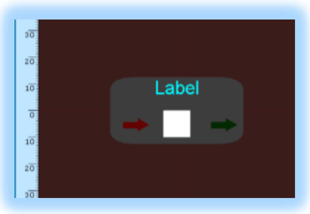
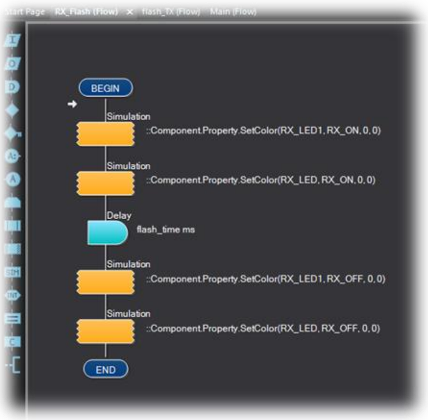
FLOWCODE 8

Component Creation

Build A Flowcode Component

Design & build your own Components



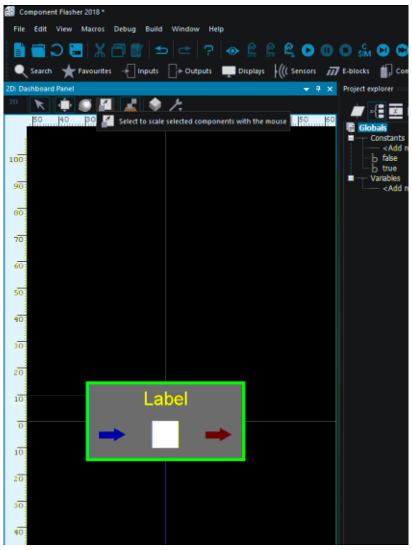


BEGIN
Simulation
::Component.Property.SetColor(RX_LED1, RX_ON, 0, 0)
Simulation
::Component.Property.SetColor(RX_LED, RX_ON, 0, 0)
Delay
flash_time ms
Simulation
::Component.Property.SetColor(RX_LED1, RX_OFF, 0, 0)
Simulation
::Component.Property.SetColor(RX_LED, RX_OFF, 0, 0)
END

Note – You should created two Floders:

1 – The working component project ***2 – Your Finished Components Library***

Discover how to build Sub-Components (advanced shapes) like this ***Flasher***, to use in your own component designs and how to give them a unique **GUID.**


Component Flasher 2018 *
File Edit View Macros Debug Build Window Help
Search Favourites Inputs Outputs Displays Sensors E-blocks
2D: Dashboard Panel
Project explorer
Select to scale selected components with the mouse
Globals
Constants
Variables
Label

GUID - globally unique identifier

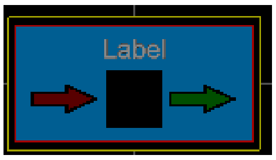
Every component is made up of various simple shapes, that you can put together in graphic sets (***see opposite***), to build your own flowcode 7/8

sub-components. These can be used to build: -

- Relays
- Solenoids
- Displays etc

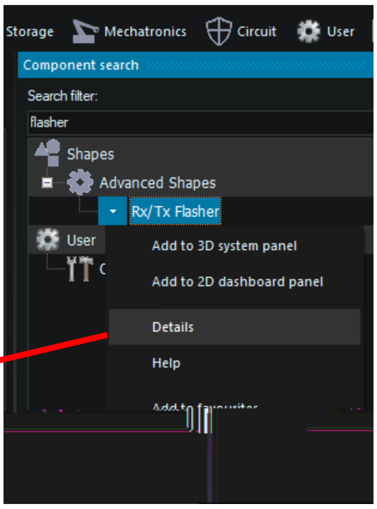
The list of possibilities is endless and entirely up to you, to build unique components to meet your own project or business needs.

Each unique graphic set/block requires an – GUID in order for flowcode API to recognise each finished sub_component or final component. There may be more than one sub_component used with its own GUID, in your project. However, when we have completed the assembly of those sub_components, the final component will have its very own GUID, that you can save in your own ***Component Library,*** that either FC7/8 can access in your projects.

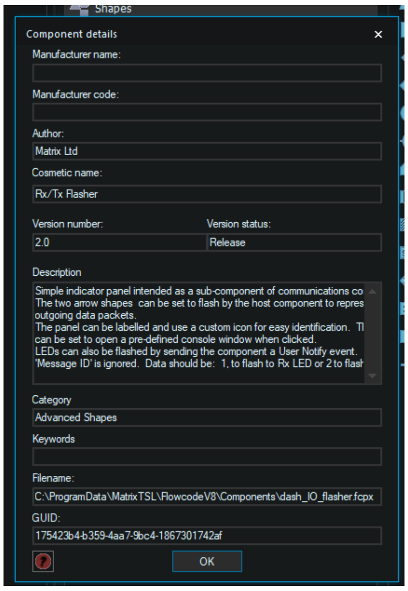


To view the information stored in each component such as this RX/TX Flasher (***a sub-component***), from the tool-bar, select the component of interest: -

- From the pull-down menu select **details**



This will give you the following:


Component details

Manufacturer name:

Manufacturer code:

Author:
Matrix Ltd

Cosmetic name:
Rx/Tx Flasher

Version number: 2.0
Version status: Release

Description
Simple indicator panel intended as a sub-component of communications co
The two arrow shapes can be set to flash by the host component to repres
outgoing data packets.
The panel can be labelled and use a custom icon for easy identification. Tl
can be set to open a pre-defined console window when clicked.
LEDs can also be flashed by sending the component a User Notify event.
'Message ID' is ignored. Data should be: 1, to flash to Rx LED or 2 to flash

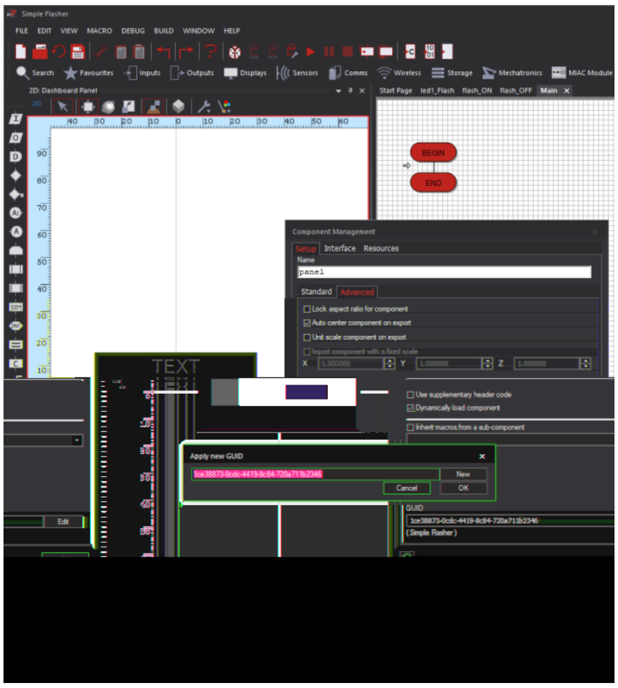
Category
Advanced Shapes

Keywords

Filename:
C:\ProgramData\MatrixTSL\FlowcodeV8\Components\dash_IO_flasher.fcpx

GUID:
175423b4-b359-4aa7-9bc4-1867301742af

All these graphics ***sub-components*** will give you endless possibilities to enhance your set of available components to be used in your projects and they will be fully compatible with Flowcode 7 & 8.


Simple Flasher
FILE EDIT VIEW MACRO DEBUG BUILD WINDOW HELP
Search Favourites Inputs Outputs Displays Sensors Comms Wireless Storage Mechatronics MIAC Module
2D: Dashboard Panel
Start Page led1_Flash flash_ON flash_OFF Main
BEGIN
END
TEXT
Apply new GUID
Edit

Component Management

Setup | Interface | Resources

Name: panel

Standard | Advanced

- ☐ Lock aspect ratio for component
- ☑ Auto center component on export
- ☐ Unit scale component on export
- ☐ Import component with a fixed scale
- X 1.000000 Y 1.000000 Z 1.000000
- ☐ Use supplementary header code
- ☑ Dynamically load component
- ☐ Inherit macros from a sub-component

Apply new GUID: 1ce38873-0cdc-4419-8c84-720a711b2346 [New] [Cancel] [OK]

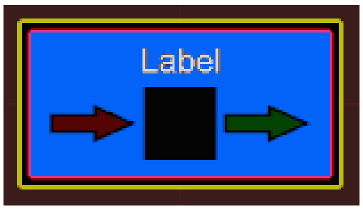
GUID: 1ce38873-0cdc-4419-8c84-720a711b2346 (Simple Flasher)

Getting Started

When you have a situations that requires a component to be built for use within your flowcode 7/8 graphical programming environment, that is not readily available in Flowcode 7/8 standard component package(s). This may be due to a specialist project you are working on or maybe it is a device that requires a unique set-up. I'm talking about devices you wish use and to simulate during development, before committing to the final production stage of the embedded microelectronic device/controller of your project. Devices that are regularly requested; displays using controller chips not found within the standard display component pack or extended I2c controllers. These are among those frequently required. I would like to add to the previous tutorials I have written and show you how to build those **Graphical Sub-Components** used to create your own FC7/8 components.

Let’s start with this graphical sub-component:

Rx/Tx Flasher – it’s not an actual component, it’s an advanced shape used to build FC7/8 components referred to as a ***Sub-Component***.



I have studied this advanced shape for quite some time in FC7 and struggled to figure out have to construct those arrows, let alone as to how to make them flash during simulation. When FC8 was released, I stopped writing the tutorial. I now have FC8, therefore I can build and test accordingly.

Thanks goes the ‘Benj’ for his help in getting me started back in FC7, so now I would like to share with you, how building sub-components in FC8 & FC7, is done.

It is very important to understand what these graphical shapes are and how they are used to build your final FC7/8 Component(s).

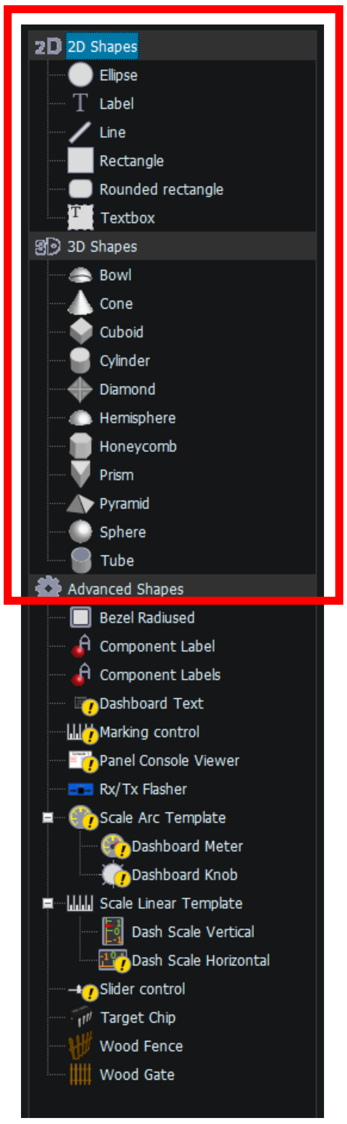
A FC7/8 component is simply a graphical representation of something that looks like or represents the device you are using, **but** is a library of **all** the connections/logic/functionality of your device, wrapped up in one nice graphical image. So, we can use lots of different graphical shapes, put them together to look like your device and simulates that devices action(s) when required during simulation/testing. It makes checking that devices functionality that much easier during the development of your project, to ensure that your final project **behaves/functions** as you expected when completed.

Flowcode comes with a set of shapes you can use for building complex FC8 components and sub-component – **2D; 3D and Shapes**

I’ll show you how to put them together to build your very own FC8 sub-components. Once you have mastered the workflow of putting them together, it should help you to get started building your very own final flowcode 7/8 components.

The Shapes

The shapes we are interested in are those 2D & 3D shapes. Often wondered what they were used for, well now I am going to show you. We are going to make our own sub_component to build graphics that simulate the devices we are using in our projects. Once we have constructed a ***sub-component*** using these shapes, you will be able to use it to make a FC7/8 components that contains all the working logic/functionality of the device you are using, wrapped up in a nice graphical image.



One very important point *– these graphical shapes/components may have a lot of programming code to get them to simulate correctly.* ***However****, this simulation code is never downloaded onto the chip/controller, so don't worry about the amount of simulation code you need to get them to simulate correctly.*

Tested On

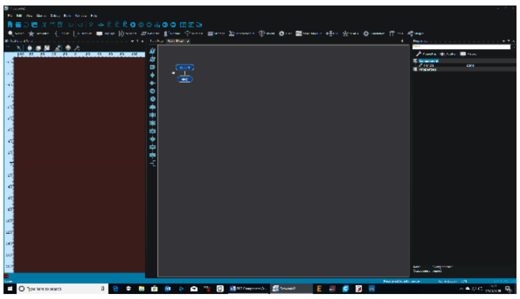
Flowcode v8.1.1.11

Built Dec 14 2018

Flowcode 7 v7.3.0.7

Built Feb 26 2018

Step 1 – The work Space



Throughout this tutorial, you will be using just the standard screen layout as shown opposite:

- 2D Dashboard Panel
- Start page
- Properties Panel

***Create a project called – Simple_Flasher** and save!*

You can use absolutely any embedded chip you like, it simply does not matter as everything you are doing does not involve chip related code during this **Advance Shape** sub-component building process.

Build a Simple Sub-Component

It's called an sub_component, simply because it uses several simple shapes put together for **one** purpose – **simulation** and **appearance**. It will be saved (with a unique ***GUID***) but will only be available as a ***sub-component*** (advanced shape) to build your final FC7/8 component that can be exported to your Component library.

Let's begin by building a simple flashing box with a label – simple! This simple shape will be further developed to build components such as relays/pumps/valves etc.

Requirements: - ***kept simple to for Illustration Purposes***

- The box can be called by the user to flash when required during the simulation of a component
- The colour (of the box) can be changed to show effects such as ON/OFF
- The duration of the flash rate of the box can be changed by the user or set to be **ON** until turned **OFF**
- The label can be edited by the user

What this sub-component (advanced shape) will gives us – ***the ability to use it for constructing any simple device graphically and simulate that device when it is turned ON or OFF*** such as a relay.

Once you have finished building this sub-component (advanced shape) you will begin to understand the process of simulation along with the use of component variables to get everything to simulate correctly. This way, you will begin to see the possibilities FC7/8 has to offer the advanced user. Yes, this simple shape is just the start but illustrates the work-flow required to build your own sub- components and final component.

Step 2 – Construction

IMPORTANT – See: Using The 2D Dashboard Panel

Begin by placing rectangles as shown onto the dashboard panel. These two rectangles, I believe create a colour scheme to suit both Flowcode environments (FC 7 & 8).

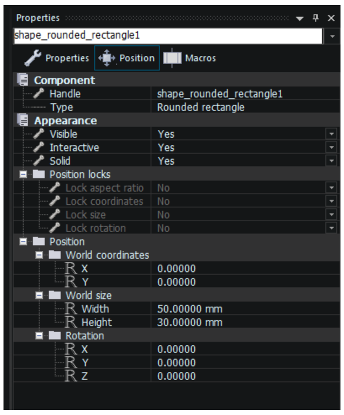
You will notice details of the rectangle(s) displayed in the properties panel:

- Size
- Colour
- Location (world location or actual location) etc

See – Component Creation pt2

First Rectangle:

Width 60.0000mm
Height 30.0000mm
Colour – Blue – FF8057
Your choice



Properties

shape_rounded_rectangle1

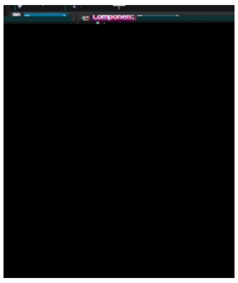
Properties | Position | Macros

Component	
Handle	shape_rounded_rectangle1
Type	Rounded rectangle
Appearance	
Visible	Yes
Interactive	Yes
Solid	Yes
Position locks	
Lock aspect ratio	No
Lock coordinates	No
Lock size	No
Lock rotation	No
Position	
World coordinates	
X	0.00000
Y	0.00000
World size	
Width	50.00000 mm
Height	30.00000 mm
Rotation	
X	0.00000
Y	0.00000
Z	0.00000

Place a second rectangle on top:

- Width 57.5mm
- Height 27.5mm
- Dark Grey - 1E1E1E

If you click anywhere onto the 2D dashboard Panel, the Properties Panel will be blank. However, the sizes and colour of all the shapes you use will still be stored in its own *hidden* library.

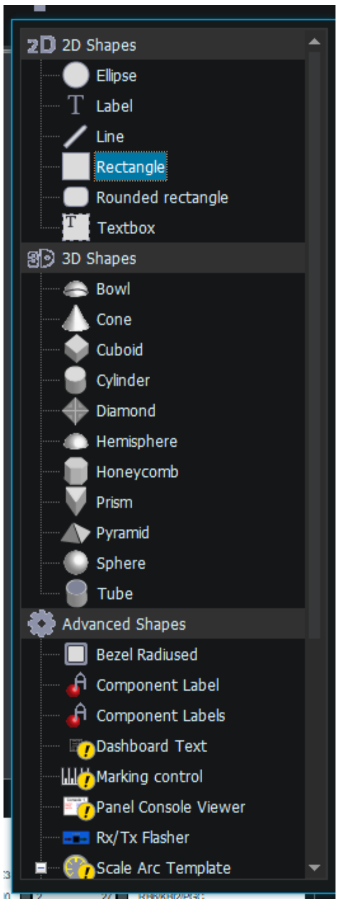


This is because, when this sub-component is exported to your component library, FC8 begins to store all its details in a library – which I will come back to later. For now, what you see graphically, will be seen by the user when building FC8 components, so we need to get this stage looking right.

The grey rectangle gives us a background to place everything on that we need for our sub-component. The properties will not be seen unless we chose specifically to do so. Think of this rectangle as your canvas to get something that looks the way you want for a simple device, any device for now.

Step 3 – The Flashing Box Construction

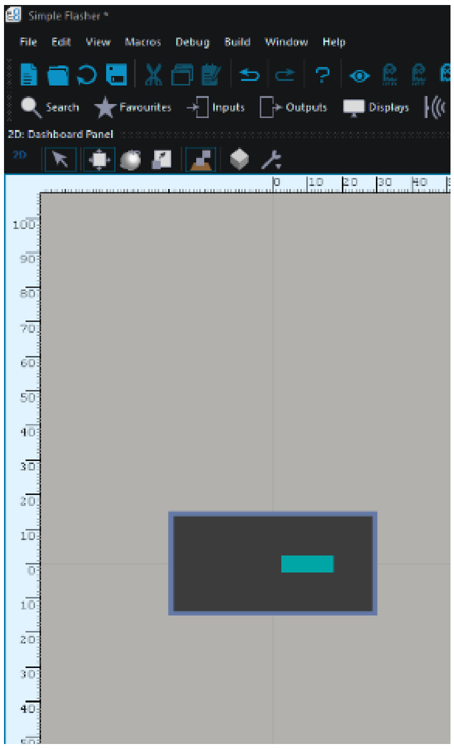
This simple shape is the start to get a working simulating in FC8/7. Looks simple enough. I am going to take you on a journey, putting more shapes together in the next tutorial, that you can control through simulation, to get some impressive results, e.g. how to get it to change colour or even appear to flash (turn ON then OFF).



This is what took me some time to understand:

First, we need to construct the complex shape we want from those simple shapes available in FC7/8.

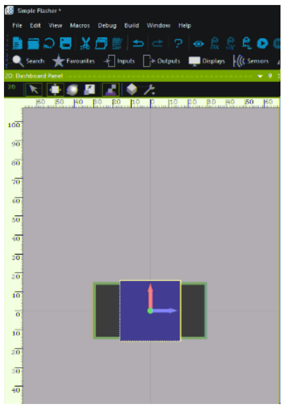
The process of building complex shapes - is by arranging each of these basic shapes in the 2D Dashboard Panel to form the final shape we want. Their sizes & orientation will need to be changed, then placed next to each other to look like your device you are using. We are going to start with a single simple shape a (**rectangle**) to begin the process. Once you have mastered this, we will begin to add further shapes to build more interesting and complex shapes for use in your FC7/8 Component creations.


Simple Flasher *
File Edit View Macros Debug Build Window Help
Search Favourites Inputs Outputs Displays
2D: Dashboard Panel

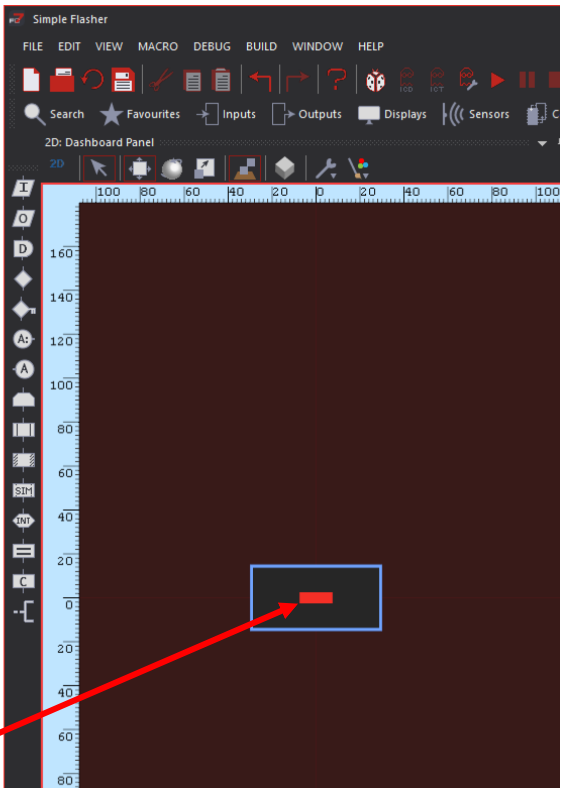
This is the above smaller rectangle placed upon our – ***canvas***. I've changed its size and colour to what I want. However, I want to be able to make its colour change during simulation.

IMPORTANT – See: Using The 2D Dashboard Panel

When you first place these shapes in the 2D Dashboard Panel it will look like something like this shown here.



You can adjust the size to make a smaller rectangle that is placed on top of the larger two rectangles – ***The Canvas***


Simple Flasher
FILE EDIT VIEW MACRO DEBUG BUILD WINDOW HELP
Search Favourites Inputs Outputs Displays Sensors
2D: Dashboard Panel

Click onto the rectangle to highlight it. Then, from the **Properties Panel**, select **Position** and change the **World size to:**

- Width 15mm
- Height 5mm

Change the **World coordinates to –**

X = 10.000mm   Y = 0.000mm

Hopefully, you have a little red rectangle that looks like that opposite.

Well, where you place it is up to you. All we need to get this red rectangle to change colour – ***Flash***.

Handle - shape_rectangle1

Next - we are going to begin to set everything up so we can control how the smaller rectangle is simulated in the flowcode 7/8 environment.

Step 4 – The Flashing Shape

Take a closer look at the Properties Panel.

There are three menus:

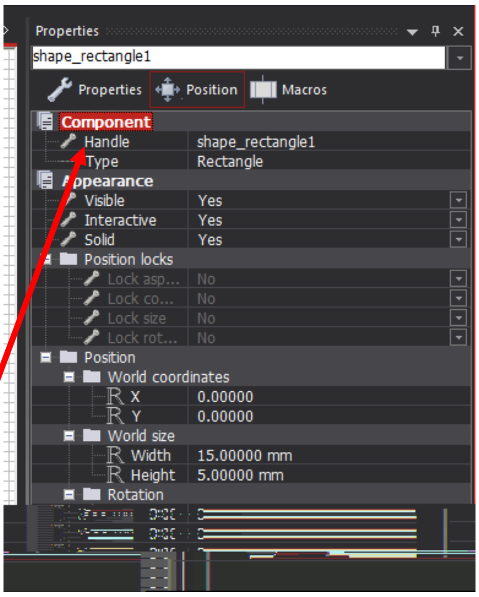
- **Properties**
- **Position**
- **Macros**

We need to select the menu – Properties.

Click anywhere onto the 2D dashboard Panel and you will see that there are no properties associated with our shapes.

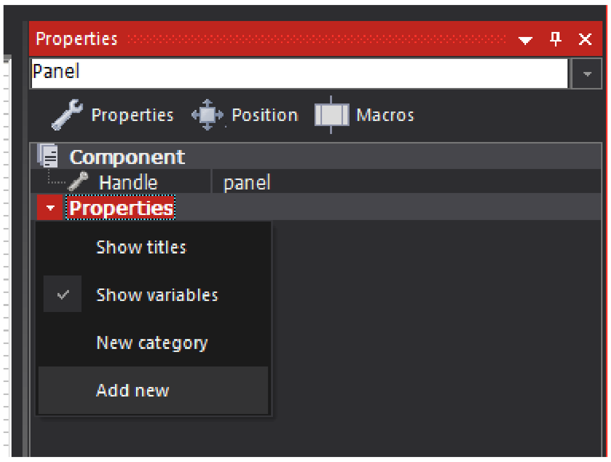
Click onto the red rectangle and change it's **Handle to – led1**

Why – this will make referencing it during simulation much easier!



Component
Handle shape_rectangle1
Type Rectangle
Appearance
Visible Yes
Interactive Yes
Solid Yes
Position locks
Lock asp... No
Lock co... No
Lock size No
Lock rot... No
Position
World coordinates
X 0.00000
Y 0.00000
World size
Width 15.00000 mm
Height 5.00000 mm
Rotation

You need to set up this graphic component block properties variables. This is necessary, for any future component built using this ***sub-component*** (simple advanced shape) flasher, will require various variables, used for simulation purposes, to be changed that will be stored in its own **library + GUID** and allow on- screen simulation effects.



Next – How to set up those variables

Step 5

From the pull-down menu on the properties panel – select ***Add New***

The first variable we are going to set up is: -

- **Flash_ON** – colour variable

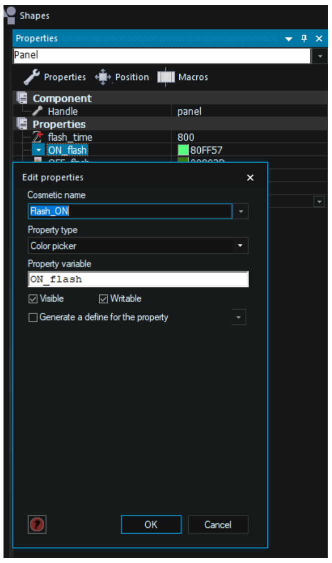
This is the colour a user will see and be able to select, during simulation when the shape (Handle – **led1)** is turned on.

The cosmetic name: - **Flash_ON**

Variable:- ***ON_flash***

I've have deliberately tried to differentiate the variable from its cosmetic name.

Property Type:- ***Color Picker (Unsigned Integer)***



Shapes

Properties

Panel

Component	
Handle	panel
Properties	
flash_time	800
ON_flash	🟩 80FF57

Edit properties

Cosmetic name: Flash_ON

Property type: Color picker

Property variable: ON_flash

[✓] Visible [✓] Writable

[ ] Generate a define for the property

ULONG

Icon:	
Range:	0 to 4294967295
Bit Depth:	32-bit unsigned integer

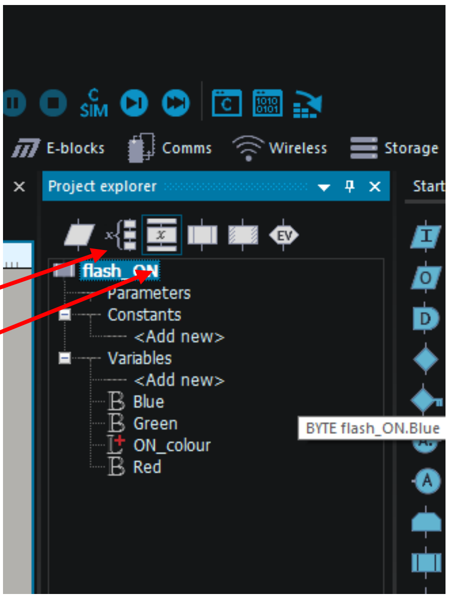
Note - An unsigned 32-bit value offers the largest integer value of any of the variable types, and because of this, many of the simulation-only functions in Flowcode use ULONG, because simulation memory is not a consideration. Remember, nothing here is down loaded onto the chip!

Click OK to confirm

There are several variable types available. We need to be able to identify each variable type, as this will be very important to us when transferring information from an ***active variable – its stored data must not be altered or destroyed by any simulation process unless it is required to do so,*** to a local temporary variable that is used to get changes necessary for simulation purposes.

Global variables are shown here:

Local variables are shown here:



Project explorer

flash_ON
- Parameters
- Constants
  - <Add new>
- Variables
  - <Add new>
  - Blue
  - Green
  - ON_colour
  - Red

BYTE flash_ON.Blue

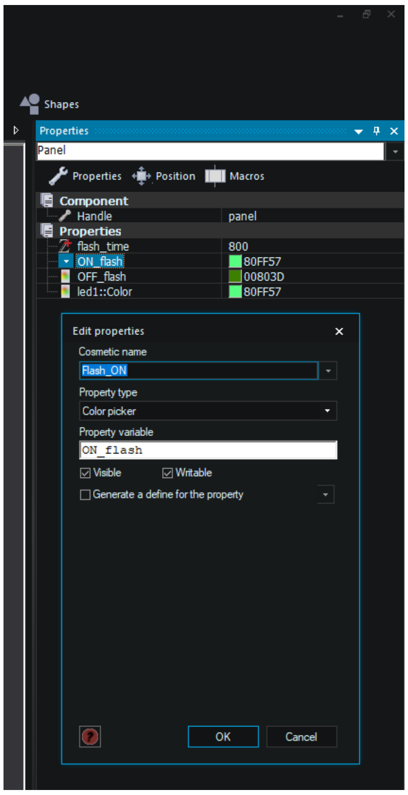
Ideally, we only need local variables to achieve simulation – i.e. what appears on the screen.

The size/type of variable is also important. If the data is in the range of – 0 to 255 or 0x00 to 0xff, then a Byte variable is ideal;

BYTE

Icon:	
Range:	0 to 255
Bit Depth:	8-bit unsigned integer

However, we are going to make the small rectangle **flash** or change colour. You may be aware that colour information is stored as either 8bit; 16bit or 32bit. The size of the variable to store that information is very important.


Shapes

Properties

Panel

Properties Position Macros

Component	
Handle	panel
Properties	
flash_time	800
ON_flash	🟩 80FF57
OFF_flash	🟩 00803D
led1::Color	🟩 80FF57

Edit properties

Cosmetic name: Flash_ON

Property type: Color picker

Property variable: ON_flash

☑ Visible ☑ Writable

☐ Generate a define for the property

OK Cancel

You should now have the following shown on your Properties Panel:

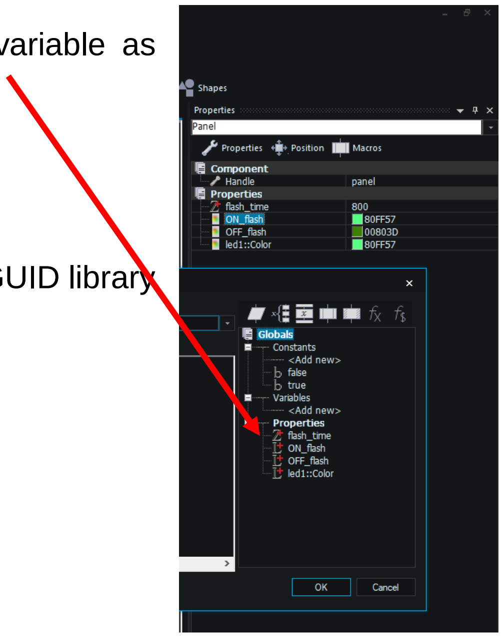
You need to set the colour value for – ***ON_flash***

This will give a **Properties variable** selected from the list – ***Colour picker***

You can change the colour to what ever you want and will be available to you – the user – when building your components. If you click anywhere in side the 2D Panel, the Property variable will still be visible. This is because it is now stored in its own unique library.

The numbers to the right for GREEN is – **80ff57**

This is a 24bit hexadecimal value.


Shapes

Properties

Panel

Properties Position Macros

Component	
Handle	panel
Properties	
flash_time	800
ON_flash	🟩 80FF57
OFF_flash	🟩 00803D
led1::Color	🟩 80FF57

```
Globals
  Constants
    <Add new>
    false
    true
  Variables
    <Add new>
  Properties
    flash_time
    ON_flash
    OFF_flash
    led1::Color
```

OK Cancel

Flowcode will store these value using a 32bit variable as shown here

You will see the icon next to the variable

This shows - the exact variable type used by the GUID library for storing the variable – ***ON flash*** color

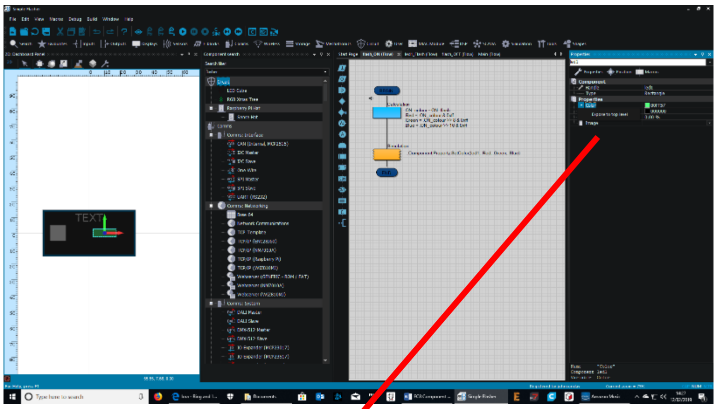
Repeat the above and set up –

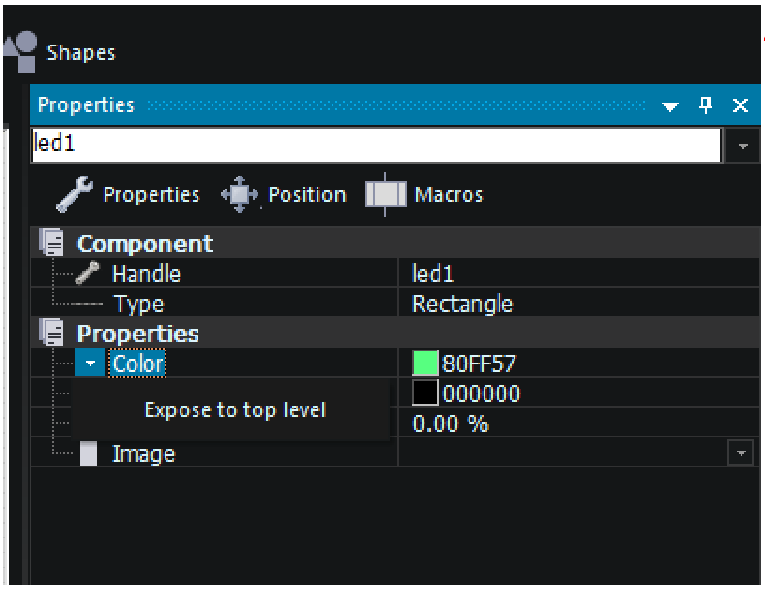
- ***OFF flash***

Click onto the small rectangle and from the Properties Panel: -

Under properties – **color** select from the drop down menu:

Expose to top level





Component	
Handle	led1
Type	Rectangle
Properties	
Color	🟩 80FF57
	■ 000000
	0.00 %
Image	

*The variable – **led1::color** is a hidden variable. You must exposed this variable to allow both observation & Simulation (how its value changes during simulation) and the transfer of data to a temporary holding variable during simulation .*

Step 6

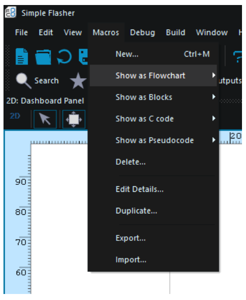
We are going to create two **Macro's** for this simple sub_component flasher that can be ***simulated (ON/OFF) using these macros*** and inserted into your final component. You will be able to change the variables to adjust the colour (***led1::color*** *)* for the shape you have used using the colour picker. However, the colour variables for the small rectangle, must be set up first.

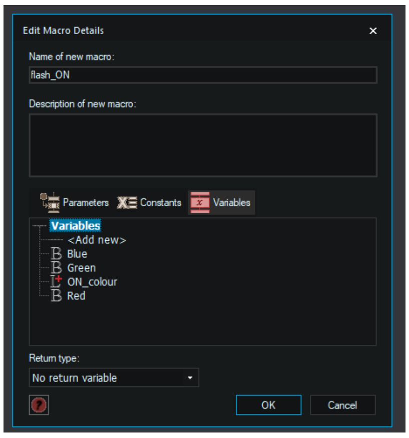
So far, the variable you have created are: ***Global Colour variables*** – store in the **Properties Panel** These are the colours you chose for ON & Off. What we need to be able to do, is transfer these ON/OFF colours to our small rectangle – here's how.

- **ON_flash** colours
- **OFF_flash**

Create two macros

add the following variables under – **LOCAL Variables**




Edit Macro Details

Name of new macro: flash_ON

Description of new macro:

Parameters Constants Variables

Variables
- <Add new>
- Blue
- Green
- ON_colour
- Red

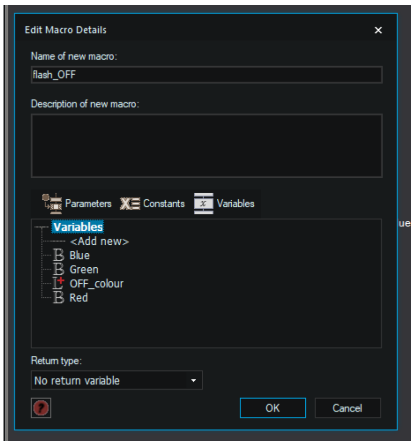
Return type: No return variable

OK Cancel

1. **flash_ON** macro

Variables - **local**

- **Blue**
- **Green**
- **Red**
- **ON_colour** **(temp store)**


Edit Macro Details

Name of new macro: flash_OFF

Description of new macro:

Parameters Constants Variables

Variables
- <Add new>
- Blue
- Green
- OFF_colour
- Red

Return type: No return variable

OK Cancel

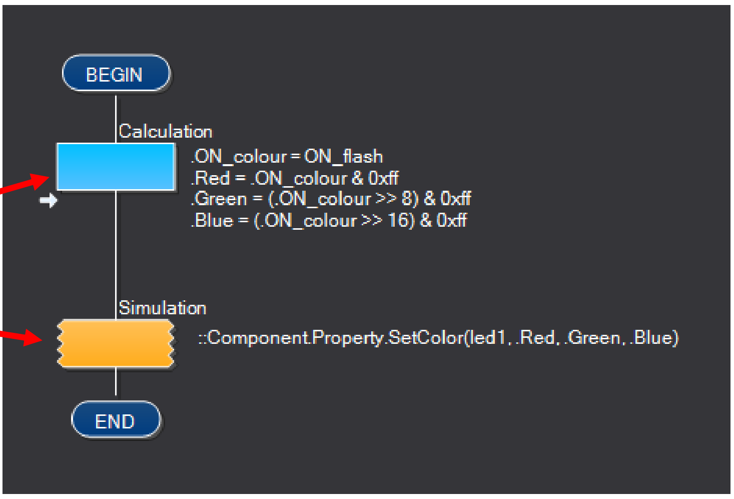
2. **flash_OFF** macro

Variables - **local**

- **Blue**
- **Green**
- **Red**
- **OFF_colour** **(temp store)**

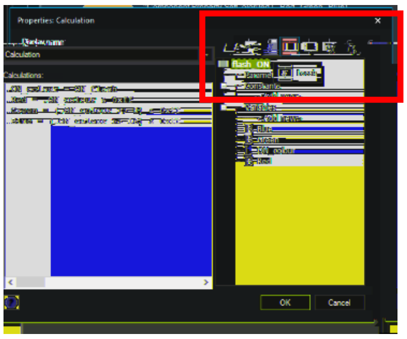
Place the following ICON's for the Macro – **flash_ON**

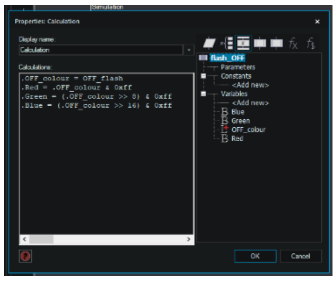
- **Calculation**
- **Simulation**



```
.ON_colour = ON_flash
.Red = .ON_colour & 0xff
.Green = (.ON_colour >> 8) & 0xff
.Blue = (.ON_colour >> 16) & 0xff
```

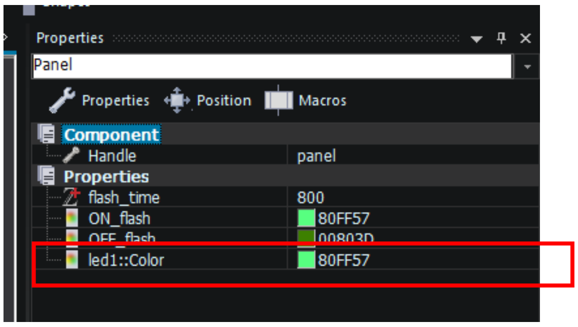
```
::Component.Property.SetColor(led1, .Red, .Green, .Blue)
```





```
.OFF_colour = OFF_flash
.Red = .OFF_colour & 0xff
.Green = (.OFF_colour >> 8) & 0xff
.Blue = (.OFF_colour >> 16) & 0xff
```

To get the small rectangle to change colour, we need to change the **property value** of – **color** associated to the shape – **led1**

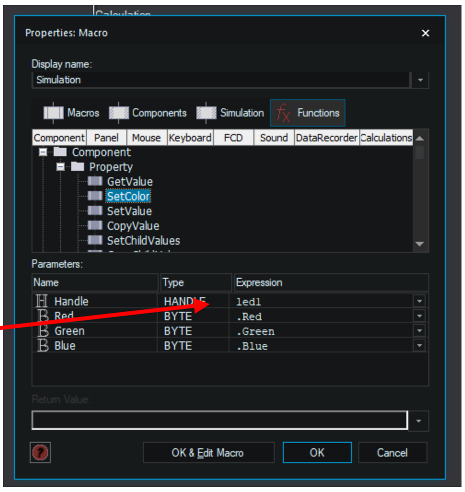


Component	
Handle	panel
Properties	
flash_time	800
ON_flash	🟩 80FF57
OFF_flash	🟩 00803D
led1::Color	🟩 80FF57

Remember that variable we – ***exposed!***

led1::color

The small rectangle shape we called **Handle - led1,** has a hidden library. That library has a variable called ***led1::color*** associated to its colour. Well, we get to change it when we added the simulation ICON.



Properties: Macro

Display name: Simulation

Parameters:

Name	Type	Expression
Handle	HANDLE	led1
Red	BYTE	.Red
Green	BYTE	.Green
Blue	BYTE	.Blue

Click onto the simulation ICON and:

- **Select Function**
- **Component**
- **Property**
- **Setcolor**
  (has the hidden colour data *led1::color*)

Parameters

Handle – **led1** this holds the data of ***(led1::color)***

Red	**.Red** (local variable)
Green	**.Green**
Blue	**.Blue**

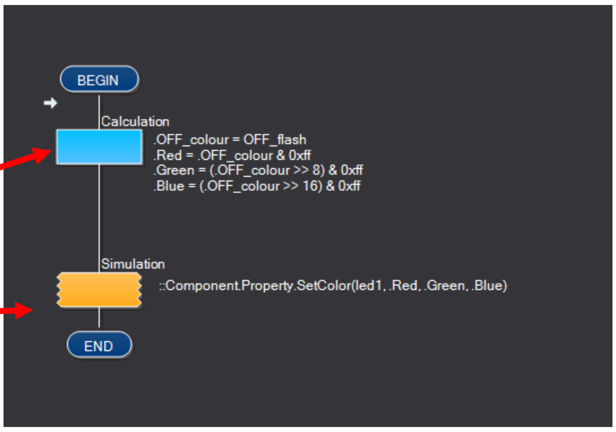
Note – All local variables are signified by the dot **.Red** etc

These 8bits colour variables will change the led1::color variable!

Repeat for the OFF Macro

Place the following ICON's for the Macro – **flash_OFF**

- **Calculation**
- **Simulation**



```
.OFF_colour = OFF_flash
.Red = .OFF_colour & 0xff
.Green = (.OFF_colour >> 8) & 0xff
.Blue = (.OFF_colour >> 16) & 0xff
```

```
::Component.Property.SetColor(led1, .Red, .Green, .Blue)
```

To extract each of the colours to change - **.Red; .Green; .Blue** of our shape, in order to simulate – **ON** colour executing the **ON Macro;** and the **OFF colour** executing the **OFF Macro,** we need do some math.

This is achieved as follows: - All the colour information is stored in a UNIT LONG variable ***led1::color***. The colour information is stored as: - ***RGB i.e.*** first 8bits - RED; second 8bits - Green followed by the last 8bits - Blue. To extract each 8bit set, we use a method such as: -

1. Transfer the ***led1::color*** variable into the temp. variable - **ON_colour**
2. Use the **AND** function to mask the bits we want
3. Shift the bits along the register until we get the next set of 8 colour bits
4. Use the **AND** function again to extract those colour bits
5. Keep doing this until all three colours have been extracted.

This is how it works

Remember, we have set up a variable to temporally store the contents of the colour – **flash_ON;** and **flash_OFF**. This is necessary to prevent errors by permanently changing the contents of the variables set by the user – **ON-flash; OFF_flash macro**

To extract the – **RED 8Bits**

24 **ON_colour** variable Bit 1

1	1	1	1	1	1	0	0		1	0	1	0	1	1	1	1	1	1	0	1	1	0	1	1

AND 0xFF

1	1	1	1	1	1	1	1

variable **.Red**

1	**1**	**0**	**1**	**1**	**0**	**1**	**1**

To extract the **-GREEN 8Bits**

Rotate Right >> 8

0	0	0	0	0	0	0	0	1	1	1	1	1	1	0	0	1	0	1	0	1	1	1	1

AND 0xFF

1	1	1	1	1	1	1	1

variable **.Green**

1	**0**	**1**	**0**	**1**	**1**	**1**	**1**

To extract the – BLUE 8Bits

Rotate Right >> 16

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	0	0

AND 0xFF

1	1	1	1	1	1	1	1

variable **.Blue**

1	**1**	**1**	**1**	**1**	**1**	**0**	**0**

The flowcode 7/8calculation to do this is as follows for the **ON** colour change: -

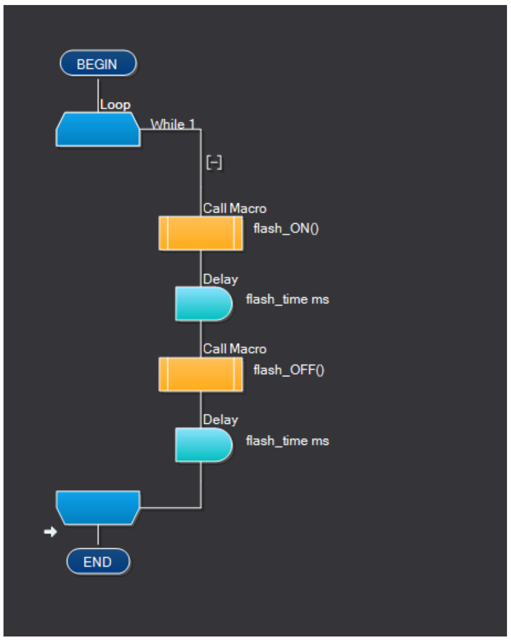
```
.ON_colour = ON_flash
.Red = .ON_colour & 0xff
.Green = (.ON_colour >> 8) & 0xff
.Blue = (.ON_colour >> 16) & 0xff
```

The flowcode 7/8 calculation to do this is as follows for the **OFF** colour change: -

```
.OFF_colour = OFF_flash
.Red = .OFF_colour & 0xff
.Green = (.OFF_colour >> 8) & 0xff
.Blue = (.OFF_colour >> 16) & 0xff
```

Step 7 – **Testing**

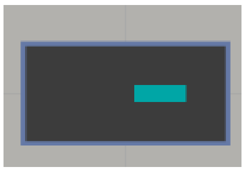
Place the following in the **Main** start page.



When you run the program, you will see the little square change colour. Experiment by changing the flashing colour and rate of the flash. You should see the colour picker of the variable ***led1::color*** change along with the hexadecimal values.

When you have finished testing, delete everything the Main Start Page

Step 8 – **Finishing Touches**

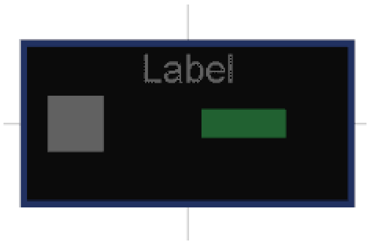


Okay, this is what we have so far, not very interesting to look. What we need to do is add something to identify what it is, when it is placed in a project as a finished component. We can achieve this by adding a label, that can be edited and a symbol of what it represents, this would finish it off nicely.

Here's how

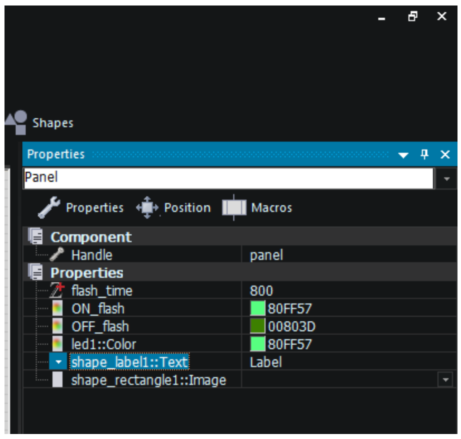
Place the following onto the canvas from the 2D shapes – label (this is for TEXT); Square (change size to 10mm x 10mm)

Place anywhere onto the canvas to suit your preference. It should look something like this:



To make them editable, highlight each in turn, then from the drop-down menu, select

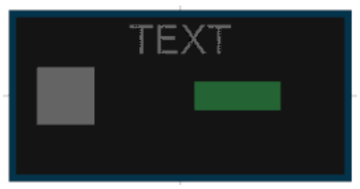
- **Expose to top level for the - Label**
- **Expose to top level - image**



Panel

Component	
Handle	panel
Properties	
flash_time	800
ON_flash	🟩 80FF57
OFF_flash	🟩 00803D
led1::Color	🟩 80FF57
shape_label1::Text	Label
shape_rectangle1::Image	

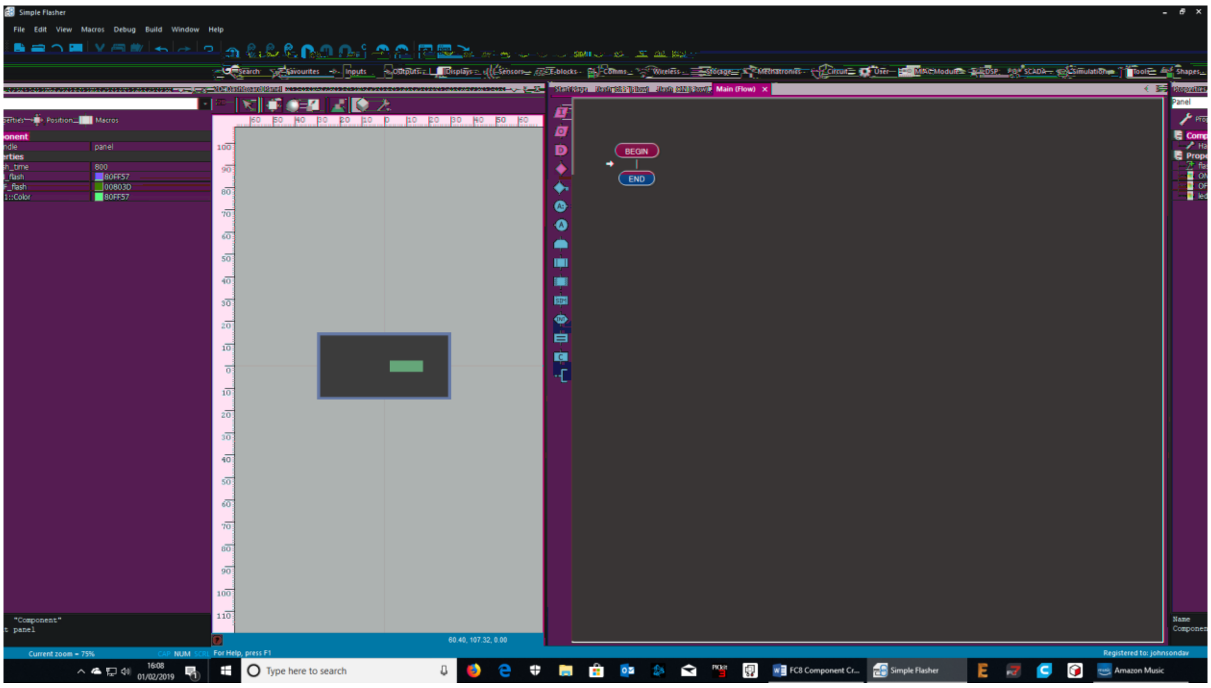
When you have done that, it should look like this: -



Step 10 – **Simple Flasher Configuration**

To begin the Sub-Component configuration, we need to tidy everything up ready to export the finish sub-component ready for use in any final Flowcode 7/8 component. I will be showing you how to use this simple sub-component to make a flowcode component for controlling a relay. Simple yes however, it will illustrate all the remaining stages necessary to get everything working correctly.

This is what your **Sub-Component** project should look like: -



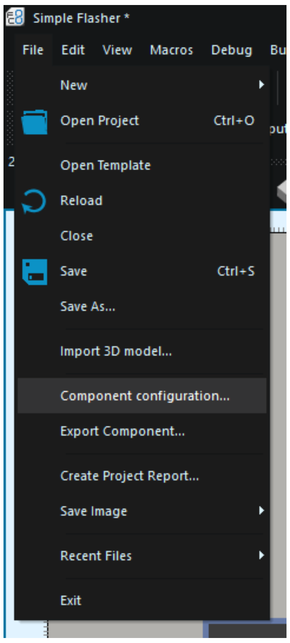
The ***Start Pages*** should have three macros: -

- Main
- flash_ON
- flash_OFF

The variables: -

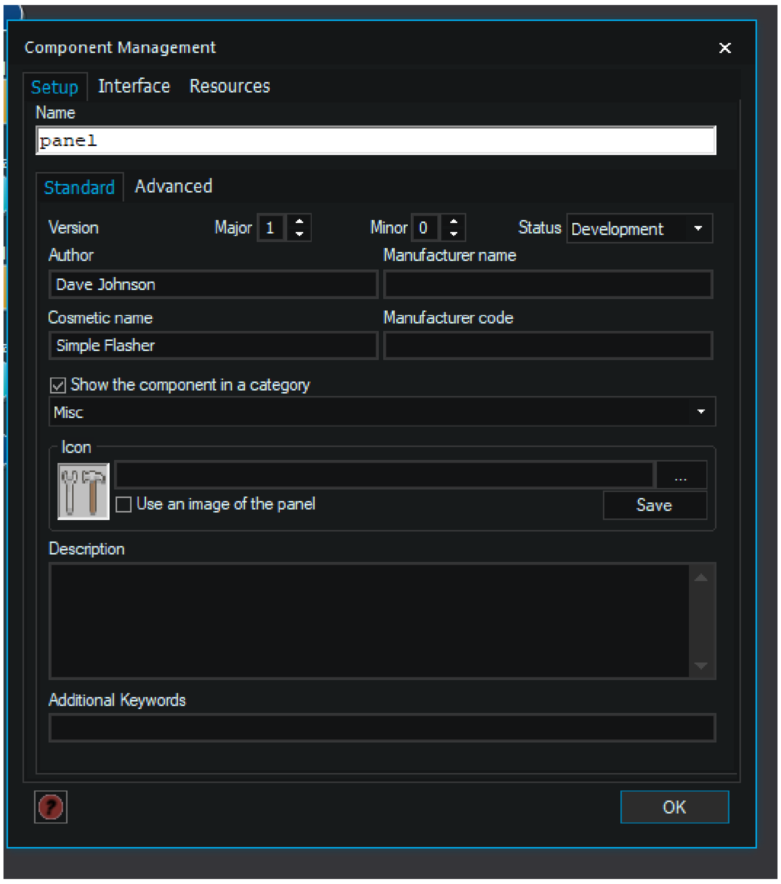
- flash_time – Optional (this will be used for the next tutorial)
- ON_flash
- OFF_flash
- Led1::color (exposed to top level)

Open – Component configuration from the menu under file: -



This opens the following – **Component Management**

TAB - **Standard**



Name: panel

Version Major 1 Minor 0 Status Development

Author: Dave Johnson

Manufacturer name:

Cosmetic name: Simple Flasher

Manufacturer code:

☑ Show the component in a category

Misc

Icon

☐ Use an image of the panel

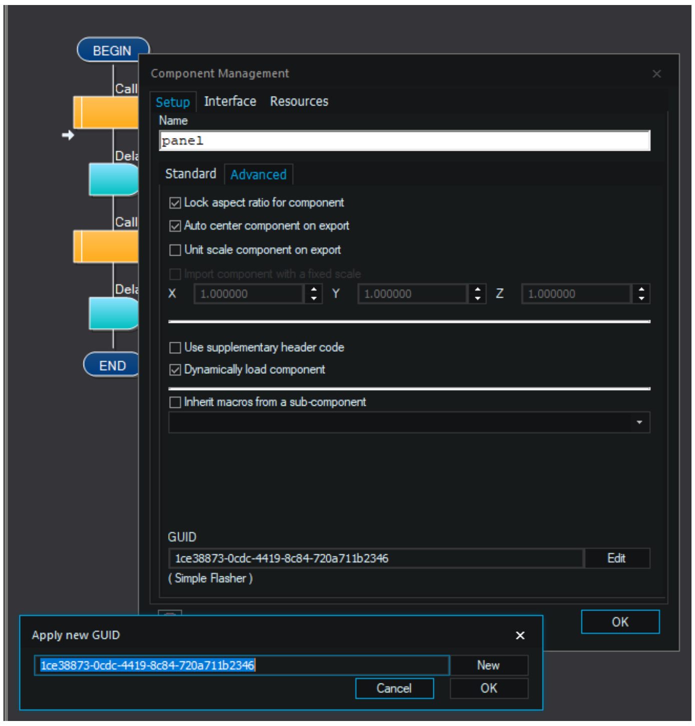
Description

Additional Keywords

Set up:

- Author
- Cosmetic name - **Simple Flasher**
- Status - **Development**
- The ICON -
- Show in the component category – **Misc**

TAB - **Advanced**



Component Management

Setup | Interface | Resources

Name: panel

Standard | Advanced

☑ Lock aspect ratio for component
☑ Auto center component on export
☐ Unit scale component on export
☐ Import component with a fixed scale
X 1.000000 Y 1.000000 Z 1.000000
☐ Use supplementary header code
☑ Dynamically load component
☐ Inherit macros from a sub-component

GUID: 1ce38873-0cdc-4419-8c84-720a711b2346 Edit
(Simple Flasher)

OK

Apply new GUID

1ce38873-0cdc-4419-8c84-720a711b2346 New

Cancel OK

Set Up: -

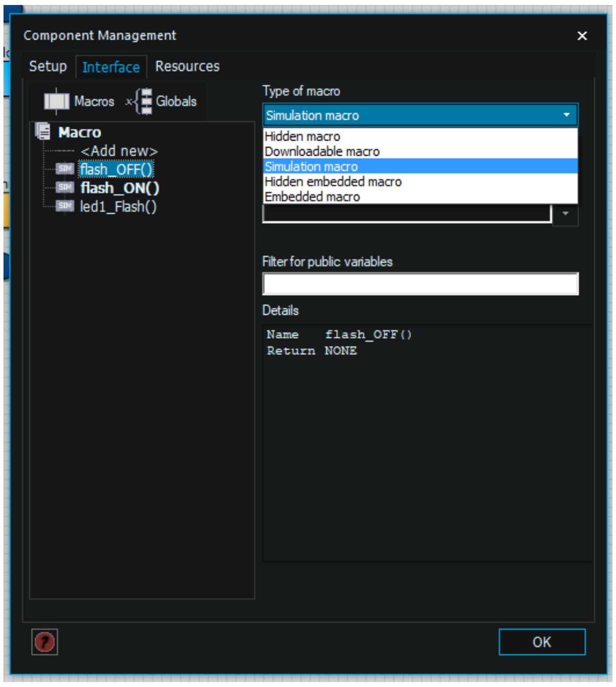
- GUID select edit; **New** then click OK (Keep a record for future reference)
- Lock Aspect ratio (optional)

TAB – **Interface**

For each of the listed Macros, using the ***Types of Macro***, from the drop down menu set each macro as follows: **-**

- flash_OFF ***Simulation macro***
- flash_ON ***Simulation macro***

Please ignore that macro shown – led1_flashl will be showing you this later in the next tutorial. Its not needed just jet!



Component Management

Setup | Interface | Resources

Macros | Globals

Macro
<Add new>
flash_OFF()
flash_ON()
led1_Flash()

Type of macro: Simulation macro

Hidden macro
Downloadable macro
Simulation macro
Hidden embedded macro
Embedded macro

Filter for public variables

Details

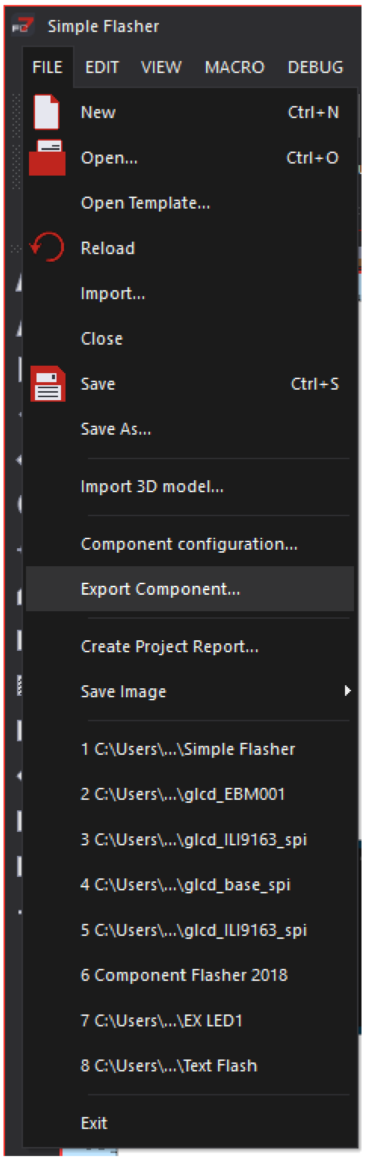
```
Name    flash_OFF()
Return  NONE
```

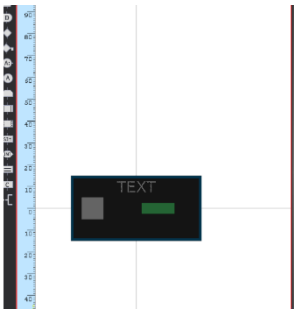
OK

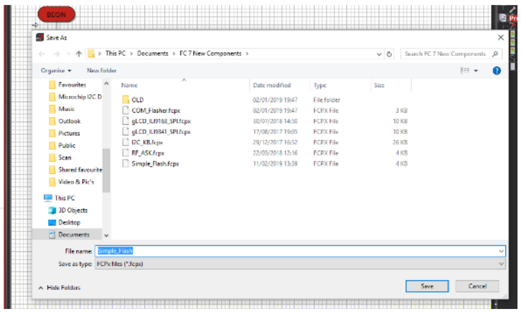
Click **OK** when done.

We need to export our simple flasher into our library that we set up earlier. To do this: -

- select ***Export Component***
- Navigate to your component library



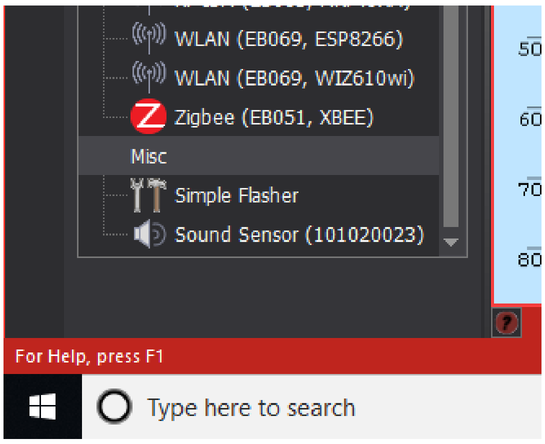



Save As
This PC › Documents › FC 7 New Components
File name: Simple_Flash
Save as type: FCPX files (*.fcpx)
Save
Cancel

Name your simple flasher component and select – OK to save.

Just to check everything is ok, restart Flowcode and, using the search option, you should be able to locate your simple_flasher component under – ***Misc***

If it is not shown after you have restarted Flowcode, check that you have assigned a **NEW** GUID for your sub_component.



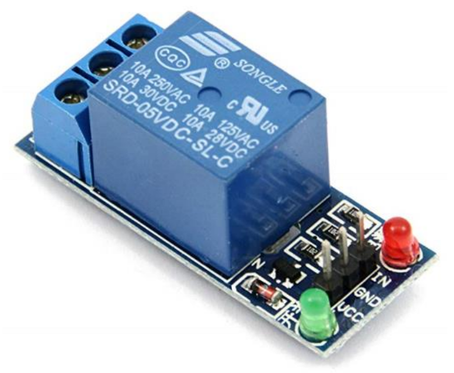
The Next Tutorial – Build a Flowcode Component for controlling a Device – A Digital Output (Relay) 7 RS232 Communication (433MHz TX/RX Modules)

Use the Sub-Component to Build Relay Component

A Digital Output Device

Now we can use our sub_component that we have created to build an actual working component for use in our projects – A Relay.

Yes, it’s a simple output device, but this next part of the tutorial I will show you how to make add connection pin to the sub_component that will allow you to switch a relay **ON** or **OFF** in your projects.



1 Channel Relay Module 5 V/230 V LED Relay for Arduino Raspberry p

by Geras-IT

RS232 TTL Communication

Further develop the Sub-Component to build a very simple 433MHz (ASK) transceiver. Tested to achieve good communication of up 20 metres.

Or this LoRa 433Mhz RS232 E32-TTL-100

Longer Range module