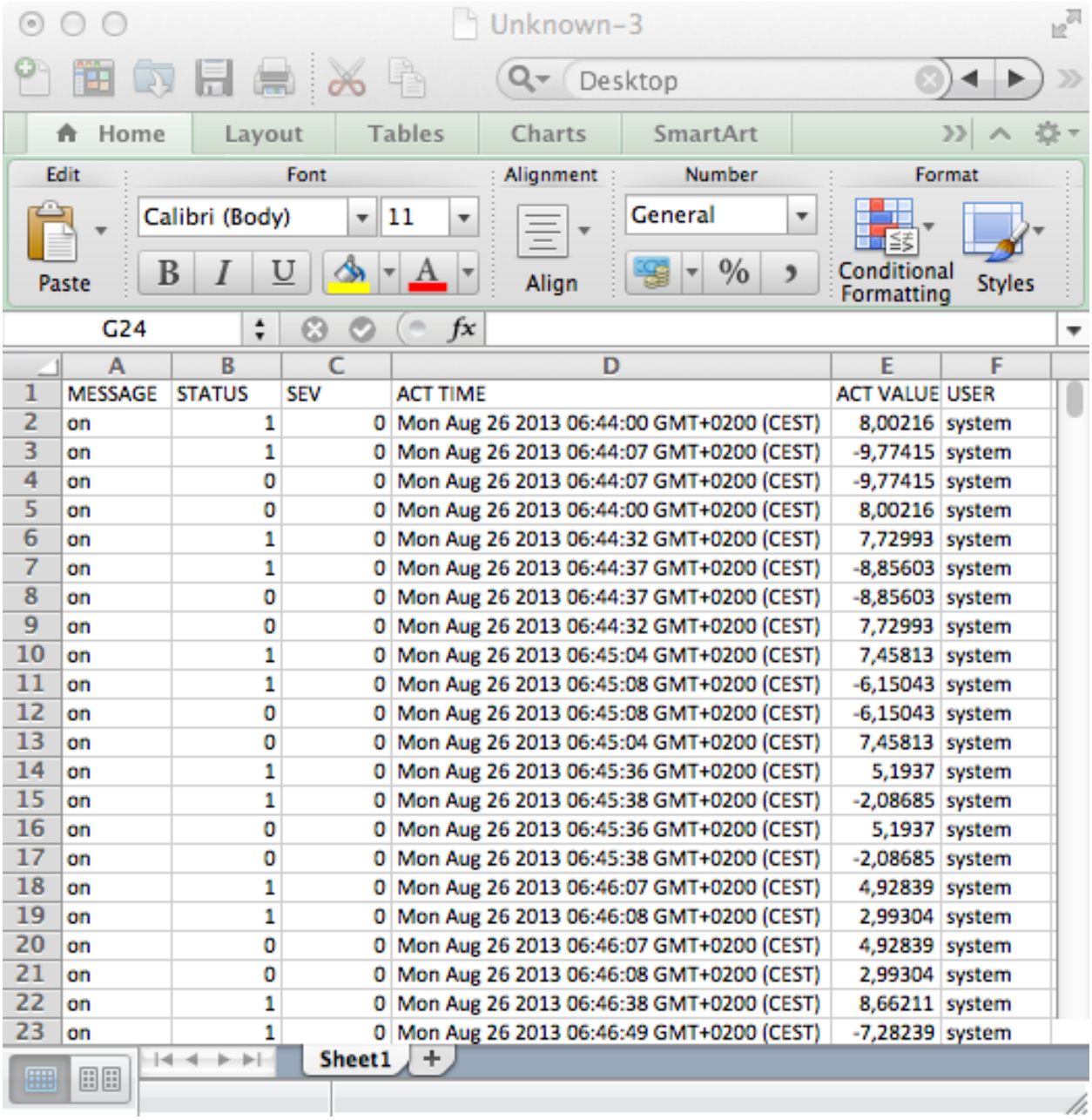
This means that alarms are presented in a chronological order: if two or more alarms have the same time and date, they are presented in order of severity; if any alarms have the same time and date and the same severity, they are presented by the area name.

16.4 Alarm History

mySCADA engine automatically (if not disabled) logs your alarms into history. Every alarm action is logged with all relevant data such as current time (with precision to 1 millisecond). You can browse through the alarm history in the *Alarm History* window. Along with direct data browsing, you can filter your data, based on the criteria and export the alarms history into a XLS file.



#	MESSAGE	STATUS	SEV	ACT TIME	ACT VALUE	USER
1	on	ACT	0	August 23 2013 08:10:06	9.82	system
2	on	ACT	0	August 23 2013 08:10:17	-4.91	system
3	on	DEA	0	August 23 2013 08:10:06	9.82	system
4	on	ACT	0	August 23 2013 08:10:38	9.97	system
5	on	ACT	0	August 23 2013 08:10:47	-9.50	system
6	on	DEA	0	August 23 2013 08:10:38	9.97	system
7	on	DEA	0	August 23 2013 08:10:47	-9.50	system
8	on	ACT	0	August 23 2013 08:11:12	4.23	system
9	on	ACT	0	August 23 2013 08:11:18	-10.00	system
10	on	DEA	0	August 23 2013 08:11:12	4.23	system
11	on	DEA	0	August 23 2013 08:11:18	-10.00	system
12	on	ACT	0	August 23 2013 08:11:44	6.32	system
13	on	ACT	0	August 23 2013 08:11:50	-9.92	system
14	on	DEA	0	August 23 2013 08:11:50	-9.92	system
15	on	DEA	0	August 23 2013 08:11:44	6.32	system
16	on	ACT	0	August 23 2013 08:12:15	8.45	system



	A	B	C	D	E	F
1	MESSAGE	STATUS	SEV	ACT TIME	ACT VALUE	USER
2	on	1	0	Mon Aug 26 2013 06:44:00 GMT+0200 (CEST)	8,00216	system
3	on	1	0	Mon Aug 26 2013 06:44:07 GMT+0200 (CEST)	-9,77415	system
4	on	0	0	Mon Aug 26 2013 06:44:07 GMT+0200 (CEST)	-9,77415	system
5	on	0	0	Mon Aug 26 2013 06:44:00 GMT+0200 (CEST)	8,00216	system
6	on	1	0	Mon Aug 26 2013 06:44:32 GMT+0200 (CEST)	7,72993	system
7	on	1	0	Mon Aug 26 2013 06:44:37 GMT+0200 (CEST)	-8,85603	system
8	on	0	0	Mon Aug 26 2013 06:44:37 GMT+0200 (CEST)	-8,85603	system
9	on	0	0	Mon Aug 26 2013 06:44:32 GMT+0200 (CEST)	7,72993	system
10	on	1	0	Mon Aug 26 2013 06:45:04 GMT+0200 (CEST)	7,45813	system
11	on	1	0	Mon Aug 26 2013 06:45:08 GMT+0200 (CEST)	-6,15043	system
12	on	0	0	Mon Aug 26 2013 06:45:08 GMT+0200 (CEST)	-6,15043	system
13	on	0	0	Mon Aug 26 2013 06:45:04 GMT+0200 (CEST)	7,45813	system
14	on	1	0	Mon Aug 26 2013 06:45:36 GMT+0200 (CEST)	5,1937	system
15	on	1	0	Mon Aug 26 2013 06:45:38 GMT+0200 (CEST)	-2,08685	system
16	on	0	0	Mon Aug 26 2013 06:45:36 GMT+0200 (CEST)	5,1937	system
17	on	0	0	Mon Aug 26 2013 06:45:38 GMT+0200 (CEST)	-2,08685	system
18	on	1	0	Mon Aug 26 2013 06:46:07 GMT+0200 (CEST)	4,92839	system
19	on	1	0	Mon Aug 26 2013 06:46:08 GMT+0200 (CEST)	2,99304	system
20	on	0	0	Mon Aug 26 2013 06:46:07 GMT+0200 (CEST)	4,92839	system
21	on	0	0	Mon Aug 26 2013 06:46:08 GMT+0200 (CEST)	2,99304	system
22	on	1	0	Mon Aug 26 2013 06:46:38 GMT+0200 (CEST)	8,66211	system
23	on	1	0	Mon Aug 26 2013 06:46:49 GMT+0200 (CEST)	-7,28239	system

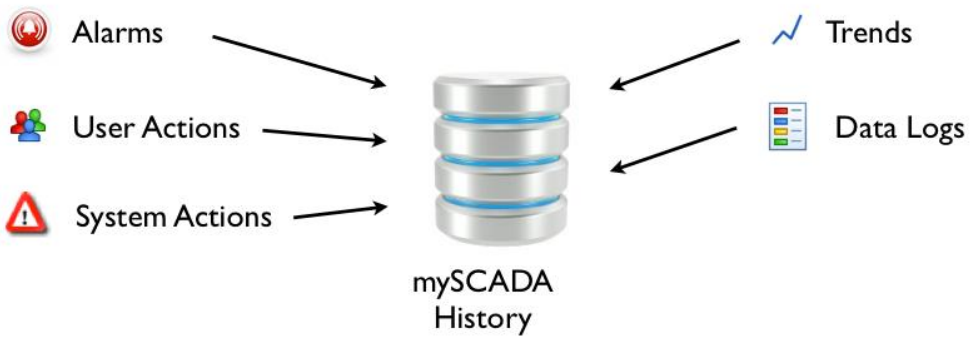
Figure 6 Exported alarm history to XLS

17. Data Logging

You can log and access a complete alarms history, all users actions, and any technological data you wish to log. The historical data are grouped into logical sections called *Data logs*.

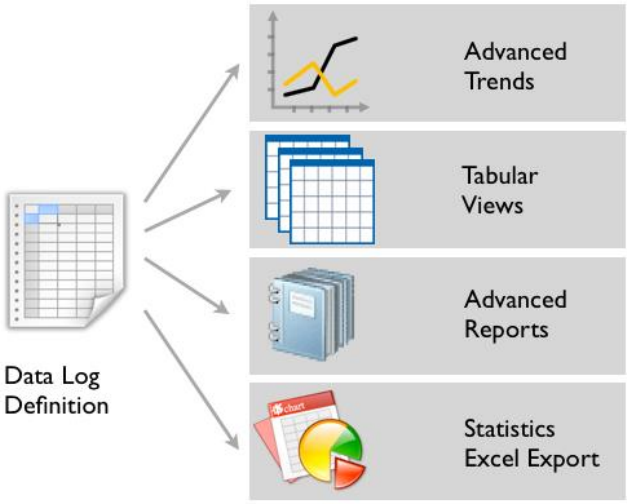
Data log has many options you can set up to tune your logging options. You can simply optimize your data logs for speed and storage.

Data logs can be periodic or event driven. Using *pre* and *post* event buffers, you can save collection of data before a specified event has happened.



17.1 Data Logs

You can log eventually any data or information available in *mySCADA*. For user convenience and easy access data are grouped into the *Data logs*. You can think of a data log as a collection of similar data. It can be for example a set of temperatures read each second from the PLC, motor start-up voltage and the current logged each 100 milliseconds, running hours of machinery, operator actions or computed production statistics. You can also log any user-defined variables from *Server-side Scripts* via a virtual PLC.



***Data logs**:* data sets grouped together. Each data log has a set of parameters, such as log period, *pre* and *post* event buffers and so on. Data logs are defined by the data you wish to read and log. Data is the collection of data points. Data points can be variables read from the PLCs, user defined variables and computed statistics from *Server-side Scripts*.

***Data point**:* can be either a numerical value or a value array. Numerical value can be of any numerical type such as *Boolean*, *Integer, Float, Double, Signed* or *Unsigned*. Numerical value is always automatically converted and logged as a double value. This way you do not have to care about the data type and its conversions. An array can represent a set of numerical values, buffer, string or date. Representing of an array is user-defined, however, you can change its type any time later - without data loss. You can freely combine numerical values and arrays in a single data log. Therefore, you can have values, strings and dates logged together.

There are two types of the *Data logs*:

Continuous Data Logs

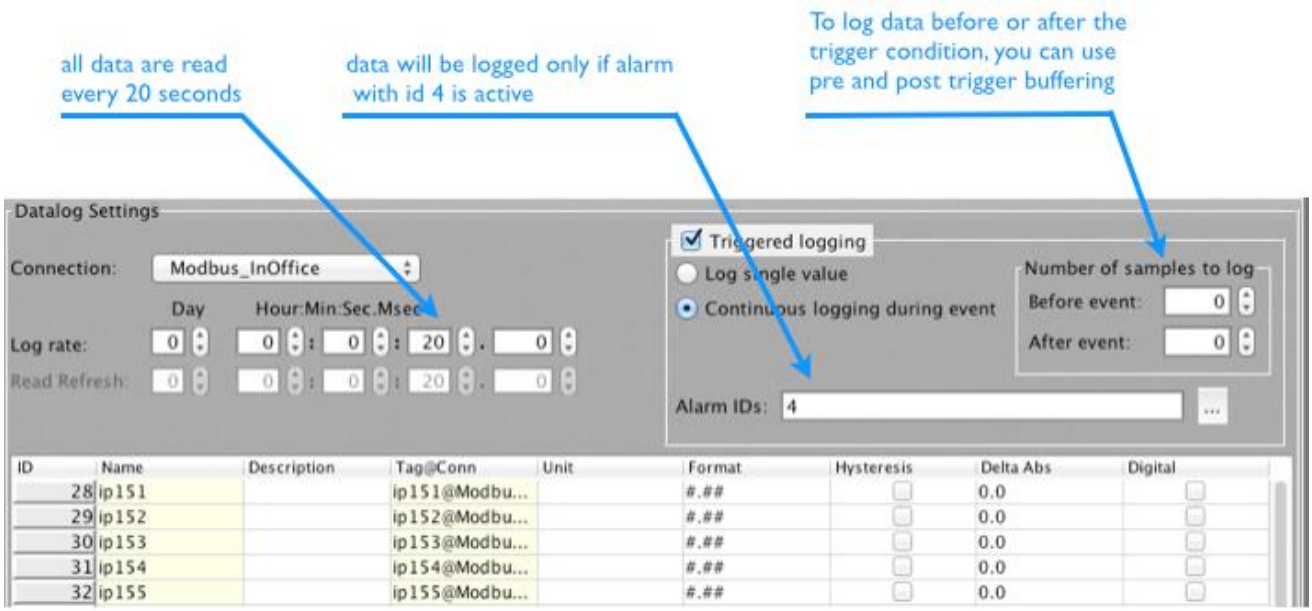
They log periodically without interruption, this type of data log is useful mainly for persistent processes.

Triggered Data Logs

Data logging, which is dependent on some event - condition. The condition is specified by the alarm ID. This type of data log is useful for repetitive or random processes where you can specify the triggering condition.

1) Go to the ***Data Logs*** section in the *Project Window*, right-click on the *Data logs* and select *Add New*. A new dialog window will show up, fill in the data log name and click on **OK**.

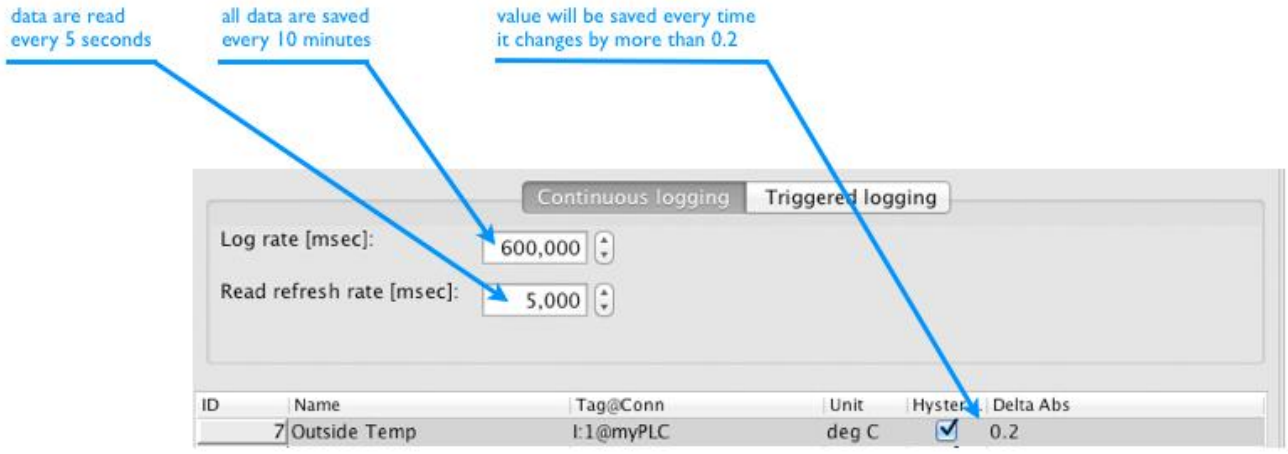
2) Double-click on your newly created data log in the ***Data Logs*** folder. A new definition window will show up. It is split into two horizontal sections: data log definition and data log views.



17.2 Continuous Data Logging

The purpose of the continuous data log is to log data periodically without interruption. This type of data log is useful mainly for persistent processes.

With the continuous data logging you define the read refresh rate in milliseconds - in this period all data defined in a data log are read from PLCs and logged into the database. For any data point you can define hysteresis of data logging. When hysteresis is enabled for a given data point it will be read each cycle, defined by the read period - however it will be logged only if its value has changed by more than of the defined hysteresis value. This can be especially useful for logging analog data that does not change so often. When you set hysteresis of some data points in the data log, you can specify the log rate – the period in which all data points of the data log will be logged, regardless whether they have changed or not. By enabling hysteresis, you can dramatically decrease the number of historical data logged while maintaining data precision and time continuity.



17.3 Triggered Data Logs

The purpose of the triggered (event driven) data log is to log data depending on some event - condition. This is useful for repetitive or random processes where you can specify the triggering condition. You can, for example, log the production data only at the time when the production line is running or log data upon the system failure and use the logged data for diagnostics. With event driven data log you specify the start of the event by a condition. If the trigger condition is met, the system starts recording the data.

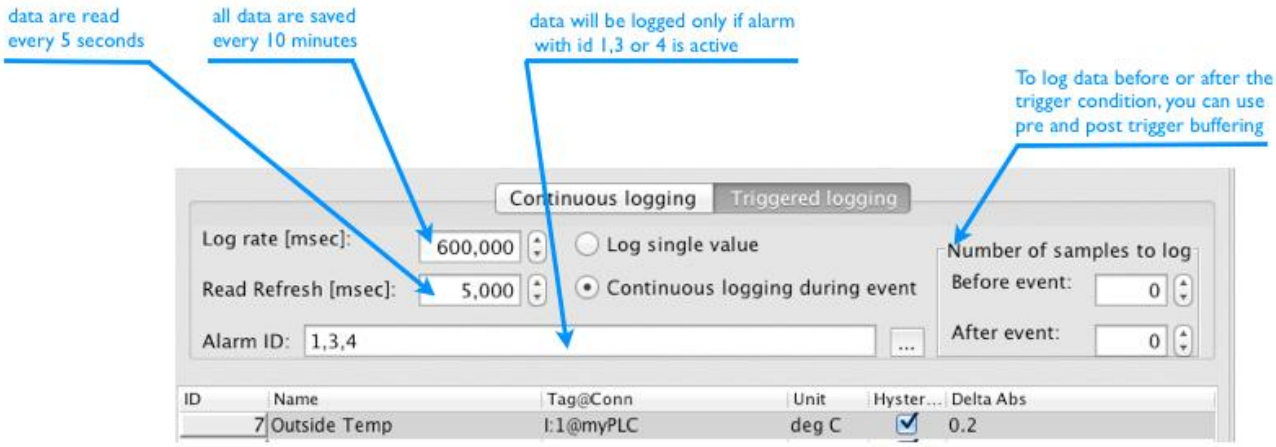


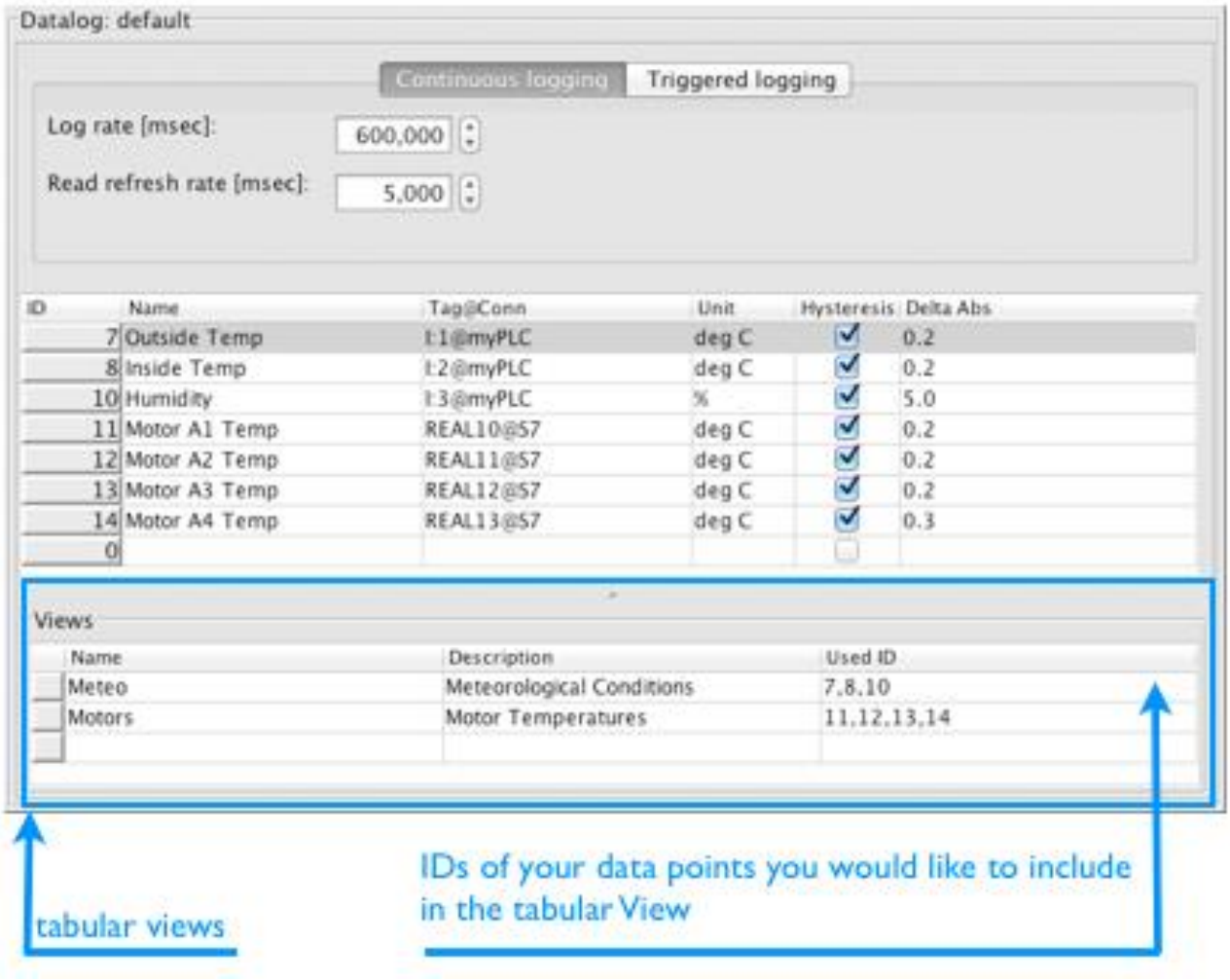
Figure 7 Triggered data log definition

Pre trigger buffering: you specify the number of time samples to log before the trigger condition is met. The system automatically keeps the number of defined time samples in the memory. If the event occurs, the system will flush all the buffered data into the database and continue logging.

Post trigger buffering: the system will continue to log your data even after the trigger condition has stopped. You should specify the number of the time samples to be logged after the trigger condition has finished. The time sample duration is equal to the read period.

17.4 Tabular Views

Each data log can have defined multiple tabular views. You can define the tabular views in the same window as you define the data log. Tabular views enable displaying all captured data from the data log in the form of a table.

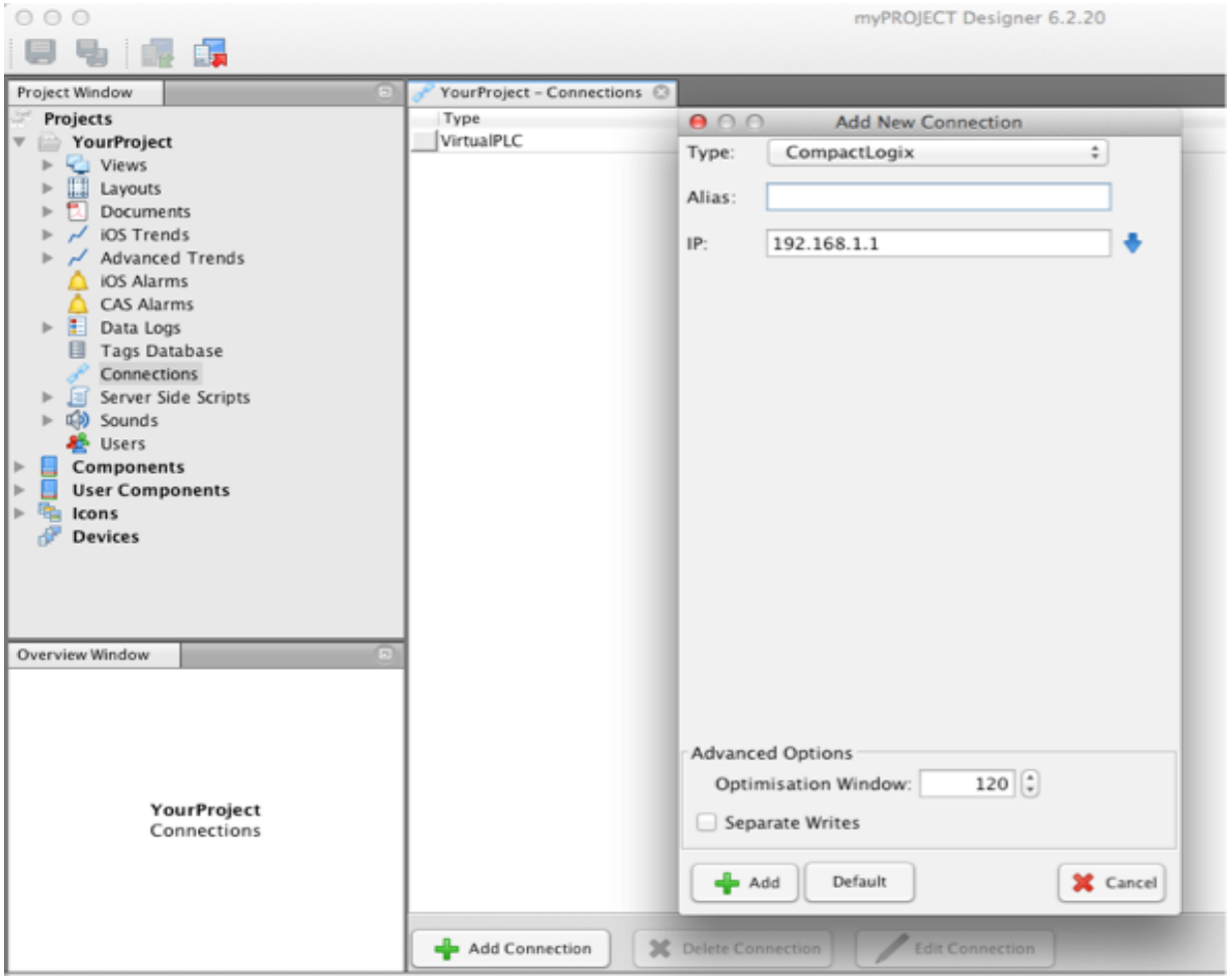


In this example, there is one data log with temperatures and humidity being logged. You can see the data are logged from two different PLCs (*myPLC* and *S7*).

There are defined two tabular views: *Meteo* and *Motors*. *mySCADA* automatically generates two tables for the user, based on the tabular views definitions. This way the user can browse and save the historical data or export them into a XLS file for further evaluation.

18. Connections

To view existing connections select the ***Connections*** folder in the *Project Window*.



1) Create a new connection by clicking on the **Add Connection** button at the bottom of the main window. A dialog window will open where you to enter specific connection parameters in order to properly create a viable communication channel with the PLC.

   - ***Type*** – select the type of PLC from the drop-down box at the top of the window
   - ***Alias*** – enter the name of the connection
   - ***IP*** – enter IP address of the PLC
   - ***Slot*** – select proper slot specification

   Note: The dialog content depends on the selected PLC type.

2) Once all the valid information is entered, click on **OK** at the bottom of the window.

If you want to delete some connections select them and click on the **Delete Connection** button right next to the **Add Connection** button at the bottom of the main window.

To edit any of the existing connections, double-click on the specific value you wish to change (*Type*, *Alias, IP, Slot*) and continue to make changes.

19. User Access Levels

Security is the major concern for every modern SCADA system. In *mySCADA* you can limit access from the whole project viewing to a single element control. This way you can easily control user access to your project and keep the security at the highest level. Also, if there is more than one person using the technology you can utilize the *user access* function to set certain limitation.

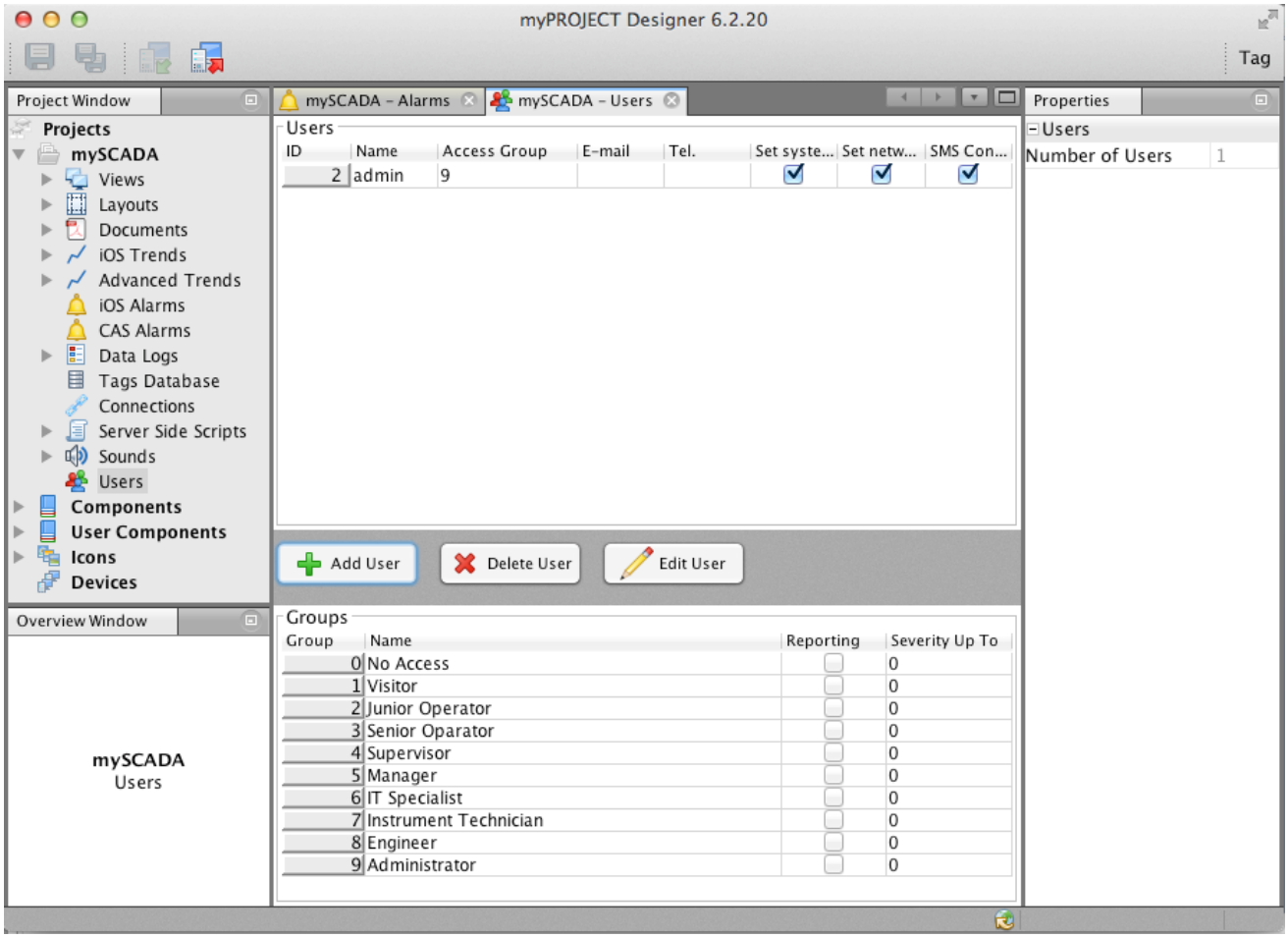
19.1 Access Levels

Access Levels are security groups with chosen level of access. You can set different access levels for different users. For example, maintenance personnel would have lower system access level than administrators, but higher than operators. This way you can control who can access your system and who can operate given technology. *mySCADA* has ten *Access Levels* numbered from "0" to "9". "0" (zero) is the lowest access available while "9" is the highest access level possible.

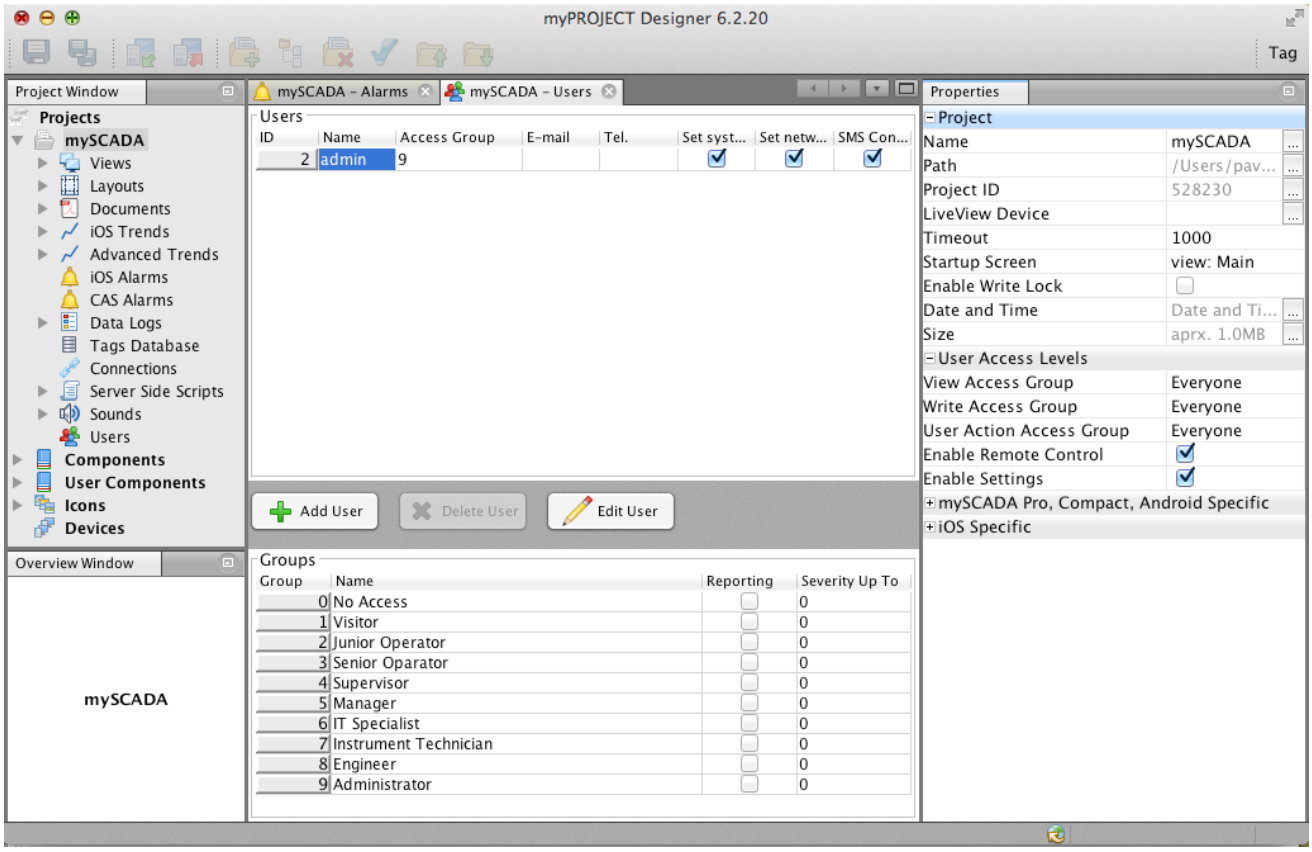
For better user orientation you can give specific names to these user access levels

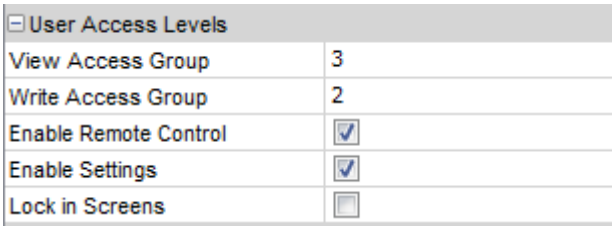
19.2 Specify User Accounts

In order to use User Access Levels, you have to specify users for your project. You can create as many users as you need. Each user must have specified an Access Group ranging from 0 to 9 - the higher the number the higher access rights for the user.



Limit access for whole project





View Access Group: Sets the minimal level of access for viewing.

Write Access Group: Sets the minimal level of access for writing to PLC.

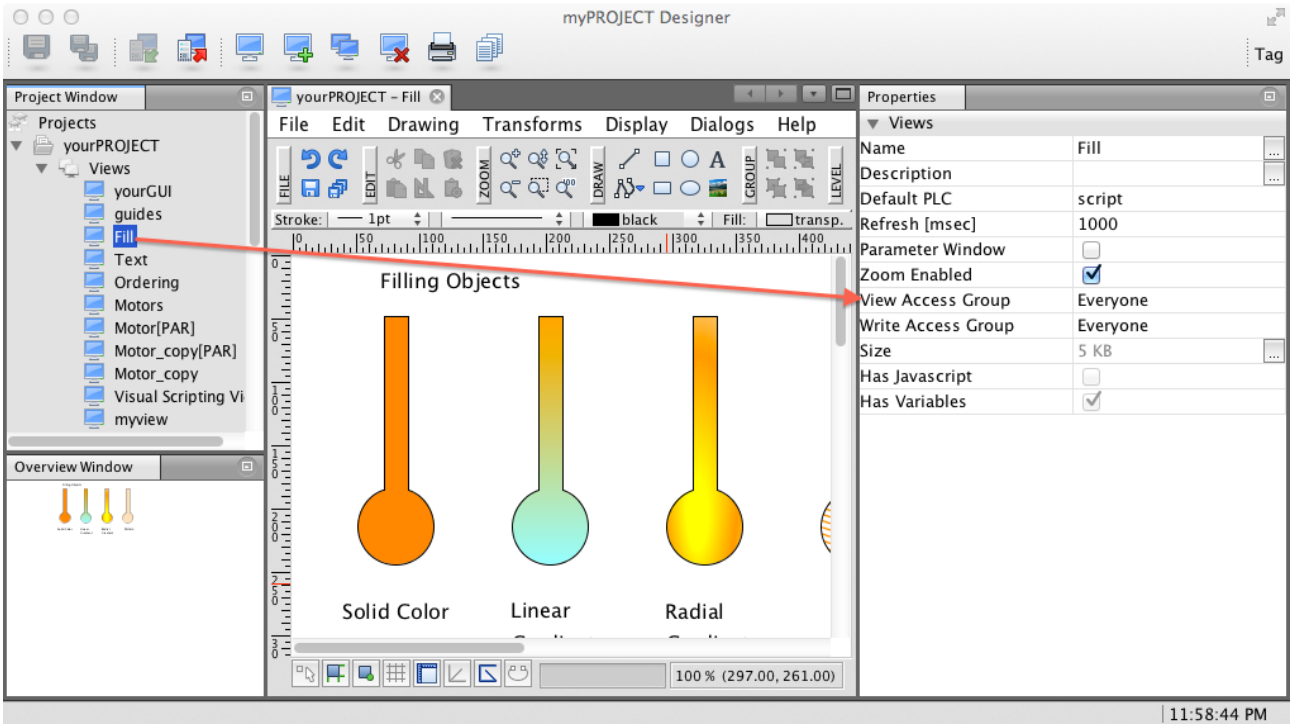
Enable Remote Control: You can disable access from *myPROJECT Designer* by unselecting this option. Once unselected, you will not be able to access *mySCADA* from *myPROJECT Designer*. You can always re-enable this option from the *mySCADA* settings page.

Enable Settings: Gives users access to the setting page on *Apple iOS* devices.

Lock in Screens: Allows users to access certain views only.

Limit access for project screens

You can set the user access level for each view separately. Select a desired view in the *Project Window* to display the view properties.

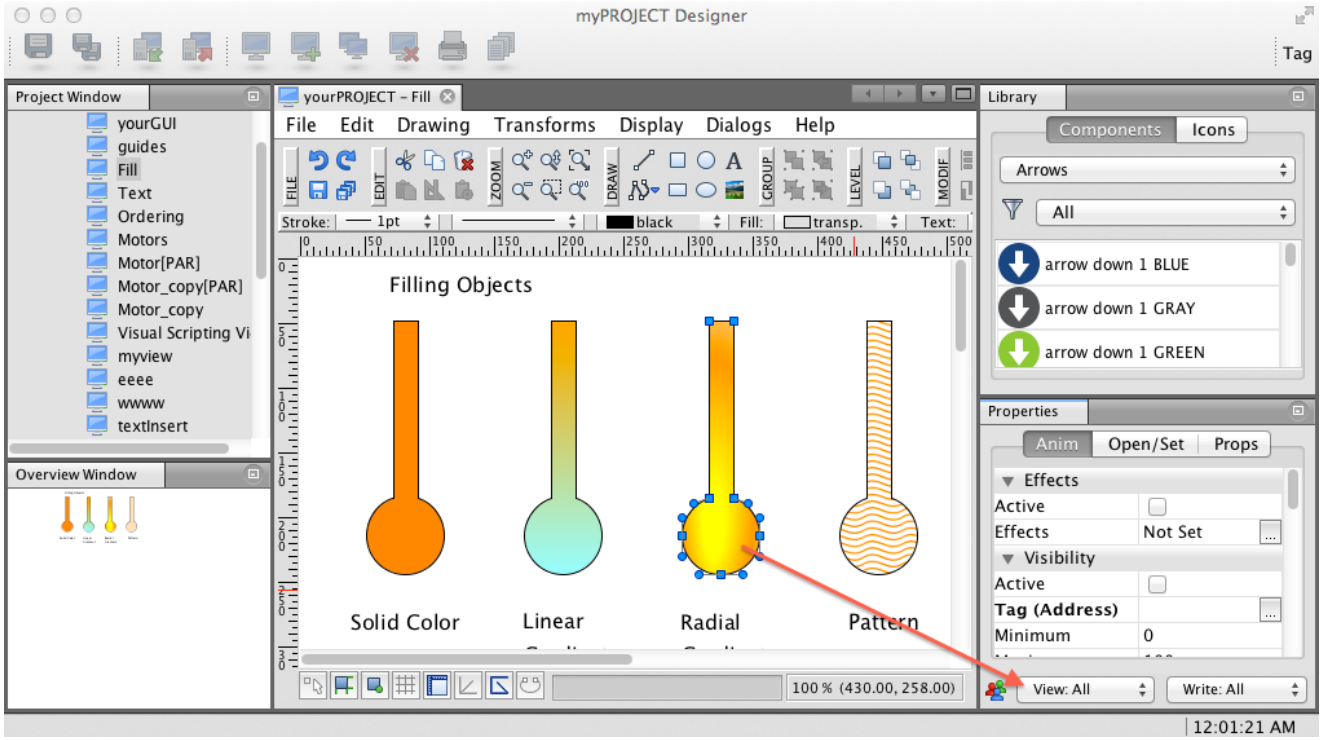


Here you can set the access levels for the screen viewing and writing to the PLC separately:

View Access Group	3
Write Access Group	5

In this example users with access the level 3 (or higher) can view the visualization screen, and users with the access level 5 (or higher) can execute the *Set commands* defined in this view.

Limit access of arbitrary object in views



You can set the user access rights even to a lower detail. The access rights can be set for each object in a view, allowing creation of complex and customizable visualizations.

Select the target object and set the *View* and *Write Access* values:

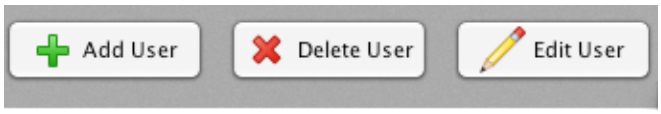
View Access Group	3
Write Access Group	5

Users

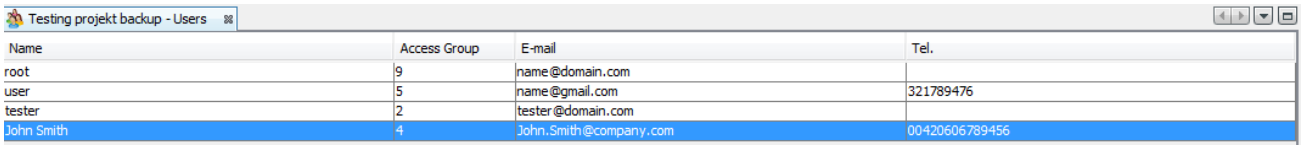
In this module you can define the users of *mySCADA* application.

After clicking on the ***Users*** folder a list of defined users will appear in the main window.

At the bottom you can find buttons for managing the user records.



Click on **Add User** and fill in the user name, password, E-mail or telephone number in the dialog window. You can also select the access level group for accessing certain project elements. The newly added users will be shown in the users list.



Name	Access Group	E-mail	Tel.
root	9	name@domain.com	
user	5	name@gmail.com	321789476
tester	2	tester@domain.com	
John Smith	4	John.Smith@company.com	00420606789456

20. Server-side Scripts

20.1 Introduction

Server-side scripting is an easy to use, yet extremely powerful option to extend the functionality of your *mySCADA* system. Server-side scripting uses *JavaScript* as a primary language for writing scripts, which is one of the most simple, straightforward, versatile and effective scripting languages. It is relatively easy to learn as it uses syntaxes close to English. Also, you can find a lot of resources and *JavaScript* libraries on the web - one of the main reasons why *JavaScript* has been chosen for server-side scripting on *mySCADA* platform.

The execution of *JavaScript* is based on excellent *V8* library from *Google,* which is now the fastest *JavaScript* engine available. Rather than interpreting *JavaScript,* as the old engines used to do, *V8* uses the *Just-In-Time compiler* to produce and execute native instructions, tailored to a processor on which the application is running. The generated instructions are cached - avoiding the overhead of repeated code generation - and deleted if no longer needed.

For networking and advanced functionalities *mySCADA* server-side scripts use the *NODE.JS* toolbox for building fast and scalable network applications. *NODE.JS* uses an event-driven, non-blocking I/O model that makes it lightweight and efficient, perfect for data-intensive real-time applications.

What can you achieve with *mySCADA* server-side scripting:

- process and analyze the PLC data
- compute statistical data
- create reports and serve them over a web server to a client
- implement complex alarming
- communicate with devices over Ethernet or Serial line
- implement own protocols for specialty devices

Look at the example below: *HTTP server retrieves data from the PLC and sends them to a user who opens the web page of mySCADA Box on the port 8000:*

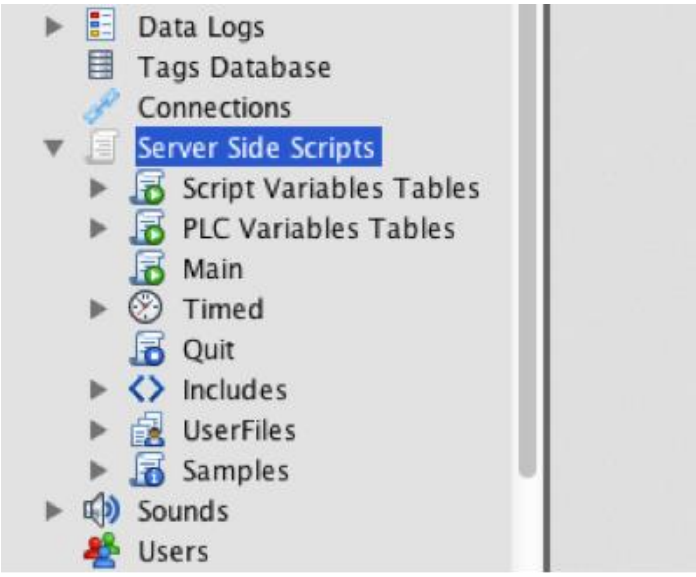
```
//we will implement our own web server
//which will accept connections on port 8000
var http = require('http');
http.createServer(function (req, res)
{
     res.writeHead(200, {'Content-Type': 'text/plain'});
     res.end("Our first useful script.");
}).listen(8000);
```

Take this function and put in the main script of your *mySCADA* system. Download the project into your *mySCADA* device and open the web browser on: *http://mySCADABoxIP:8000.*

As you can see, you can simply extend your *mySCADA* system functionality. Now, you can for example save PLC data every minute, compute statistics and use your web server for showing data in PDF.

20.2 Server-side Scripts folder

The ***Server-side Scripts*** folder is located in the *Project Window* menu, as displayed in the picture below. After opening this section the *Scripts toolbar* will appear.



20.3 Scripts Toolbar

Open Script – opens a current script in the script editor

Add Script – creates a new script

Import Script – imports a single file or folder into the current folder.

Duplicate Script – creates a copy of a current script

Delete Script – deletes a current script

Report Wizard – this feature helps you create a script report step-by-step

Open Table – opens a table

Add Table – creates a new table

Delete Table – deletes a current table

Export to Excel – exports data to XLS file

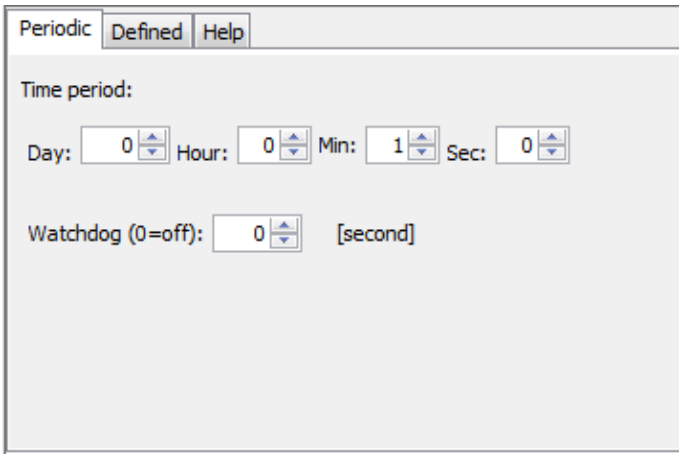
Import from Excel – imports data from XLS file

20.4 Server-side Scripts Folder Structure

Main – the main script executed only once, it cannot be deleted but can be left empty

Timed – contains timed scripts, executed periodically - the period settings can be executed by two modes, *Periodic* and *Defined*. In the *Periodic* mode you specify the delay between executions, in the

Defined mode you specify the times when the script should be executed. These options can be selected by bookmarks in the *Variables Window* at the bottom of the main window - settings can be done in four ways (Hourly, Daily, Weekly, and Monthly), also you can set the *Watchdog*, which terminates the process if the run-time exceeds the user-specified time.



Quit – executed upon the system reboot or server-side scripts restart, so it can be used e.g. for correct terminating of communication or reports generating

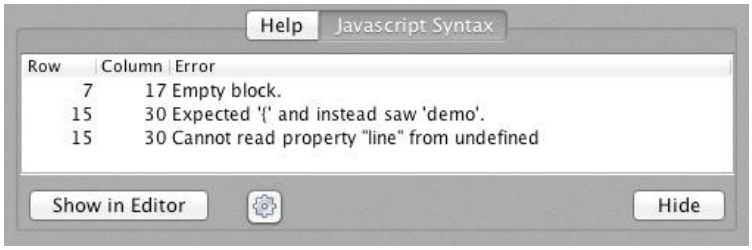
Includes – directly linked with *NODE.JS*, where you can store modules you want to use in scripting and call them only by their name, i.e. without the path - like the *NODE.JS* built-in modules

UserFiles – used for the script files, it can be accessed from server-side scripts with a relative address "*./UserFiles*"

Samples – examples of basic server-side scripting functions that can be used as a base for more complex user scripts, if you want to use them create a copy into the *Main* script or the ***Includes*** folder

20.5 Script Editor Functions

Check syntax and ***Check syntax setting*** – as described previously (see the chapter *Check Project)* this function checks if the script syntax is correct, if some errors found they will be listed in a small window at the bottom of the main script window



Save – this button saves the current script

Undo and Redo – these buttons allow you to reverse or repeat the last editing actions

fx – this field shows a list of all used functions in the script

Find Next and Find Previous – use these buttons for searching certain words in the script

Replace – this function allows you to replace words, functions or symbols with different ones at once (similar to the *"Find and Replace"* function in any other text editor)

20.6 Event-driven Asynchronous Callbacks

The *NODE.JS* approach is not unique, but the underlying execution model is different from other runtime environments like *Python, Ruby, PHP* or *Java*.

Let us take a look at the following part of the code:

```
var result = database.query("SELECT * FROM hugetable");
console.log("Hello World");
```

The first line queries the database for lots of rows and the second line puts *"Hello World"* to the console.

Let us assume that the database query is really slow, due to the number of rows and it takes too long to execute.

With codes written this way the *JavaScript* interpreter of *NODE.JS* has to read a complete result set from the database and then execute the *console.log()* function.

If this piece of code were written e.g. in PHP it would work the same way -> *"read all the results at once, then execute the next line of the code".* If this code were part of a web page script, the user would have to wait several seconds for this page to load. However, in the PHP execution model, this would not become a "global" problem, i.e. the web server starts its own PHP process for every HTTP request it receives. If one of these requests results in a slow execution of the code it slows down the page loading only for a particular user and does not affect other users.

The execution model of *NODE.JS* is different - there is only one single process. If there is a slow database query somewhere in the process, it affects the whole process - everything comes to a halt until the slow query has finished.

To avoid this, *JavaScript* and therefore *NODE.JS* introduce a concept of event-driven, asynchronous callbacks by utilizing an event loop.

We can understand this concept by analyzing the re-written version of the problematic code:

```
database.query("SELECT * FROM hugetable", function(rows)
{
 var result = rows;
});
console.log("Hello World");
```

Instead of expecting *database.query()* to return the result directly, we pass it the second parameter - an anonymous function.

In the previous form the code was synchronous: *"first do the database query, and only when this is done write to the console".*

Now, *NODE.JS* can handle the database request asynchronously, provided that *database.query()* is part of the asynchronous library. It takes the query and sends to the database, but instead of waiting for it being finished it makes a 'mental note' saying "*when at some point in the future the database server is done and sends the result of the query, then I have to execute the anonymous function passed to database.query().*"

Then, *NODE.JS* immediately executes *console.log()* and enters the event loop. It continuously cycles through this loop again and again, even though there is nothing else to do - events such as "*slow database query*" finally deliver their results.

Note: This asynchronous, single threaded, event-driven execution model is not infinitely the only and the best option - it is just one of several models with its limitations (one being that NODE.JS is just a single process capable to run on one single CPU core). However, this model is quite approachable, because it allows writing applications that must deal with completion in an efficient and relatively straightforward manner.

You might want to take time to read Felix Geisendörfer's post on "Understanding NODE.JS" for additional background explanation.

20.7 Organizing Applications into Modules

Example:

Let us have a look how to organize the application containing a code for a very basic HTTP server:

```
var http = require("http");
function onRequest(request, response){
      response.writeHead(200, {"Content-Type": "text/plain"});
      response.write("Hello World !!!");
```

```
        response.end(); }
http.createServer(onRequest).listen(port, host);
```

Now, let us turn this code to a real *NODE.JS* module that can be used by the main script or the timed scripts.

As you may have noticed the modules are used in the code as follows:

```
var http = require("http");


...

http.createServer(…);
```

In *NODE.JS* there is a module called *"http",* that can be used in the code by requiring it and assigning the result of this requirement to a local variable. This turns the local variable into an object carrying all the public methods that the *"http"* module provides.

It is a common practice to use the module name as the local variable name, however you may choose whatever you like:

```
var foo = require("http");
...
foo.createServer(...);
```

Now, it should be clear how to use internal *NODE.JS* modules.

To create and use your own modules you do not have to change that much. Making some code a module means that you need to export those parts of its functionality that you want to provide to scripts requiring this module.

For now, the functionality the HTTP server needs to export is simple: scripts requiring the server module simply need to start the server.

To make this possible, put your server code into a function named *start()* and export it:

```
var http = require("http");

function start() {
        function onRequest(request, response) {
        response.writeHead(200, {"Content-Type": "text/plain"});
        response.write("Hello World !!!");
        response.end();
        }
http.createServer(onRequest).listen(8888);
}

exports.start = start;
```

Now, you just save this code into the ***Includes*** folder and name it *"server.js"*. You can start the HTTP server from the main or timed scripts:

```
var server = require("server");
server.start();
```

Note: You can use the server modules as internal modules - require their files and assign to variables -> exported functions then become available.

20.8 Creating Reports

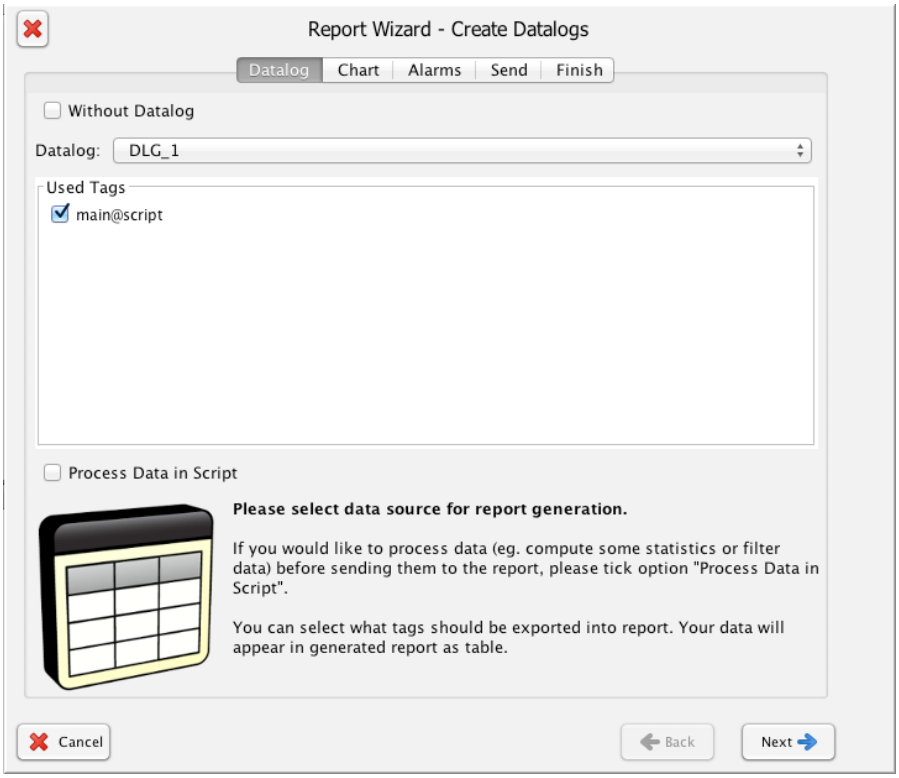
There are two ways of report creating in server-side scripts:

- Using the *Report Wizard*
- Manual report creating

Report Wizard

This feature helps you create a report in the scripts step-by-step which saves you plenty of time spent if creating the reports manually.

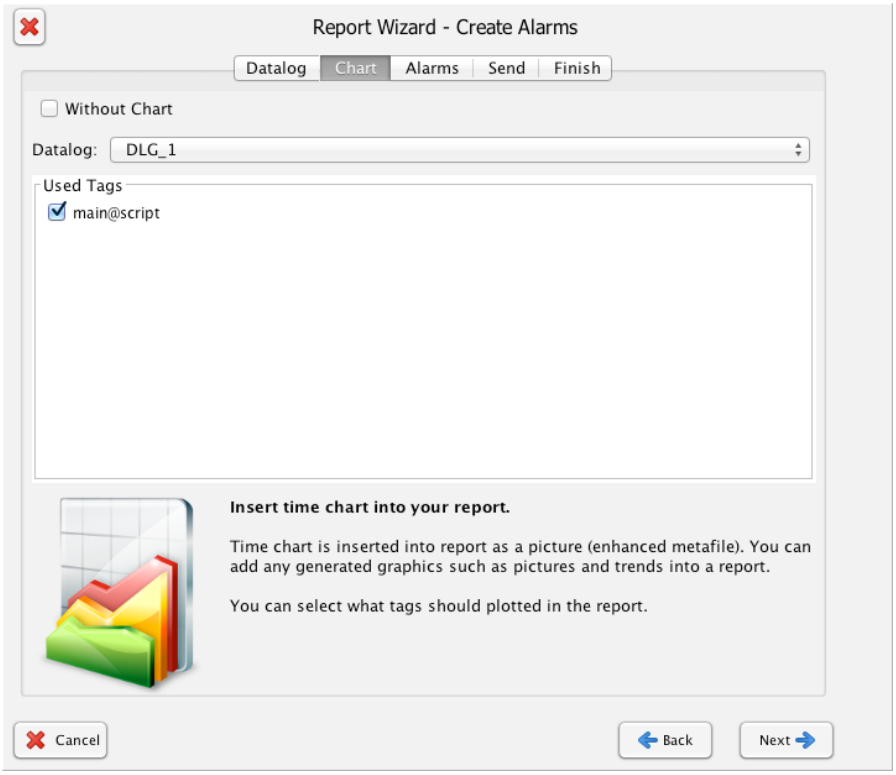
1) Select the ***Server-Side Scripts*** folder in the *Project Window* and click on the **Report Wizard** icon in the toolbar. The following window will appear:



2) In the tab **Datalog** select data-logs and tags to be included in the report table. You can also activate data processing in the script - tick off the box *Process Data in Script*. If you wish to

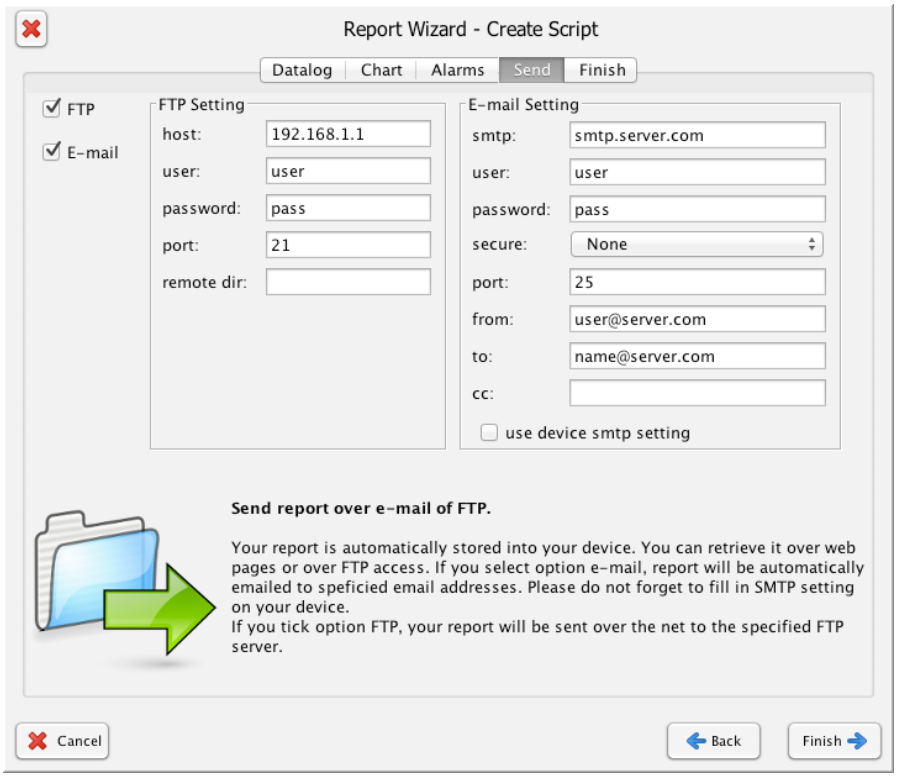
create a report without data-logs tick off the box *Without Datalog,* then click on the button **Next**.

3) The following tab **Chart** allows you to include a time chart of data-logs in the report. You can select data-logs and tags to be displayed. If you do not wish to include the charts in the report tick off the box *Without Chart*. Click on the **Next** button.

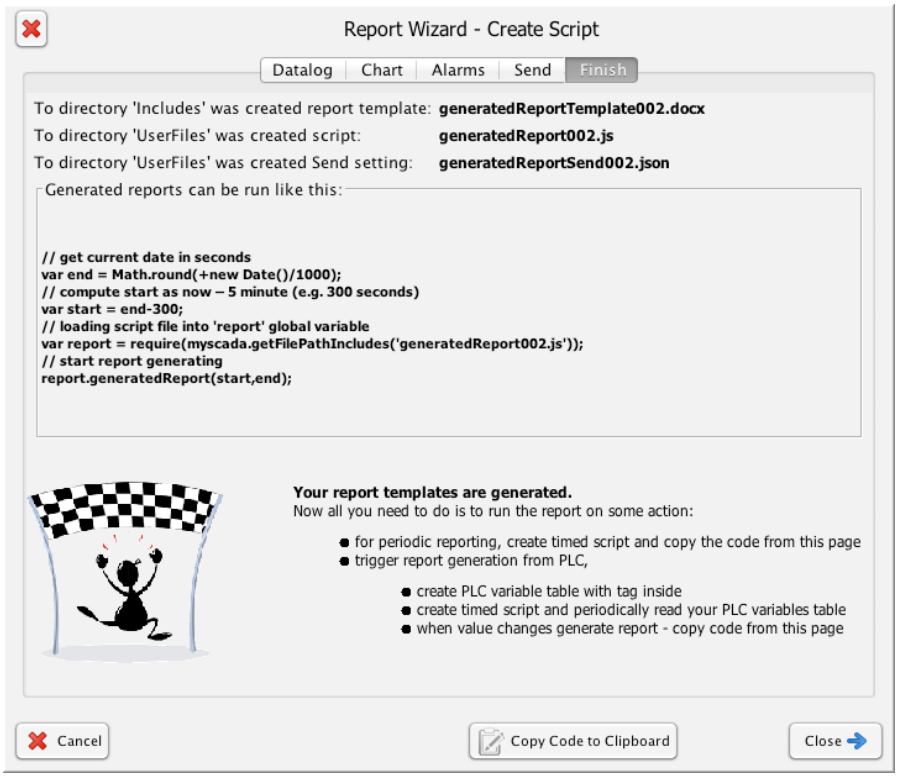


4) In the tab **Alarms** you can select alarms to be included in the report, then click on the button **Next**.

5) In the **Send** tab you can select the way the report from your project will be sent. Fill in the necessary information as shown on the picture below and click on **Next**.

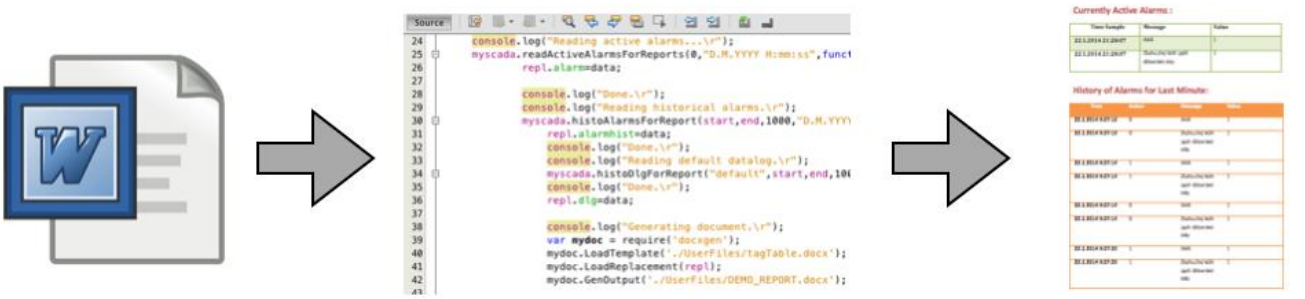


6) In the last tab you will see the list of generated report template files. Now, you have to set triggering actions for your report to be sent. Click on the button **Copy Code to Clipboard** to use a generated report script for a later use and click on **Close**.



Manual report creating

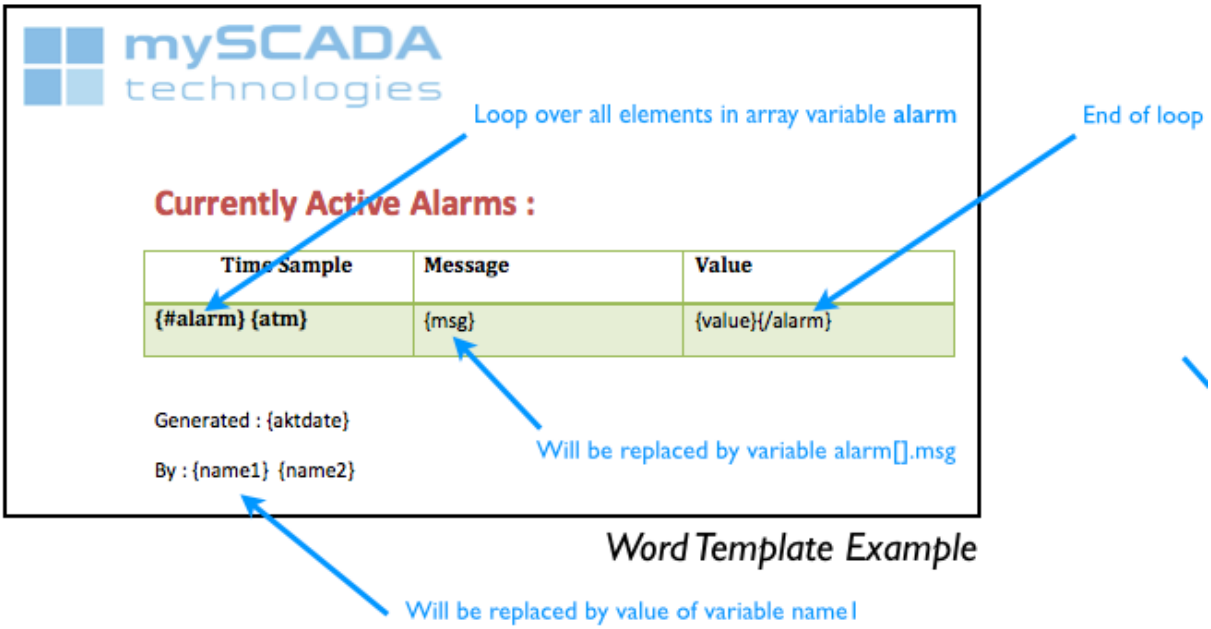
Report templates are created in *MS Word* (and other compatible text editors) with a simple syntax. During report processing all defined variables inside a Word document are replaced with the script data.



Example:

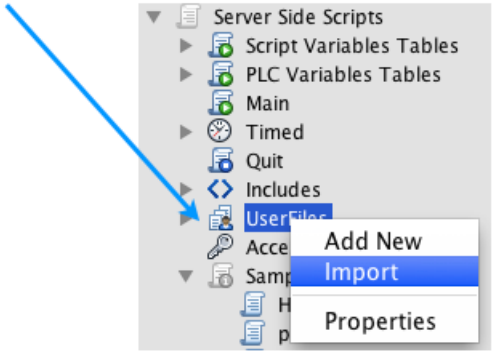
1) Prepare a report in MS Word (or any compatible text editor)

   a) To replace a single variable put it in a single bracket: {variable}
   b) To create a loop you can use loop opening {#} and loop closing {/} brackets



Word Template Example

c) Import the document into ***UserFiles*** in the ***Server-side Scripts*** folder:



2) Create a script to fill in the *MS Word* report with data:

script example which reads active alarms from the system and fills table in word template

a) create replacement variable

b) fill it with data

c) require "docxgen"

d) open MS Word template file

e) generate report

f) save it to file

```
// A) Create Replacement variable repl and fill it with
// some data such as Name and Surname
repl = {
        "name1" : "Petr",
        "name2" : "Svoboda"
        };

 //Read active alarms from the system
 myscada.readActiveAlarmsForReports(0,"D.M.YYYY H:mm:ss",
function(status,data) {
    //add data into replacement variable under the name alarm
    //use same variable name as in script
    repl.alarm=data;

    //create the document engine by require docxgen
    var mydoc = require('docxgen');
    //load your template saved in UserFiles
    mydoc.LoadTemplate('./UserFiles/Template.docx');
    //perform the replacements with our repl variable
    mydoc.LoadReplacement(repl);
    //save newly created report into file
    mydoc.GenOutput('./UserFiles/DEMO_REPORT.docx');

    myscr.A.value=0;
    myscada.writeTable("myscr",function(err){});
});
```

Note: You can use the timed scripts for report generating in predefined intervals or certain dates.

20.9 Scripting Functions

GENERAL FUNCTIONS

myscada.logFile(log);

This function prints the default log into the report, each log entry ends with a new line. You can access the log file from the status on *mySCADA Box* or directly - *mySCADA Desktop*, it is useful for debugging.

myscada.logFileClear();

This function clears the default log file (see the function *logFile(log)*).

myscada. getFilePathUserData(filename);

This function returns the full path with a specified filename to a default user folder for storing generated data, which can be accessed by users. The user folder can be accessed by FTP or WWW (if enabled in the section *Server-side Scripts -> Accesses*).

myscada.getFilename(filename);

This function is identical to *getFilePathUserData(filename)*.

myscada. getFilePathUserFiles(filename);

This function returns the full path with a specified filename to a default ***UserFiles*** folder.

myscada. getFilePathUserIncludes(filename);

This function returns the full path with a specified filename to a default ***Includes*** folder.

myscada.requireIncludes();

This function provides a way to load modules from the ***Includes*** folder.

Example:

```
var module = myscada.requireIncludes('myModule.js');
module.myfunction();
```

myscada.deleteUserData();

This function deletes all data from the user data folder.

myscada.usedSpaceUserDataFolder();

This function returns a used space in the user data folder in MB (available only on *mySCADA Box*).

myscada.freeSpaceUserDataFolder();

This function returns a free space to the user data folder in MB (available only on *mySCADA Box*).

myscada.getStringFromArray(values);

This function converts an array into a string and back, e.g. values from *readTable()*.

Example:

```
myscada.readTable('myTable',function(status) {
   if (!status) {/* read error processing */}
   else {/* successful read processing */
      var values = myTable.myTag.values;
      var text = myscada.getStringFromArray(values);
   }
});
```

myscada.sendSMS(number,text);

This function sends a message SMS to a set phone number.

Example:

```
var telNumber = '123456789';
var smsText = 'SMS text from mySCADA';
myscada.sendSMS(telNumber,smsText);
```

myscada.sendMail(filename, filenamepath, mailsubject, mailtext, mailto, cc, host, secure, port, user, pass, from);

This function sends an email to a defined email address.

Example:

myscada.sendMail("Report.docx", myscada.getFilename("Report.docx"), "Report mail", "This is mySCADA generated Report.", 'mailto@server.com', 'mailcc@server.com', 'smtp@server.com', 'None', 25, 'user', 'pass', 'from@server.com')

myscada.sendMailJson(filename, filenamepath, mailsubject, mailtext, config);

This function sends an email to a defined email address.

The parameters are the same as for the previous function with the **config** parameter, setting the *Json* email structure.

Example:

var config = require(myscada.getFilePathUserFiles("smtp.json"));

myscada.sendMailJson("Report.docx", myscada.getFilename("Report.docx"), "Report mail", "This is mySCADA generated Report.", config);

ONLINE DATA

myscada.readTable(table_name,callback)

This function reads a complete table (all variables from the table) with a table name provided as the first function parameter, as it is defined in the *PLC Variables Tables* or the *Script Variables Tables*. The *Callback* function will be supplied with the read status (1 – read successful, 0- read error). Upon a successful reading, after the *Callback* function has finished, the value(s) of all variables will be filled with the read data.

Note: The read data are available inside the Callback function also - for multiple tables call this function multiple times with required parameters.

Example:

```
myscada.readTable('myTable',function(status) {
	if (!status)
	{/* read error processing */
	}
	else
	{/* successful read processing */
	var myReadValue=myTable.myTag.value;
	var myReadStatus=myTable.myTag.status;
	}
});
```

myscada.writeTag(tag_name,callback);

This function writes a single tag, already defined in one of the tables - *PLC Variables Tables* or *Script Variables Tables*. The *Callback* function will be supplied with the write status (1 – write successful, 0- write error). If you do not need checking for writing errors you can pass null instead of the *Callback* function.

Example:

```
myscada.writeTag('myTable.myTag',function(status) {
	if (!status)
	{/* write error processing */}
	else
	{/* successful write processing if needed */ }
});
```

myscada.writeTable(table_name,callback);

This function writes all tags, defined in the PLC or *Script Variables Table,* which is supplied as the first parameter. Only the tags with value(s) different from null will be written, the *Callback* function will be supplied with the write status (1 – write successful, 0- write error).

Example:

```
myTable.myTag_1.value=100;
myTable.myTag_2.value=null;
/* only myTable.myTag_1 variable will be written as it's not null
*/

myscada.writeTable('myTable',function(status) {
	if (!status)
	{/* write error processing */
	else
	{/* successful write processing if needed */ }
});
```

myscada.readActiveAlarms(severity,callback);

This function reads all currently active alarms. The first parameter filters out all active alarms with a severity lower than the value of a supplied parameter. The *Callback* function will be supplied with the read status (1 – read successful, 0 - read error) and the list of active alarms as a data object.

Example:

```
myscada.readActiveAlarms(0,function(status,data) {
	if (!status)
	{/* read error processing */}
	else {/* successful read processing */ }
});
```

The active alarms are saved in a data object structure you receive in the *Callback* function. The data object is an array with the following object structure:

data[].

atm activation time [Date]

dtm deactivation time [Date]

acktm acknowledge time [Date]

msg text description of alarm [String]

area area (geografical or logical) [String]

dev device [String]

stat alarm status [Integer]

(0 – non-active,1 – active, 2 – acknowledged ,8 – suppressed)

sv severity

val current value

acktval value when alarm become active

deactval value during deactivation

myscada.readAlarmsStatus(severity,callback);

This function returns a number of active and acknowledged alarms. The *Callback* function will be supplied with the read status (1 = read successful, 0 = read error) and with a data object as the second parameter.

Example:

```
myscada.readAlarmsStatus(0,function(status,data) {
        if (!status)
        {/* read error processing */}
        else
        { /* successful read processing */
        var number_of_active_alarms=data.active;
        var number_of_acknowledged_alarms=data.ack; }
});
```

The data object has the following structure:

data

.active number of active alarms [Integer]

.ack number of acknowledged alarms [Integer]

HISTORICAL DATA

myscada.histoDlg('dlg_name',start,end,limit,callback);

This function returns historical data from a given data-log. In the parameter **dlg_name** you specify the data-log name, the parameter **start** is the starting date, specified as a number of seconds since 1970/01/01, the parameter **end** is the ending date, again specified as a number of seconds since

1970/01/01. The**Limit** parameter is the maximal number of records returned. You should also specify the **callback** function to execute when the file is saved.

Example:

```
//get current date in seconds
var end = Math.round(+new Date()/1000);
//compute start as now - 1 minute (e.g. 60 seconds)
var start = end-60;
myscada.histoDlg("default",start,end,10,function(status,data) {
       if (!status)
       {/* read error processing */}
       else
       { /* successful read processing */ }
});
```

The returned object data is an array where each element has the following structure:

data[].

tm time [Date]

1 first column [Double]

2 second column [Double]

… … …

n last column [Double]

myscada.histoDlgCSV('filename','dlg_name','format',start,end,limit,callback)

This function reads historical data from a given data-log and saves to a newly created file, specified in the first parameter, **filename** - if the file already exists it will be overwritten (the saved data format is CSV). In the parameter **dlg_name** you specify the data-log name. The **format** parameter is used for setting of the date format (see *Formatting Dates* section in this manual). The **start** parameter is the starting date, specified as a number of seconds since 1970/01/01. The **end** parameter is the ending date, also specified as a number of seconds since 1970/01/01. The **tags** parameter stands for the number of columns you want to read. The **limit** parameter is the maximal number of records returned. You can also specify the **callback** function to execute when the file is saved.

Note: If you specify only the file name it will be automatically saved into the user folder, which you can access by FTP or WWW (if enabled in the section Server-side Scripts → Accesses). If you like to use a different location, specify the complete path to the file.

Example:

```
//get current date in seconds
var end = Math.round(+new Date()/1000);
//compute start as now - 1 minute (e.g. 60 seconds)
var start = end-60;
```

```
myscada.histoDlgCSV(myscada.getFilename('example.csv'),'default',
'MMMM Do YYYY, h:mm:ss a',start,end,3,function(status) {
        if (!status)
        {/* read error processing */}
        else
        { /* successful read processing */ }
});
```

Output file “example.csv” :

timestamp;v1;v2;v3;v4;v5

January 26th 2014, 8:37:15 pm;31.9244;33.0909;33.4595;34.1234;30.2989

January 26th 2014, 8:37:10 pm;34.4499;31.0198;33.9934;33.3424;33.5842

January 26th 2014, 8:37:00 pm;30.9228;31.3836;31.4809;30.5054;33.3162

myscada.histoDlgAppendCSV('filename','dlg_name','format',start,end,limit,callback)

This function is identical to *histoDlgCSV()*, but instead of creating a new file, it appends data into an already existing file. If the file does not exist it will be created. If the file already exists it will append the data to the file end.

myscada.histoDlgTags(dlg_name,start,end,”tags1,tags2”,limit,callback);

This function returns historical data from a given data-log. In the parameter **dlg_name** you specify the data-log name, the **start** parameter is the starting date, specified as a number of seconds since 1970/01/01. The **end** parameter is the ending date, also specified as a number of seconds since 1970/01/01. The **tags** parameter stands for the number of columns you want to read. The **limit** parameter is the maximal number of records. The **limit** parameter is the maximal number of records returned. You should also specify the **callback** function to execute when the file is saved.

Example:

```
//get current date in seconds
var end = Math.round(+new Date()/1000);
//compute start as now - 1 minute (e.g. 60 seconds)
var start = end-60;
myscada.histoDlg("default",start,end,”1,2”,10,function(status,data)
{
        if (!status)
        {/* read error processing */}
        else
        { /* successful read processing */ }
});
```

The returned object data is an array and every object in this array has the following structure:

data[].

tm time [Date]

1 first column [Double]

2 second column [Double]

… … …

n last column [Double]

myscada.histoDlgTagsCSV(filename,dlg_name,format,start,end,tags,limit);

This function reads historical data from a given data-log and saves to a newly created file, specified in the first parameter, **filename** - if the file already exists it will be overwritten (the saved data format is CSV). In the second parameter, **dlg_name** you specify the data-log name. The **format** parameter is used for setting of the date format (see *Formatting Dates* section in this manual). The **start** parameter is the starting date, specified as a number of seconds since 1970/01/01. The **end** parameter is the ending date, also specified as a number of seconds since 1970/01/01. The **tags** parameter stands for the number of columns you want to read. The **limit** parameter is the maximal number of records returned. You can also specify the **callback** function to execute when the file is saved.

Note: If you specify only the file name it will be automatically saved into the user folder, which you can access by FTP or WWW (if enabled in the section Server-side Scripts → Accesses). If you like to use a different location, please specify the complete path to the file.

Example:

```
//get current date in seconds
var end = Math.round(+new Date()/1000);
//compute start as now - 1 minute (e.g. 60 seconds)
var start = end-60;
myscada.histoDlgTagsCSV(myscada.getFilename('example.csv'),
'default','MMMM Do YYYY, h:mm:ss a',start,end,"1,2",3,
function(status) {
        if (!status)
       {/* read error processing */}
       else
       { /* successful read processing */ }
});
```

Output file “example.csv” :

timestamp;v1;v2

January 26th 2014, 8:37:15 pm;31.9244;33.0909

January 26th 2014, 8:37:10 pm;34.4499;31.0198

January 26th 2014, 8:37:00 pm;30.9228;31.3836

myscada.histoDlgTagsAppendCSV(filename,dlg_name,format,start,end,tags,limit);

This function is identical to **histoDlgTagsCSV**, but instead of creating new file, it appends data to an already existing file. If the file does not exist it will be created, if the file exists it will append the data to the file end.

myscada.histoAlarms(start,end,limit,callback);

This function returns history of alarms in a defined time interval. The parameter **start** is the starting date, specified as a number of seconds since 1970/01/01, the parameter **end** is the ending date again specified as a number of seconds since 1970/01/01. The **limit** parameter is the maximal number of records returned. You should also specify the **callback** function to execute when the file is saved.

Example:

```
//get current date in seconds
var end = Math.round(+new Date()/1000);
//compute start as now - 1 minute (e.g. 60 seconds)
var start = end-60;
myscada.histoAlarms(start,end,10,function(status,data) {
	if (!status)
	{/* read error processing */}
	else
	{ /* successful read processing */ }
});
```

History of alarms is saved in the data object structure you will receive in the *Callback* function. The data object is an array where each object has the following structure:

data[].

atm activation time [Date]

dtm deactivation time [Date]

acktm acknowledge time [Date]

msg text description of alarm [String]

area area (geografical or logical) [String]

dev device [String]

stat alarm status [Integer]

(0 – non-active,1 – active, 2 – acknowledged ,8 – suppressed)

sv severity

actval activation value

ackval value when alarm become active

deactval value during deactivation

user user which has confirmed alarm

myscada.histoAlarmsCSV('filename',format,start,end,limit,callback);

This function returns history of alarms in a defined time interval and save them to a newly created file, specified in the first parameter, **filename**. If the file already exists it will be overwritten (the saved data format is CSV). The **format** parameter sets the date format (see *Formatting Dates* section in this manual). The **start** parameter is the starting date, specified as a number of seconds since 1970/01/01. The **end** parameter is the ending date, also specified as a number of seconds since 1970/01/01. The **limit** parameter is the maximal number of records returned. You can also specify the **callback** function to execute when the file is saved.

Note: If you specify only the file name it will be automatically saved into the user folder, which you can access by FTP or WWW (if enabled in the section Server-side Scripts → Accesses). If you like to use a different location, specify the complete path to the file.

Example:

```
//get current date in seconds
var end = Math.round(+new Date()/1000);
//compute start as now - 1 minute (e.g. 60 seconds)
var start = end-60;
myscada.histoAlarmsCSV(myscada.getFilename('alarms.csv'),'MMMM Do
YYYY, h:mm:ss a',start,end,2,function(status) {
      if (!status)
      {/* read error processing */}
      else { /* successful read processing */ }
});
```

Output file “alarms.csv” :

atm;dtm;acktm;id;msg;area;dev;stat;sv;atm;dtm;acktm;av;dv;ackv;f;user

January 26th 2014, 8:36:15 pm;January 26th 2014, 8:37:15 pm; ;1;var1;area0; ;0;0;5;1;0;#.#;system

January 25th 2014, 1:00:15 pm;January 25th 2014, 1:30:00 pm; ;2;var2;area0; ;0;0;2;1;0;#.#;system

myscada.histoAlarmsAppendCSV('filename',format,start,end,limit,callback);

This function is identical to *histoAlarmsCSV()*, but instead of creating a new file, it appends data to an already existing file. If the file does not exist it will be created, if the file exists it will append the data to the file end.

REPORTING FUNCTIONS

myscada.readActiveAlarmsForReports(severity,format,callback);

This function reads all currently active alarms and a format for a direct use in the MS Word Template Report. The first parameter filters out all active alarms with a severity lower than a value of a supplied parameter. The second parameter is used for adjusting the date format (see the *Formatting Dates* section in this manual). The **callback** function will be supplied with the read status (1 – read successful, 0 - read error) and the list of active alarms as a data object.

The active alarms are saved in the data object structure you will receive in the *Callback* function. The data object is an array with the following object structure:

data[].

atm activation time [Date formatted as string]

dtm deactivation time [Date formatted as string]

acktm acknowledge time [Date formatted as string]

msg text description of alarm [String]

area area (geographical or logical) [String]

dev device [String]

stat alarm status [Integer]

(0 – non-active,1 – active, 2 – acknowledged ,8 – suppressed)

sv severity

val current value

acktval value when alarm become active

deactval value during deactivation

*Note: Save your DOCX document into **UserFiles**, inside the **Server-side Scripts** folder (use Import from the context menu)*

Example:

```
myscada.readActiveAlarmsForReports(0,function(status,data) {
       if (!status)
       {/* read error processing */}
       else
       { /* successful read processing */
       //create replacement variable
       var repl=new Object();
       repl.alarm=data;
       //create DOCX object
       var mydoc = require('docxgen');
       //load MS Word Report Template
       mydoc.LoadTemplate('./UserFiles/template.docx');
       //process alarms - save them into a table inside template
       mydoc.LoadReplacement(repl);
       //save report into file
       mydoc.GenOutput(getFilename('DEMO_REPORT.docx'));
      }
```

```
});
```

MS Word template.docx should contain a table defined like this:

Time	Message	Value
{#alarm} {atm}	{msg}	{value}{/alarm}

myscada.histoDlgForReport('dlg_name',start,end,limit,format,callback);

This function returns historical data from a given data-log and a format for a direct use in the MS Word Template Report. In the first parameter **dlg_name** you specify the data-log name, the parameter **start** is the starting date, specified as a number of seconds since 1970/01/01, the parameter **end** is the ending date, again specified as a number of seconds since 1970/01/01. The **limit** parameter is the maximal number of records returned. The **format** parameter sets the date format (see *Formatting Dates* section in this manual). You should also specify the **callback** function to execute when the file is saved. The returned values are formatted accordingly to the set format, specified in the data-log definition.

The returned object data is an array where each element has the following structure:

data[].

tm time [Date formatted as string]

1 first column [Double]

2 second column [Double]

… … …

n last column [Double]

Note: Save your DOCX document into UserFiles inside Server-side Scripts (use import context menu)

Example:

```
//get current date in seconds
var end = Math.round(+new Date()/1000);
//compute start as now - 1 minute (e.g. 60 seconds)
var start = end-60;
myscada.histoDlgForReport('default',start,end,10,'MMMM Do YYYY,
h:mm:ss a',function(status,data) {
       if (!status)
       {/* read error processing */}
       else { /* successful read processing */
       //create replacement variable
       var repl=new Object();
       repl.dlg=data;
       //create DOCX object
       var mydoc = require('docxgen');
```

```
        //load MS Word Report Template
        mydoc.LoadTemplate('./UserFiles/template.docx');
        //process datalogs - save them into a table inside template
        mydoc.LoadReplacement(repl);
        //save report into file
        mydoc.GenOutput(getFilename('DEMO_REPORT.docx'));
        }
});
```

MS Word template.docx should contain a table defined like this:

Time	First	Second
{#dlg} {tm}	{1}	{2}{/dlg}

myscada.histoDlgForReportWithTags('dlg_name',start,end,['1','2','3',...],limit,format,callback);

This function is identical to *histoDlgForReport()*, but adds a new array parameter ['1','2','3',...] for specifying the tags to be read. In the data-log definition each row (tag) has its unique identification number - use this number to select a given tag.

myscada.histoDlgForReportNoRounding('dlg_name',start,end,limit,format,callback);

This function is identical to *histoDlgForReport()*, but does not format data (speficied by format in the data-log definition).

myscada.histoDlgForReportRounded('dlg_name',start,end,limit,format,decimalplaces,callback);

This function is identical to *histoDlgForReport()*, but adds the parameter **decimalplaces**, which specifies a number of decimal places of returned data values.

myscada.histoAlarmsForReport(start,end,limit,format,callback);

This function returns history of alarms in a defined time interval and format for a direct use in MS Word Template Report. The first parameter **start** is the starting date, specified as a number of seconds since 1970/01/01, the second parameter **end** is the ending date, again specified as seconds since 1970/01/01. The **limit** parameter is the maximal number of records returned. The fourth parameter is used for adjusting the date format (see the *Formatting Dates* section in this manual). You should also specify the **callback** function to execute when the data file is ready.

History of alarms is saved in the data object structure you will receive in the *Callback* function. The data object is an array where each object has the following structure:

data[].

atm activation time [Date]

dtm deactivation time [Date]

acktm acknowledge time [Date]

msg text description of alarm [String]

area area (geografical or logical) [String]

dev device [String]

stat alarm status [Integer]

(0 – non-active,1 – active, 2 – acknowledged ,8 – suppressed)

sv severity

actval activation value

ackval value when alarm become active

deactval value during deactivation

user user which has confirmed alarm

Note: Save your DOCX document into UserFiles inside Server-side Scripts (use import context menu)

Example:

```
//get current date in seconds
var end = Math.round(+new Date()/1000);
//compute start as now - 1 minute (e.g. 60 seconds)
var start = end-60;
myscada.histoAlarmsForReport(start,end,10, 'MMMM Do YYYY, h:mm:ss
a',function(status,data) {
        if (!status)
        {/* read error processing */}
        else
        { /* successful read processing */
        //create replacement variable
        var repl=new Object();
        repl.histalarm=data;
        //create DOCX object
        var mydoc = require('docxgen');
        //load MS Word Report Template
        mydoc.LoadTemplate('./UserFiles/template.docx');
        //process history of alarms - save them into a table inside
        template
        mydoc.LoadReplacement(repl);
        //save report into file
        mydoc.GenOutput(getFilename('DEMO_REPORT.docx'));
        }
});
```

The MS Word template.docx should contain a table, defined as follows:

Time	Message	Value
{#histalarm} {atm}	{msg}	{value}{/histalarm}

myscada.histoDlgCreatePicture('dlg_name',start,end,filename,limit,w,h,format,callback)

This function creates a time chart (graph) from the historical data saved in a data-log, named 'dlg_name' and saves it as a picture with a type specified by the **format** parameter. The second parameter **start** is the starting date, specified as a number of seconds since 1970/01/01, the third parameter **end** is the ending date, again specified as seconds since 1970/01/01. The **filename** parameter is the file path (including the file name) for saving the picture to. The **limit** parameter is the maximal number of records to use for the picture rendering. The **w** and **h** parameters are the width and height of the picture. The **format** is a file type generated, you can use the following formats: *jpeg,svg,pdf, png* and *emf*. If you want to use the generated picture in the report, use the format *emf*.

Example:

```
//get current date in seconds
var end = Math.round(+new Date()/1000);
//compute start as now - 1 minute (e.g. 60 seconds)
var start = end-60;
myscada.histoDlgCreatePicture('default'start,end,myscada.getFilenam
e('picture.emf'),10,640,480,'emf',function(status,data) {
      if (!status)
      {/* read error processing */}
      else
      { /* successful read processing */
      }
});
```

myscada.histoDlgCreatePictureForPens('dlg_name',start,end,filename,keys,limit,w,h,format,callback)

This function is identical to *histoDlgCreatePicture()*, but adds the parameter **keys**, where you can specify what pens to include in the chart. The **keys** parameter is defined as an array, where the values specify the tags to be read. In data-log definition each row (tag) has its unique identification number, use this number to select a given tag.

Example:

```
//get current date in seconds
var end = Math.round(+new Date()/1000);
//compute start as now - 1 minute (e.g. 60 seconds)
var start = end-60;
myscada.histoDlgCreatePictureForPens('default'start,end,myscada.get
Filename('picture.emf'),['1','2'],10,640,480,'emf',function(status,
data) {
      if (!status)
```

```
        {/* read error processing */}
        else
        { /* successful read processing */
        }
});
```

myscada.histoDlgCreatePictureForPensWithNames(datalogname,start,end,filename,names,keys,limit,w,h,format,callback)

This function is identical to *histoDlgCreatePictureForPens()*, but adds an array parameter **names**, where you can specify names of the pens shown in the legend.

Example:

```
//get current date in seconds
var end = Math.round(+new Date()/1000);
//compute start as now - 1 minute (e.g. 60 seconds)
var start = end-60;
myscada.histoDlgCreatePictureForPensWithNames('default'start,end,myscada.getFilename('picture.emf'),['name1','name2'],['1','2'],10,640,480,'emf',function(status,data) {
        if (!status)
        {/* read error processing */}
        else
        { /* successful read processing */
        }
});
```

myscada.createReport(reportTemplate,callback)

This function creates an object for report generating. Specify the file path to the report template.

Example:

```
myscada.createReport(myscada.getFilePathUserFiles('reportTemplate.docx'),function(status,doc) {
        if (!status)
        {/* read error processing */}
        else
        { /* successful read processing */
        }
});
```

myscada.createReportSync(reportTemplate)

This function creates an object for report generating. Specify the file path to the report template. This function does not use a callback!

Example:

```
var doc =
myscada.createReport(myscada.getFilePathUserFiles('reportTemplate.docx'));
```

myscada.generateReport(doc,reportname)

This function creates reports (use the function *createReport* to get a report first and then fill it with data). Input parameters: report object created by function *createReport* and the file where to save the generated report.

Example:

```
var doc =
myscada.createReport(myscada.getFilePathUserFiles('reportTemplate.do
cx'));
doc.setData({
             "aktdate":aktdate,
             "timeFrom":timeFrom,
             "timeTo":timeTo
             });
myscada.generateReport(doc,myscada.getFilename("Report.docx"));
```

FORMATTING DATES

Every function converting data from *Date* to *String* requires you to provide the format. Generally, all functions saving data to a CSV file, and all the reporting functions require you to provide the format parameter. The format takes a string of tokens and replaces them with their corresponding values.

Example:

'MMMM Do YYYY, h:mm:ss a'	January 26th 2014, 11:35:33 pm
'dddd'	Monday
'MMM Do YY'	Jan 26th 14

For the format parameter provide a string with specified format, the following tokens are valid:

	Token	Output
Month	M	1 2 ... 11 12
	Mo	1st 2nd ... 11th 12th
	MM	01 02 ... 11 12
	MMM	Jan Feb ... Nov Dec
	MMMM	January February ... November December
Quarter	Q	1 2 3 4
Day of Month	D	1 2 ... 30 31
	Do	1st 2nd ... 30th 31st

	DD	01 02 ... 30 31
Day of Year	DDD	1 2 ... 364 365
	DDDo	1st 2nd ... 364th 365th
	DDDD	001 002 ... 364 365
Day of Week	d	0 1 ... 5 6
	do	0th 1st ... 5th 6th
	dd	Su Mo ... Fr Sa
	ddd	Sun Mon ... Fri Sat
	dddd	Sunday Monday ... Friday Saturday
Day of Week (Locale)	e	0 1 ... 5 6
Day of Week (ISO)	E	1 2 ... 6 7
Week of Year	w	1 2 ... 52 53
	wo	1st 2nd ... 52nd 53rd
	ww	01 02 ... 52 53
Week of Year (ISO)	W	1 2 ... 52 53
	Wo	1st 2nd ... 52nd 53rd
	WW	01 02 ... 52 53
Year	YY	70 71 ... 29 30
	YYYY	1970 1971 ... 2029 2030
Week Year	gg	70 71 ... 29 30
	gggg	1970 1971 ... 2029 2030
Week Year (ISO)	GG	70 71 ... 29 30
	GGGG	1970 1971 ... 2029 2030
AM/PM	A	AM PM
	a	am pm
Hour	H	0 1 ... 22 23
	HH	00 01 ... 22 23
	h	1 2 ... 11 12

	hh	01 02 ... 11 12
Minute	m	0 1 ... 58 59
	mm	00 01 ... 58 59
Second	s	0 1 ... 58 59
	ss	00 01 ... 58 59
Fractional Second	S	0 1 ... 8 9
	SS	0 1 ... 98 99
	SSS	0 1 ... 998 999
Timezone	z or zz	EST CST ... MST PST
	Z	-07:00 -06:00 ... +06:00 +07:00
	ZZ	-0700 -0600 ... +0600 +0700
Unix Timestamp	X	1360013296

Given that a preferred formatting differs, there are a few tokens that can be used for the time formatting, based on its language. There are uppercase and lowercase variations for the same formats. The lowercase version is intended to be a shortened version of its uppercase counterpart.

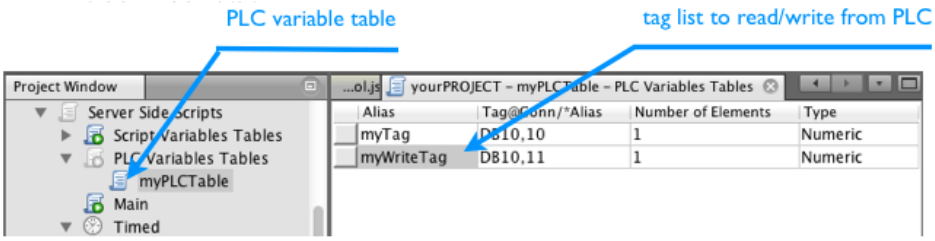
Time	LT	8:30 PM
Month numeral, day of month, year	L	09/04/1986
	l	9/4/1986
Month name, day of month, year	LL	September 4 1986
	ll	Sep 4 1986
Month name, day of month, year, time	LLL	September 4 1986 8:30 PM
	lll	Sep 4 1986 8:30 PM
Month name, day of month, day of week, year, time	LLLL	Thursday, September 4 1986 8:30 PM
	llll	Thu, Sep 4 1986 8:30 PM

Example:

In this example, the task is to read and write data to the PLC with a 1-minute interval:

1) Create a PLC variable table and give it a name.

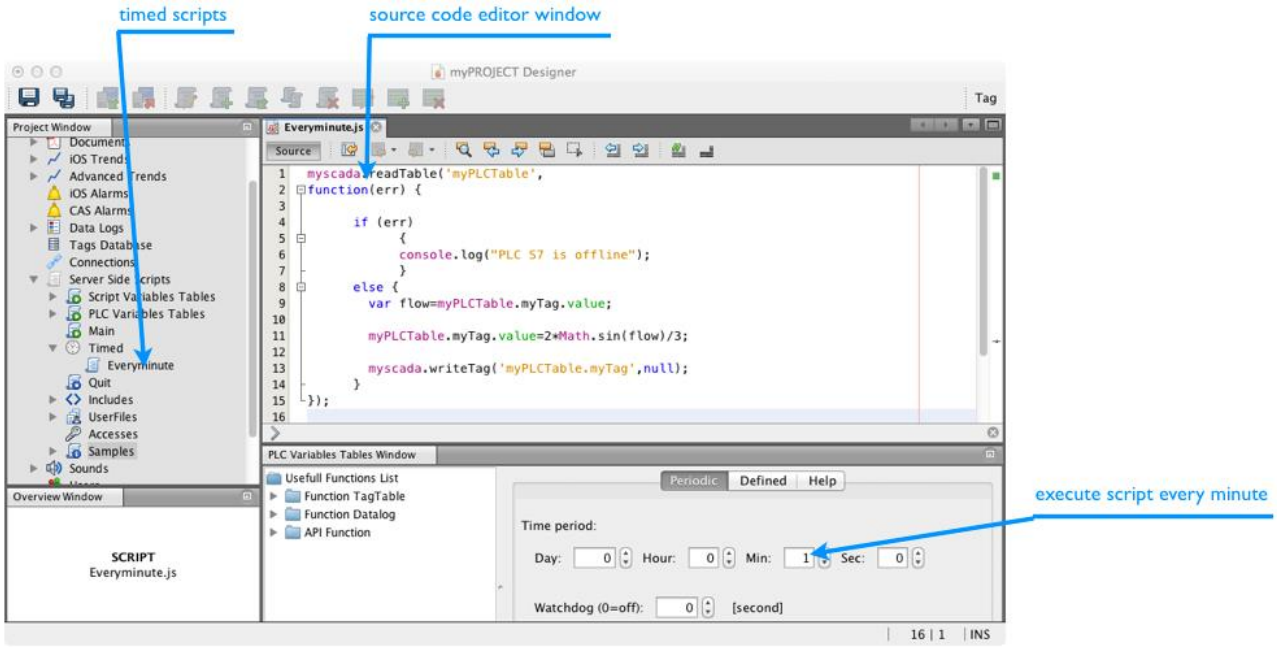
2) Fill in the tags you want to read from the PLC, alias is the variable name under which you will access your defined tag in the scripts.

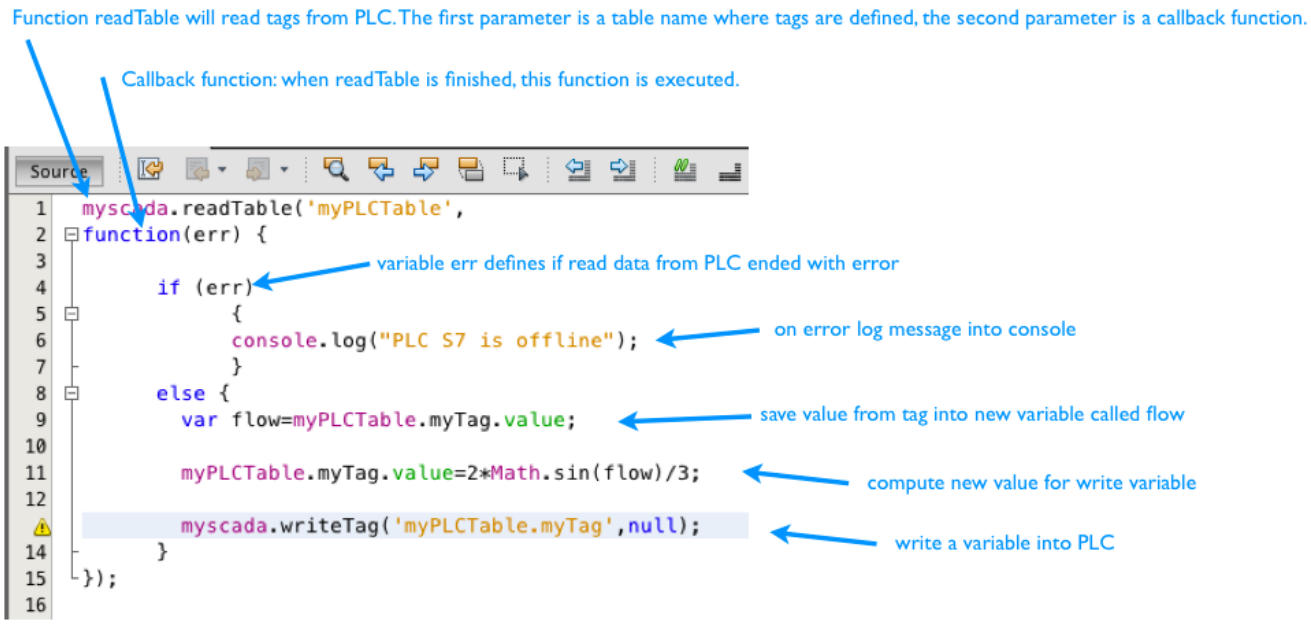


3) Create a timed script and set the refresh period to 1 minute.

4) Fill in the code into the code window.

Code description:





5) Now you are done. Download the project to your *mySCADA* device and you will see that every minute the tags defined in the *myPLCTable* will be read and written, based on your formula evaluation.

20.10 Debugging

With this feature you can manually debug your written scripts. Debugging uses Node Inspector.

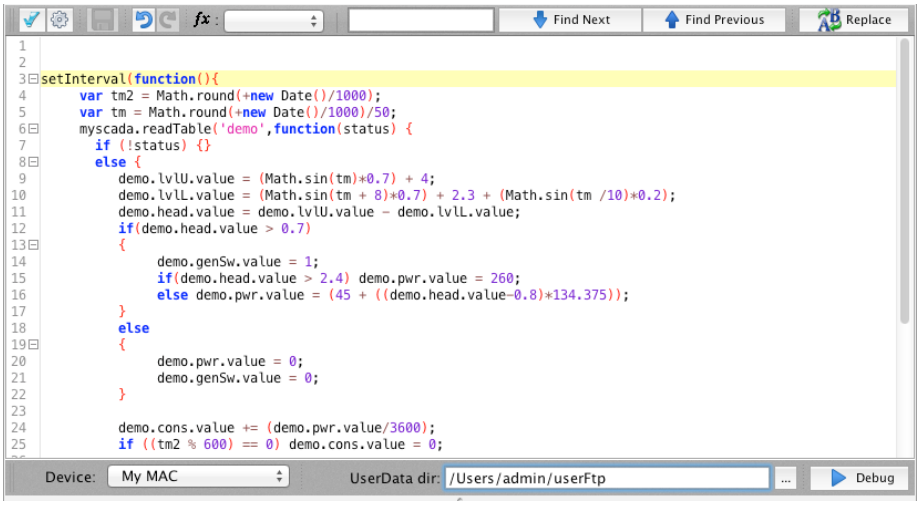
Debugger integrated in myPROJECT Designer is a powerful JavaScript debugger interface. Node Inspector supports almost all of the debugging features of DevTools, including:

- Navigate in your source files
- Set breakpoints (and specify trigger conditions)
- Step over, step in, step out, resume (continue)
- Inspect scopes, variables, object properties
- Hover your mouse over an expression in your source to display its value in a tooltip
- Edit variables and object properties
- Continue to location
- Break on exceptions
- Disable/enable all breakpoints

Before you start debugging:

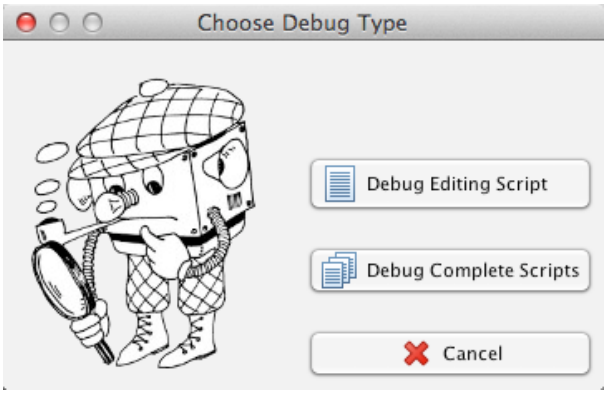
- Make sure you have same project downloaded to your device as you have in myPROJECT Designer
- Select correct device in bottom of the script editor

1) To start debugging click on the **Debug** icon in right bottom corner of the *Script window*.



TIP: if your script generates some files, you will find them in the path specified in ***UserData dir***

2) In the following window select whether you want edit the current script or all the project scripts.



Note:

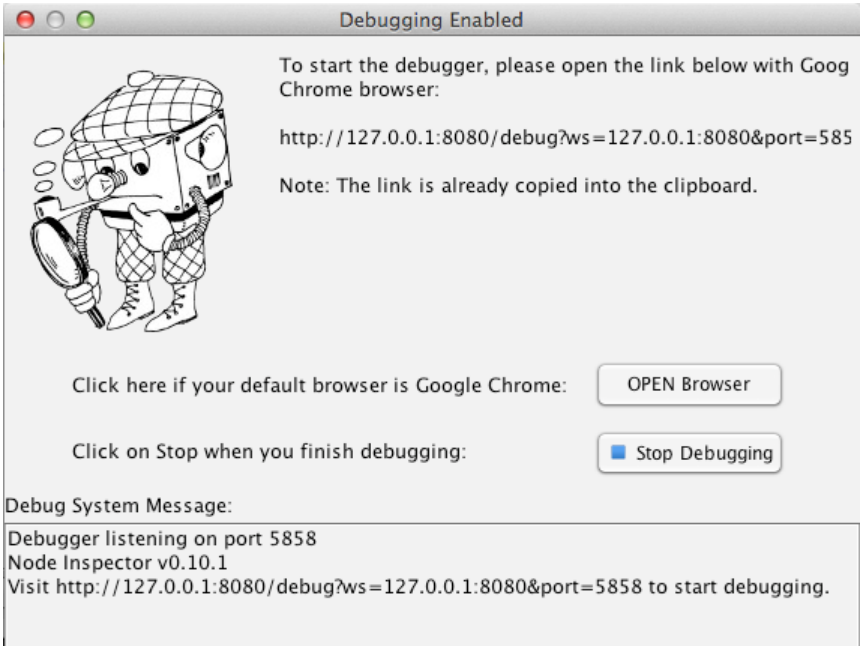
When downloading a project into the device all the scripts merge into one.

Debug Editing Script: this option will run only the current script you are editing other scripts will not be executed.

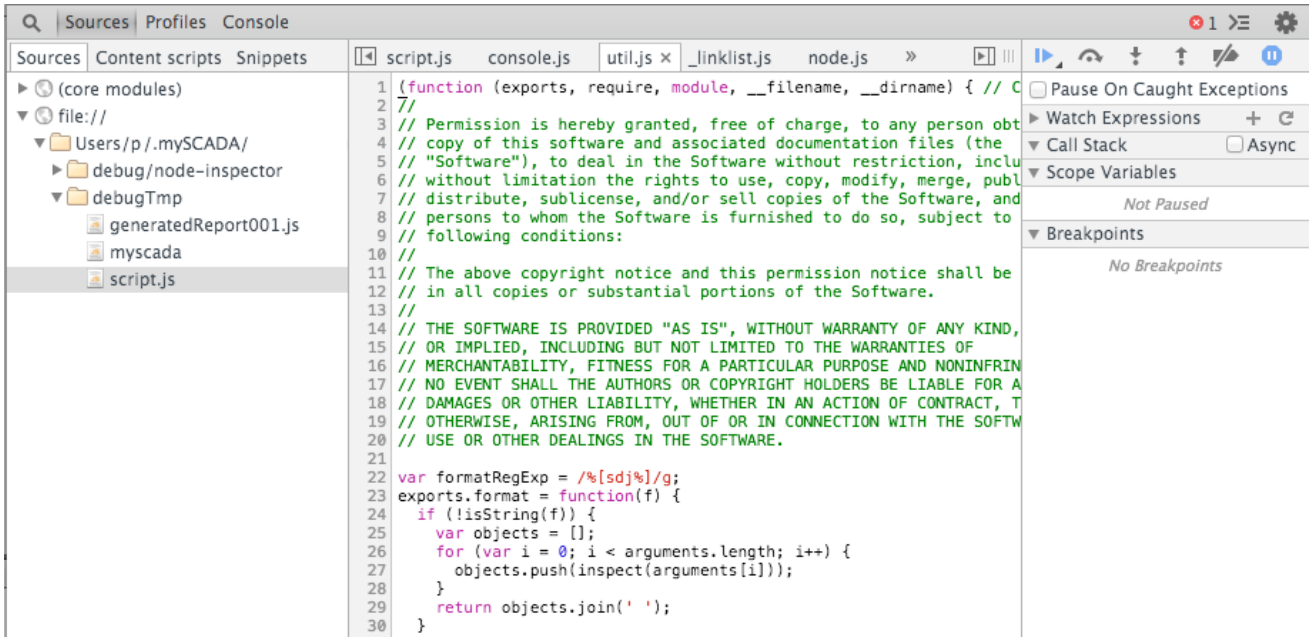
Debug Complete Scripts: this option will run complete solution, eg. It will run all your scripts exactly same way as they will run on your device when you download a project to it.

TIP: If you have some dependency among your scripts (eg. Variable declarations in main script etc.) always use Debug Complete Scripts.

3) After a short initialization start the debugger by clicking on the button **OPEN Browser**. Alternatively, you can open Chrome browser and paste the link copied to the clipboard.



The debugging console will open in the *Chrome* browser *(you need to have it installed and set as default browser to be able to run the debugger).*



In the debugging console you can:

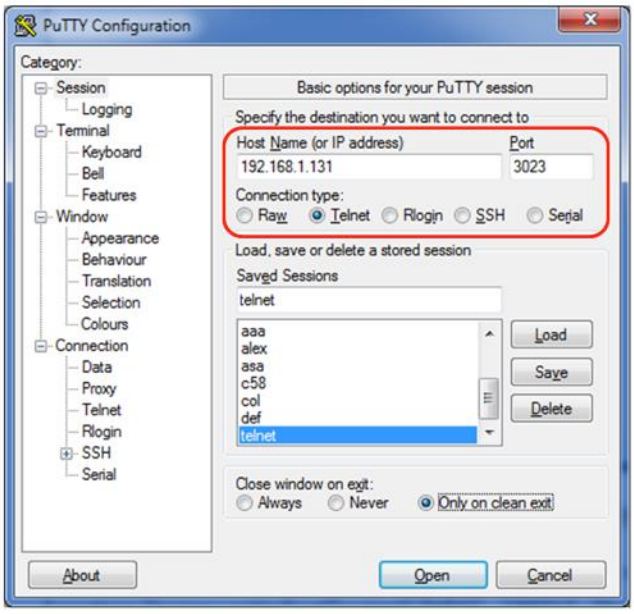
- browse through the script row by row or set the break points to which it will be checked automatically
- enter into *caller functions* to debug *called functions*
- view the local and global variables on the right side of the console window etc.

4) When you finish debugging go back to *myPROJECT Designer* and click on the button **STOP Debugging** and close the Chrome browser window.

Console log

For debugging of server-side scripts on *mySCADA Compact* devices, you can use the *Console log*, accessible via a web interface of *mySCADA Box* in the menu *SYSTEM*, and the bookmark *STATUS.*

The console is accessible through any *mySCADA* runtime with a telnet connection on the port **11015**. The windows users can use a freeware utility *Putty* (see picture below) and the UNIX users (including MAC) can use a telnet command with a specific port.



Also a useful tool for debugging of server-side scripts is to define a function for logging of uncaught exceptions:

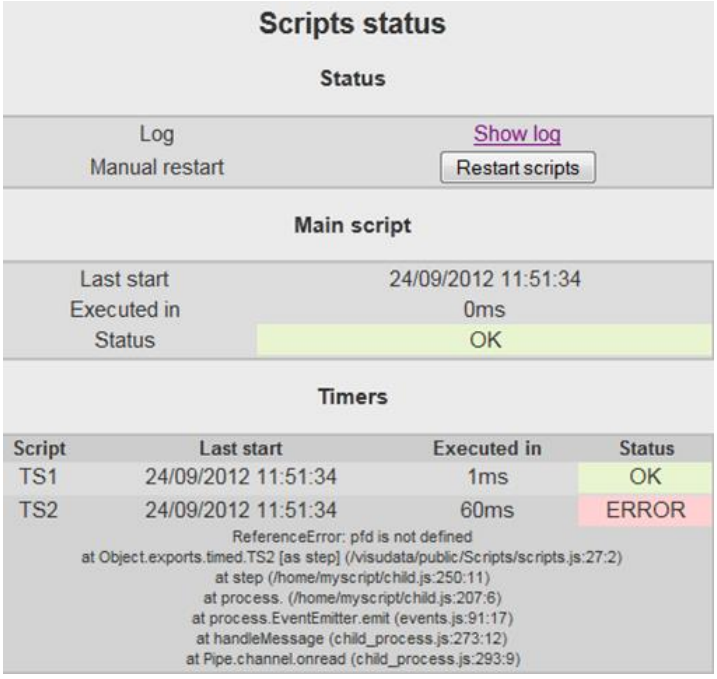
```
process.on('uncaughtException', function (err) {
console.log(err);
});
```

This function will log a description of an uncaught error into the console, you can, of course define any action inside this function.

20.11 Script Status

Scripts status and web log

In the menu *SYSTEM* and the bookmark *STATUS* you can find the scripts *status*, which can be useful for the scripts troubleshooting - an example output is shown in the picture below:



In the *Status* column you can see the web log and restart of all server-side scripts. You can use this log from your scripts with the function *myscada.logFile ("your text");* which will add a time stamp to your text. This function is embedded in *NODE.JS* so you can call it without requiring.

In the next parts there are reports on the *Main script* and on *Timed scripts*. There are three parameters in this report:

Last start – the last time when the script started

Executed in – run time of the script

Status – status of the script execution, if execution fails the error log is displayed (as shown in the previous picture for the script TS2)

Global variables

For interconnecting server-side scripts you can use global variables. To define a global variable use “globVar = 5;” instead of “var globVar = 5;” in the *Main script* so it will be accessible from all the other scripts.

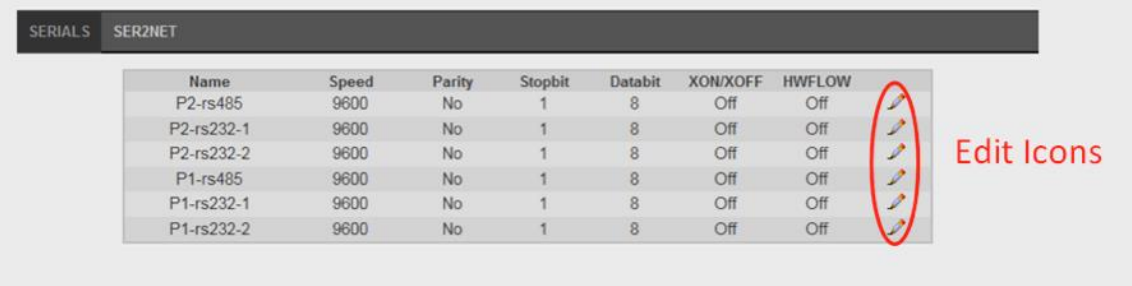
Virtual PLC

This is a powerful utility to bind outputs of user scripts with *mySCADA* HMI, alarms or live data. First step to start using a virtual PLC is to create a virtual tag – it can be created the same way as real PLC tags – the only difference is that you select the script as a source, instead.

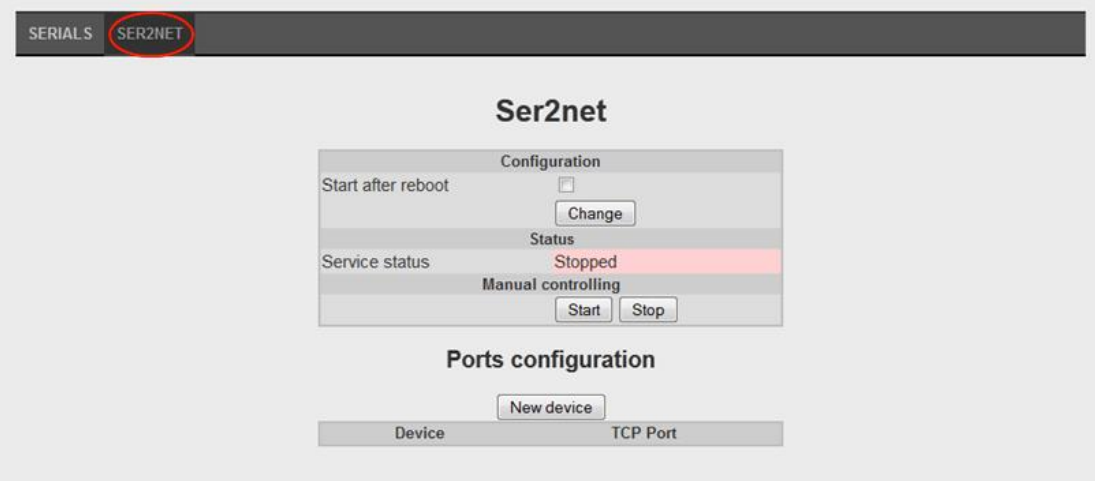
Note: When there is no real PLC added a virtual PLC (script) is the only option.

20.12 Ser2Net

Ser2Net is a very useful service, which allows using serial ports of *mySCADA Box* in *Server-side Scripts*. At first it is necessary to setup the parameters of serial ports in the *COM* bookmark of the *mySCADA Box* menu. A number of available serial ports depends on the hardware configuration of *mySCADA Box*. Clicking on the "edit" icons on the right can change the settings of each port.



After configuring the desired serial ports you can switch to the *SER2NET* bookmark and configure thr *Ser2Net* service. Thr *Ser2Net* configuration window is shown in the picture below:



By default the *Ser2Net* service is stopped and the service auto-start is disabled. To enable auto-start of *Ser2Net* you have to tick off the *Start after reboot* box and click on the **Change** button to save the settings. The **Start** and the **Stop** buttons can control the *Ser2Net* service manually. To enable access to the serial port via *Ser2Net* you have to set up a new device by clicking on **New device**. The *Add device* dialog is shown on the following picture.

Device – list of available serial ports

Timeout – time (in seconds) before the port will be disconnected if there is no activity on it. A zero value disables this function

Allow remote control – allows remote control of the serial port parameters via RFC 2217

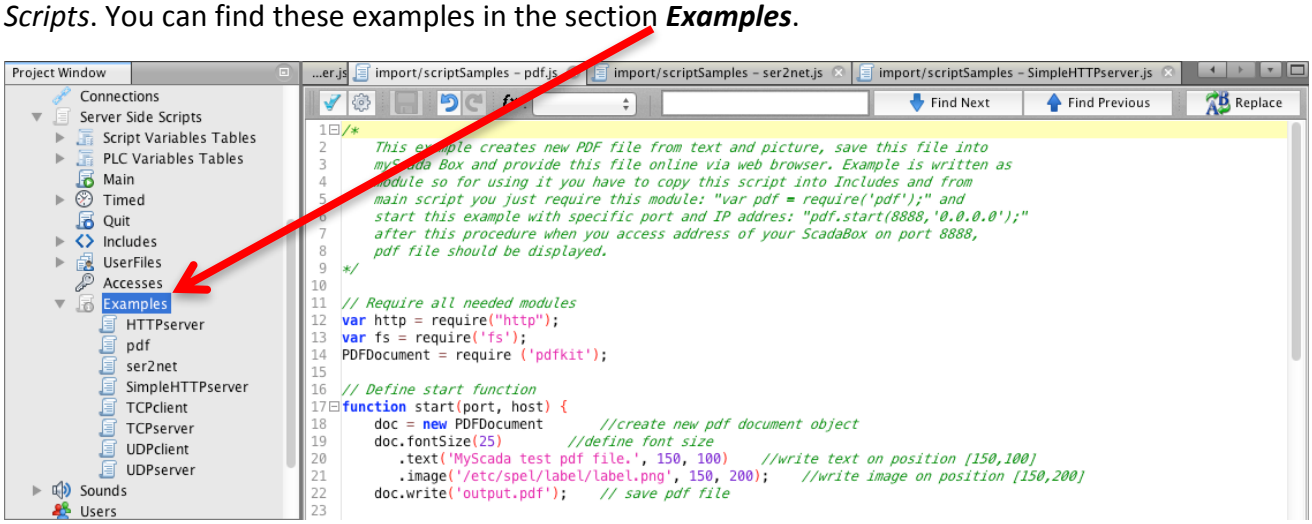
TCP Port – number of the TCP/IP port to accept connections

State – specifies operation mode of device:

- *Raw* – enables the port and transfers all data as it is between the TCP and the serial port
- *Telnet* – enables the port and runs the telnet protocol on the port to set up telnet parameters
- *Off* – disables the port from accepting connections
-

20.13 Script Samples

myPROJECT Designer contains many useful examples to help you get started with *Server-side Scripts*. You can find these examples in the section ***Examples***.

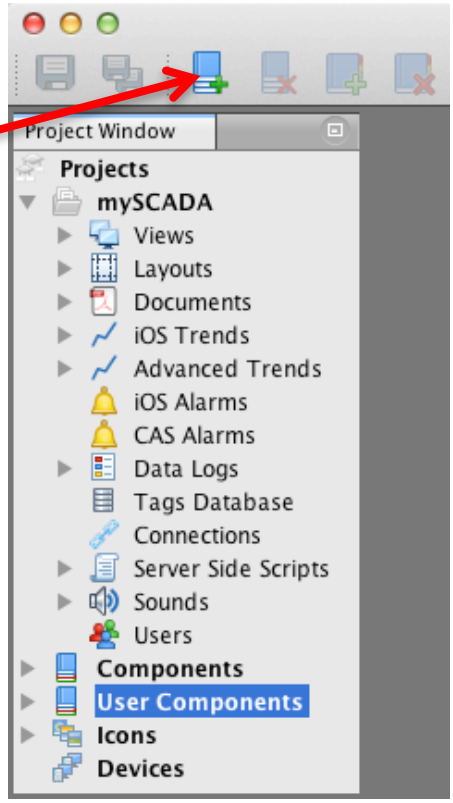


21. Creating and Testing Components

With *myPROJECT Designer* you can create rich feature components, suitable for creation of animated repeatable objects like buttons, gauges, charts etc. The main idea is that you create a group of animated objects, where the animations are tied to some variables names instead of a direct tag address. After adding such component into a view you simply enter the target tag address in one place (component properties) as a variable value and this change will apply to all tied animations without an actual need to set the tag address for each object separately.

21.1 Creating Components

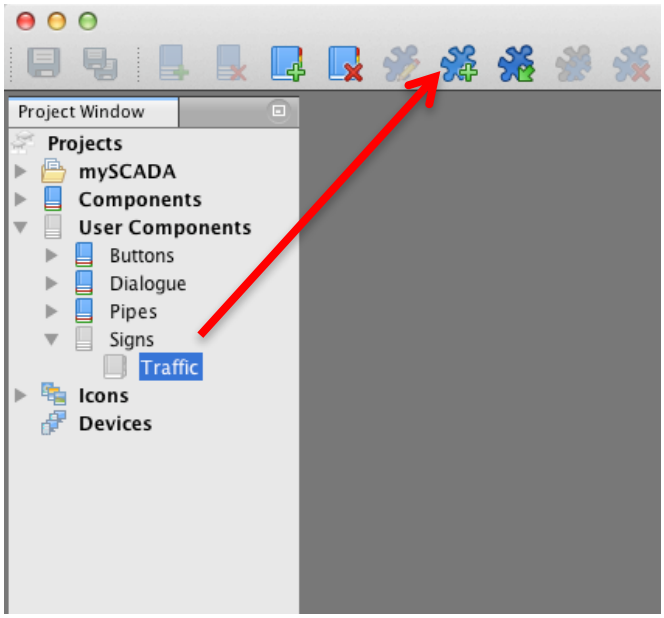
New components can be added to existing groups or created as a part of the new components group.



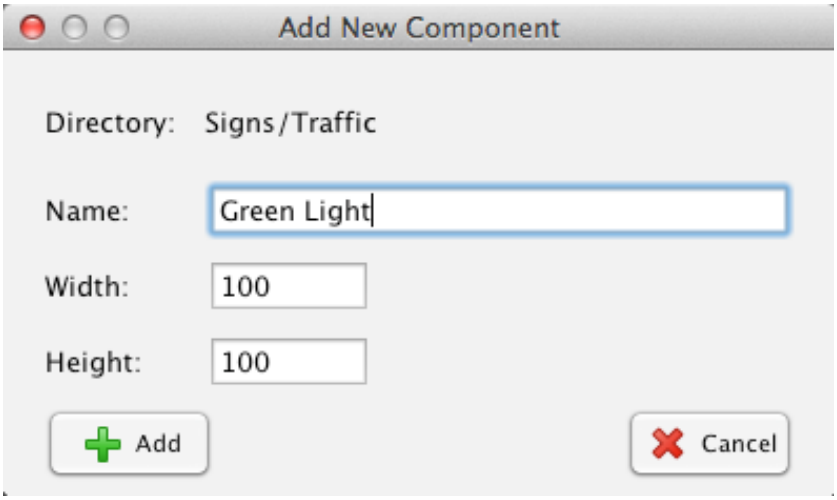
1) Select the *User Components* in the *Project* window and click on the icon **Add Component Directory** in the main toolbar.

2) A new dialog opens where you can enter the name for the new directory and click on the button **Add**. Clicking on the icon  and select the source folder to import the components from.

3) Select the group, where the new component should be placed and click on the **Add Component** icon in the main toolbar.



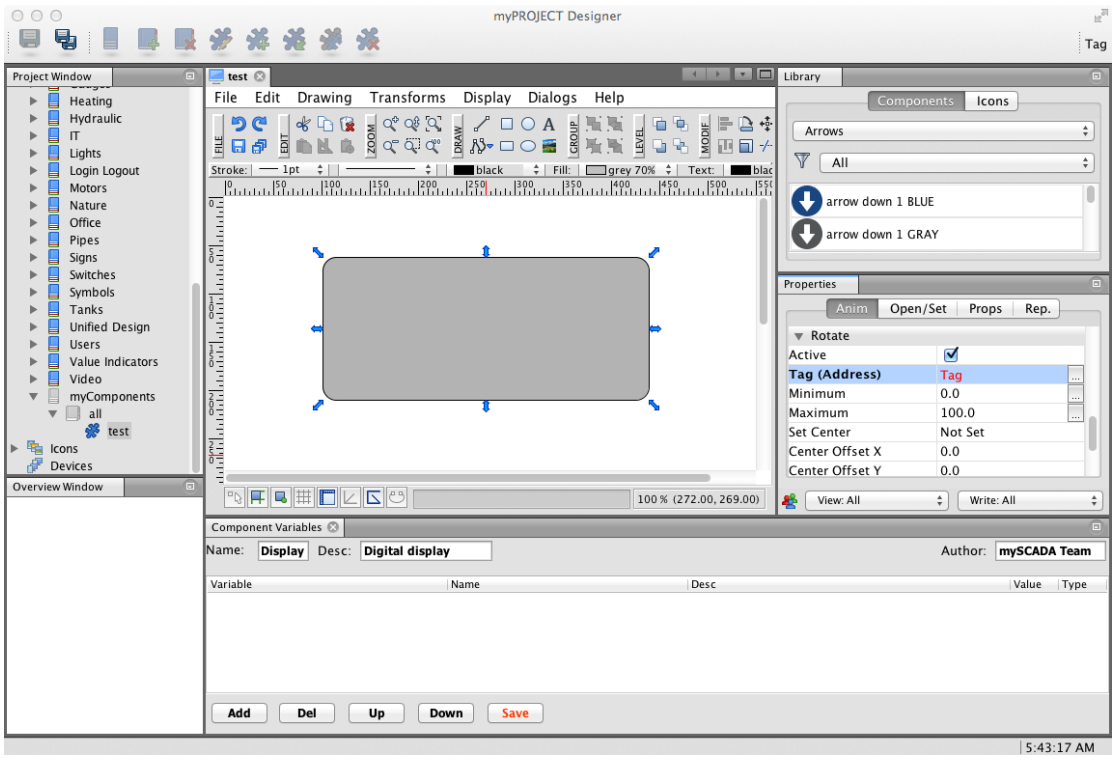
4) A new dialog will be shown, where you can fill in the component name, specify the width and height of the drawing area.



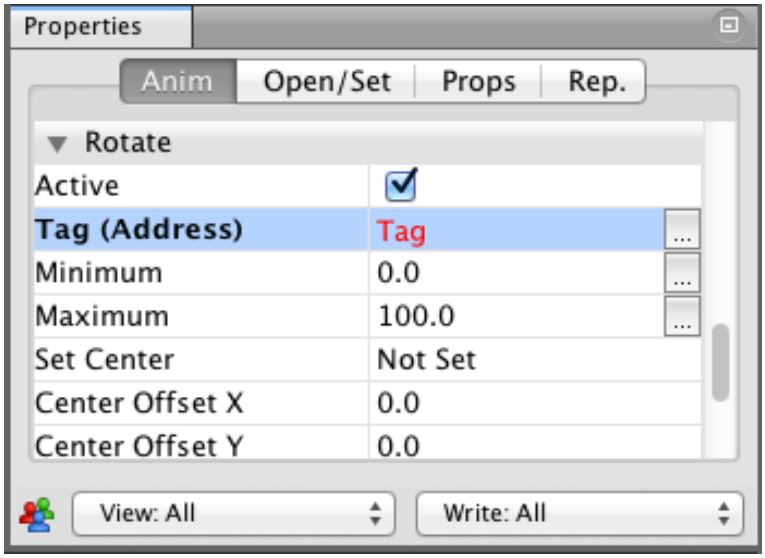
5) Click on **Add** to create the component.

Now when a new component is created double-click on its name, which will prompt the graphic editor.

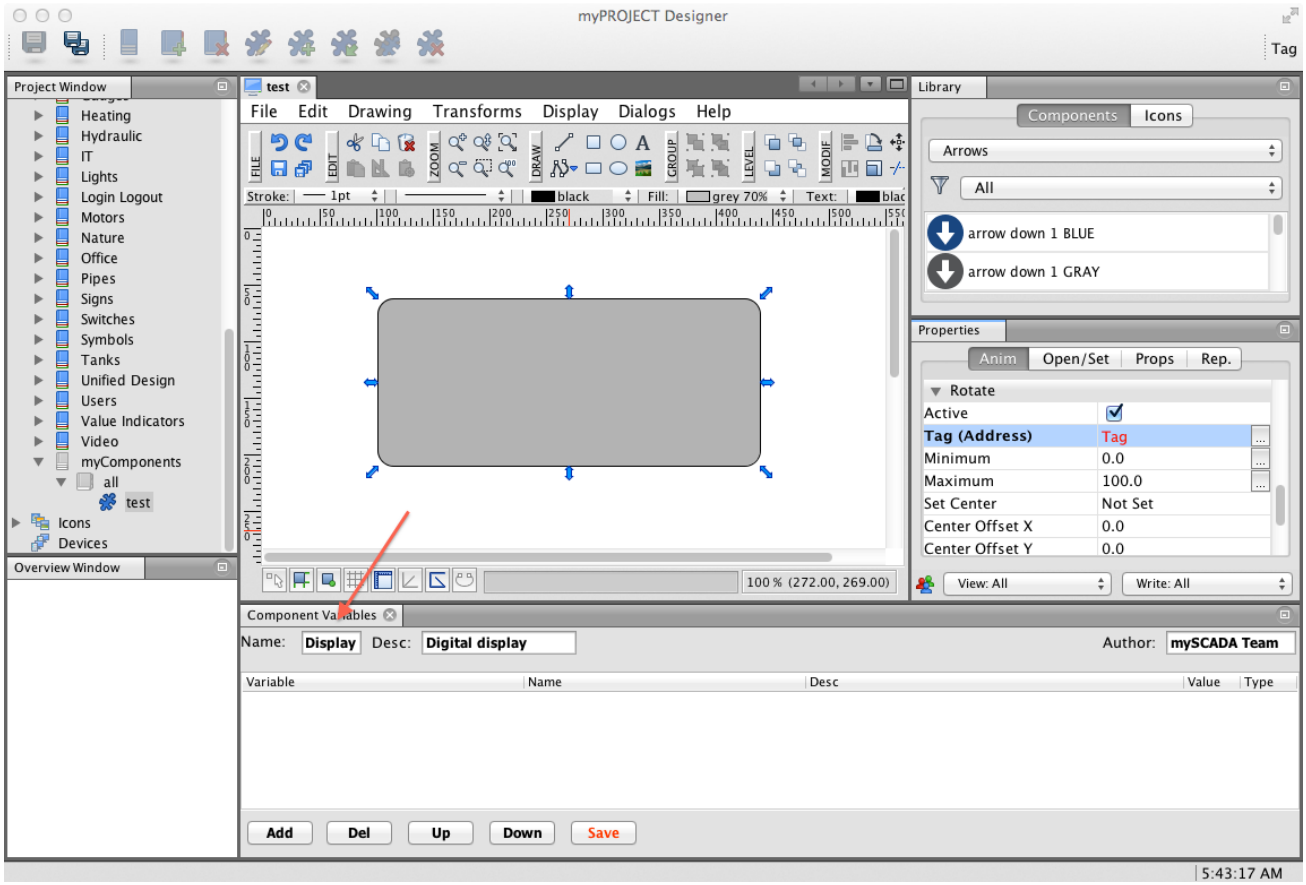
Let us start with a simple rectangle, animated by rotation:



Create a rectangle and set the *Rotate* animation on it in the *Properties window*, where you input the **Tag (Address)**, set the **Minimum** and **Maximum** expected tag values and the rotation axis if needed.



Now take a look at the *Component Variables* window.



21.2 Component Variables

In this section you define all local variables (visible only inside of the component) by specifying a name, short description and type for each variable. These variables can contain tag values read from the PLC, color and fill values for different elements, calculation results etc.

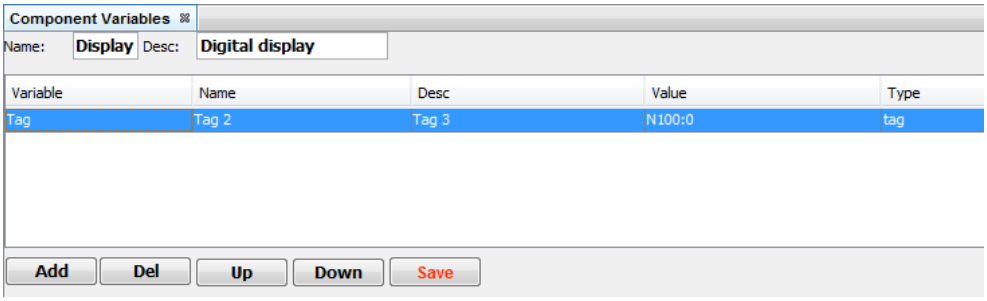
Each component can have as many local variables as you need. You have to save your progress after addition of each new variable by clicking on **Save** in the tab toolbar. The **Color** button opens the color palette when an appropriate type of variable is selected.

Each local variable has following properties:

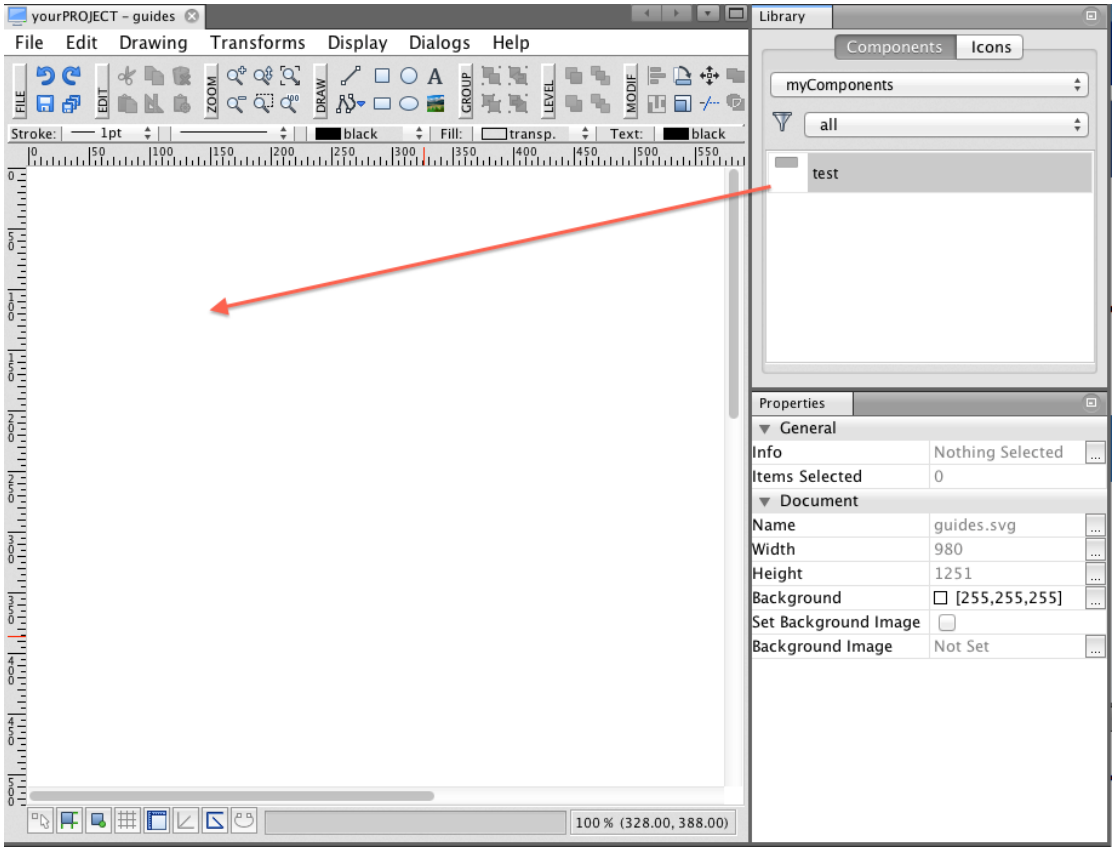
Property	Description
Variable	Name of the variable as internal value. It is used for a reference in animations and properties of your component. Create variable named “Tag” to continue in this example
Name	Name of the variable seen by the user, this will be shown in the component properties later, name the variable as “Tag2”
Desc	Description of the variable
Value	Variable default value, for this example purposes set as “N100:0”
Type	Type of variable, supported types are: *Tag, String, Int, Double (Float), Color, Boolean, Object*

You can refer to the local variable anywhere in the animations, effects or properties across the edited component.

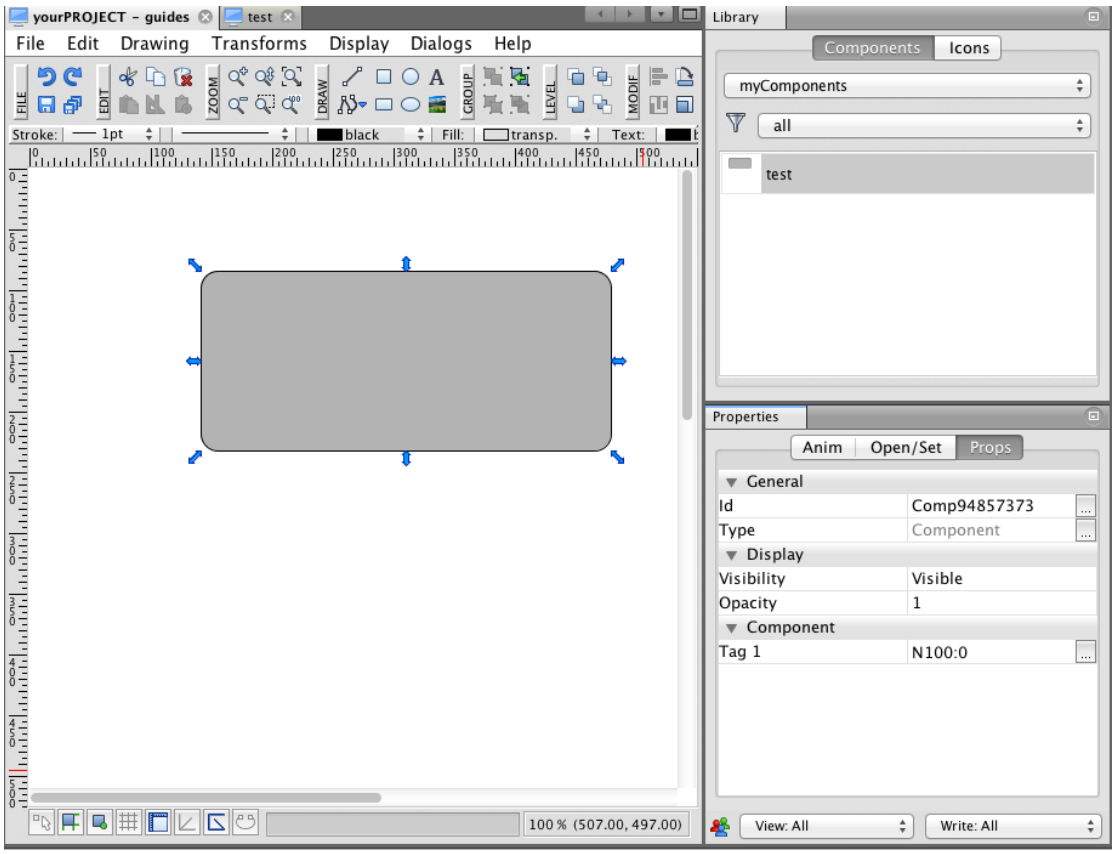
Click on **Save** to confirm variable changes and then save the whole component.

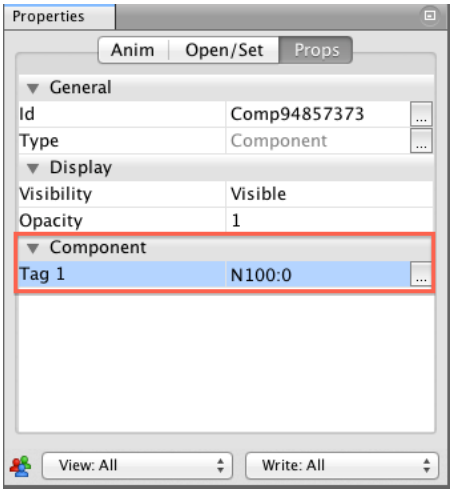


Return back to the view and test the new component.



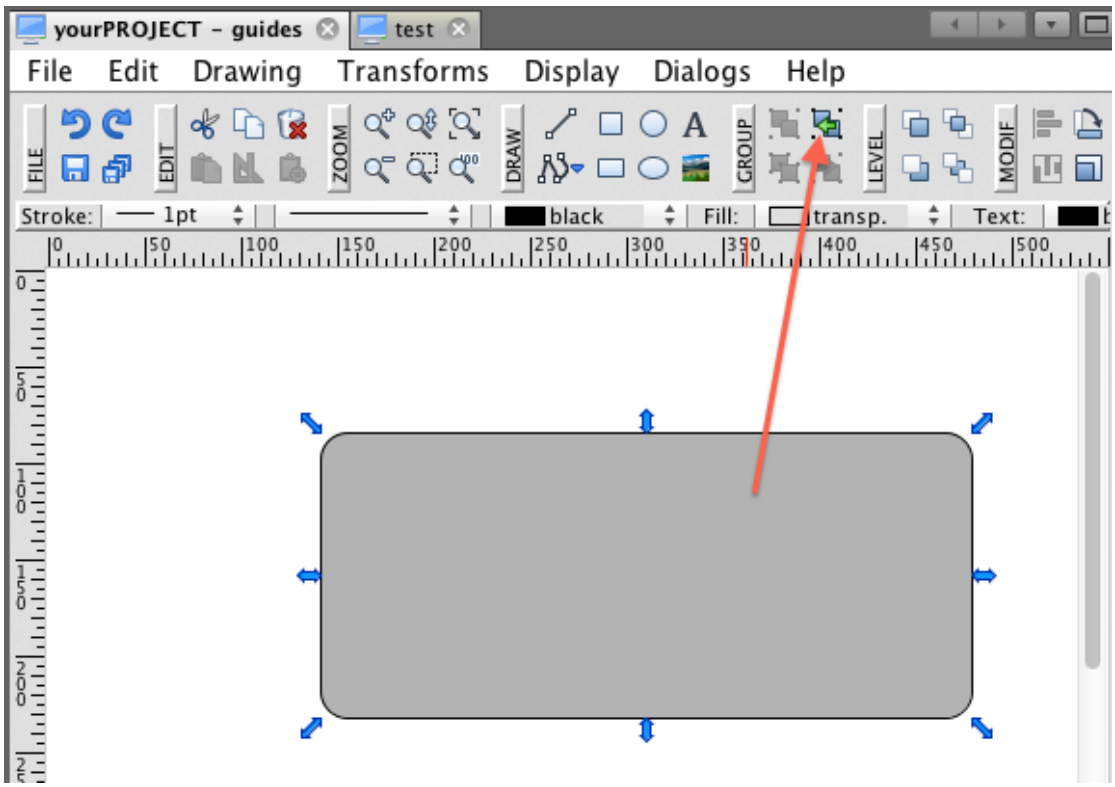
Now open the *Components* library in the *Properties window* and select your created component. Drag the component to the opened view and click on it component to see its properties.



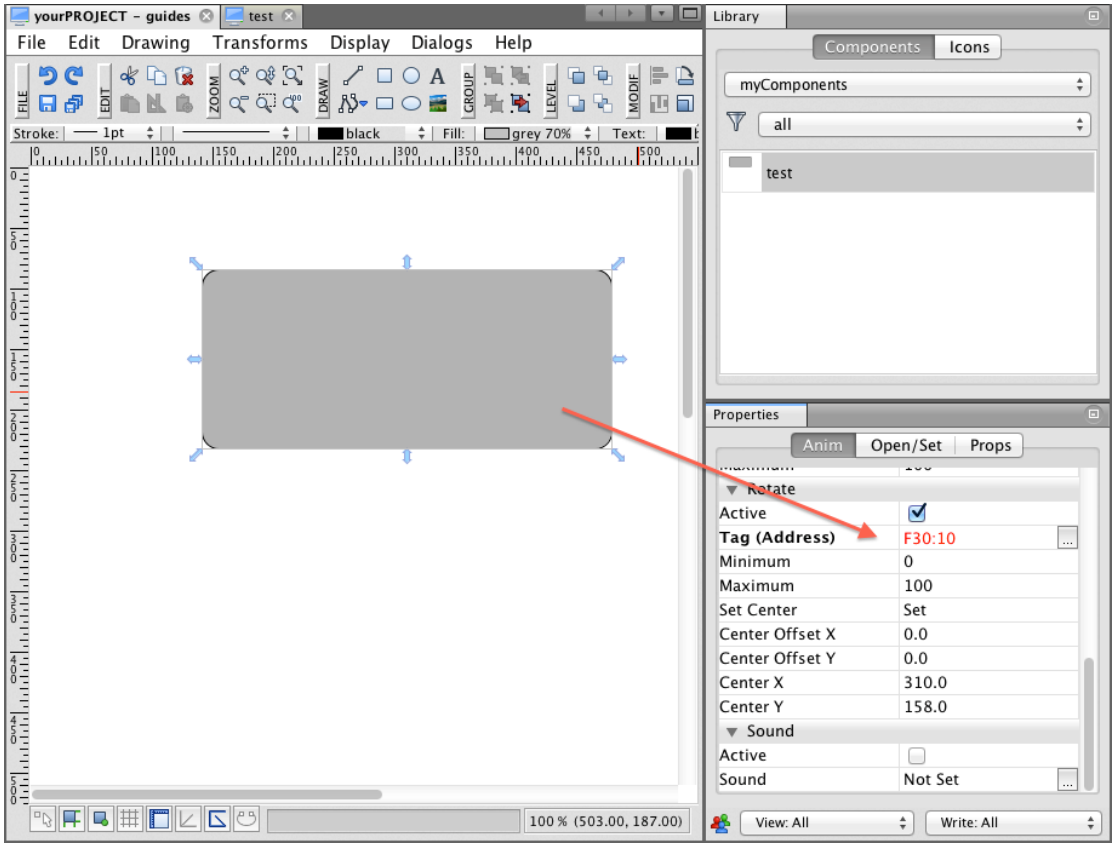


In the *Properties window* you can see the parameters of the component - the name of the component variable *"Tag2"*, which has a default value of *"N100:0"*. To override the default tag address enter any other tag you want to connect your component with, for example *"F30:10"*.

To make a quick check if the tag address in the animation has been changed right-click on the component and click on **Enter group**.



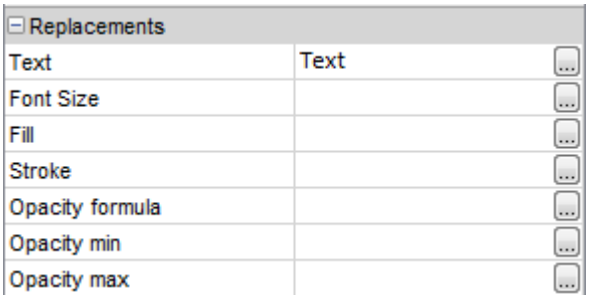
Select the rectangle again, open its properties in the *Properties window* and set the *Rotate* animation.



You should see the tag address used for this animation has changed to the requested “F30:10”.

21.3 Replacements

This feature should ease replacing of a predefined text in a component. You can change the text, font size, color, stroke or opacity of the text element.

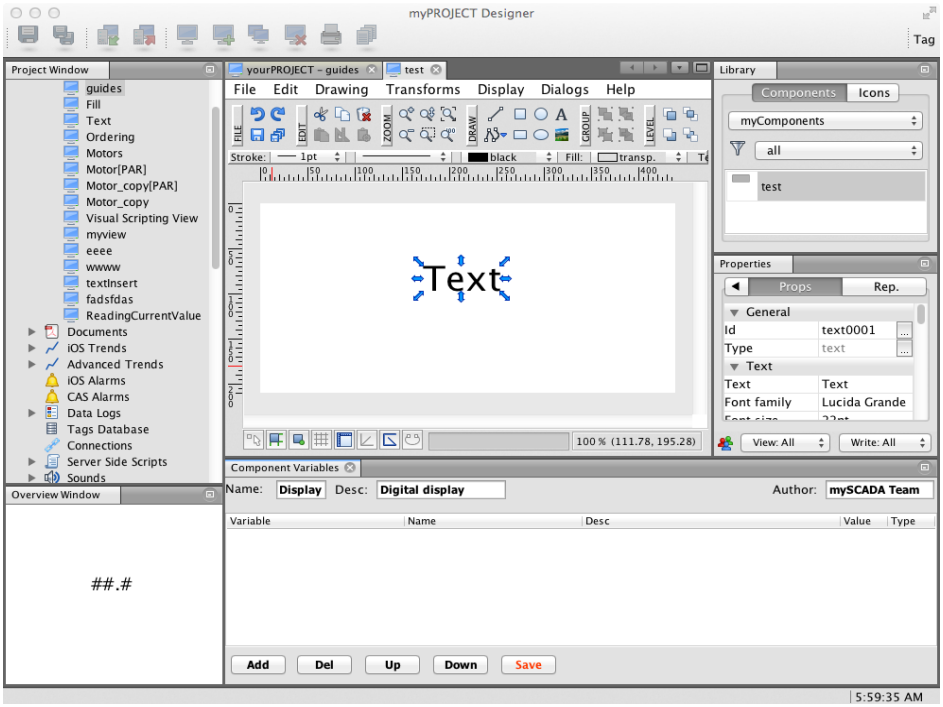


1) Start with the component variables again and create the following variables:

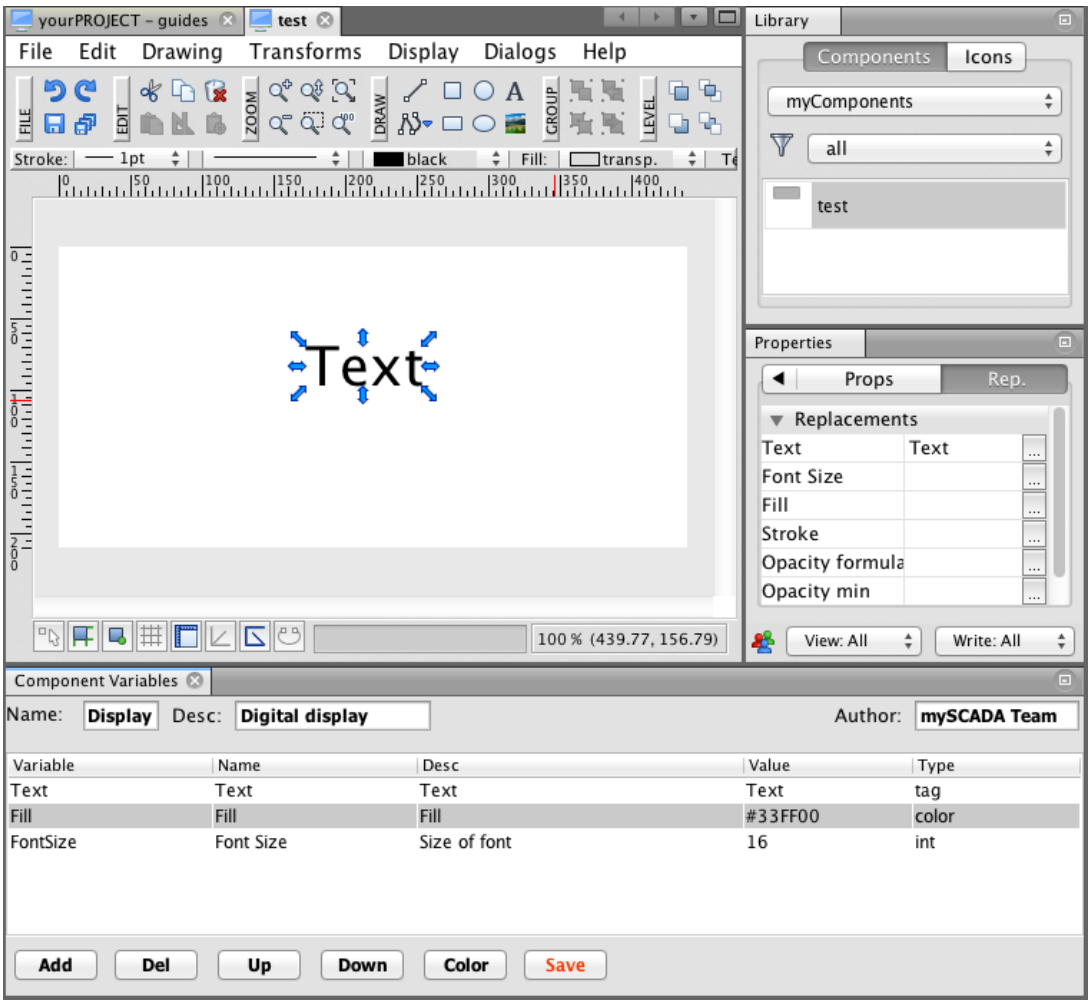
- Variable Text with type *String*
- Variable Font Size with type *Int*
- Fill with type *Color*
- Stroke with type *Color*
- Opacity Formula with type *Int* or *Double*
- Opacity min/max with type *Int* or *Double*

The *Opacity* here is similar to *Opacity* animation, which means that the text will be visible if the value of Opacity formula will be within Opacity min/max range.

2) Now create a text element:



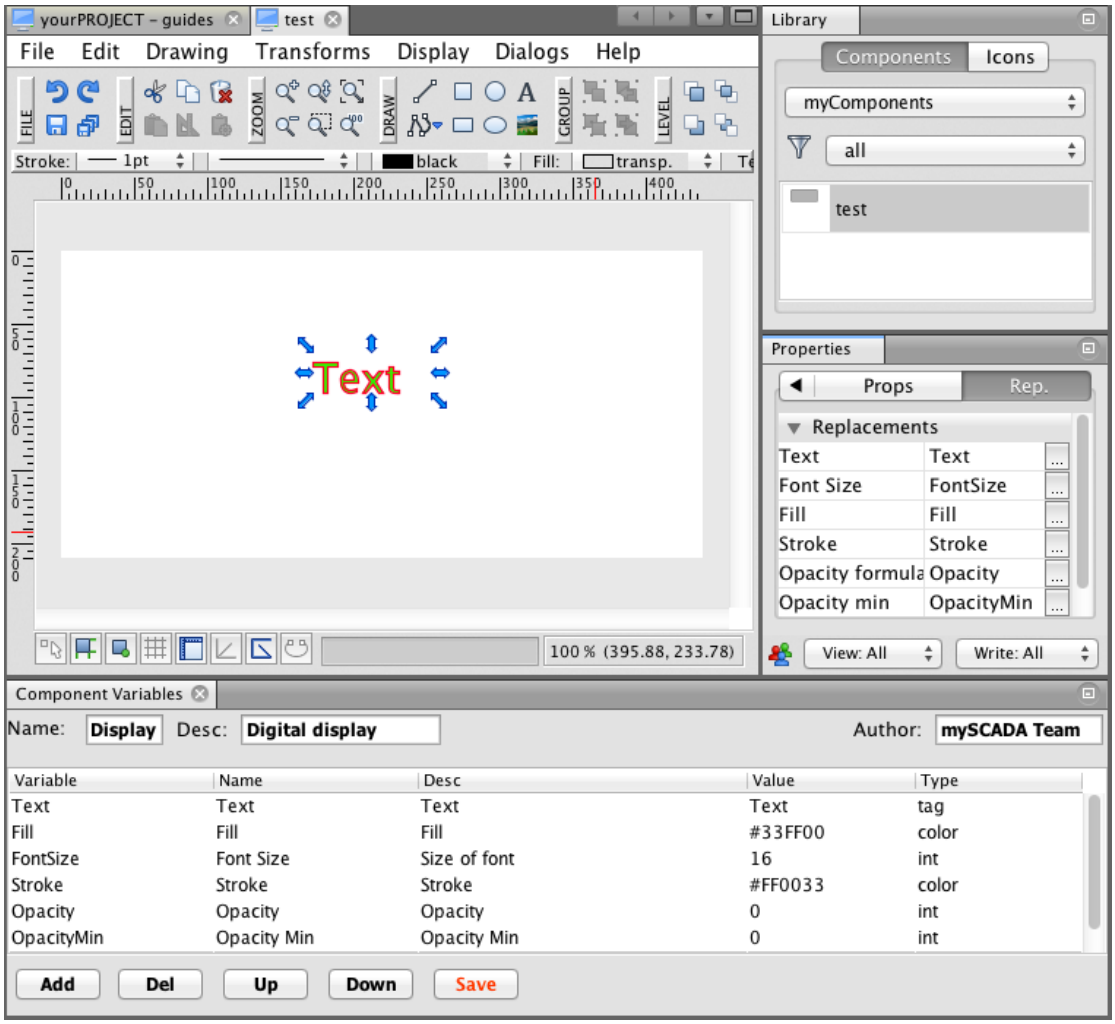
3) Fill in the *Replacement -> Text* in the *Properties window*.



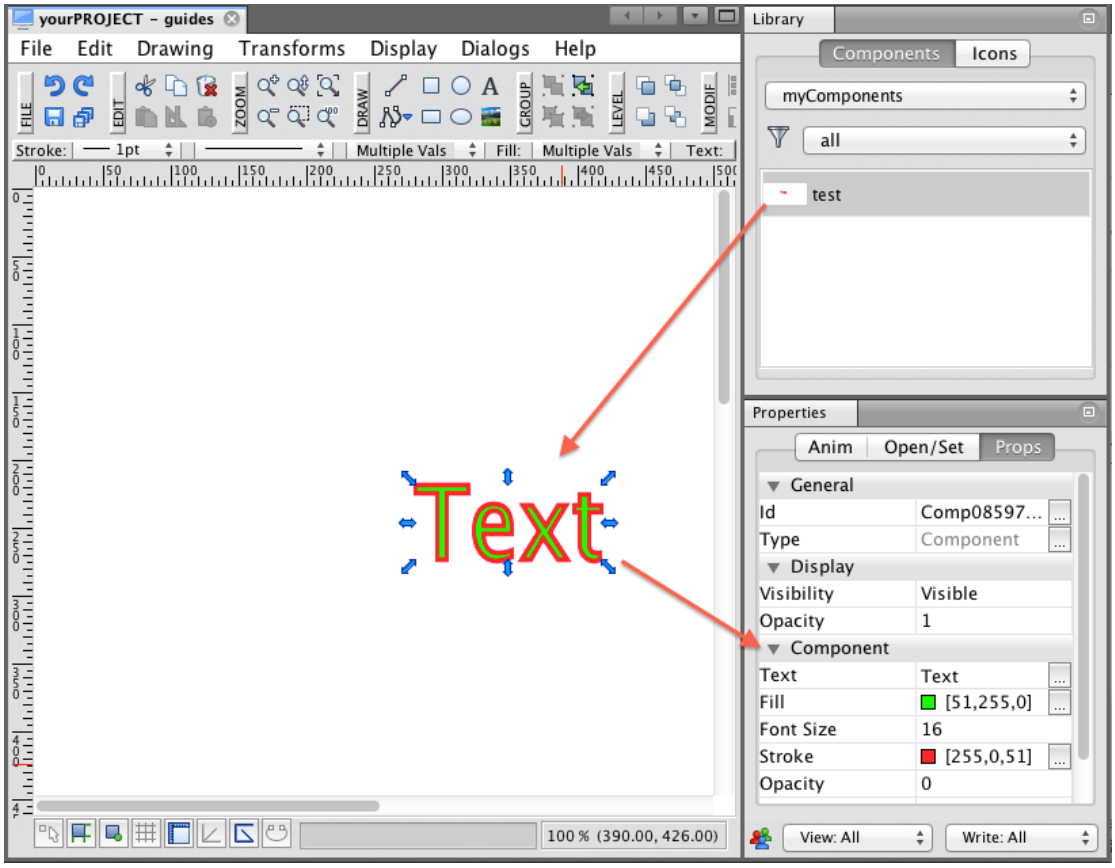
4) Input the *Component Variable* that you want to use for the text -*"Text"* in this case and fill the *FontSize* and other parameters alike**.**

   Note: The variable names are upper case sensitive.

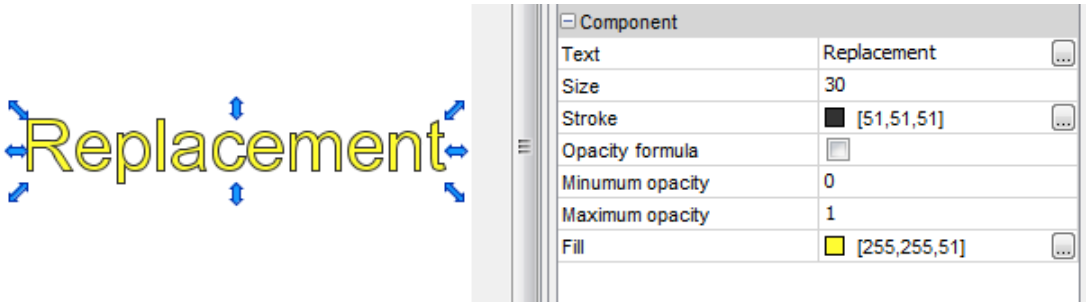
5) When you fill all properties save your component.



6) Text will change according to the value of an attached component variable. After saving the component, open the *Library* and import it.



- Open the *User Components* library in the *Properties window* and select the created component. Drag the component into an opened view and click on it to see its properties.
- Now, navigate to the *Component* section, where you can change the value for all component variables.

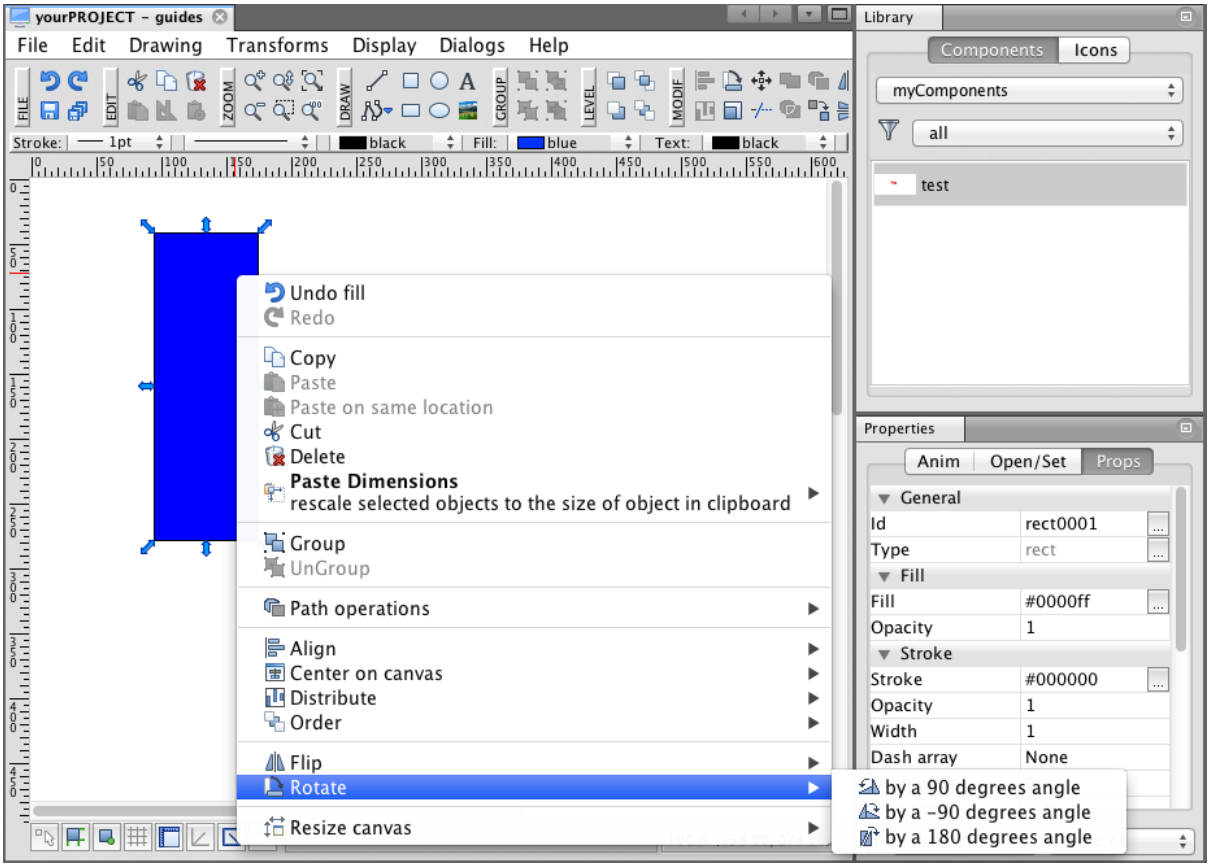


Replacements are important for complex components, where you can replace the component text (button text, for example) by just one click.

Example:

In the following example we create a simple slide bar showing the variable progress and its **Min** and **Max** values.

1) Create a new component and draw a rectangle as seen in the picture below.
2) Right-click on the rectangle and select *Rotate -> by 90 degrees angle* from the menu.



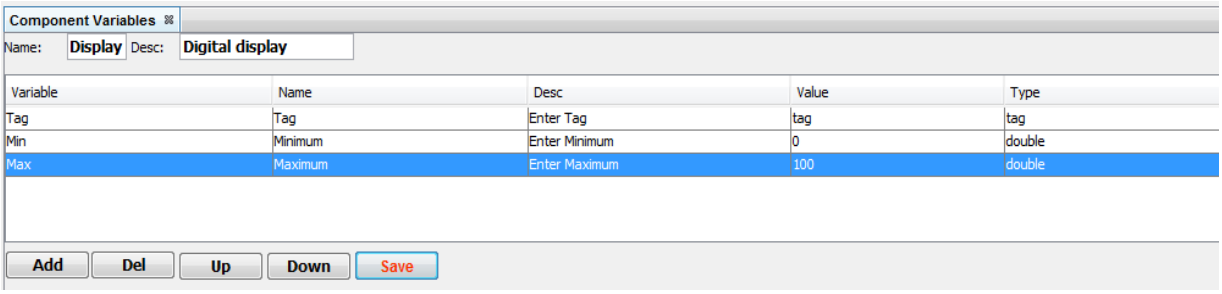
3) Copy the selected rectangle and apply ***Paste on same location*** on it.

   Set the **Fill** property to "none" then put the element to the background with the function *Lower to the background.*

4) We have now created a frame of the scale.

5) Now create 3 text fields and add the animations:

   For the animation of the slide bar we will our rectangle. Fill the *Component Variables* as shown in the picture below:

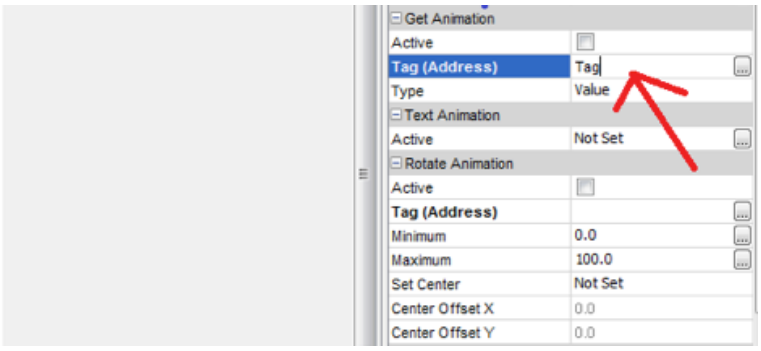


As you can see we have created three local variables, for the tag value you have used a variable tag, for specifying the minimum and maximum values you have used the min and max variables.

*Note: Keep saving your project during such a complex creation with the button **Save**.*

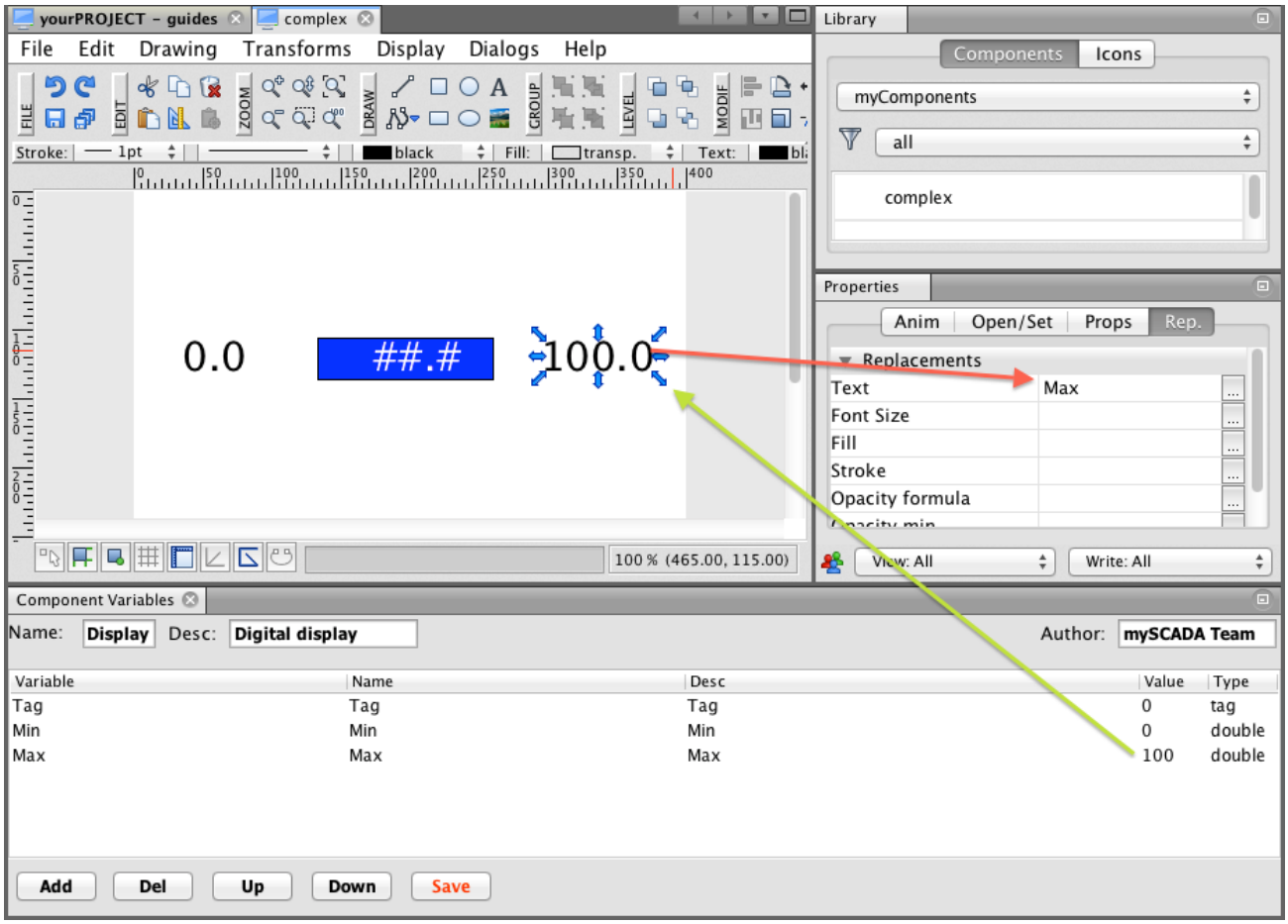
To make indicator out of the rectangle by resizing it, according to the value read from the PLC, set the *Scale animation* as shown in the following picture:

We also want to show the actual value in the middle of the slide bar. Click on the text field in the middle of the rectangle, select the **Anim** tab in the *Properties window* and click on the **Tag (Address)** field in the *Get Animation* section. Then fill in the **Tag** field with "Tag", which corresponds to our local variable name.



As you can see we have used one of our local variables, you can use the same variable (especially the tag type) for multiple animations/effects to achieve better effect.

We will also show the *Min* and *Max* limits of our tag. Click on the right text field, then select the **Rep.** tab from the *Properties window* and set *"Max"* for *Text* in the *Replacements.*



7) Now do the same for the other text field but place the *Min* variable there.

   Let us finish the component by drawing a simple rectangle around the slider as a background (*Transforms –> Order –> Lower to background*) to create a boundary of the component.

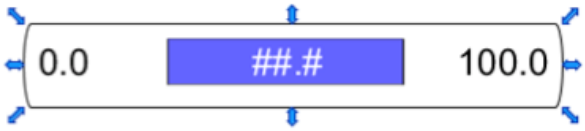
   The final component should as follows:

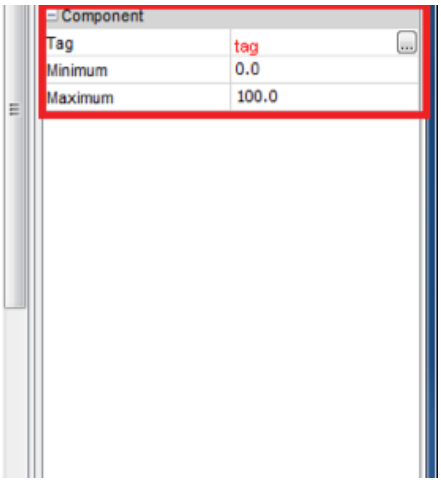
8) Save the component and test its functionality.

21.4 Testing Component

Create a new view and open it, then navigate to the *Library*. Select your component and drag it to the canvas.

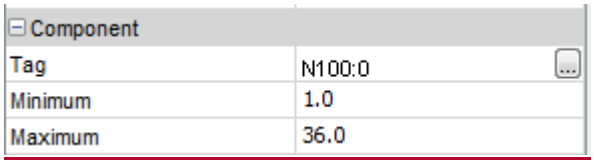
Now click on the component and look into the *Properties window*:



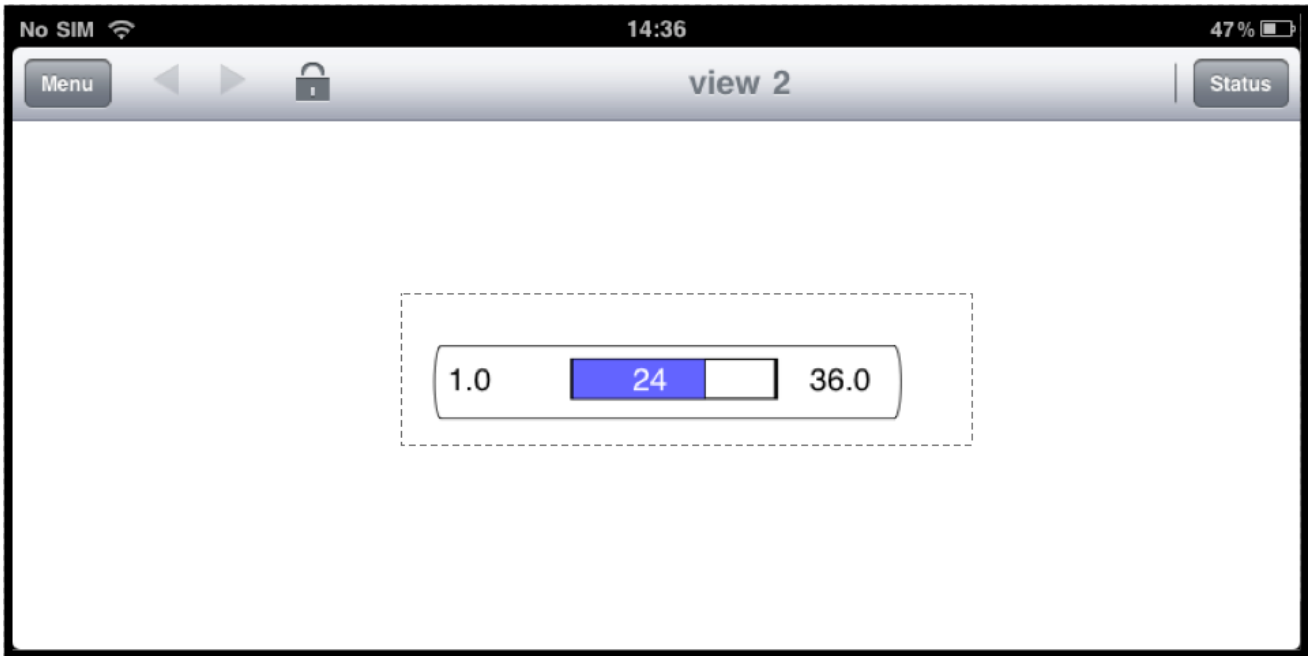


In the *Component* section you can see all local variables you have defined with variables of the tag type displayed in bold letters.

Set the **Tag (address),** the **Minimum** and **Maximum** values:



Now save the view and send to a supported device, the screen should look as follows:

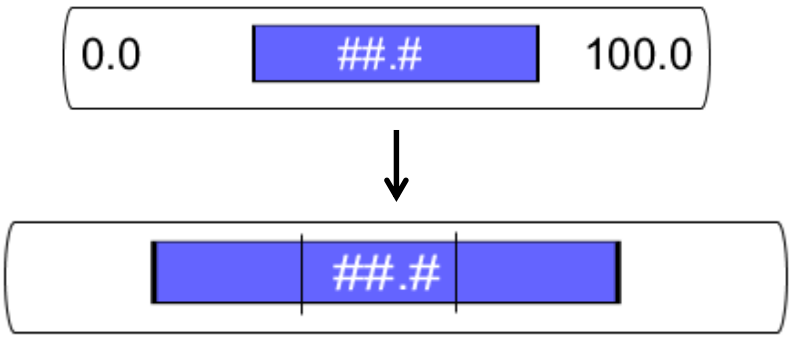


21.5 Entering Advanced Functions (Equations)

To create complex and feature rich components you need to enter complex formulas instead of simple variables. You can use functions anywhere you like: inside animations, instead of constants or as a text replacement of the text fields.

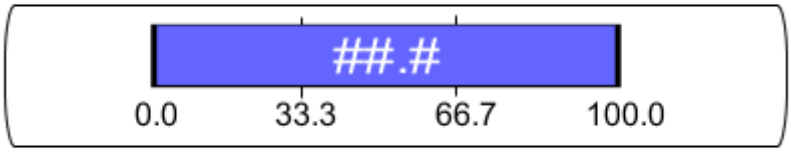
To show the functionality we will extend our example by implementing the scale along the slide bar we have just created.

1) Open the slide bar created in the previous chapter and adjust as follows:



2) Delete the *Min* and *Max* fields and stretch the blue rectangle, including the frame. After that, insert additional lines with the *Drawing-Line* tool to divide the rectangle into thirds. With *Transforms -> Order -> Lower to background* send both lines and frame into the background

3) Now insert additional text fields.



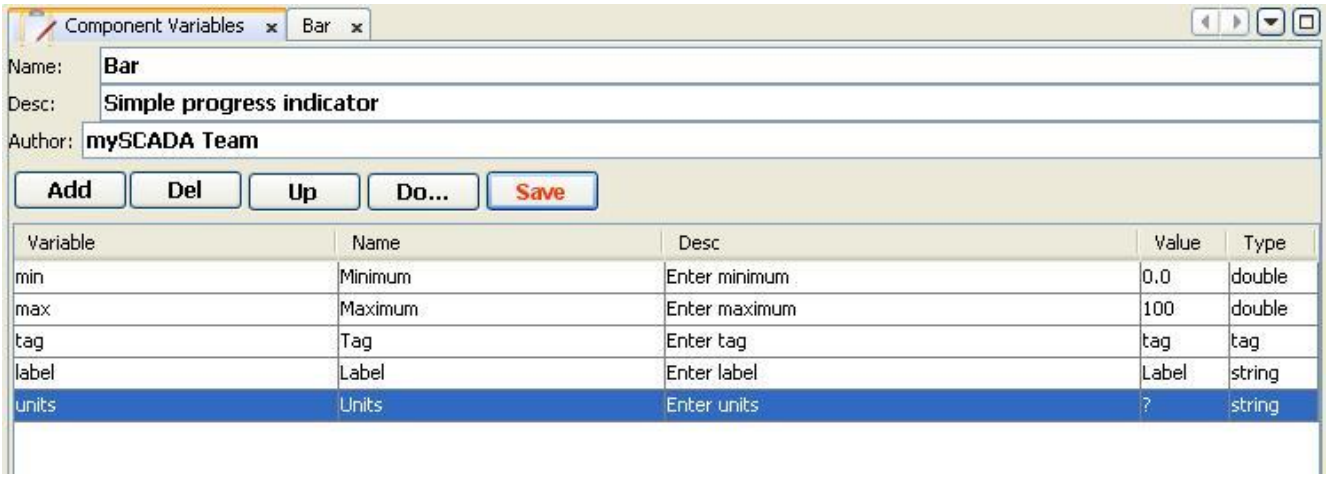
For each text field we need to specify the *Text* in the *Replacement* in the **Rep.** tab:

- Left text field: *(min).toFixed(1)*
- Second text field (from left): *((min+max)/3+min).toFixed(1)*
- Second text field (from right): *(2*(min+max)/3+min).toFixed(1)*
- Right text field: *(max).toFixed(1)*

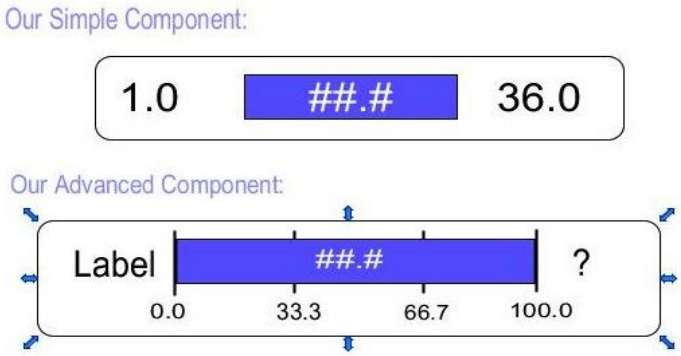
The left text field will show the value of the variable *min.toFixed(x)*, which is a function that rounds the result to x decimal places. In our case, the value will be formatted to show one decimal place.

The text field second from left should show 1/3 of the scale, therefore we have used the formula *min+(min+max)/3* to get 1/3 of the scale. We will use the *toFixed(1)* function to round the result.

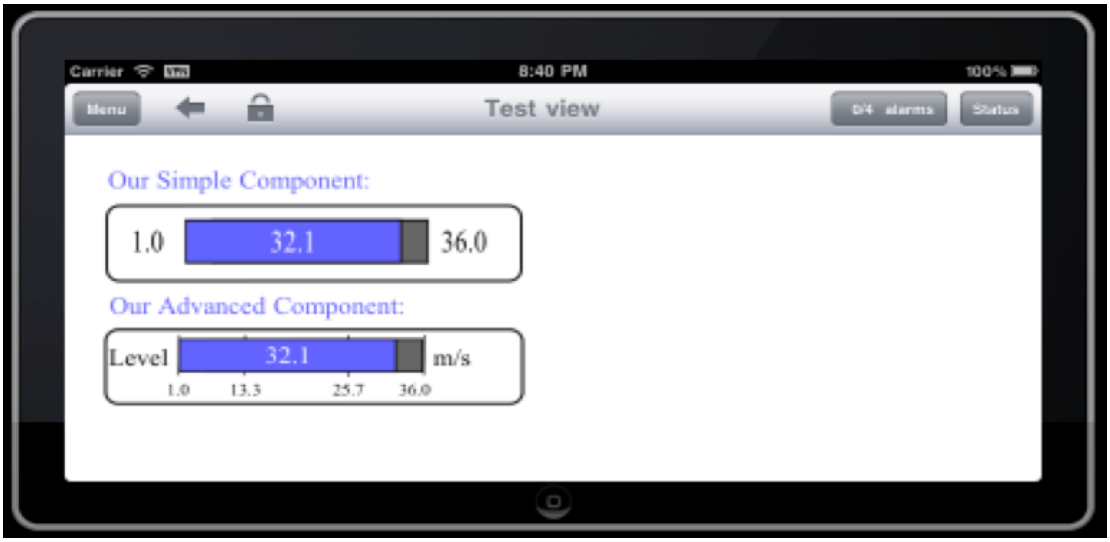
The other two text fields are similar to those two, add units and a label to our component to finish:



To use the Label and Units text fields we need to create two new variables. Both will be of the type string. The first variable will be named *label* and the second will be named *unit*. For the text field label add Text Replacement equal to variable label. For Units text field add Text Replacement equal to variable units. Finally, click on the **Save** button to see the changes.

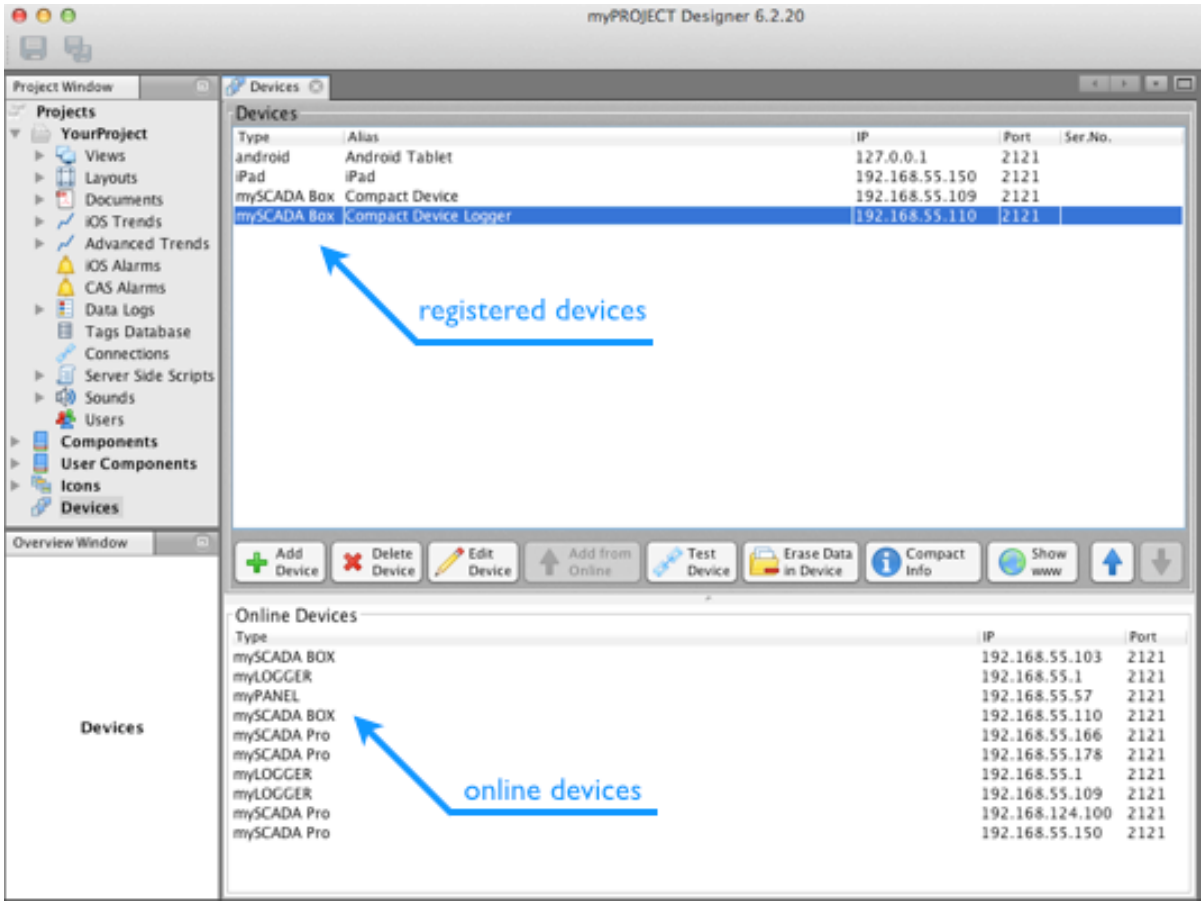


The visualization running on iPad:



22. Devices

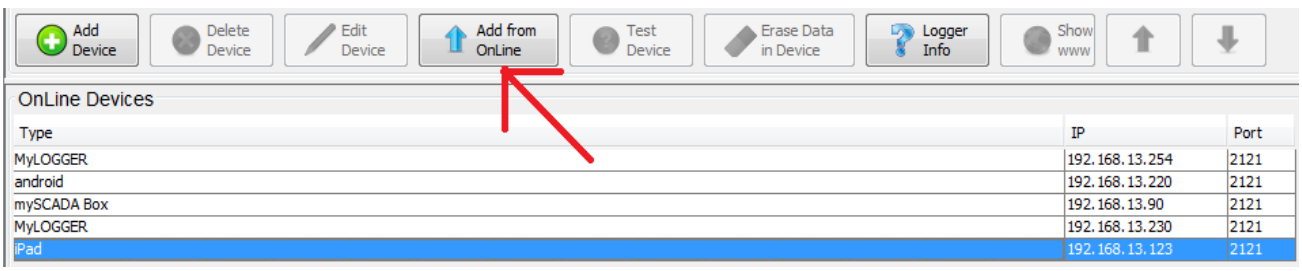
This module allows a quick overview of all available *mySCADA* device connections - manually added and connected to the local network.



The main window of the *Devices* window is split into two parts:

- Upper part displays all permanently defined devices
- Lower part displays the list all compatible devices in the local network (iOS or Android devices with the *mySCADA* app running in the foreground, *mySCADA* boxes and *mySCADA Desktop* and *Server*)

You can easily store online devices the defined by selecting them from the *Online Devices* list and clicking on the **Add from OnLine** icon.



1) Click on the **Add Device** button to add a device manually and fill in the dialog window:

2) Select the device type first:

- iPad
- iPhone
- iPod touch
- mySCADA Box
- myLOGGER
- Desktop
- Server

3) Set the device parameters:

- *Alias* – name describing the device
- *IP* – fill in the IP address of the devices
- *Ser. No.*: – serial number of the device (mySCADA box/myLOGGER only). You can find serial number in **System-Status** menu of the device.

23. Communications

The user is not forced to enter the data type during the screen drawing. The proper data type is fetched during the data reading from the PLC. There are some limitations and only some data types are supported. The following paragraph specifies all supported data types for each PLC type. Each supported PLC protocol is described in the subchapter below.

23.1 EtherNet/IP (ControLogix)

This EtherNet/IP enabled PLC brand supports all numeric data types:

Tag Data Type	Description
DINT	An atomic data type consisting of a DWORD used for storing a 32-bit signed integer value (-2,147,483,648 to +2,147,483,647).
SINT	An atomic data type that stores an 8-bit signed integer value (-128 to +127).
INT	An atomic data type consisting of a word used for storing a 16-bit signed integer value (-32,768 to +32,767).
REAL	An atomic data type that stores a 32-bit IEEE floating point value.
BOOL	The BOOL data type is an atomic data type consisting of a single bit.

The access to user defined data types like structures or arrays are supported. To read or write the elements inside the structured tags you should use the “dot” notation, e.g.

tank[1].volume is a proper syntax to read the volume of the 1st tank.

Reading Controller Tags:
To read a controller tags just specify the tag name.

Reading Program Tags:
To read a program tag, you have to specify the program name before a tag name. You must use the following syntax:

Program: name.tag
name - program name
tag - tag name

So if you want to communicate with tag named valve from a program named control you will have the following syntax: Program:control.valve

Limitation of the BOOL arrays in ControLogix

The different syntax to access the elements in the BOOL arrays is used in the *myPROJECT Designer,* compared to the programs stored in the controller. In *myPROJECT Designer* you have to address the word and bit position separately. The word consists of the 32 bits – BOOL elements and the range of the BOOL array is split into multiple words.

Example:

If you have a BOOL array TestBool with 54 elements you can access its elements using the following address:

3rd element (bit)	TestBool[0].3
34th element (bit)	TestBool[1].2
54th element (bit)	TestBool[1].22
Out of range, bad address	TestBool[2].0
Out of range, bad address	TestBool[50]

Recommendation on improvement of communication speed

The communication with the *ControLogix* controller has been designed in such way that reading of arrays together is faster than reading each element separately. This means that if you use the tags that can be arranged into the arrays you will notice significant increase of speed.

For example, to read the status of levels in all 10 tanks you should design your data types so that all levels are stored in an array with 10 elements. Below you will see an example of how to implement the faster method:

Level[1]	
Level[2]	FAST
Level[3]	

If you prefer structures you can use the following way, however, addressing of this type will slow down the communication. Therefore, we do not recommend to use this type of data addressing.

BigTank[1].Level	
BigTank[2].Level	SLOW
BigTank[3].Level	

23.2 MicroLogix and SLC

For *MicroLogix* PLC the data types are defined by means of the used file types. In PLC programs you can use many different file types, but only the types specified in the following table can be read or written by the *mySCADA* application.

File Type	Description	Use with these controllers
N – Integer	16-bit long numbers, signed, this file is used for storing numeric values or bit information, read/write support.	SLC and MicroLogix
B – Binary	16-bit long numbers, signed, this file stores internal relay logic, read/write support.	SLC and MicroLogix
F – Float	This file stores numbers with a range of 1.1754944e-38 to 3.40282347e+38, read/write support.	SLC and MicroLogix
L - Long	32-bit long numbers, signed, also called double word, read/write support.	MicroLogix 1100, 1200, 1400 and 1500
O - Output	This file stores the state of output terminals for the controller, read only.	SLC and MicroLogix
I- Input	This file stores the state of input terminals for the controller, read only.	SLC and MicroLogix

The other file types like *Strings, Timer, Counter* and *Control* are not supported.

There is a limitation that certain outputs can be read-only. To set outputs you should use the other file type and then adjust the program design.

To access the file elements you should specify the element position after the colon, i.e. to read or write to the 3rd element of the integer file no. 100, use the following syntax:

N100:3

You can access each bit directly by defining its position after the slash, i.e. to access the 7th bit of the 3rd element in the integer file no. 100 use the following syntax:

N100:3/7

23.3 Modbus

The *Modbus* communication protocol describes the interactions of each device on a *Modbus* network. In the protocol description you can find all details about establishing address, device recognitions and all other important parts of the *Modbus* communication. Here we will concentrate on protocol usage by *mySCADA* application.

Tag name syntax

In the following table the available syntaxes of the tag names are summarized.

<table>
<tr><td>Tag</td><td>Meaning</td><td></td></tr>
<tr><td>Rn:e</td><td>16-bit signed integer stored in input register at address e</td><td rowspan="2">Read access only.

Read function: 4

Standard address range:

30001-39999</td></tr>
<tr><td>R:e</td><td>16-bit unsigned integer stored in input register at address e</td></tr>
<tr><td>H:e</td><td>16-bit unsigned integer stored in holding register at address e</td><td rowspan="15">Read/write access.

Read function: 3

Write function: 16

Standard address range:

40001 – 49999(extended to 499999 by some manufacturers)</td></tr>
<tr><td>Hn:e</td><td>16-bit signed integer stored in holding register at address e</td></tr>
<tr><td>Hf:e</td><td rowspan="4">Float number stored in holding register at address e</td></tr>
<tr><td>Hfs:e</td></tr>
<tr><td>Hfsb:e</td></tr>
<tr><td>Hfsw:e</td></tr>
<tr><td>Rf:e</td><td rowspan="4">Float number stored in input register at address e</td></tr>
<tr><td>Rfs:e</td></tr>
<tr><td>Rfsb:e</td></tr>
<tr><td>Rfsw:e</td></tr>
<tr><td>Hd:e</td><td rowspan="4">32-bit integer stored in holding register at address e</td></tr>
<tr><td>Hds:e</td></tr>
<tr><td>Hdsb:e</td></tr>
<tr><td>Hdsw:e</td></tr>
<tr><td>Rd:e</td><td>32-bit integer stored in input register at</td></tr>
</table>

Rds:e Rdsb:e Rdsw:e	address e	
I:e	Discrete input, boolean value	Read access only. Read function: 2 Standard address range: 10000-19999
O:e	Discrete Output Coils	Read/write access: Read function: 1 Write function: 15 Standard address range: 1-9999

A lot of manufacturers use addresses instead of tags when describing the *Modbus* communication. *mySCADA* currently does not support direct access by address, so you have to convert such addresses to tags, according to the provided table above. Be aware that specific read/write function is needed to access each range, so the correct choice of the tag type is important.

32-bit registers in Modbus

The *Modbus* protocol was designed to operate with devices of 16-bit register length. Consequently, special considerations are required when implementing 32-bit data elements. Most of implementations use two consecutive 16-bit registers to represent 32 bits of data, or eventually 4 bytes of data. Within these 4 bytes data single-precision floating-point data can be encoded into a Modbus RTU message.

Modbus itself does not define floating-point data type but it is widely accepted that it implements 32-bit floating-point data, using the IEEE-754 standard. However, the IEEE standard has no clear definition of the byte order. Therefore, the most important consideration when dealing with 32-bit data is that the data will be addressed in the correct order.

The following table shows two adjacent 16-bit registers conversion to a 32-bit floating-point or 32-bit integer value:

Register suffix	Swap mode	16-bit registers	32-bit floating point or integer
-	N/A	[a b][c d]	[a b c d]
s	Byte and word swap	[a b][c d]	[d c b a]
sb	Byte swap	[a b][c d]	[b a d c]
sw	Word swap	[a b][c d]	[c d a b]

The following section describes the 32-bit data types implementation to the *mySCADA* application.

Floating point numbers

Floating point data type is possible to use in both input and holding registers and all possible byte swap combinations are supported.

Register	Mapping	Swap mode	Bytes in 2 16-bit registers	Resulting 32-bit floating point
Hf	Holding registers	N/A	[a b][c d]	[a b c d]
Hfs	Holding registers	Byte and word swap	[a b][c d]	[d c b a]
Hfsb	Holding registers	Byte swap	[a b][c d]	[b a d c]
Hfsw	Holding registers	Word swap	[a b][c d]	[c d a b]
Rf	Input registers	N/A	[a b][c d]	[a b c d]
Rfs	Input registers	Byte and word swap	[a b][c d]	[d c b a]
Rfsb	Input registers	Byte swap	[a b][c d]	[b a d c]
Rfsw	Input registers	Word swap	[a b][c d]	[c d a b]

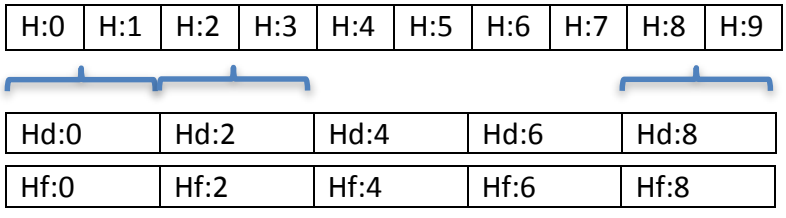
32-bit Integers

The long integers are implemented in the same manner as the floating-point numbers.

Register	Mapping	Swap mode	Bytes in 2 16-bit registers	Resulting 32-bit integer
Hd	Holding registers	N/A	[a b][c d]	[a b c d]
Hds	Holding registers	Byte and word swap	[a b][c d]	[d c b a]
Hdsb	Holding registers	Byte swap	[a b][c d]	[b a d c]
Hdsw	Holding registers	Word swap	[a b][c d]	[c d a b]
Rd	Input registers	N/A	[a b][c d]	[a b c d]
Rds	Input registers	Byte and word swap	[a b][c d]	[d c b a]
Rdsb	Input registers	Byte swap	[a b][c d]	[b a d c]
Rdsw	Input registers	Word swap	[a b][c d]	[c d a b]

Address mapping

On the *Modbus* server-side only 16-bit long holding and input registers are used. The 32-bit long data types are just the interpretation of the two adjacent registers. The concept of address mapping is clearly shown on the following figure. Three tables are shown there. All of them are addressing the same place in the memory: holding registers. In the first table the holding registers are shown, in the second the 32-bit integers and in the last table the floating-point data type are shown.



Signed and unsigned numbers

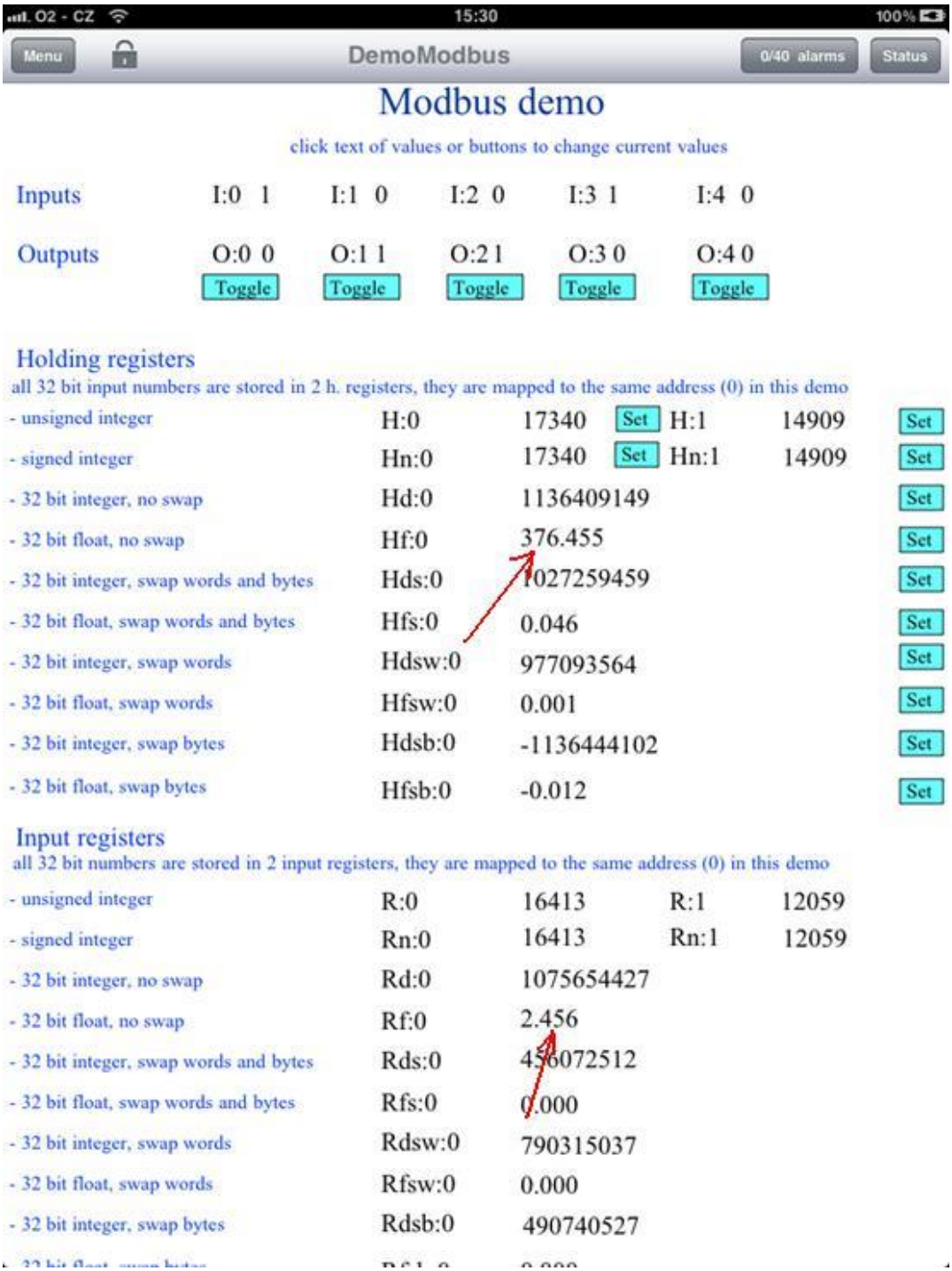
Signed and unsigned integers can be stored in the registers. The unsigned numbers are read from or written to the *Modbus* device using simple addressing, like H or R to holding and input registers. To use signed integers add the suffix “n” to the address. Using tag Hn:5 means that access the holding register at address 5 and the number will be interpreted as the signed integer.

In 32-bit data type numbers, the signed integers are used by default and the suffix “n” is not used.

Example:

An example of advanced *Modbus* functionality can be seen on *ModbusDemo* screen, which can be downloaded as a part of the demo project from *http://www.myscada.org/downloads*.

The current values of inputs and outputs are visible in the upper part of the screen. Further, the current values of holding registers H:0 and H:1 and its data interpretation if the stored numbers are signed integers, float numbers or 32 bit integers. To set the register, click on the **Set** or **Toggle** buttons. Figure 46 shows the storage of a floating point number 376.455 stored using Hf:0 register and the floating-point value 2.456 is stored in the input register Rf:0.



Siemens S7 family PLCs
Standard S7-200/300/400/1200 LOGO! item Syntax

Address Syntax

Input, Output, Flag Memory Types

<memory type><S7 data type><address>

<memory type><S7 data type><address><.bit>

DB Memory Type

DB<num>,<S7 data type><address>

DB<num>,<S7 data type><address><.bit>

where <num> ranges from 1 to 65535.

Memory types

Memory Type	Description	Address Range	Data Type	Access
I **E**	Inputs	Dependent on S7 Data Type (see table below)		Read/Write
Q **A**	Outputs			Read/Write
M **F**	Flag Memory			Read/Write
V	Variable Memory			Read/Write
DB	Data Blocks			Read/Write

Note: The Variable Memory is available only for S7-200 and LOGO!.

S7 Data types

S7 Data Type	Description	Address Range	Data Type
X	Bit	Xo. b-X65534.b .b is Bit Number 0-7	**Boolean**
B **BYTE**	Unsigned Byte	Bo-B65535 BYTEo-BYTE65535	**8-bit unsigned integer**
		Bo.b-B65535.b BYTEo.b-BYTE65535.b .b is Bit Number 0-7	Boolean
C **CHAR**	Signed Byte	Co-C65535 CHARo-CHAR65535	**8-bit signed integer**
		Co.b-C65535.b CHARo.b-CHAR65535.b .b is Bit Number 0-7	Boolean

W **WORD**	Unsigned Word	Wo-W65534 WORDo-WORD65534	**16-bit unsigned integer**
		Wo.b-W65534.b WORDo.b-WORD65534.b .b is Bit Number 0-15	Boolean
I **INT**	Signed Word	Io-I65534 INTo-INT65534	**16-bit signed integer**
		Io.b-I65534.b INTo.b-INT65534.b .b is Bit Number 0-15	Boolean
D **DWORD**	Unsigned Double Word	Do-D65532 DWORDo-DWORD65532	**32-bit unsigned integer**
		Do.b-D65532.b DWORDo.b- DWORD65532.b .b is Bit Number 0-31	Boolean
DI **DINT**	Signed Double Word	DIo-DI65532 DINTo-DINT65532	**32-bit signed integer**
		DIo.b-DI65532.b DINTo.b-DINT65532.b .b is Bit Number 0-31	Boolean
REAL	IEEE Float	REALo-REAL65532	**Float**

Note: Be cautious while modifying WORD, INT, DWORD, and DINT type, as each address starts at a byte offset within the device. Therefore, words MW0 and MW1 overlap at byte 1. Writing to MW0 will also modify the value held in MW1. Similarly, DWORD and long types can also overlap. It is recommended that these memory types should be used in such way that overlapping does not occur. As an example DWORD MD0, MD4, MD8 ... etc. can be used as a prevention of bytes against overlapping.

Note (iOS only): Please be aware of difference in addressing X type of variables (like IX0.0) between TIA portal/Step 7 and mySCADA. The problem is that in TIA portal variable of X type defined as of 8-bit size. In mySCADA we treat them as 16-bits with swap byte order. For example if you define db1.dbx1 and .dbx2 in TIA portal you can access each of them in .0-.7 bit range. In mySCADA you can access both variables as db1.dbx1 with .0-.15 bit range. Because of swap bit order db1.dbx1.0-.7 in mySCADA would relate to TIA portal db1.dbx2.0-.7 and db1.dbx1.8-.15 would access db1.dbx1.0-.7 in TIA portal addressing. We made such difference in mySCADA to fully accommodate analog inputs readings, like IX64, for example. At the moment to achieve same addressing to BOOLEAN variables as TIA portal please use B type instead of X in mySCADA, as B type don't have swap bytes order.

S7 1200/1500 notes

To access the DB in S71200/1500 some additional setting PLC-side are needed.

1. Only global DBs can be accessed.
2. The optimized block access must be turned off.
3. The access level must be “full” and the “connection mechanism” must allow GET/PUT.

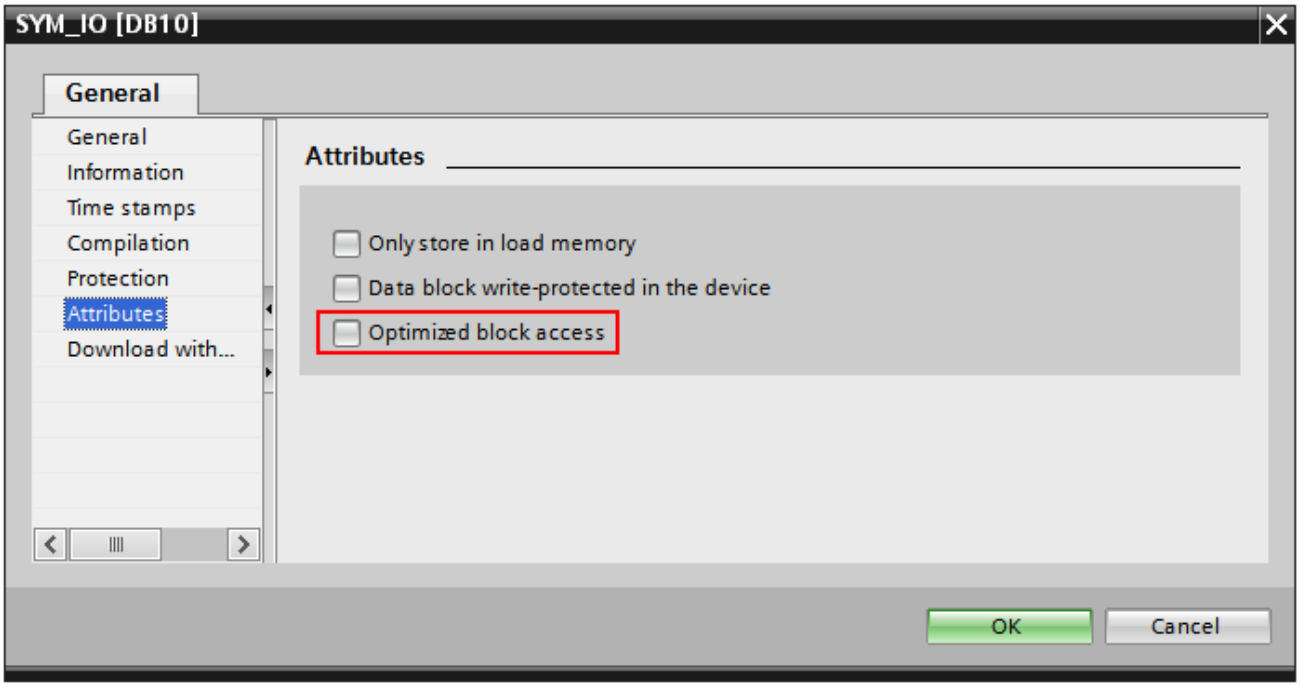
Set the previous in TIA Portal (shown version V12)

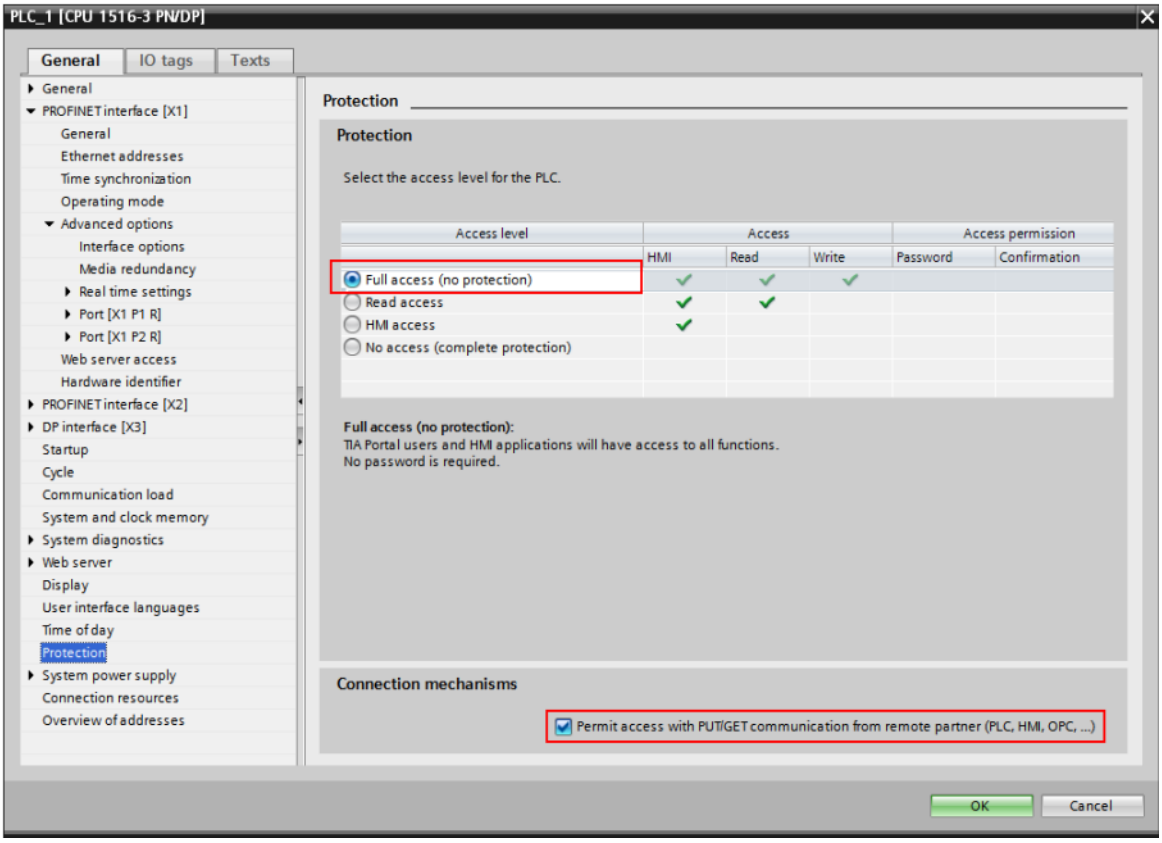
Select the DB in the left pane under “Program blocks” and press **Alt+Enter** (or in the contextual menu select “Properties…”)

Uncheck Optimized block access, (by default it is checked).

Protection

Select the CPU project in the left panel and press **Alt+Enter** (or in the contextual menu select "Properties…")
In the item *Protection*, select "Full access" and Check "Permit access with PUT/GET ….", as pictured.





LOGO! 0BA7/0BA8 configuration

Configuring the **server connection** allows you to connect *LOGO!* with *mySCADA* devices for reading and writing to the memory, just like a HMI panel would do.

- In the **Tools** menu choose the **Ethernet Connections** item.
- Right-click on the Ethernet Connections and select *Add connections* to add a connection
- Double-click on the new connection created and edit its parameters by selecting **Server Connection**.

Note:
The Local TSAP in corresponds to the Remote TSAP of LOGO! and vice-versa. ***This is the key concept for the S7 connections!*** *If you uncheck "Accept all connections" you must specify the mySCADA device address. The "Connect with an operator panel" checkbox can be checked or unchecked.*

- Confirm the dialog, close the connection editor and **download** the configuration into LOGO!.

S7-200 (via CP243-1) configuration

Configuration of *S7 200* is very similar with *LOGO!*

- In the **Tools** menu select the **Internet Wizard** item.
- Set up the CP243 position and IP configuration, in the next window select at least one *peer-to-peer* connection and set it up the same way as with *LOGO!*.

23.4 OPC UA

OPC Unified Architecture is an interoperability standard developed by OPC Foundation. It is the successor to OLE for process control (OPC). Although developed by the same organization, OPC UA differs significantly from its predecessor. The old DCOM-based version is not supported by *mySCADA* products. Using this protocol has two main advantages:

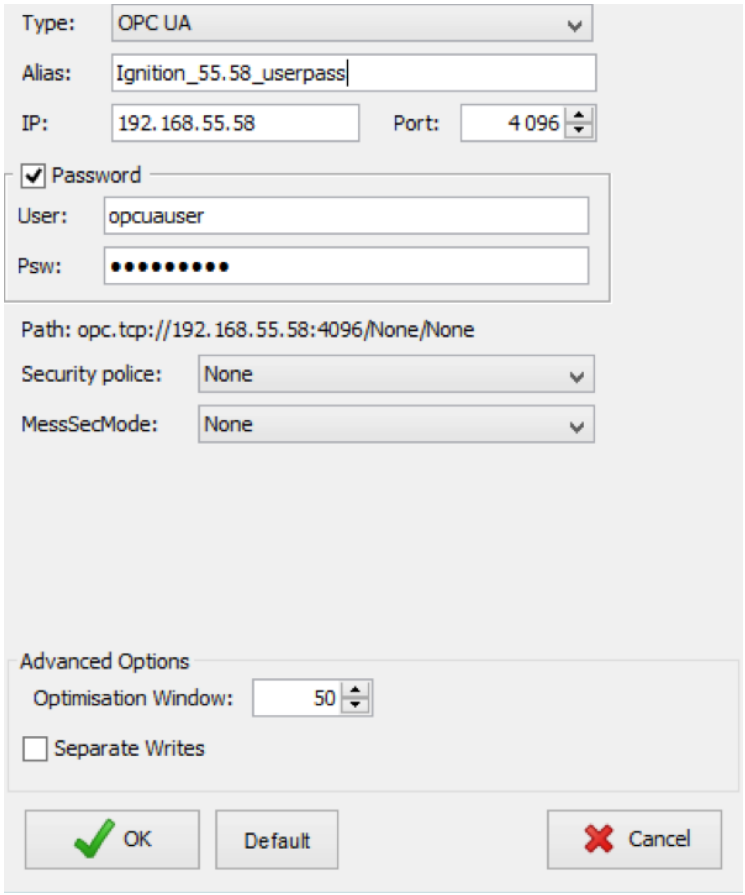
- **Security** – configures a secure message interchange, based on a contemporary cryptography
- **Scalability** – OPC server can communicate with a different vendors infrastructure

You should keep in mind that when using OPC you are not communicating with PLCs directly. The OPC server stands in the middle and all communications are dependent on the OPC server configuration.

mySCADA implementation of the OPC UA standard supports only the binary protocol - *opc.tcp://Server*. We do not support the web version – http://Server.

Connection configuration

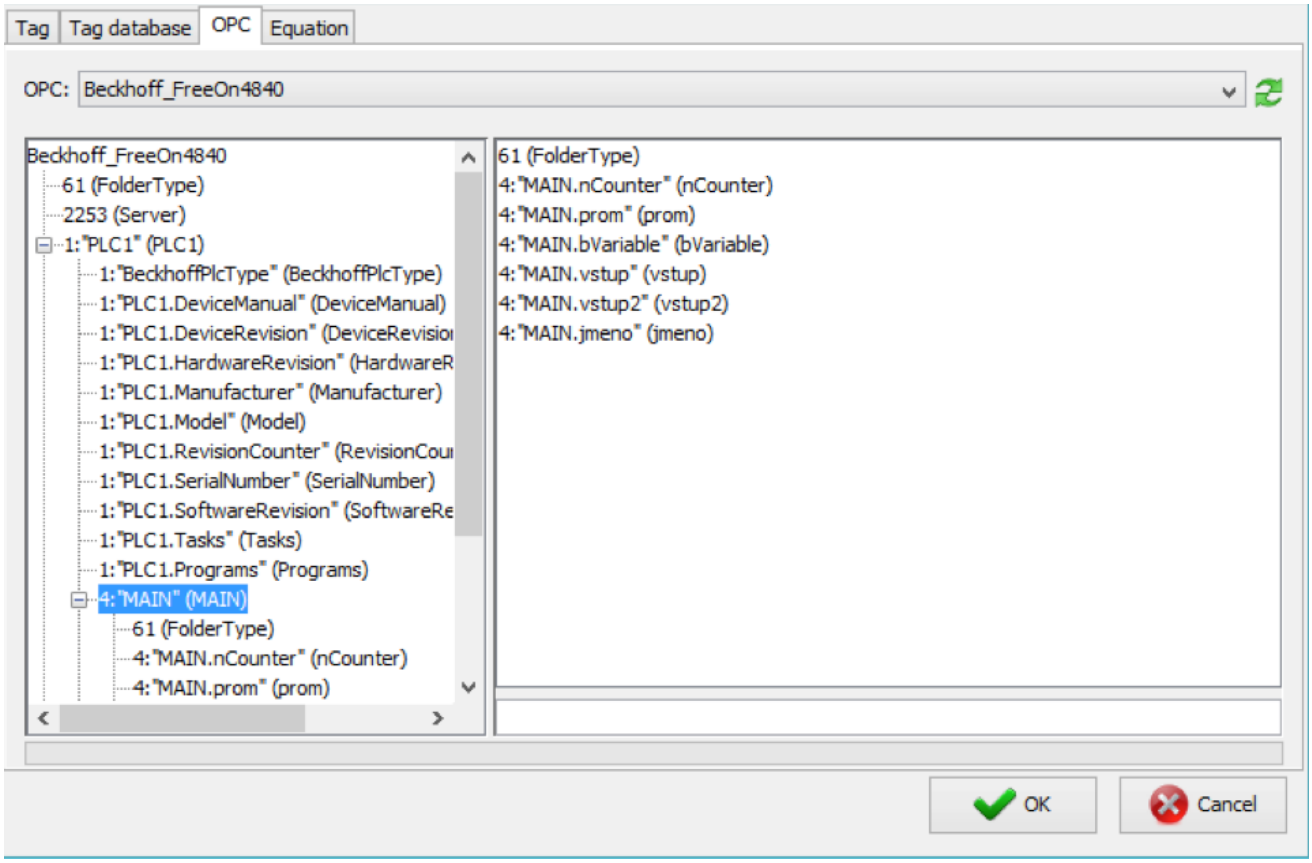
With *myPROJECT Designer* you can set connections to the *OPC* servers and the following parameters.



- **IP Address, Port** – IP address and port of the *OPC UA* server
- **User, Password** (Optional) – credentials to the *OPC UA* server, if configured on the server
- **Security Police** (Optional)
  - **Basic128Rsa15 -** uses 128-bit cryptography
  - **Basic256 -** uses 256-bit cryptography
  - **None** (Default)
- **Message Security Mode** (Optional)
  - **SignEnc -** Signs and encrypts the messages
  - **Sign** - Signs the messages
  - **None** (Default) - No security
- **Certificates** (Optional) - you should provide a private key and associated certificate in the PEM format, you can easily recognize PEM format, that the certificate file will begin with: "-----BEGIN CERTIFICATE-----" and end with "-----END CERTIFICATE-----" sentence.

Tag name syntax

Tag name is any string, which is supported and accepted by the connected *OPC UA* server. You can use the browse function in the designer to get the list of tags at your disposal. You can use the dialog to select the proper tag, later after selecting, you can edit the tag name. Usually the subsequent edition is not needed, but sometimes is, e.g. to target the proper tag address by adding the suffix.



23.5 MELSEC-Q

Currently *mySCADA* supports only a part of the *MELSEC-Q* protocol, namely 3E type of packets, originally intended for E71 type of adapters. In the table below you can find the list of both basic (original) tag syntaxes and some extensions, introduced by our team.

Tag name syntax

In the following table the available syntax of the tag names is summarized.

Tag	Meaning	Address range (decimal)
SM (91h)	Special relay, 1-bit value	0000 to 2047
SD (A9h)	Special register, 16-bit value	0000 to 2047
X (9Ch)	Input, 1-bit value	0000 to 8191
Y	Output, 1-bit value	0000 to 8191

(9Dh)		
M (90h)	Internal relay, 1-bit value	0000 to 8191
L (92h)	Latch relay, 1-bit value	0000 to 8191
F (93h)	Annunciator, 1-bit value	0000 to 2047
V (94h)	Edge relay, 1-bit value	0000 to 2047
B (A0h)	Link relay, 1-bit value	0000 to 8191
D (A8h)	Data register, 16-bit value, unsigned	000000 to 012287
W (B4h)	Link register, 16-bit value, unsigned	0000 to 8191
TS/TC (C1h)/(C0h)	Contact/Coil timer, 1-bit value	0000 to 2047
TN (C2h)	Current value timer, 16-bit value, unsigned	0000 to 2047
SS/SC (C7h)/(C6h)	Contact/Coil retentive timer, 1-bit value	0000 to 2047
SN (C8h)	Current value retentive timer, 16-bit value, unsigned.	0000 to 2047
CS/CC (C4h)/(C3h)	Contact/Coil counter, 1-bit value	0000 to 1023
CN (C3h)	Current value counter, 16-bit value, unsigned.	0000 to 1023

SW (B5h)	Link special register, 16-bit value, unsigned.	0000 to 2047
SB (A1h)	Link special relay, 1-bit value	0000 to 2047
DX (A2h)	Direct input, 1-bit value	0000 to 8191
DY (A3h)	Direct output, 1-bit value	0000 to 8191

Our implementation of MELSEC protocol supports also the following extension tags:

Tag	**Meaning**	**Address range (decimal)**
DS (A8h)	Data register, 16-bit value, signed	000000 to 012287
DD (A8h)	Data register, 32-bit value, signed	000000 to 012286 Use regular D registers, address increased by 2
FL (A8h)	Data register, 32-bit float value	000000 to 012286 Use regular D registers, address increased by 2
FD (A8h)	Data register, 64-bit float value	000000 to 012284 Use regular D registers, address increased by 4
WS (B4h)	Link register, 32-bit signed value	0000 to 8191 Use regular W registers, address increased by 2

All custom tags consist of multiple regular 16-bit tags in the *least to the most significant* word order.

Connection settings

IP Address – IP address of PLC.

Port – port for the TCP connection to the PLC, please note that all *Mitsubishi* PLCs support only one connection / port. For multiple connections to the same PLC set a different port for each *mySCADA* device.

MultiBatch Optimised – enable this function if the PLC supports "Multiple block batch read/write" functions (codes 0406/1406). Enabling this option for a PLC that does not support such functions will cause a communication error.

CPUTimer – set this parameter if you expect an extensive communication with the PLC or communicate with a heavily loaded PLC to increase the response preparation time on the PLC side.

Optimisation Window – this setting influences how "aggressive" the optimization of your request will be. If it is set to 0 each tag will be requested in a separate packet (slow, but helps to find errors in the syntax). With default value of 1 all "adjusted" tags (D0000 and D0001, for example) will be requested in one packet. Set to 2 allows optimization to skip one tag address to the count tags as "adjusted" (for example D0000 and D0002 will be requested in one packet with such setting). 3 allows to skip 2 addresses etc.. Maximal value for this setting is 20. Please note that if the error returns in response to request, all tags included in this request will not be read. Use this feature carefully.

The following block of settings is required only for a hierarchy access within the *MELSEC* network of multiple PLCs - for a direct (local) communication leave the settings at their default values.

- **PC Number –** sets number of requesting station
- **Network number –** sets the network number to send the request to
- **Request dest.Module No. –** sets the number of requested modules for multi-CPU PLC connection
- **Request dest.Module I/O -** sets the number of requested modules for connecting via multi-drop

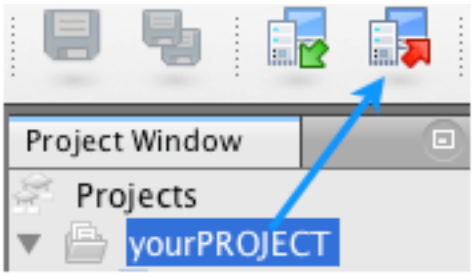
24. Download/Upload from/to Device

24.1 Download to Device

When you have finished designing and setting your project you are ready to download it to your operating device.

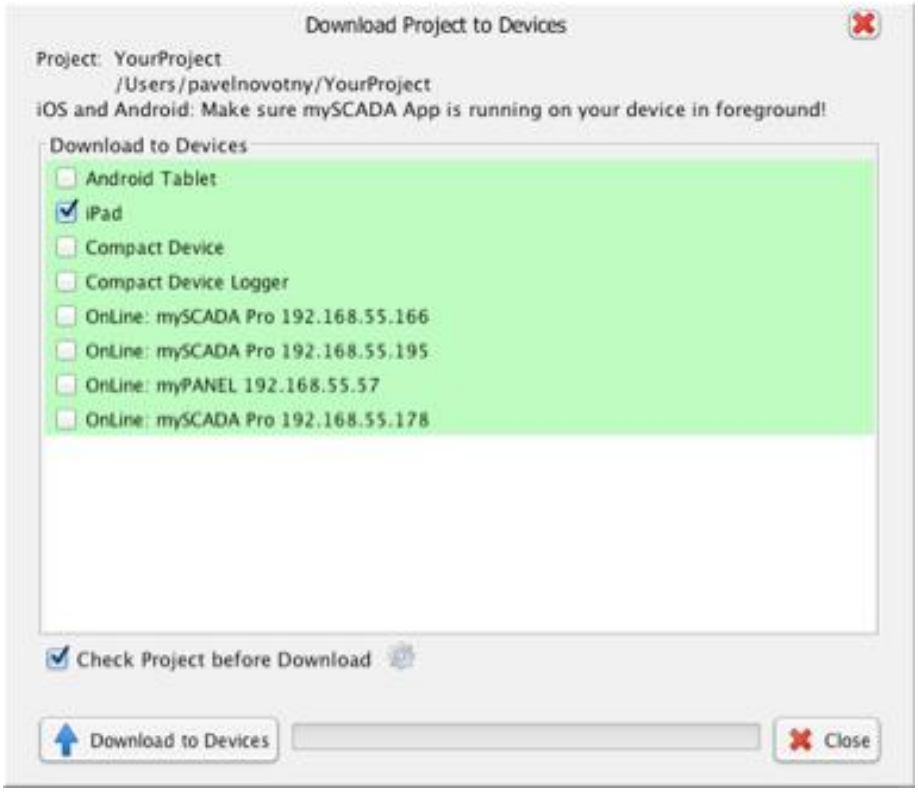
1) Select the project you wish to download to the device from the ***Projects*** folder in the *Project Window* - this will render the **Upload/Download** icons in the toolbar. 

2) Click on the **Download to Devices** button to load the selected project to the device. 

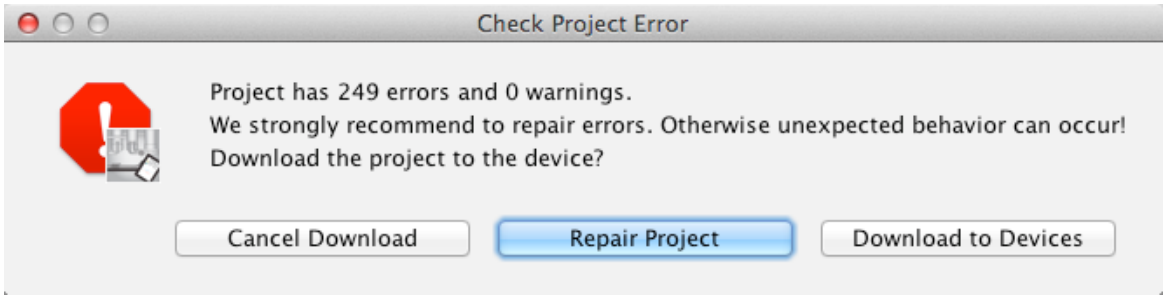


You can also right-click on the project from the list and select *Download* from the menu.

3) In the following dialog window you will see the list of all defined devices and all available **On-line** devices you can load your project onto (except *mySCADA Box* and *myLOGGER* - these need a serial number for communication).



4) Before downloading tick off the box **Check Project before Download** -> if there are any errors found the following dialog will show up:



Note: You can download your project to multiple devices at once. To do that, check multiple devices.

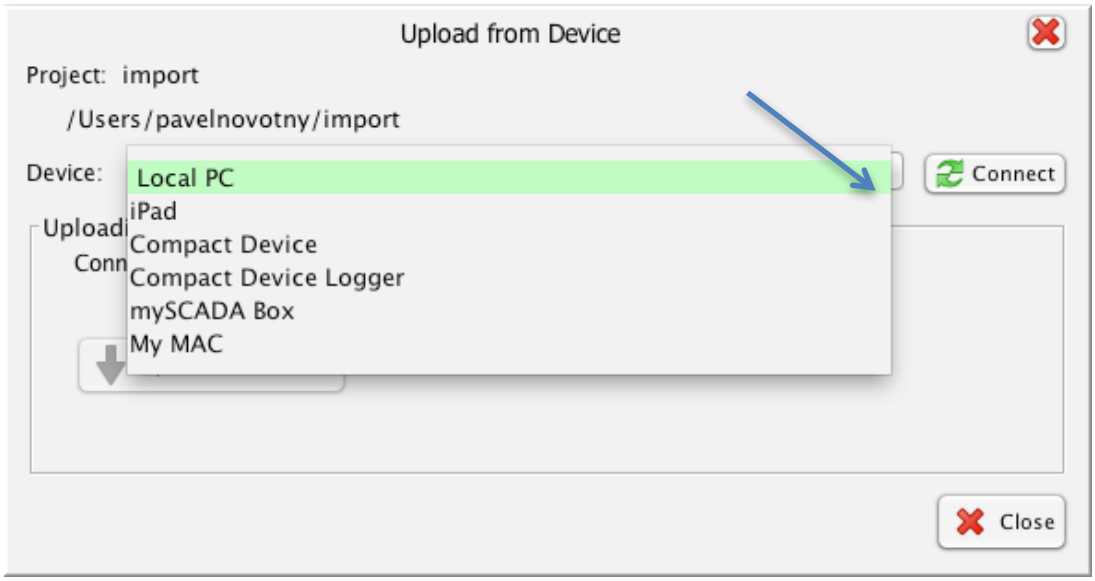
Warning: when loading a new project to your device **all existing projects loaded on the device will be overwritten!** Thus, back-up all important project information from the device before loading a new project to it.

24.2 Upload from Device

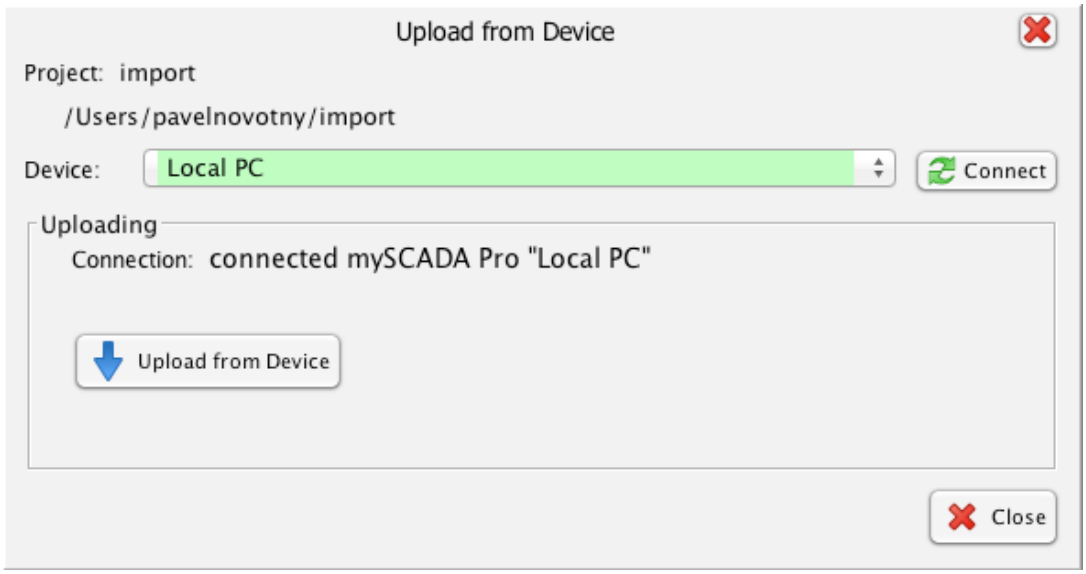
You can also upload a *mySCADA* project from your device to *myPROJECT Designer*.

Select one of the existing projects:

1) Click on the **Upload from Device** button  or right-click on the ***Projects*** folder in the *Project Window* and select *Upload from Device* from the context menu**.** The following dialog window will show up:



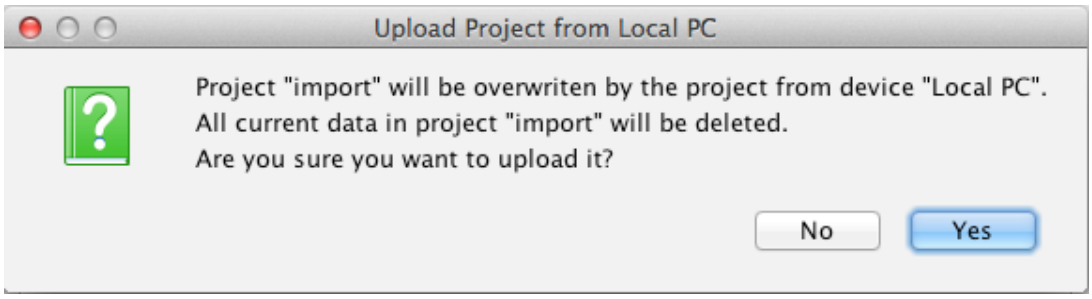
2) Select the device you want to upload the project from and click on the **Connect** button. If the connection is successful the following message will be displayed:



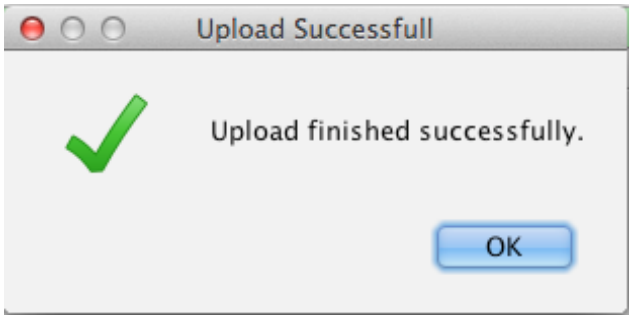
3) Click on the **Upload from Device** button 

The warning dialog notifying a possible project data loss will appear.

Warning: **All data saved in the project folder will be lost!**



4) After successful project uploading the confirming dialog will be shown.



5) Now you can start working with your created project!