

User documentation

FLEXGUI INDUSTRIAL 4.0

PPM AS |

VERSION 3486

formázott: Tabulátorok: 15,75 cm, Jobbra zárt + Nincs
15,5 cm

Contents

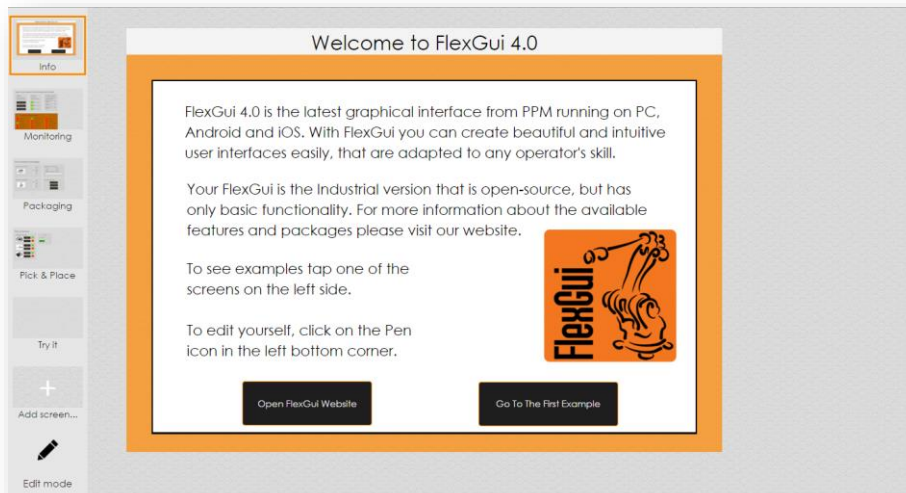
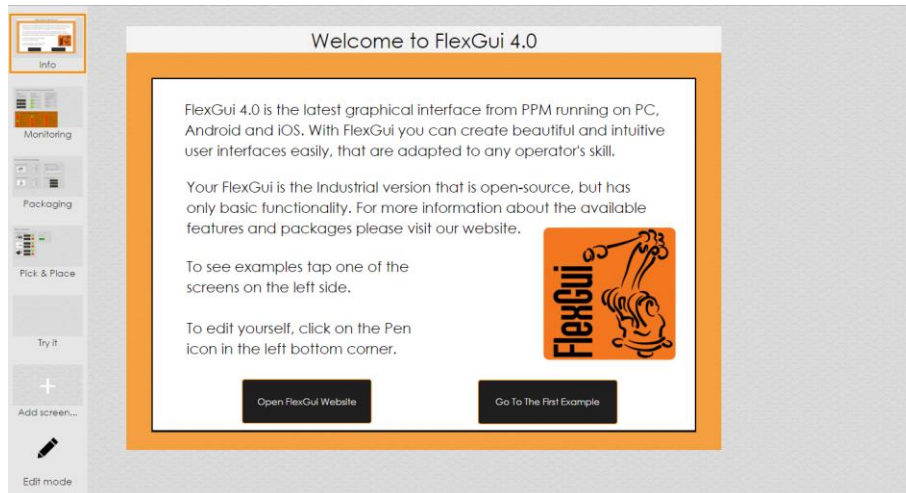
1	BeltTechnology introduction	4
1.1	Screen belt	6
1.1.1	Editing screen properties	7
1.2	Edit belt	8
1.2.1	Elements of Edit belt	8
1.3	Delete belt	8
1.4	Properties belt	8
1.4.1	Image Explorer Window	9
1.4.2	Saving Changes	9
1.4.3	Arranging Fidgets	10
1.5	Fidget belt	10
1.5.1	Fidgets	10
2	Modal windows	12
2.1	Help	12
2.2	Settings	13
2.2.1	General	13
2.2.2	Init script	14
2.2.3	Nodes	16
2.2.4	Connection settings	18
2.2.5	Language	20
2.2.6	Project	22
2.2.7	Diagnostics	24
2.2.8	Backup	26
2.3	Properties	27
2.4	Popup messages	28
2.5	Add screen	28
3	Moving Fidgets	28
4	Programming FlexGui	29
4.1	JavaScript function reference	29
4.1.1	#autoInit	29
4.1.2	#setScreenByName	29
4.1.3	#setScreenByIndex	29
4.1.4	#message	30
4.1.5	@friendlyName	30
4.2	JavaScript basics	30

4.2.1	Statements	30
4.2.2	Comments	30
4.2.3	Variables	31
4.2.4	Operators	31
4.2.5	Conditional statements	32
4.2.6	Loops	32
5	ROS-I connection	33
5.1	Overview	33
5.2	Publishing a Topic	35
1	BeltTechnology introduction	32
1.1	Screen belt	43
1.1.1	Editing screen properties	43
1.2	Edit belt	54
1.2.1	Elements of Edit belt	54
1.3	Delete belt	54
1.4	Properties belt	54
1.4.1	Image Explorer Window	65
1.4.2	Saving Changes	76
1.4.3	Arranging Fidgets	76
1.5	Fidget belt	76
1.5.1	Fidgets	76
2	Modal windows	98
2.1	Help	98
2.2	Settings	98
2.2.1	General	109
2.2.2	Init script	109
2.2.3	Nodes	119
2.2.4	Connection settings	1110
2.2.5	Language	1211
2.2.6	Project	1211
2.3	Properties	1311
2.4	Popup messages	1312
2.5	Add screen	1412
3	Moving Fidgets	1413
4	Programming FlexGui	1413
4.1	JavaScript function reference	1413

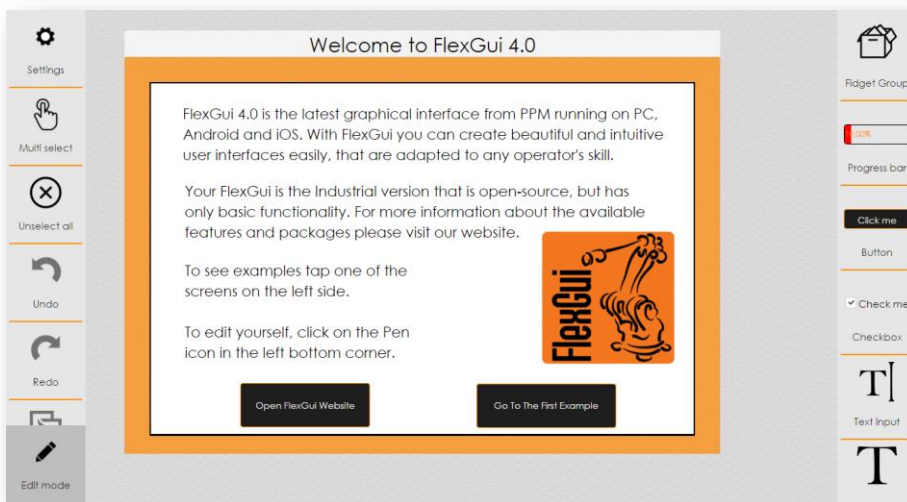
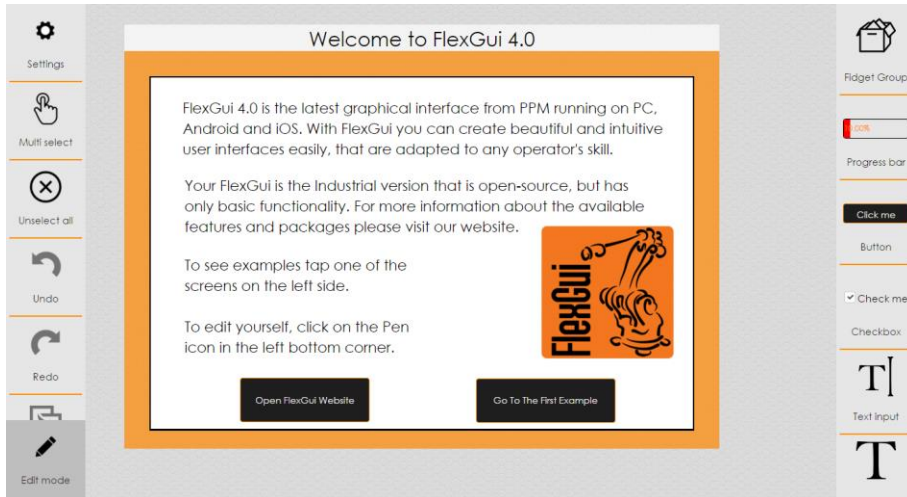
4.1.1	#autoInit	1413
4.1.2	#setScreenByName	1513
4.1.3	#setScreenByIndex	1513
4.1.4	#message	1514
4.1.5	@friendlyName	1514
4.2	JavaScript basics	1514
4.2.1	Statements	1514
4.2.2	Comments	1614
4.2.3	Variables	1615
4.2.4	Operators	1615
4.2.5	Conditional statements	1716
4.2.6	Loops	1816
5	ROS I connection	1817
5.1	Overview	1817
5.2	Publishing a Topic	1918

1 BeltTechnology introduction

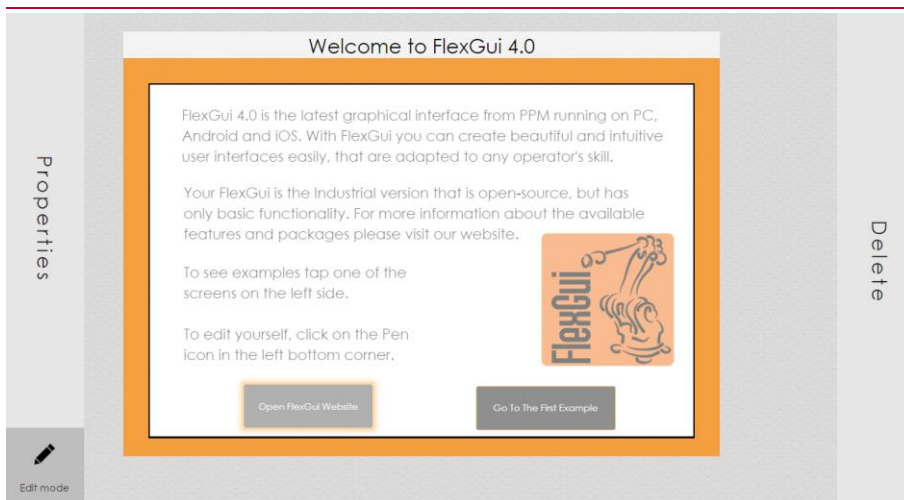
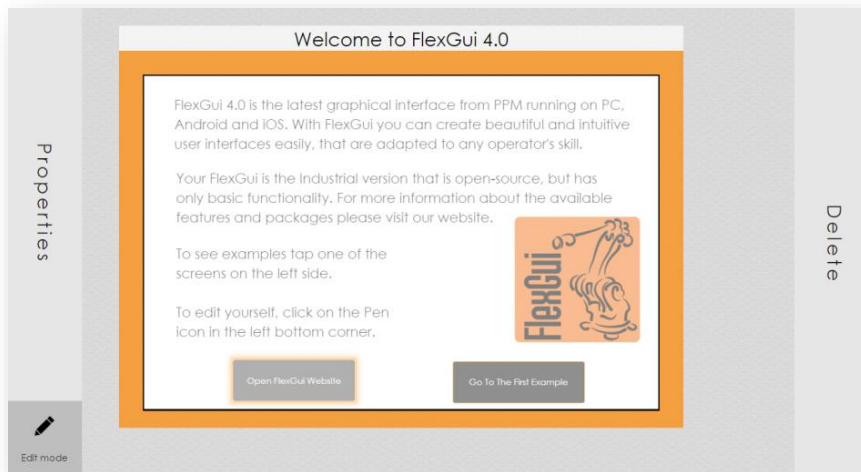
FlexGui contains 5 different belts for touch, drag&drop optimized user experience.



1 The Screen belt to access the screens in the project. A screen can contain many Fidgets, serve a purpose.



2 The Edit and Fidget belt that appear when the edit mode is turned on using the pencil icon in the left bottom corner. The edit belt contains useful editing fetures, while the Fidget belt the usable Fidgets that the user can drag and drop.



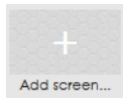
3 The Properties and Delete belt that appear when the user drags a Fidget in edit mode. They serve as goals of dropping to edit or remove a Fidget.

Let's see these items in more details.

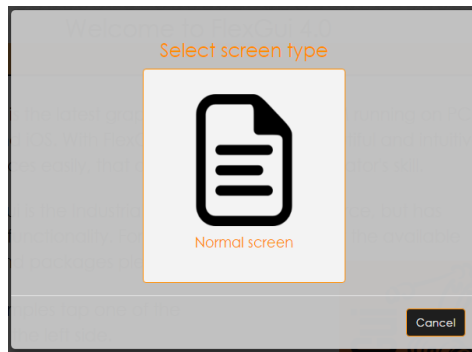
1.1 Screen belt

Screen belt is on the left side of the screen if the user is not in Edit Mode.

The Screen belt contains all of the available screens in the project. The logged-in users can add new screens by clicking the Add screen button.



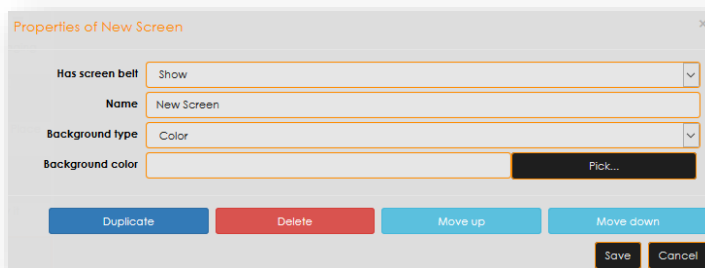
and selecting the type of the screen on the popup window:



formázott: Középre zárt

1.1.1 Editing screen properties

After adding the new screen, the Screen properties window will be shown. After adding a screen, long pressing the selected screen it will open the Screen properties window again.



Screens have name and background color properties and the position on the screen belt. [The screen belt can be switched off to all screens, so the screen has more space, or the integrator can decide to limit the user's available screens to a selected set.](#)

Both properties can be set up here and it is also possible to delete [and move](#) the selected screen.

1.2 Edit belt

The Edit belt is on the left side of the screen in **Edit Mode**.

1.2.1 Elements of Edit belt

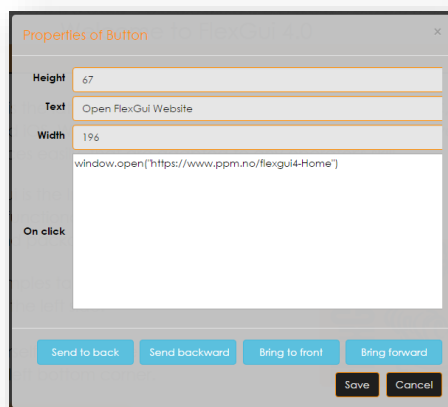
- **Settings:** opens the settings window
- **Multi select:** enables multiple fidget selection. ***Please note**, that the Properties belt is disabled while moving more than one Fidgets.*
- **Delete:** [Deletes all selected Fidgets](#)
- **Unselect all:** removes the current selection
- **Undo:** undo last action
- **Redo:** redo last action
- **Copy:** copies selected Fidgets to the clipboard
- **Cut:** copies selected Fidgets to the clipboard and removes them from the screen
- **Paste:** pastes Fidgets from the clipboard (only available if one or more items are on the clipboard)

1.3 Delete belt

The Delete belt is on the right side of the screen while dragging one or more Fidgets in **Edit Mode**. The user can drop the Fidgets on this belt to remove them from the screen.

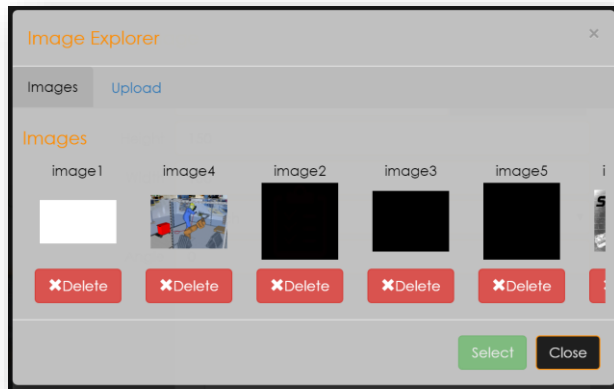
1.4 Properties belt

Dragging Fidgets to Properties belt opens the property window

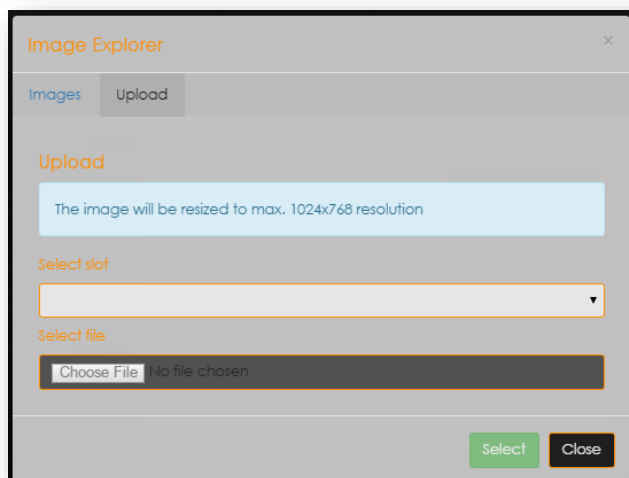


The elements of the property windows depends on the type of the Fidget. You can read more about the available properties in the next chapter. The type of a property editor field depends on the type of the property (e.g. for images we can open an Image Explorer window and add/delete images on the backend and use as a source for a Fidget).

1.4.1 Image Explorer Window



On the picture above you can see the currently available pictures stored on the backend [in ROSparams](#). It is possible to delete them or overwrite them by uploading.



1.4.1.1 Upload steps:

1. **Select slot:** on the backend there are [10-20](#) available slots for images, the user has to select which slot should be used.
2. **Select file:** the user has to select an image, which will be resized and uploaded to the server. After the upload the image will be automatically selected on the Images tab.

1.4.2 Saving Changes

Save: save the current setup and close the property window

Cancel: restore the previous setup and close the property window

1.4.3 Arranging Fidgets

Fidgets can be arranged by these four buttons:

Send to back: go to the lowest layer of the screen

Send backward: go one layer down

Bring to front: go to the topmost layer of the screen

Bring forward: go one layer up

1.5 Fidget belt


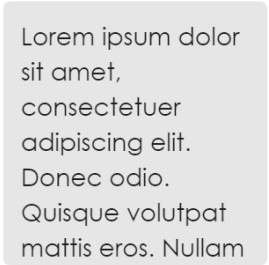



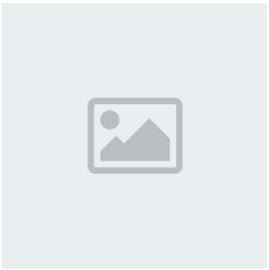
The fidget belt is on the right side of the screen in Edit Mode. The user can add new Fidgets to the current screen with drag & drop. To grab a Fidget from the belt, long tap / click has to be used. Releasing the tap / click adds the Fidget to the screen.



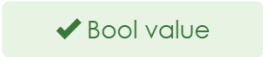

1.5.1 Fidgets

1.5.1 Every fidget has 4 default properties: width, height, top and left. With this property the user can set the size and the position of the fidget. The position of a fidget is calculated from the top-left corner of it's parent: a fidget group or the screen.

Name	Fidget	Properties
Fidget group		Size Color Opacity Border color Border width On click
Button		Size Text On click Font Font size Font color
Text		Font Font size Color Text Text align Size Font size Color Text On click

táblázatot formázott


Text input		Size Text On click
Scrollable text		Size Text On click TextAlign Font Font Size Color
Progress		Size Value Color Font color On click
Radio button		Size Options Font Font size Font color Value Options Font color Value On click
Checkbox		Text Value Font Font size Font color Size Text Value Font color On click
Image		Size FillScale Image On click

Gauge		Size Value Color Minimum Maximum Angle arc Angle offset
Indicator lamp		Size Blinking Blink period On/off color Text Font color On click
Bool		Size Value Text Font Font size Font color On click On click
Slider		Size Minimum Maximum Value Step Precision On click

2 Modal windows

2.1 Help

On-screen contextual help is available and can be enabled / disabled in the settings window.

The user can get instant help by pressing one of the -icons anywhere in FlexGui to get contextual help.



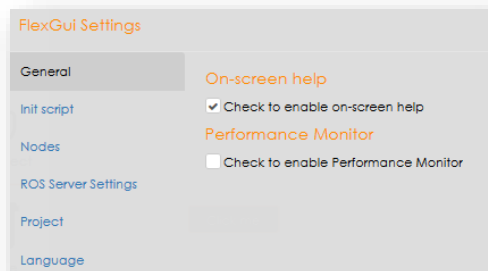
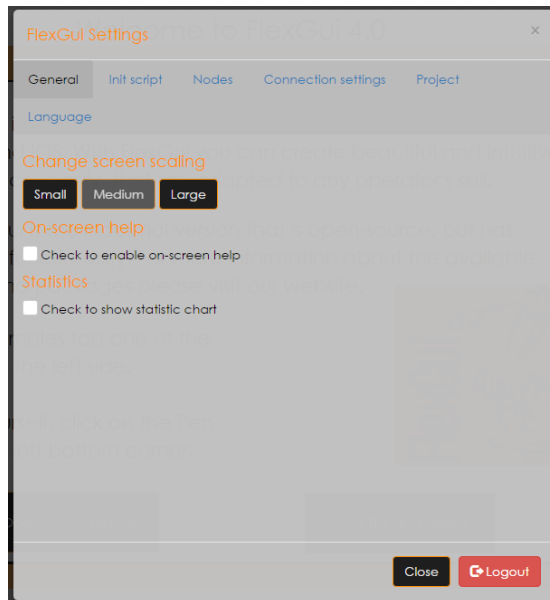
2.2 Settings

To open the settings window the user has to enable **Edit Mode** and press the settings on the top of the Edit Belt.

2.2.1 General

General settings of FlexGui:

- ~~Screen scaling (zoom on mobile devices),~~
- ~~On-screen help on and off,~~
- ~~Performance Monitor on and off~~
- ~~Grid size for edit mode.~~



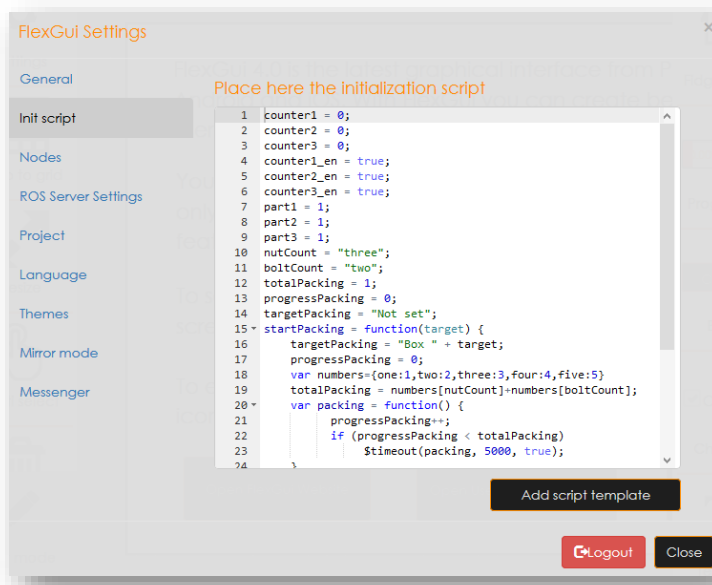
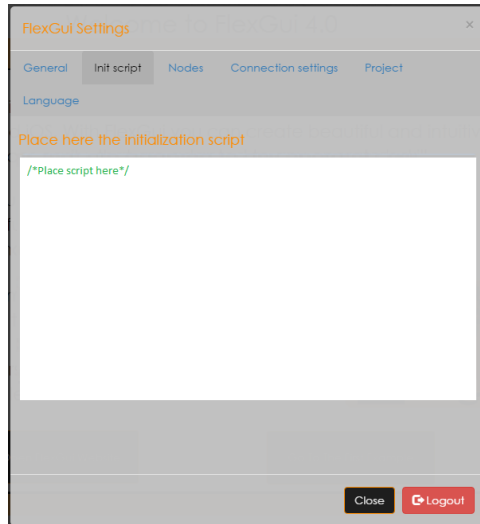
2.2.2 Init script

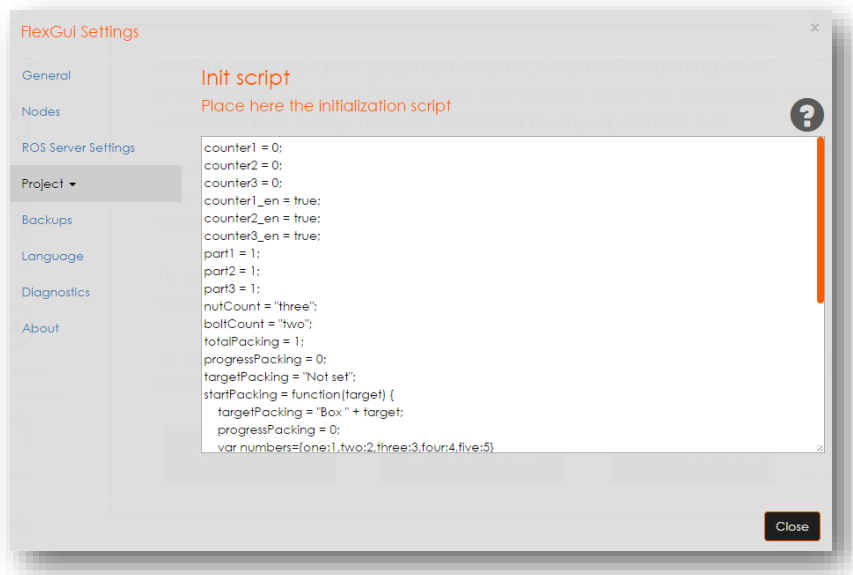
The init script editor is located under the project menu in Settings window.

The init script will run when loading a project. It's used to initialize custom variables, functions. The init script is synchronized between devices, the value of the local variables after initializing them are not. If you want to synchronize values between two clients, please use server side variables in ROS or a device.

Init script

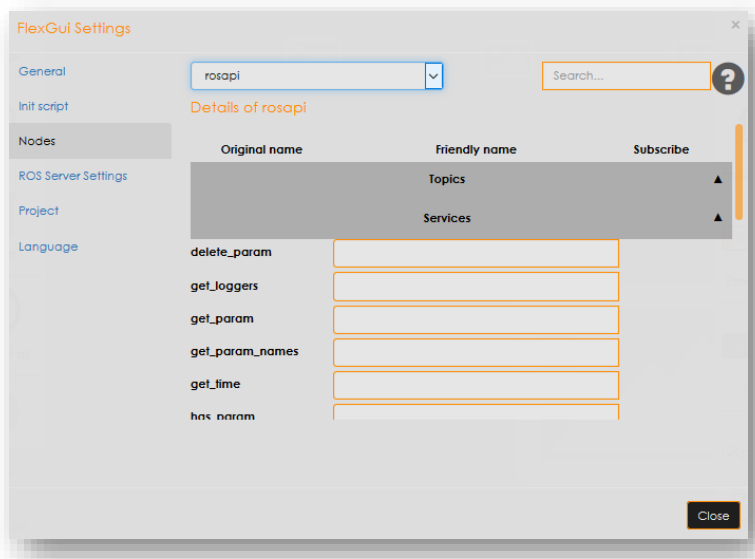
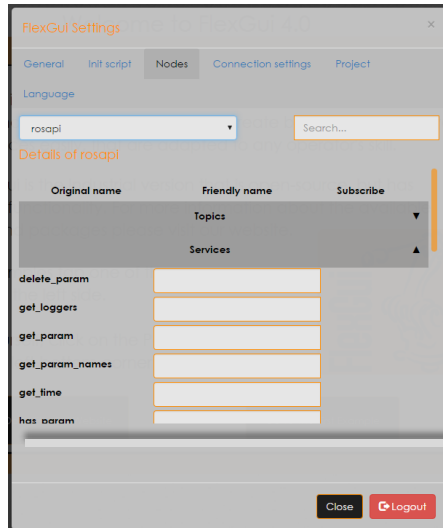
The init script will run when loading a project. It's used to initialize custom variables, functions. The init script is synchronized between devices, the value of the local variables after initializing them are not. If you want to synchronize values between two clients, please use server side variables in ROS or a device.

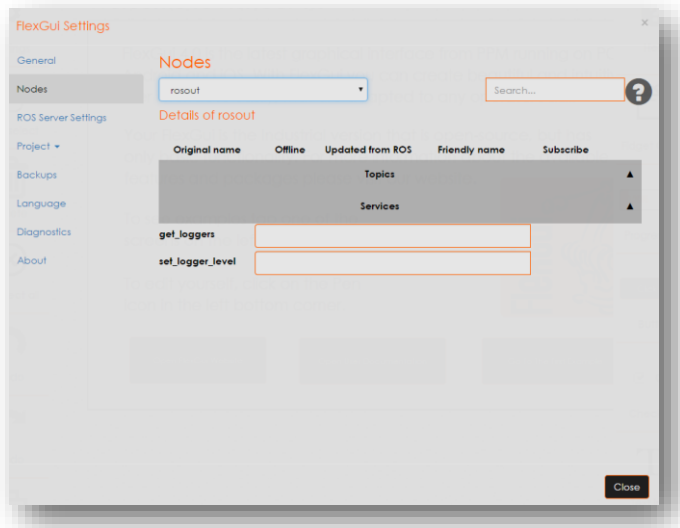




2.2.3 Nodes

In this tab the user can see the available topics and services in ROS, so it is not necessary to open ROS or node documentation to see what is available on each item.





Both topics and services can be renamed for easier access, then the @friendlyName syntax is to be used instead of the full path. If a topic is subscribed, all changes will be received using push method, and all user interface elements will update automatically that use this value.

2.2.4 Connection settings

In the connection settings tab the user can set up the address of the backend server or enable demo mode for local usage. The backend server should run roscore and rosbridge as well. [In case you want secure communication, you need to enable WebSocket Security. This is required for the online trial as well.](#)

FlexGui Settings

General

Init script

Nodes

Connection settings

Messenger

Language

Project

IP

192.168.0.175

Port

9090

Demo mode

☐ Check to enable demo mode

Save

CloseLogout

FlexGui Settings

General

Init script

Nodes

ROS Server Settings

Project

Language

Set the location of the ROS server to use, so you can access the nodes on it. If you want to use FlexGui offline, check the box below.

IP

192.168.1.69

Port

9090

Secure mode

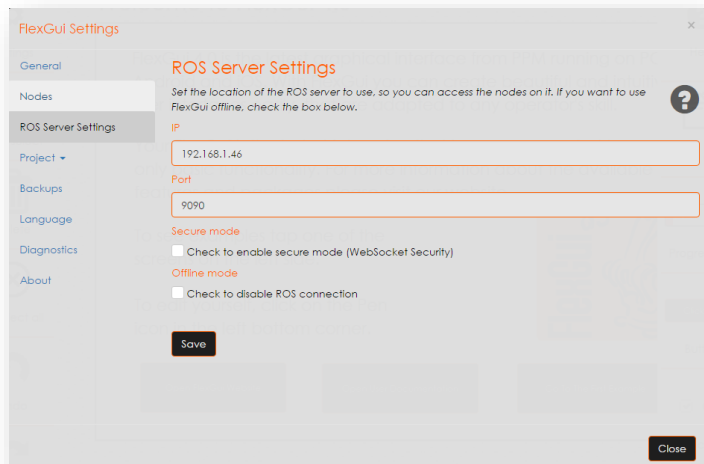
☐ Check to enable secure mode (WebSocket Security)

Demo mode

☐ Check to enable demo mode

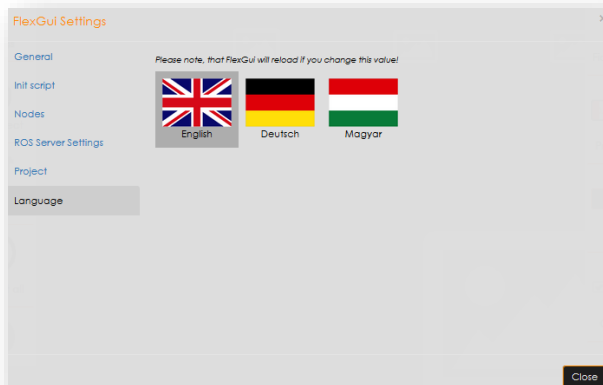
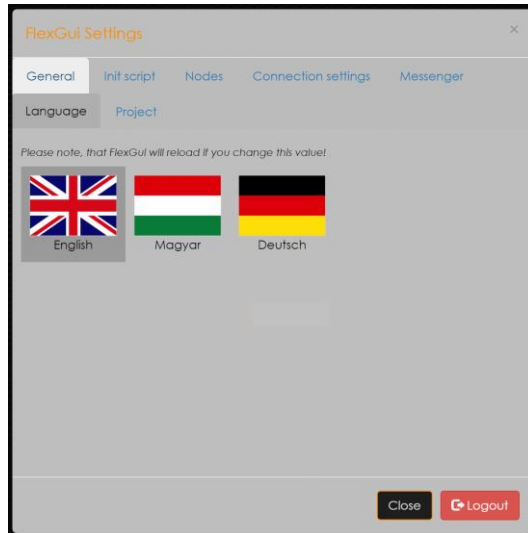
Save

Close



2.2.5 Language

FlexGui supports multiple languages. The user can set up the currently used language in the Language tab.

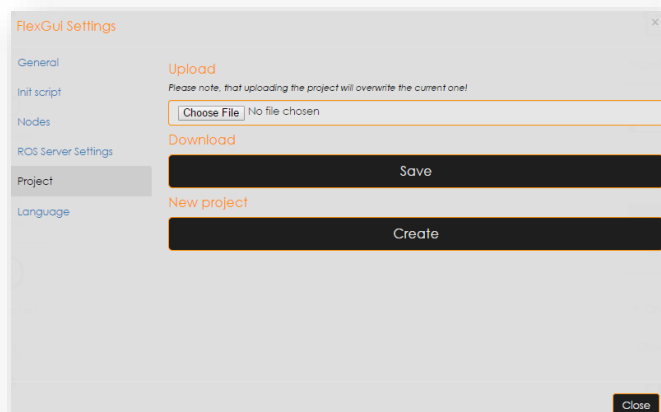
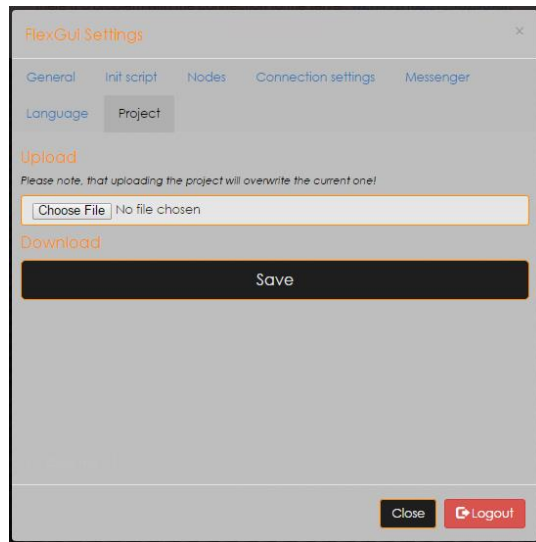


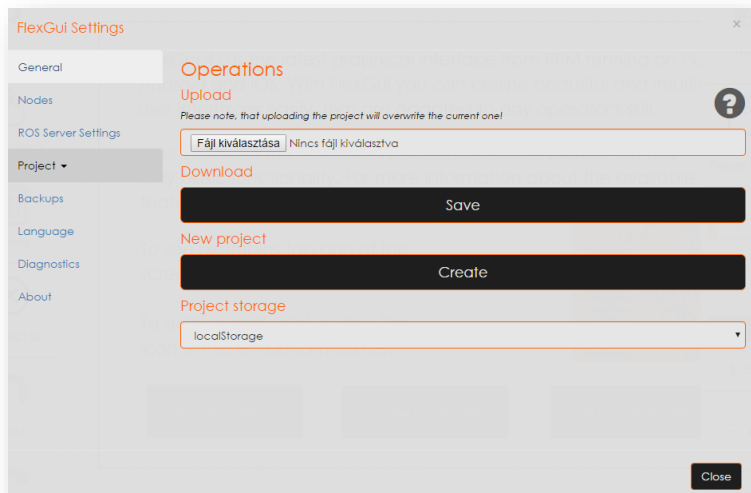


2.2.6 Project

To upload and download projects, the user can use the operations tab under Project menu. This makes it easy to backup or copy projects between devices, useful for debugging too. Demo mode also has this feature, so you can simply assemble a project in demo mode, then copy it to the working system or vica versa. If ROS is connected the user can decide where to store the project. If more user is on the same ROS server, but want to use different projects localStorage has to be chosen. With this, the project stays on the local computer and not synchronised to the ROS server. If more user want to use the same project, rosparam has to be chosen. With rosparam storage mode, the project is synchronised (manually or automatically) to the ROS server as a ROS parameter and each user will get the same project. Only one user can edit the project with rosparam storage mode.

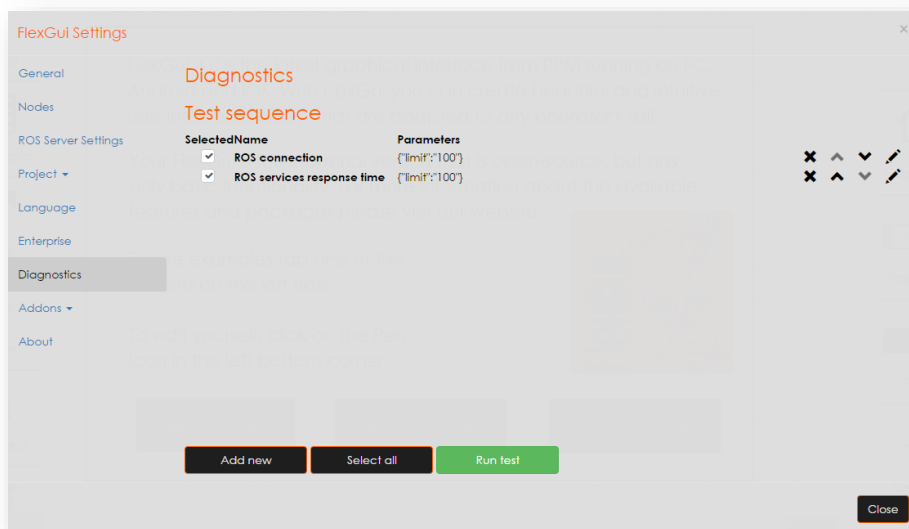
To upload and download projects, the user can use the project tab. This makes it easy to backup or copy projects between devices, useful for debugging too. Demo mode also has this feature, so you can simply assemble a project in demo mode, then copy it to the working system or vica versa.





2.2.7 Diagnostics

With FlexGui 4.0 Diagnostics the users can create test sequences to indentify to problems runtime. Each test starts individually after each other and they are completely separated. The result of one is not effecting of the others, thus with a good sequence it is easy to find the problems in the nodes, connections or other parts of the system.



It is also possible to run only one specific part of the sequence, by removing / adding a test based on the selected checkbox on the left. Select all button will check all of the checkbox, thus all of the tests will be selected to run.

Pressing the Add new button opens a dialog window.

Edit test

☐ Check this if you want to make an own script

Offline topics

Gets the offline topics in a selected node

Parameters

nodeName

The name of the tested node, e.g. FlexGuiNode

Close Save

There are two types of test:

Predefined tests can be selected from the drop down list and each test has some predefined and required paramters (e.g. Topic update rate's parameters are the nodeName, the timeout and the limit). Each parameter has a description which helps the user to fill in the form.

The time based tests has a limit parameter, which supposed to target the maximum response time of a selected call.

User defined tests can be written by checkin the own script checkbox on the top.

New test

☒ Check this if you want to make an own script

Name

myDiagnostics

Script

```
console.log('Hello');
```

Close Save

formázott: Középre zárt

Each user defined test has to fulfill some criterias, to guarantee the proper operation. All of them can be found in the [Help](#).

Please note: use the following variables for optimal result

- **test.result:** this will be shown on the UI under the name of your script
- **srv.testFinished();** Call this function when your script finished to be able to jump to the next test.
- **\$rootScope.\$apply();** call this function, when the script ends if you have any async calls (timeout, interval or e.g. waiting for a ROS event to end)

Pressing the Run button will run the selected tests. The result will be visible in the Result modal:

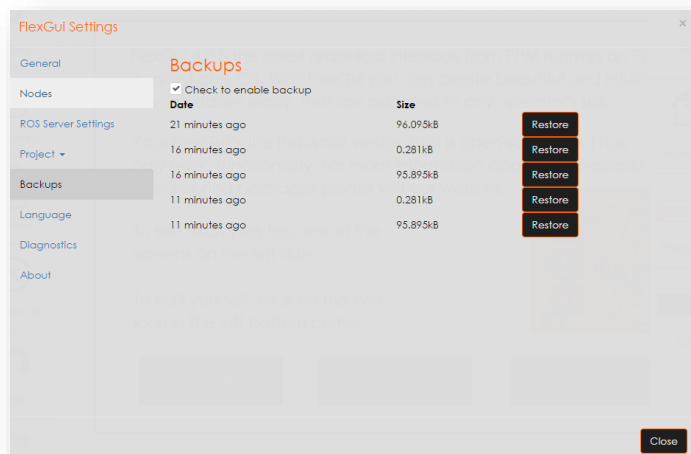


The result clearly shows the outcome of each tests. It is not possible to close the result window until all of the tests ended, unless the user presses the Cancel button which stops the tests. **Please note:** press the cancel button won't stop the current asynchronous call, but the next test case won't start.

2.2.8 Backup

FlexGui 4.0 saves the current project to localStorage in every minute automatically to backup the current work. In case of any problem, the use can go to Settings / Backup and choose from the last 5 backup.

formázott: Címsor 3

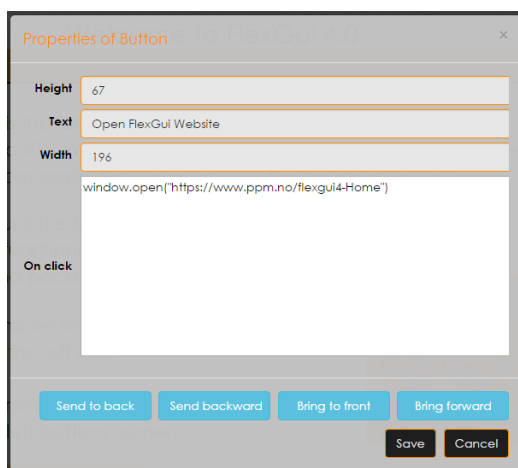


formázott: Középre zárt

formázott: magyar

2.3 Properties

To edit properties of screens or Fidgets, the user can use the Property window. The content of the property window depends on the edited item. The example shows the properties of a button.



Arranging a Fidget [\(in z-order\)](#):



Ordering a Screen [\(moving it on the screen belt up and down\)](#):

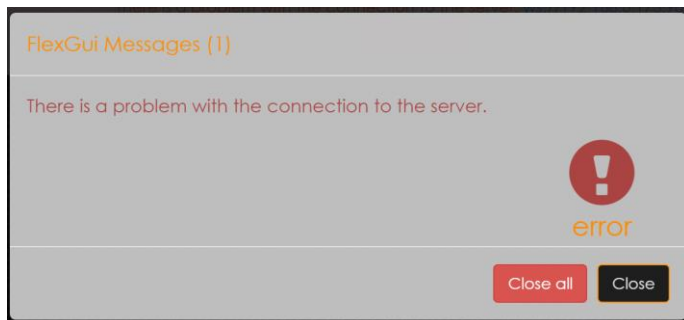


2.4 Popup messages

There are different types of popups in FlexGui:

- Info
- Warning
- Error

It is used e.g. if we cannot connect to the server. The user can show such screens also programatically using the #message call: `#message<("some text", "#infoMessage")`



2.5 Add screen

Defined earlier.

3.0 Login

Defined earlier.

5.3 Moving Fidgets

FlexGui uses drag & drop technology. This means, to move a Fidget the user has to grab it:

- From the fidget belt with a long press
- From the screen with a drag or a click.

While the mouse is pressed or the screen is touched, the user can drag Fidgets on the screen. After the drag is released:

- opens the property window, if the Fidget is over the property belt (the position won't update)
- deletes the Fidget, if the cursor is over the delete belt
- otherwise the Fidget will move to the current position on the screen

6.4 Programming FlexGui

JavaScript is used in FlexGui to specify behaviour on specific events like event handlers or button presses. The following section will explain the basic supported language elements through examples for easier understanding of the system.

6.14.1 JavaScript function reference

FlexGui 4.0 contains some predefined functions, to help the users' work. All these functions start with #:

6.1.14.1.1 `#autoInit`

This function initializes a variable with a value, if it is not initialized before.

6.1.1.14.1.1.1 *Parameters*

name: string

value: object

6.1.1.24.1.1.2 *Example:*

`#autoInit("int0", 42);` //add int0 variable with 42, if it is not yet defined

4.1.2 `#setScreenByName`

This function changes the current screen to the given one using it's name.

4.1.2.1 *Parameters*

screen: string

4.1.2.2 *Example*

`#setScreenByName("New Screen")`

4.1.3 `#setScreenByIndex`

This function changes the current screen to the given one using it's name.

4.1.3.1 *Parameters*

screen: string

4.1.3.2 *Example*

`#setScreenByIndex(1)` //sets the screen to the 2nd on the screen belt, because the index starts from 0.

6.1.2 `#setScreen`

~~This function changes the current screen to the given one.~~

6.1.4.0 *Parameters*

screen: object

6.1.6.0 *Example*

`#setScreen(projectService.screens[1]);` //set the current screen to the 2nd screen in the project

[6.1.84.1.4](#) `#message`

This function pops up a message with the given string. It is possible to define the type of the message:

- `#infoMessage` pops up an info message
- `#errorMessage` pops up an error message
- `#warningMessage` pops up a warning message

[6.1.8.14.1.4.1](#) `Paramteres`

text: string

[6.1.8.24.1.4.2](#) `Example`

`#message("Hello world");` //pops up "Hello world" az an info message

`#message("Hello world", #errorMessage);` //pops up an errorMessage.

[6.1.94.1.5](#) `#@friendlyName`

With this notation the user can access the ROS topics/[services](#) using a friendly name.

[6.1.9.14.1.5.1](#) `Example`

`#friendly["myVariable"].value.data = 12;` //set myVariable's value to 12

`#friendly.myVariable.value.data = 12;` //same result as the first

`@myTopicVariable.value.data = 12;` // [set myTopic's value to 12, publishes the new valuesame result as the first](#)

Note: a variable is a class, that has several member variables as type, path, value. We use the value member, that is also a class. This, according to the documentation (http://docs.ros.org/api/std_msgs/html/msg/Int32.html) has a data member storing the value we need. Complex data types might have more member variables (e.g. http://docs.ros.org/api/geometry_msgs/html/msg/Pose.html).

[6.24.2](#) JavaScript basics

[6.2.14.2.1](#) `Statements`

JavaScript statements are separated by a semicolon (;). States are executed in the sequence they are written. Blocks in JavaScript are surrounded by curly brackets ({}). You can use blocks if you want to group statements.

[6.2.1.1.14.2.1.1.1](#) `Example`

```
a = 5;
b = 6;
c = 12;
for (a = 5; a < 10; a++) //Explained later
{
    c = a + 2;
}
```

Please Note: JavaScript is case sensitive, variable **A** and **a** are different variables, while white space does not matter: **a = 5;** is the same as **a=5;**.

[6.2.24.2.2](#) `Comments`

Comments can be placed in the code, just surround any text with `/*` and `*/`. If you write `//` anywhere in the code, the following characters are considered to be comments until the end of the line.

formázott: magyar

formázott: magyar

[6.2.2.1.14.2.2.1.1](#) Example

```
a = 5; //This is a variable  
*****  
This is a comment that is  
going through multiple lines  
*****  
b = 6;
```

[6.2.34.2.3](#) Variables

Variables are present in JavaScript for storing data. Variable names can be letters, numbers and underscores (`_`), but they can't start with a number. The declaration of a creates the variable. It is possible to assign a value right at the declaration or after. The value of a variable can be modified infinite times. If you put quotes (simple or double) around the value of a variable it will be a string, if numbers are used only, it will be a number. If letters are used without quotes, it will be interpreted as a variable, throwing an error if it doesn't exist. Types of variables are dynamic, assigning text to a number variable will make it a string.

[6.2.3.1.14.2.3.1.1](#) Example

```
//Declaring a single number variable:  
var a;  
//Declaration of several variables with or without value  
var a = 5, b = "something", c, d = false;  
c = a + 5;  
a = b;
```

Please Note: redeclaration of a variable is possible, if there is no new value, the variable will remain exactly the same as before.

[6.2.44.2.4](#) Operators

Let's go through operators using an example. The comments next to statements show their results.

[6.2.4.1.14.2.4.1.1](#) Arithmetic and assignment operators

```
//Arithmetic operators:  
var a = 2, b = 3, c;  
c = a + b; //c will be 5 (addition)  
c = a - b; //c will be -1 (subtraction)  
c = a * b; //c will be 6 (multiplication)  
c = a / b; //c will be 0.666 (division)  
c = a % b; //c will be 2 (modulus)  
c++; //c will be 3 (incrementation)  
c--; // c will be 2 (decrementation)  
//Assignment operators:  
c = 5; //c will be 5  
c += 2; //c will be 7 (equals to c = c + 2)  
c -= 3; //c will be 4 (equals to c = c - 3)  
c *= 3; //c will be 12 (equals to c = c * 3)  
c /= 4; //c will be 3 (equals to c = c / 4)  
c %= 2; //c will be 1 (equals to c = c % 2)  
a = "Hello", b = "FlexGui ", c = 3, d;  
d = a + " " + b + c; //d will be "Hello FlexGui 3"
```

[6.2.4.1.24.2.4.1.2](#) Comparison, logical and conditional operators

```
//Comparison operators:  
a = 3, b;
```



```

b = a == 2; //b will be false (equal)
b = a === "3"; //b will be false (value and type equal too)
b = a != 2; //b will be true (not equal)
b = a !== "2"; //b will be true (not equal neither value or type)
b = a > 2; //b will be true (greater than)
b = a < 2; //b will be false (less than)
b = a >= 2; //b will be true (greater then or equal to)
b = a <= 2; //b will be false (less then or equal to)
//Logical operators:
a = true, b = false, c;
c = a && b; //c will be false (and)
c = a || b; //c will be true (or)
c = !a; //c will be false (not)
//Conditional operator:
//(condition) ? value1 : value2;
var a = 3 > 5 ? 1 : 2; //a will be 2

```

[6.2.54.2.5](#) Conditional statements

Conditional statements can be used to determine if a code should run or not depending on a logical statement.

[6.2.5.14.2.5.1.1](#) Example of conditional statement

```

if (4 > 6)
{
    var a = 5;
}
else
{
    var a = 6;
}

```

Because the result of $4 > 6$ is false the else part will be executed, a will be 6. It is possible to use the statement without the else branch.

Please Note: curly brackets are used to group more statements, but not obligatory for one (if $4 > 6$ a = 5; is ok)

[6.2.64.2.6](#) Loops

[6.2.6.14.2.6.1](#) The for loop

The for loop executes the code block while the middle statement is true. The first statement is executed before the loop, the last one is executed after each loop.

[6.2.6.14.2.6.1.1](#) Example of a for loop

```

for (var i = 0; i < 5; i++)
{
    //Statements
}

```

[6.2.6.24.2.6.2](#) The while loop

The while loop executes the code block while the statement is true.

[6.2.6.214.2.6.2.1](#) Example of a while loop

```

while (i < 5)
{

```

```
}  
  //Statements
```

[6.2.6.3.4.2.6.3](#) The do/while loop

The do/while loop executes the code block while the statement is true. The condition is checked after each run, so the body will be ran at least once.

[6.2.6.3.4.2.6.3.1](#) Example of do/while loop

```
do  
{  
  //Statements  
}  
while (i < 5)
```

75 ROS-I connection

[7.15.1](#) Overview

Supported devices

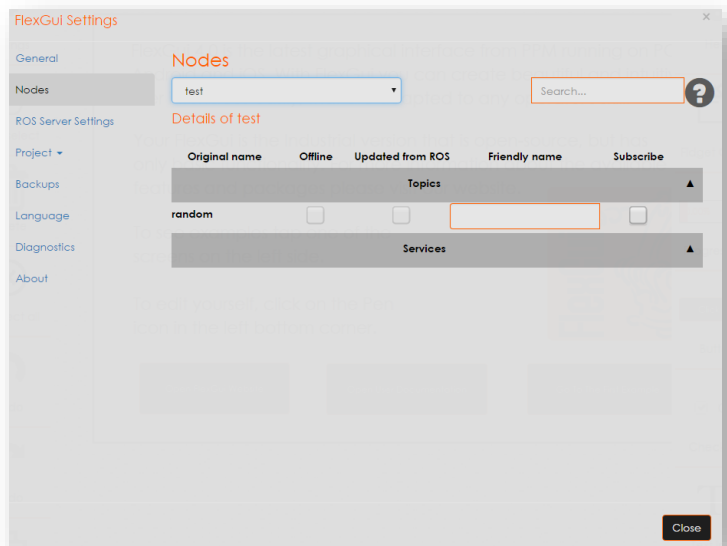
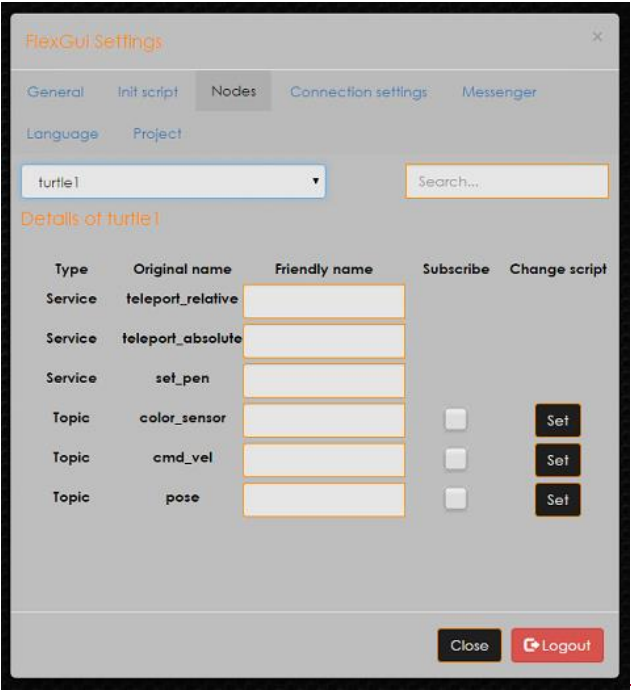
- NACHI robot connection,
- PLCs through modbus,
- KUKA implemented by Narvik
- ABB, Adept, Fanuc, Motoman, Universal Robot implemented by ROS

Additional features:

- Virtual environment simulation using Gazebo,
- One click variable access
- Custom names to variables => short and clear code
- Normal PC or RaspberryPi both supported as ROS server

When the correct connection settings of the rosbridge server are set in the Connection settings tab, the available ROS nodes are shown in the Nodes tab.

Services and Topics are shown when selecting a node. The original names can be renamed and then accessed in FlexGui. In order to receive data from a Topic the subscription checkmark has to be set.



7-25.2 Publishing a Topic

Reading a topic is done using the @topicName syntax. To publish values, first you need to create and (if it doesn't exist yet) advertise the topic. This can be done in the initScript or for debug purposes in any control that has onClick script.

```
// PLC register topic creation
registerTopic = new ROSLIB.Topic({
  name: '/plc1/register1',
  messageType: 'std_msgs/Int32',
  ros: variableService.ros
});
// Advertise the new topic
registerTopic.advertise();
```

Then, to publish values we need to use the topic we just created. This is preferably placed in an onClick script, but the initScript can have custom code like this as well.

```
//Publishing Int32 typed message
registerTopic.publish(new ROSLIB.Message({ data: 0 }));
```

Note: If you would publish another message type, you would need to use another syntax. Like a Pose has more members, a Point and a Quaternion.