Flowcode 7 – How to Make Your Own Components

If you are interested in making your own Flowcode components based upon devices which are not currently included in the FC7 component library, then this guide will show you the journey I have taken to build my own FC7 component. This enabled me to use any microcontroller using the FC7 graphical programming language to control my new device. The problem is, where do you begin to get the building process started. I have chosen this very simple device ONLY to illustrate the component building steps involved, and is my visualisation of Flowcode's inner workings.

This device is a very simple ASK - RF transmitter and receiver which uses the 433MHz band (amplitude shift keying). These devices (transmitter and Receiver modules) have only one data connection each - plus power. Data is applied as a data stream – 1 Bit at a time and 8 Bits per packet. No clock pin or chip select – nothing, simple. So, how can I control it using Flowcode?

I could simply use the RS232 port, but what if I wanted to use any available micro-controller, or use software only. That's to say, some controllers do not have a RS232 port. This is where the power of FC7 comes in to play. If I built a component using FC7 internal tools, I will be able to attach this device to any controller and use either the hardware on the chip or let FC7 build a software option to control it. Building my own component will allow me to include all the programming logic I need to control this device, wrapped up in one GUI component stored in my component library, brilliant, where do I start?

Well, this is what I have discovered is possible with FC7.

To get the component building process started, it is worth pointing out now what I believe Flowcode has under the HUD. This way, you will begin to understand what I did and how it related to how the FC7 technology works.

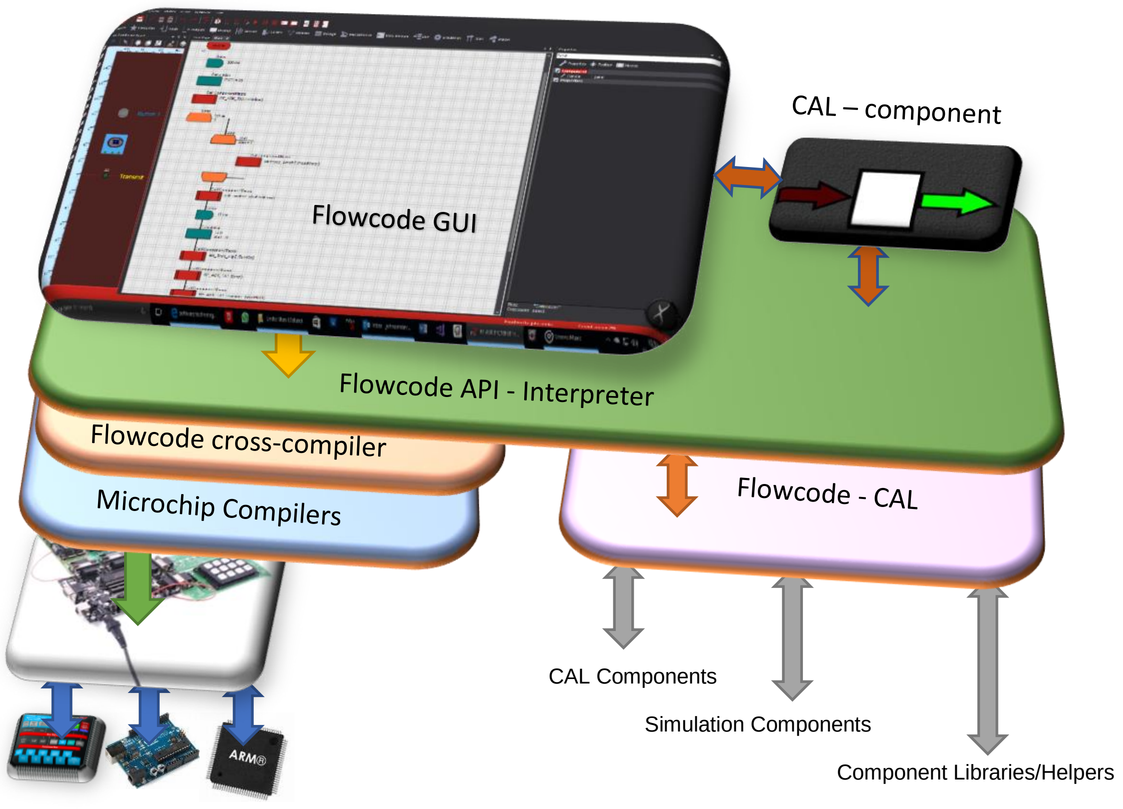
First – FC7 has - GUI; CAL API (Graphics User Interface; Application programming Interface and Code Abstraction Layer).

Flowcode 7 allows those on screen graphical symbols to be used for both Simulation (the visual effects as a program is run before loading to the chip) and Conversion to micro controller specific code, plus all of the programming logic you need to control the device. However, because there are many controllers, the conversion process goes through several stages or cross compilers. So, our component must follow a set of rules to allow this multi task environment to work correctly. Shown below is my illustrated view of those links within FC7.

Flowcode has all the tools you need to build your own functional graphical components, that can be saved to a library, ready to be inserted into your programs. The components you can build include LCD displays; Stepper motors; Communication devices and just about anything you can think of, currently available on today's market. However, they need to be programmed using Flowcode's powerful API and it needs to know how to communicate with your new component. This is where Flowcode's extensive library of tools can be accessed to help build your graphical GUI image/footprint for your component.

Shown below is a graphical view of what I think lies beneath Flowcode interface.

Flowcode uses technology layers (API; CAL; GUI) help to organise and keep track of the various features of the target controllers and the hardware that each controller has (memory; pin connections; RAM; EPROM;UART's etc). Flowcode will then take what features you want for your component and begin to link the technologies together using its built-in tools.



<u>**STEP 1**</u> – choose your CAL component first!

The component building process begins with the – CAL (code Abstraction layer).

*(The **CAL** (Code Abstraction Layer) <u>**components**</u> are designed to simplify the creation of controller programs or other related **components**.*

- *They encapsulate the internal functionality of the supported micro-controller device features.*
- *In this way your Flowcode program, or component, uses a consistent interface to the functionality irrespective of the actual target micro-controller device.*

It does sound a bit geeky, but it is simply the start of our first Flowcode building block for our new component. (*there are other base CAL's available such as – gLCD; ADC etc)*

Example -If my component needs to send data in a bit stream, I have the choice of using an UART – RS232 or SPI or Software (FC7 can replicate both using software techniques!) – both are serial interfaces and both have their own CAL.

Here are a few the generic building blocks - CAL's available in Flowcode: (I'll be showing you where to find them later)

Peripheral CAL's

ADC	A low-level implementation giving direct access to the CAL Analogue-to-digital converter
CAN	Low level routines for controlling the CAN interface
EEPROM	A low-level implementation giving direct access to the CAL EEPROM memory
I2C	Chip Abstraction Layer for Two Wire I2C Communications
PWM	A low-level implementation giving direct access to the CAL Pulse-Width-Modulation hardware
SPI	A low-level implementation giving direct access to the CAL SPI - Serial Communication hardware
UART	A low-level implementation giving direct access to the CAL RS232 hardware

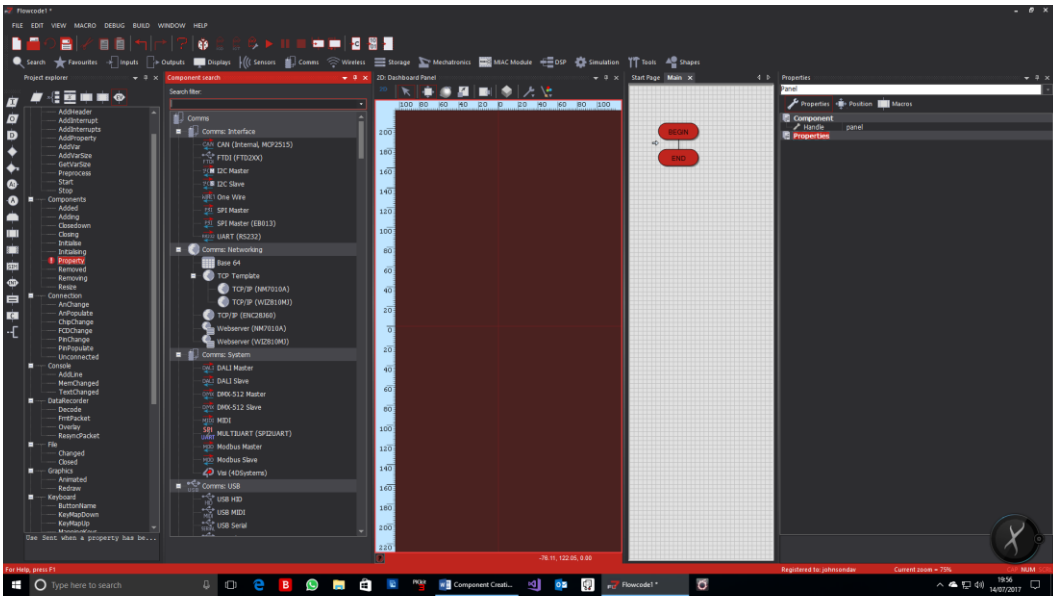
Begin by using one of these components to start the building process:

Go from this base component:- to this GUI component:-

The final GUI component has all the Macros/functions and programing logic you need, wrapped up in a nice looking graphical component that controls your new device.

STEP 2 – Start building your component

Launch FC7 and start a new project. It does not matter what chip you use (8Bit PIC etc.), but it may make sense to start with the chip you intend to be using with your component. The chip configuration is not required as nothing about the chip is used when your component is created, only the links within the CAL and the API are used to build your component. Begin by laying out your FC7 screen like this: -



Set up the view for- **Project explorer; Component search; 2D: Dashboard Panel; Start Page and Properties**

It may be a bit tight on some screens but once you get building underway you can close some of the panels.

Component Search – use this to find your chosen **CAL**. If you want to follow my example, search for **UART** and place onto your 2D Dashboard. This small graphic **image** is never used to create your image of the component, only the settings/properties and parameters it contains – it's a **CAL** component, meaning FC7 uses it to store **base component settings and functions**. So, it will be there during the whole component building process, even when exporting your finished component, but never displayed with your finished component.

That's the **base component** (bottom Layer), now the component you want to see on the screen when your component is finished. Search for **Flasher** (top layer Component) and place it onto your 2D Dashboard. If you look closely at this graphic and click onto it, the **Properties** shows it has some features that you can control: - **Label** the name that will be shown on the component**; ICON** image (that's the black square in the middle, you get to place any icon you wish to make it resemble what your component will do); **flash_time** this is for simulation purposes when it is used in your programs, any Macro which calls your component, you can make the appropriate LED flash. You can now close the Component search.

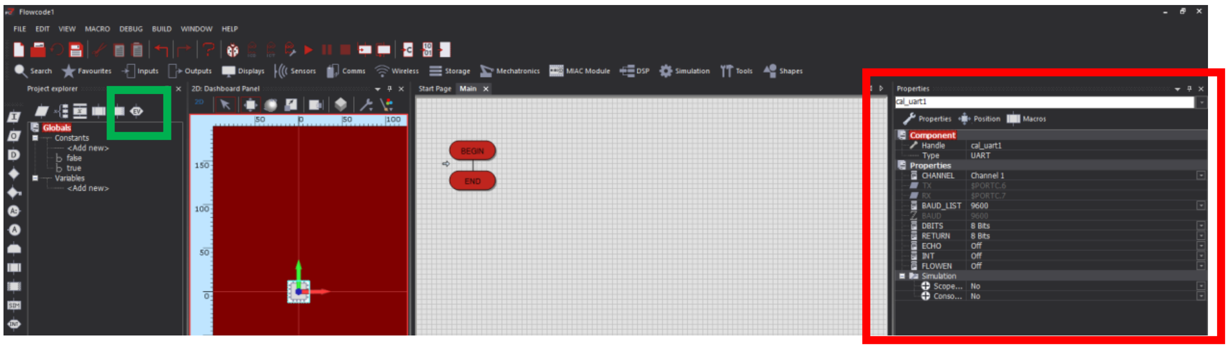
Well, what you have done so far: -

- Started with base CAL - containing all the files at the bottom layer
- Place the top GUI image at the top layer – containing the simulation files

What we are about to do now, involves linking everything together. It involves building your component using MACRO's which are available at the CAL level (those function files and API system variables) and bring them into use for your component. FC7 comes with other Components you can use such as ***switches; led's; LCD; gLCD's etc*** to enable you to build a whole range of new components. I have not used any of these yet, but it would be good to explore these later.

<u>**STEP 3**</u> – Component Properties/Parameter/Events

Now we start with those all important component properties contained within the UART **CAL.** Begin by clicking on the **UART** CAL to list what properties are contained within it and displayed in - **Properties**



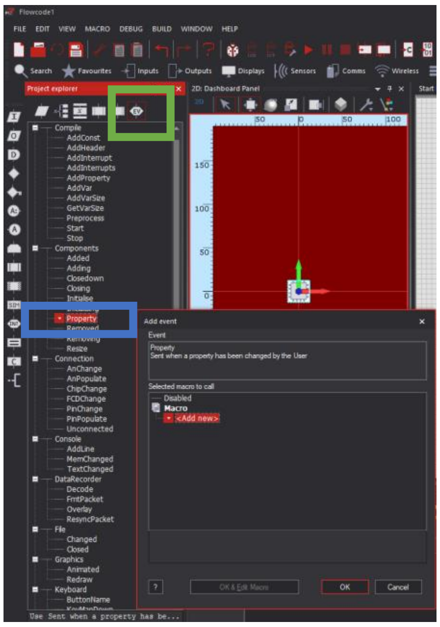
Component	
Handle	cal_uart1
Type	UART
Properties	
CHANNEL	Channel 1
TX	$PORTC.6
RX	$PORTC.7
BAUD_LIST	9600
BAUD	9600
DBITS	8 Bits
RETURN	8 Bits
ECHO	Off
INT	Off
FLOWEN	Off
Simulation	
Scope...	No
Conso...	No

What we see here is only a basic set of properties, however, FC7 has a valuable tool you can use to inspect in much more detail what is contained in this **CAL**, we'll look at that tool later. For now, at the top of the list of the component properties are – **CHANNEL.** It contains a **Handle Properties**. This is a linking event that we need to bring across to our component.

On the other side of the screen is – **Project explorer**

You will find a menu at the top labelled **<EV>** click on this to display a list of options available.

This should display the following: -


Add event

Event

Property
Sent when a property has been changed by the User

Selected macro to call

Disabled
Macro
<Add new>

The **<EV>** parameters contains what we need to control our component. We will store these setting in the following macro called **Ev_Property**

Under the heading **Components** double click the option – **Property –** select add new

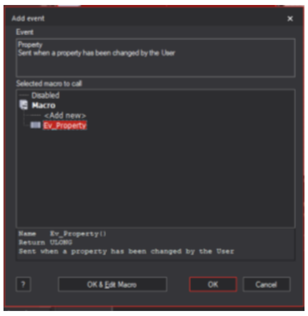
What this is going to do is to create an event **MACRO** which will contain those **properties and parameters.** The **event** can then be placed in our component. That way, the **events/parameters/properties** can be selected and used to build our component.

Click on the **Add New** and accept the default setting which are: -

- Macro name - **Ev_Property**
- Return type – **ULONG**

Your Macro should look like this: -

Now click OK


Add event

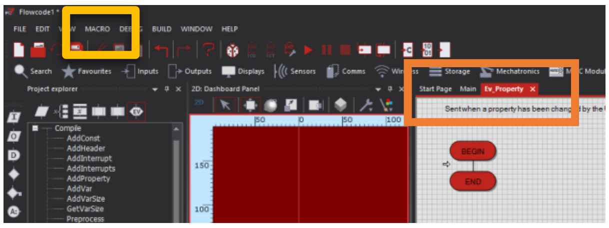
Event

Property
Sent when a property has been changed by the User

Selected macro to call

Disabled
Macro
<Add new>
Ev_Property

```
Name   Ev_Property()
Return ULONG
Sent when a property has been changed by the User
```


Flowcode1
FILE EDIT VIEW MACRO DEBUG BUILD WINDOW HELP
Project explorer
2D: Dashboard Panel
Start Page
Main
Ev_Property
Sent when a property has been changed by the User
BEGIN
END

Now, if you look at your Start page you may still only see **Main** displayed. If that is true, then from the menu **Macro** at the top,

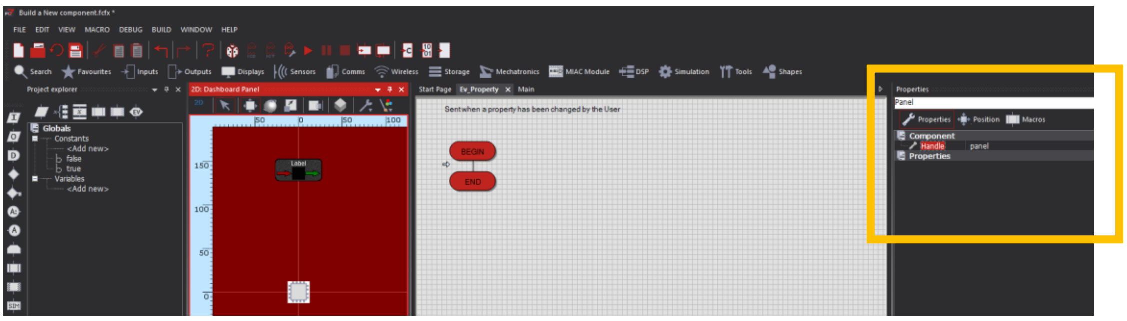
select from the drop-down menu – Show as Flowchart and click **Ev_Property**

Now your Start page will have the Ev_Property **Macro** shown.

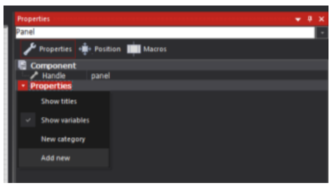
Our component will now have the correct **Properties** and the correct **RETURN** for the event ready to be included into your component. That's our first link done. Don't worry if nothing is making much sense thus far. It is only important that you know this needs to be done to begin building our component. We have placed a link that is needed to take information from the bottom **CAL** layer to make it possible to pick what we need for our new component.

STEP 4 – Selecting Our Parameters and properties

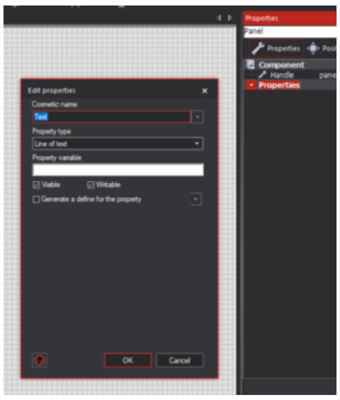
Our screen should look like this shown below. Make sure our new component is selected, just click anywhere inside of the 2D Dashboard to make sure. This way, we know we are working on our component. You will notice our **properties** list is empty.



Click on the **Properties** down arrow and select **Add New**



This will give you the following



The cosmetic name – enter **Channel**

and the property type – **Fixed list of Ints**

Property variable – **CHANNEL** (must be upper case)

Click OK

The cosmetic name is literally cosmetic but the Property variable is case sensitive – be careful!

What we are about to do, is bring across all the instances that are associated with the property called 'CANNEL'., for which there is two (these are the properties for the RS232)

- **Hardware** (fixed by the system) **Software** (we get to pick which pins on the controller to use).

You can now check the following by clicking onto the UART **CAL**, and under properties called **Channel** you will see two option **Software** and **Channel 1**. It is these properties we need to pull across. If we compare that with what we see on our components properties under '**Channel**', the list is empty.

The problem is though, whatever we do with our component in the way of changes, these changes **MUST** be transferred back to the base **CAL** component. This is because the base component is used by the **API** and are therefore **Dynamic**.

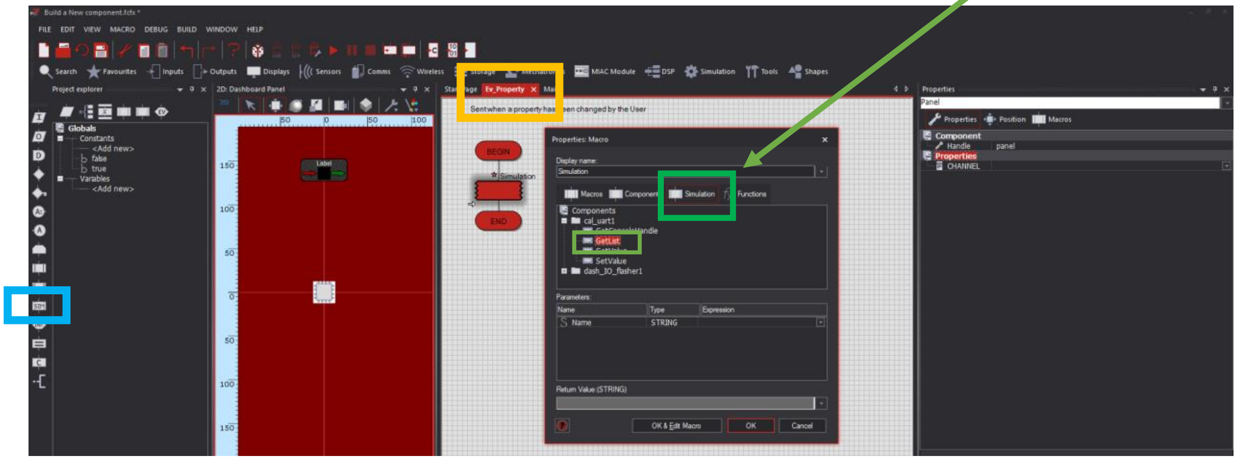
Don't worry if this sound odd, as long as you know this to be true, it will make sense of what we are about to do next to get that dynamic transfer of information, from the **CAL** based component to our component and any changes we do with our component are transferred back to the based **CAL** component. In other words, it must not be possible for our component to be in a different state to the base component. To get this dynamic information flow, there is only one way, coding.

STEP 5 – Building the dynamic properties list

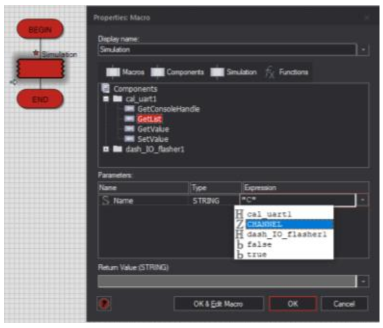
To code the **Ev_Property Macro**, make sure you are working inside this Macro from the start page. This Macro is invisible to the end user when the component is finished. Any coding we do here, is **never** compiled to the chip, it is only used to enable the end user of the component to control its properties and other settings.

From the Flowcode ICON's on the left, select **Simulation** and drag it across as shown below. When you click onto it, you will see both of the following components – the base **cal_uart1** and the **Flasher**.

Select the **cal_uart1** and along the top of the Macro, select **Simulation** as shown here;



Select the option – **GetList**



The Parameter type you need to select from those that are associated with **GetList**. If, like me you have no idea of what they should be, simply place two quotation marks and click between them. Now, when you type a capital letter such as **"C"**, a list appears giving what it expects to be type in this text field – select **"CHANNEL"**

Now we need somewhere to store that information. In the Return Value, we will create and use a Local variable called **.tempstring**

(if it places [20] at the end, remove them using edit)

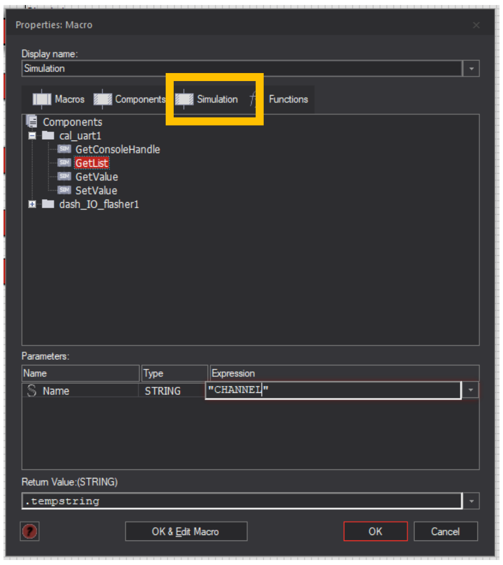
Parameter Name	STRING	"CHANNEL"

Return Value

- Local variable **.tempstring**

Now we need to place that in the Return value box

You can get the Local variable by using the Full Stop It should then appear. If not, you need to create a local variable by that name to store the information you need.



Parameters:

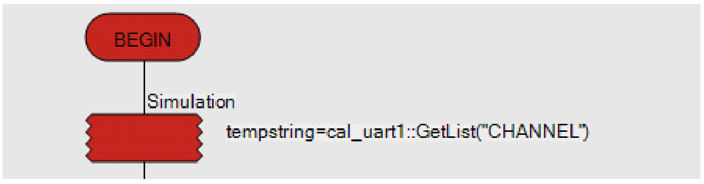
Name	Type	Expression
Name	STRING	"CHANNEL"

Return Value:(STRING)

.tempstring

Now, the Simulation Macro for **cal_uart1** and has **GetList** and the **parameter** called **CHANNEL** with the return value called **.tempstring** containing all the information you need for your component.

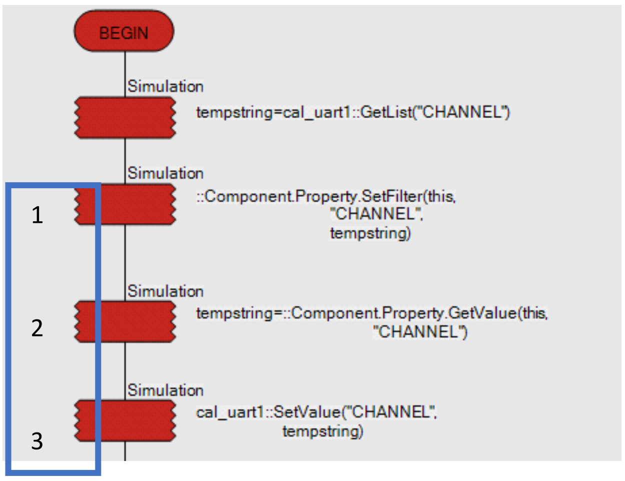
Now we need to define the **Functions** for the above, in other words, what we need to do once we have the parameters stored in the **.tempstring**.



STEP 6 – Define the Functions for the CHANNEL

We now need to use the variable called **.tempstring**

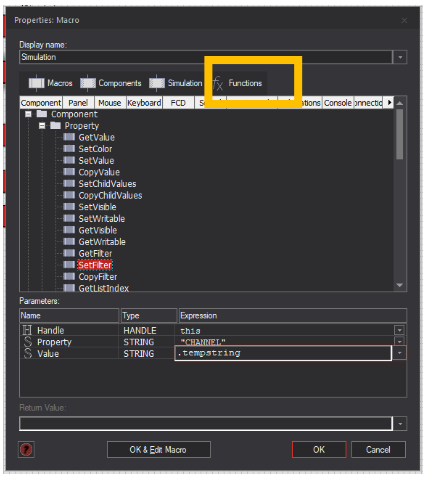
We don't know what is in it, however, we want to extract - **Filter** what it contains and place it in our component. If it is changed, then the contents of the string will change making it possible to put these changes back into the base **CAL.** Our code to do this will look like this:


BEGIN
Simulation
tempstring=cal_uart1::GetList("CHANNEL")
Simulation
::Component.Property.SetFilter(this, "CHANNEL", tempstring)
1
Simulation
tempstring=::Component.Property.GetValue(this, "CHANNEL")
2
Simulation
cal_uart1::SetValue("CHANNEL", tempstring)
3

Insert three more Simulation code icon's and set them as shown below:

- Make sure you click anywhere inside of the 2D Dashboard Panel to work on your component

Make sure the correct TAB **Functions** is selected to make the settings you need.



Name	Type	Expression
Handle	HANDLE	this
Property	STRING	"CHANNEL"
Value	STRING	.tempstring

1 - Under the **Functions TAB–** Select **Component – Property –** then **SetFilter** and for each of the following:

Handle	HANDEL	– **this**
Property	STRING	- **"CHANNEL"**
Value	STRING	- **.tempstring**

And then OK

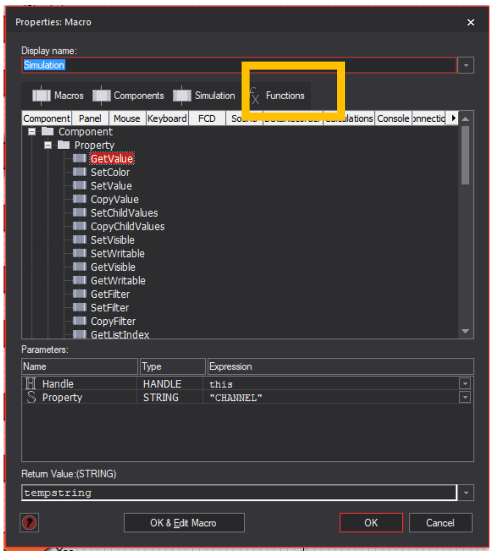
What this Macro does, is to filter out whatever the **CAL** settings are for the CHANNEL from the **.tempstring**

2 – Next Simulation ICON, Under **Functions – Component – Property**
– select **GetValue** and for each of the following

Handle	HANDLE	- **this**
Property	STRING	- **“CHANNEL”**

Return Value .**tempstring**

Click OK



Parameters:

Name	Type	Expression
Handle	HANDLE	this
Property	STRING	"CHANNEL"

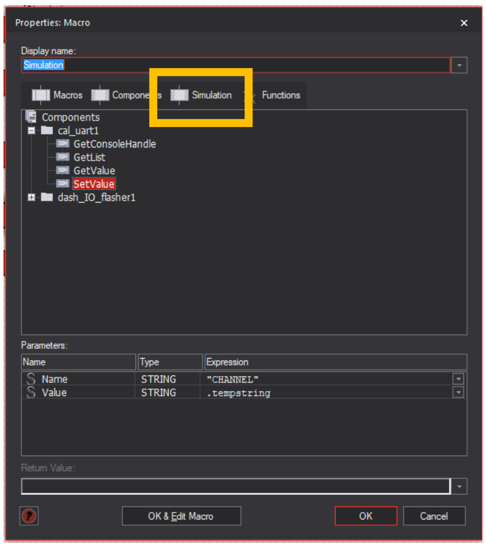
Return Value:(STRING)

tempstring

3 - Make sure **Simulation** TAB is selected and from the option select the **cal_uart1** image first
Now select **Setvalue** and for each of the following

Name	STRING	- **“CHANNEL”**
Value	STRING	-.**tempstring**

Click OK

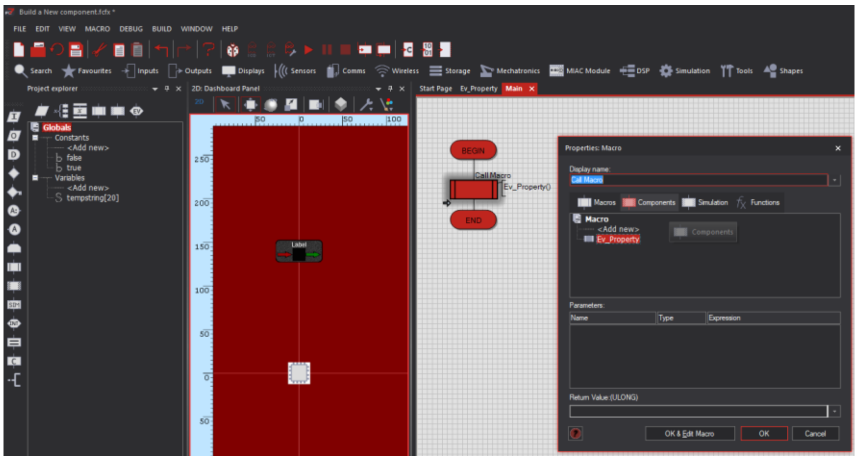


Parameters:

Name	Type	Expression
Name	STRING	"CHANNEL"
Value	STRING	.tempstring

You now have **four** Simulation code ICON’s which
1: copy what’s in the bottom **CAL** layer using **tempstring**
2: places it in our top GUI layer for our component using **Filter**
3: copies whatever changes have been made in our components
4: places these changes into the bottom **CAL** layer using **tempstring**.

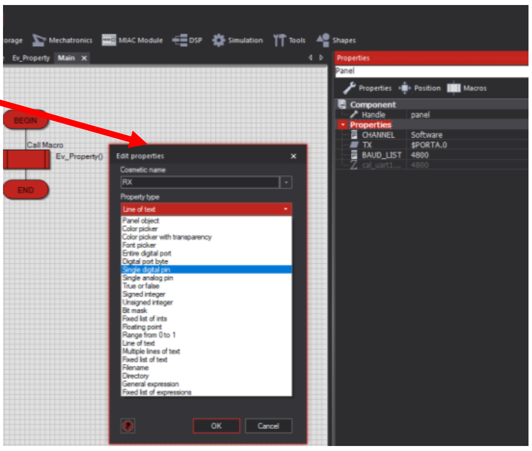
To make this work, you need to place a Macro call in the **Main** Start Page and select the macro **EV_Property**
This will make the CHANNEL selection **Dynamic**



Now we need to put in those physical connections to the Chip. This has been made much easier with FC7. Simply click **Properties** and select add new. For the Cosmetic name **RX** and variable name **RX. The** Property **type, select** Single Digital Pin as shown.

Do the same for the **TX.**

The cosmetic name is exactly that – cosmetic!



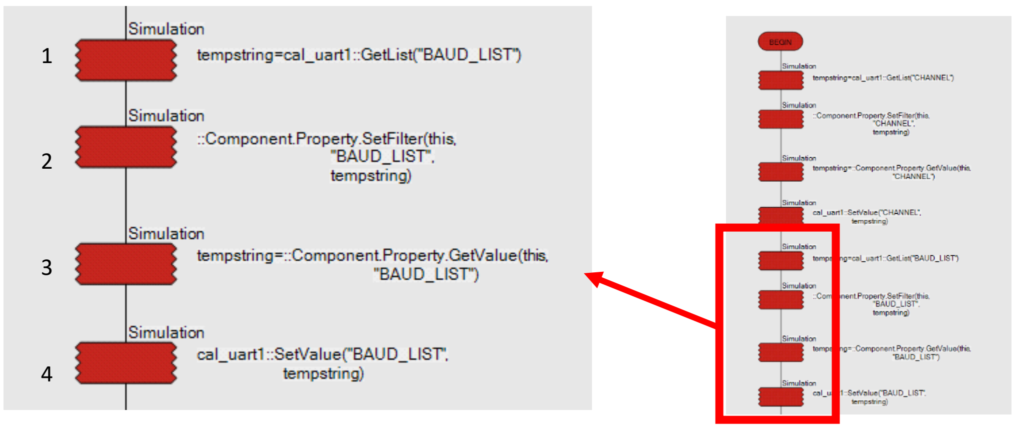
If you press **RUN**, then when you select your component you will see the options of **Channel 1** or **Software**. If you select the software option you will be able to select which pins of the chip are used for the RS232 port. Do that and then click onto the **Cal_uart1** base component icon. Here you will see that the pins match what you have selected by your component. Remember though, you are controlling the connection of the base **CAL** from your component only!

<u>**STEP 7**</u> – Now the Baud Rate

We can repeat the above procedure to get the Baud rate options for our component. Remember, this procedure consists of: -

• From the **CAL**	– **Getlist**
• Our **component**	– **Setfilter**
• Our **component**	**– Getvalue**
• Base **CAL**	**- Setvalue**

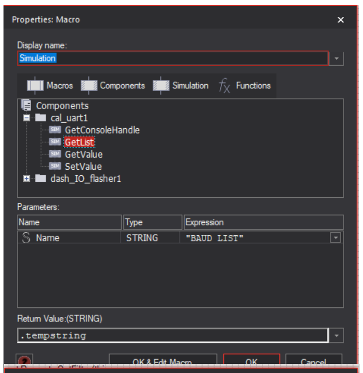
Bring four more simulation icons' across to the EV_Property macro and set them as shown below: -


1
Simulation
tempstring=cal_uart1::GetList("BAUD_LIST")
2
Simulation
::Component.Property.SetFilter(this,
"BAUD_LIST",
tempstring)
3
Simulation
tempstring=::Component.Property.GetValue(this,
"BAUD_LIST")
4
Simulation
cal_uart1::SetValue("BAUD_LIST",
tempstring)
BEGIN
Simulation
tempstring=cal_uart1::GetList("CHANNEL")
Simulation
::Component.Property.SetFilter(this,
"CHANNEL",
tempstring)
Simulation
tempstring=::Component.Property.GetValue(this,
"CHANNEL")
Simulation
cal_uart1::SetValue("CHANNEL",
tempstring)
Simulation
Simulation
Simulation
Simulation

1 - Under the **Simulation TAB–**
Select **cal_uart1**
Property – then **GetList** and for each of the following:

Name	STRING	– **"BAUD_LIST"**
Reurn Value	STRING	- **.tempstring**

Make sure you use the under-score key "_"
And then OK


Properties: Macro
Display name: Simulation

Parameters:

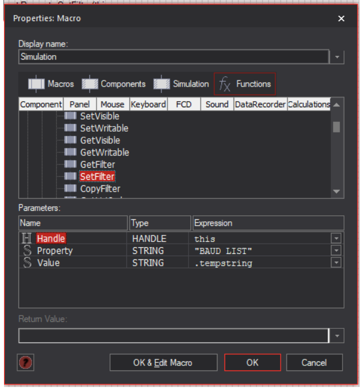
Name	Type	Expression
Name	STRING	"BAUD LIST"

Return Value:(STRING) .tempstring

Select our component by clicking anywhere inside of the 2D Dashboard Panel
2 - Under the **Functions TAB**
Select **Component**
Property – then **SetFilter** and for each of the following:

Handle	HANDEL	– **this**
Property	STRING	- **"BAUD_LIST"**
Value	STRING	- **.tempstring**

And then OK


Properties: Macro
Display name: Simulation

Parameters:

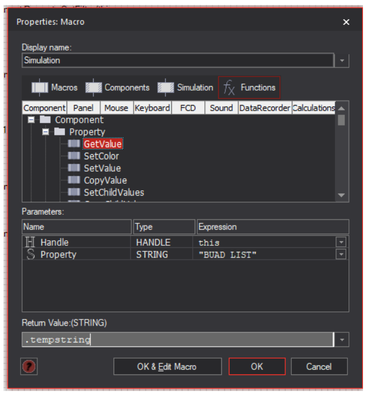
Name	Type	Expression
Handle	HANDLE	this
Property	STRING	"BAUD LIST"
Value	STRING	.tempstring

Return Value:

Select our component
3– Next Simulation ICON, Under **Functions TAB**
Component
Property
select **GetValue** and for each of the following

Handle	HANDLE	- **this**
Property	STRING	- **"BAUD_LIST"**

Return Value .tempstring


Properties: Macro
Display name: Simulation

Parameters:

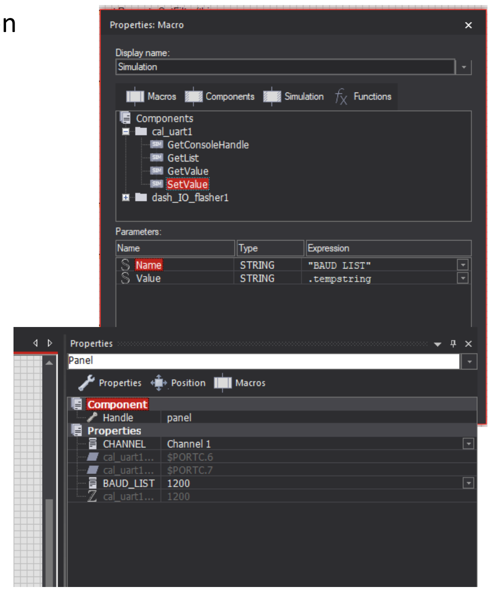
Name	Type	Expression
Handle	HANDLE	this
Property	STRING	"BUAD LIST"

Return Value:(STRING) .tempstring

4- Make sure **Simulation** TAB is selected and from the option select the **cal_uart1**
Now select **Setvalue** and for each of the following

Name	STRING	- **"BAUD_LIST"**
Value	STRING	-.**tempstring**

Click OK


Properties: Macro

Display name: Simulation

Parameters:

Name	Type	Expression
Name	STRING	"BAUD LIST"
Value	STRING	.tempstring

Properties

Panel

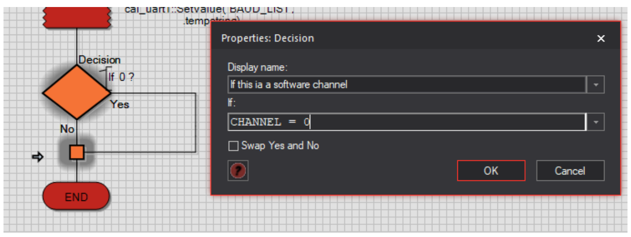
Component	
Handle	panel
Properties	
CHANNEL	Channel 1
cal_uart1...	$PORTC.6
cal_uart1...	$PORTC.7
BAUD_LIST	1200
cal_uart1...	1200

You can check that everything is working ok, by clicking on the Baud rate label under the **cal_uart1** and select **Expose**. This will reveal the baud rate in your component. Now when you change the baud rate the base **CAL UART** baud rate will change to the value you have selected.

What we need to do now is make sure that whichever **Channel** is selected, the data is sent to the correct channel's **Pin connections**– either **Channel1** or **Software.** To do this we first place a channel switch into our **EV_Property** Macro as shown below then send data to the correct pins for that channel. Channel 0 = Software Channel 1 = Hardware

For the software option, the digital pins must be writable!

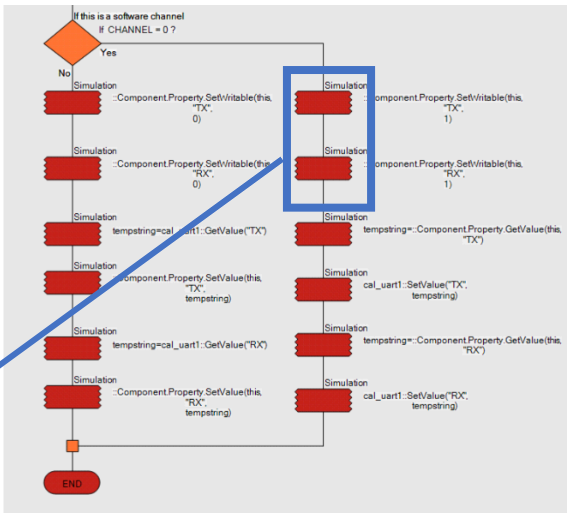
STEP 7 – To insert a **Switch** to control data flow


Properties: Decision

Display name: If this is a software channel

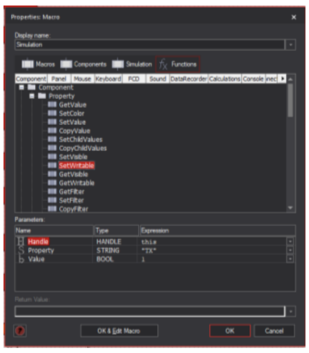
If: CHANNEL = 0

☐ Swap Yes and No


If this is a software channel
If CHANNEL = 0 ?
Yes
No
Simulation
::Component.Property.SetWritable(this, "TX", 0)
::Component.Property.SetWritable(this, "RX", 0)
tempstring=cal_uart1::GetValue("TX")
::Component.Property.SetValue(this, "TX", tempstring)
tempstring=cal_uart1::GetValue("RX")
::Component.Property.SetValue(this, "RX", tempstring)
::Component.Property.SetWritable(this, "TX", 1)
::Component.Property.SetWritable(this, "RX", 1)
tempstring=::Component.Property.GetValue(this, "TX")
cal_uart1::SetValue("TX", tempstring)
tempstring=::Component.Property.GetValue(this, "RX")
cal_uart1::SetValue("RX", tempstring)
END

To set up the data flow, the top two simulation icons are copied across to the other side but the true/false is the opposite value as shown below.

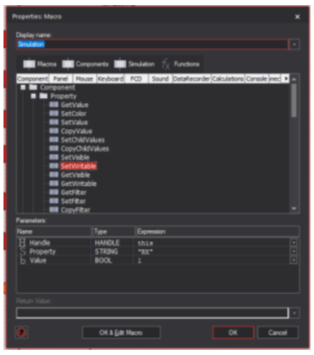
TX **RX**


Properties: Macro

Display name: Simulation

Parameters:

Name	Type	Expression
Handle	HANDLE	this
Property	STRING	"TX"
Value	BOOL	1


Properties: Macro

Display name: Simulation

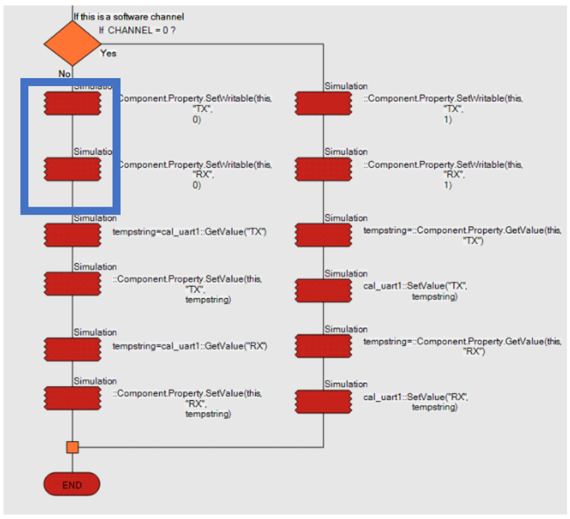
Parameters:

Name	Type	Expression
Handle	HANDLE	this
Property	STRING	"RX"
Value	BOOL	1

Component
Property
SetWritable

Handle	HANDLE	**this**
Property	STRING	**"TX"** or **"RX"**
Value	BOOL	**1**

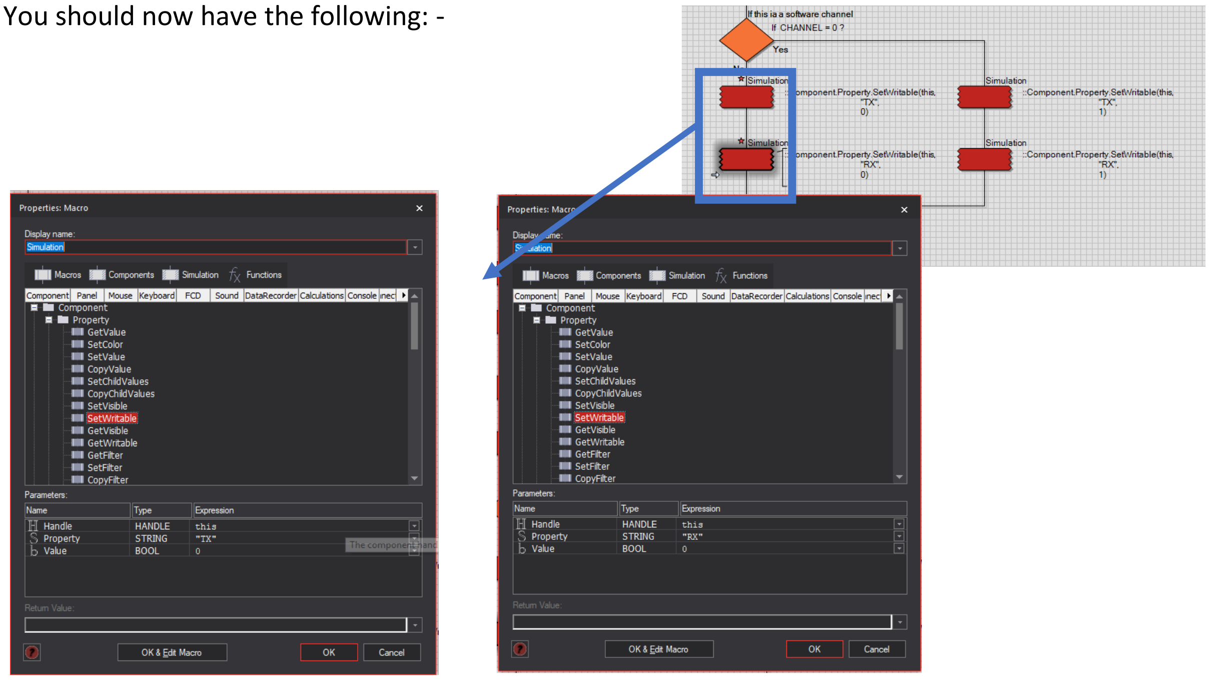
On the other side, copy those icons across and set the Value BOOL to **0**: -


If this is a software channel
If CHANNEL = 0 ?
Yes
No
Simulation
::Component.Property.SetWritable(this, "TX", 0)
::Component.Property.SetWritable(this, "RX", 0)
tempstring=cal_uart1::GetValue("TX")
::Component.Property.SetValue(this, "TX", tempstring)
tempstring=cal_uart1::GetValue("RX")
::Component.Property.SetValue(this, "RX", tempstring)
::Component.Property.SetWritable(this, "TX", 1)
::Component.Property.SetWritable(this, "RX", 1)
tempstring=::Component.Property.GetValue(this, "TX")
cal_uart1::SetValue("TX", tempstring)
tempstring=::Component.Property.GetValue(this, "RX")
cal_uart1::SetValue("RX", tempstring)
END

Component
Property
SetWritable

Handle	HANDLE	**this**
Property	STRING	**"TX"** or **"RX"**
Value	BOOL	**0**

You should now have the following: -



Name	Type	Expression
Handle	HANDLE	this
Property	STRING	"TX"
Value	BOOL	0

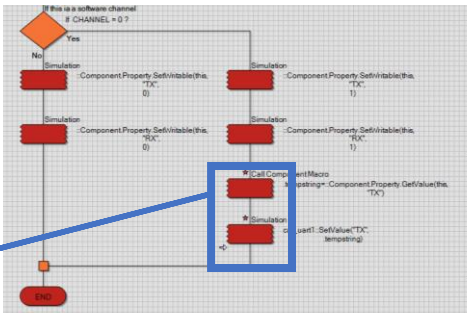
Name	Type	Expression
Handle	HANDLE	this
Property	STRING	"RX"
Value	BOOL	0

What we have achieved now is the ability to write to these **chip connections**, whatever Channel we use – **Channel1** or **Software.** We do not know what these pins are yet so we will need to set them up.

If either of the channels is set to **Channel1** or **Software**, yes we can write to them, but the **API** needs to know to which pin in the chip they are connected, or even what device is being used. This is of course another **Dynamic** property. To get this dynamic feature working, we need to check (**GetValue**) and then (**SetValue**) for both the TX and the RX pins. These will of course be different for when either option (**Channel1 – Software**) is selected and vice versa. If you study the following solution, it should become clear to how this all works, hopefully.

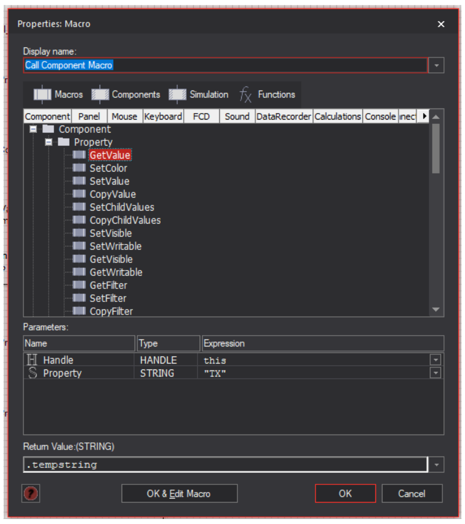
STEP 8 – Get the pin connections and set appropriately

Place two more simulation icons as shown and set to the following: -


If this ia a software channel
If CHANNEL = 0 ?
Yes
No
Simulation
::Component.Property.SetWritable(this, "TX", 0)
Simulation
::Component.Property.SetWritable(this, "TX", 1)
Simulation
::Component.Property.SetWritable(this, "RX", 0)
Simulation
::Component.Property.SetWritable(this, "RX", 1)
Call Component Macro
.tempstring=::Component.Property.GetValue(this, "TX")
Simulation
cal_uart1::SetValue("TX", .tempstring)
END

Component
Property
GetValue

Handle	HANDLE	**this**
Property	STRING	**"TX"**
Return Value	**.tempstring**	



Properties: Macro

Display name: Call Component Macro

Macros | Components | Simulation | Functions

Component | Panel | Mouse | Keyboard | FCD | Sound | DataRecorder | Calculations | Console | inec

- Component
  - Property
    - GetValue
    - SetColor
    - SetValue
    - CopyValue
    - SetChildValues
    - CopyChildValues
    - SetVisible
    - SetWritable
    - GetVisible
    - GetWritable
    - GetFilter
    - SetFilter
    - CopyFilter

Parameters:

Name	Type	Expression
Handle	HANDLE	this
Property	STRING	"TX"

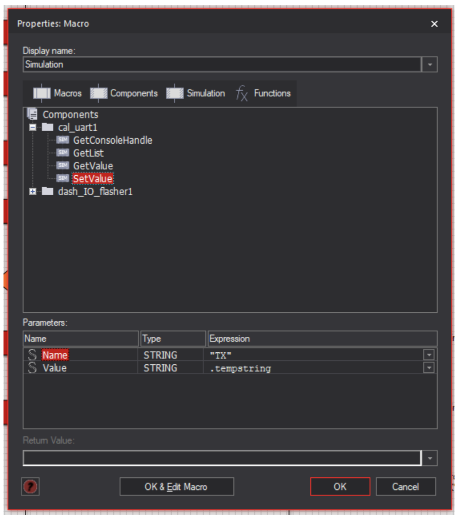
Return Value:(STRING)

.tempstring

OK & Edit Macro | OK | Cancel

Cal_uart1
SetValue

Name	STRING	"TX"
Value	STRING	**.tempstring**



Properties: Macro

Display name: Simulation

Macros | Components | Simulation | Functions

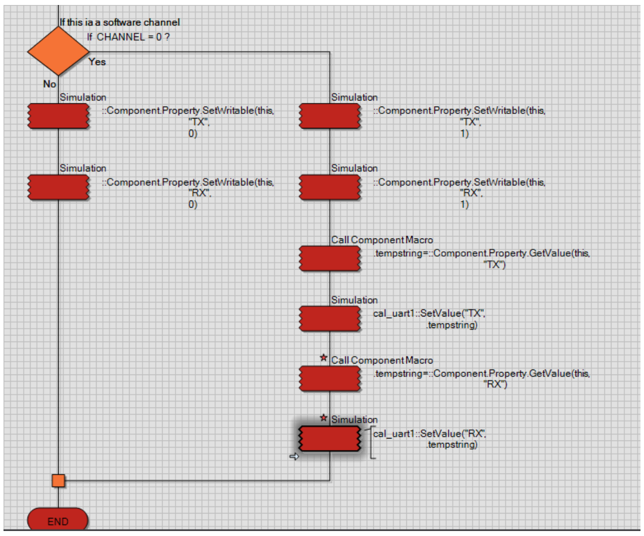
- Components
  - cal_uart1
    - GetConsoleHandle
    - GetList
    - GetValue
    - SetValue
  - dash_IO_flasher1

Parameters:

Name	Type	Expression
Name	STRING	"TX"
Value	STRING	.tempstring

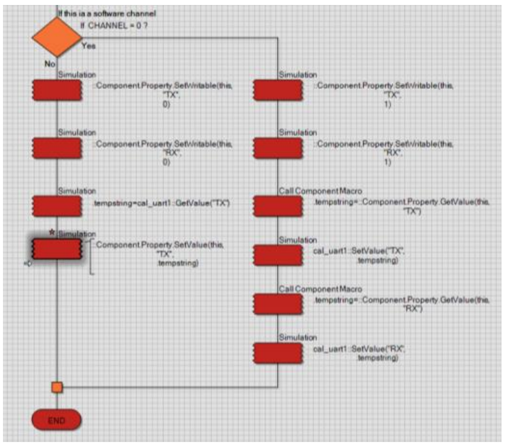
Return Value:

OK & Edit Macro | OK | Cancel


If this ia a software channel
If CHANNEL = 0 ?
Yes
No
Simulation
::Component.Property.SetWritable(this, "TX", 0)
Simulation
::Component.Property.SetWritable(this, "TX", 1)
Simulation
::Component.Property.SetWritable(this, "RX", 0)
Simulation
::Component.Property.SetWritable(this, "RX", 1)
Call Component Macro
.tempstring=::Component.Property.GetValue(this, "TX")
Simulation
cal_uart1::SetValue("TX", .tempstring)
Call Component Macro
.tempstring=::Component.Property.GetValue(this, "RX")
Simulation
cal_uart1::SetValue("RX", .tempstring)
END

Repeat the above two icon following on but this time for the **"RX"** pin.
You should have this
Now, what ever you chose, software or hardware as the **Channel**, the setting are copied from what you want to the base **CAL** (code Abstraction layer)

Now we need to set up the other side to do the switching around of the data for the other option. Start by placing another two simulation icons on the other side like this.

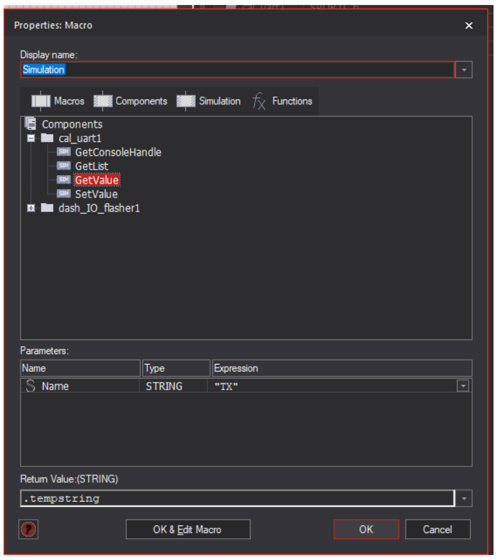

If this is a software channel
If CHANNEL = 0 ?
Yes
No
Simulation
::Component.Property.SetWritable(this, "TX", 0)
Simulation
::Component.Property.SetWritable(this, "RX", 0)
Simulation
.tempstring=cal_uart1::GetValue("TX")
Simulation
::Component.Property.SetValue(this, "TX", .tempstring)
Simulation
::Component.Property.SetWritable(this, "TX", 1)
Simulation
::Component.Property.SetWritable(this, "RX", 1)
Call Component Macro
.tempstring=::Component.Property.GetValue(this, "TX")
Simulation
cal_uart1::SetValue("TX", .tempstring)
Call Component Macro
.tempstring=::Component.Property.GetValue(this, "RX")
Simulation
cal_uart1::SetValue("RX", .tempstring)
END

This time they are the other way round. We start with the **cal_uart1** first followed by the **component**- GetValue and then SetValue as shown below: -

This is because, if the **Hardware Channel1** is selected, these setting need to copied into our component settings.

cal_uart1
GetValue

Name	STRING	"TX"
Return Value		.tempstring


Properties: Macro

Display name:
Simulation

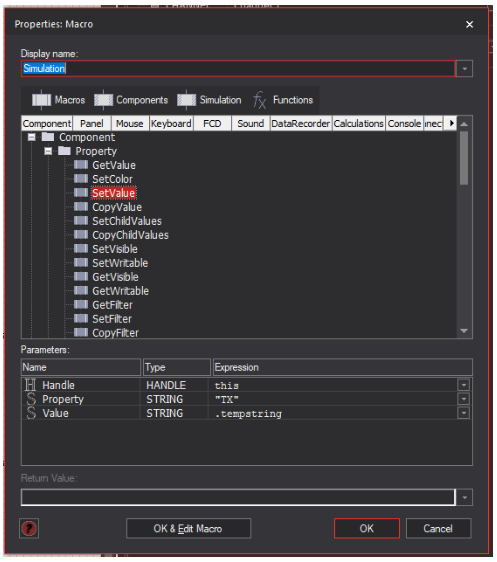
Parameters:

Name	Type	Expression
Name	STRING	"TX"

Return Value:(STRING)
.tempstring

Component
Property
SetValue

Handle	HANDLE	this
Property	STRING	"TX"
Value	STRING	.tempstring


Properties: Macro

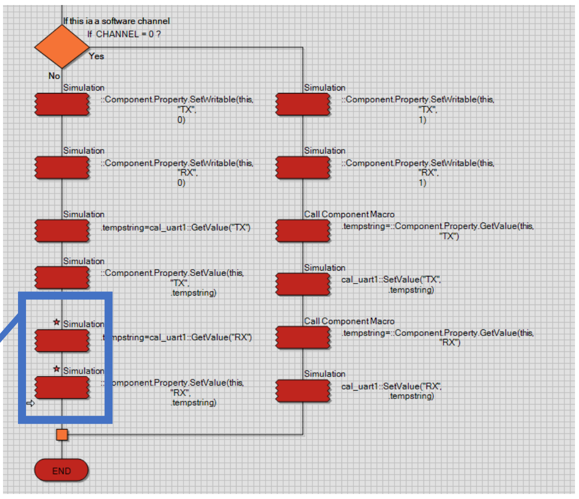
Display name:
Simulation

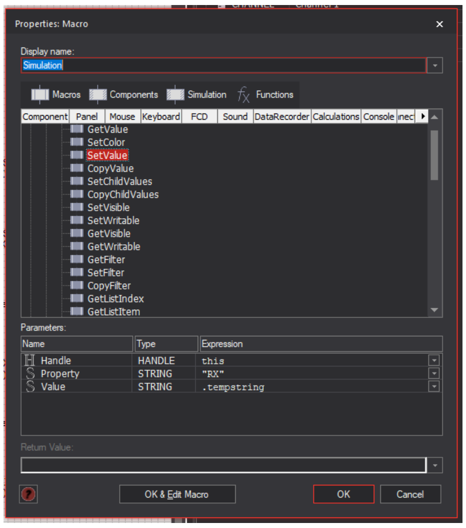
Parameters:

Name	Type	Expression
Handle	HANDLE	this
Property	STRING	"TX"
Value	STRING	.tempstring

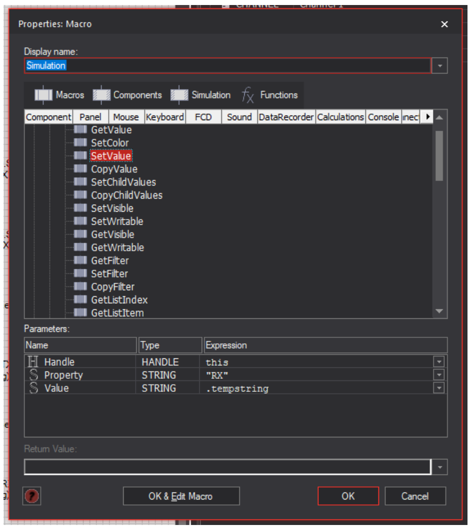
Return Value:

Copy those last two simulation icons and paste below what you have just done and change the **TX** to **RX**. You should have the following setup as shown.


If this ia a software channel
If CHANNEL = 0 ?
Yes
No
Simulation
::Component.Property.SetWritable(this, "TX", 0)
Simulation
::Component.Property.SetWritable(this, "RX", 0)
Simulation
.tempstring=cal_uart1::GetValue("TX")
Simulation
::Component.Property.SetValue(this, "TX", .tempstring)
Simulation
.tempstring=cal_uart1::GetValue("RX")
Simulation
::Component.Property.SetValue(this, "RX", .tempstring)
Simulation
::Component.Property.SetWritable(this, "TX", 1)
Simulation
::Component.Property.SetWritable(this, "RX", 1)
Call Component Macro
.tempstring=::Component.Property.GetValue(this, "TX")
Simulation
cal_uart1::SetValue("TX", .tempstring)
Call Component Macro
.tempstring=::Component.Property.GetValue(this, "RX")
Simulation
cal_uart1::SetValue("RX", .tempstring)
END



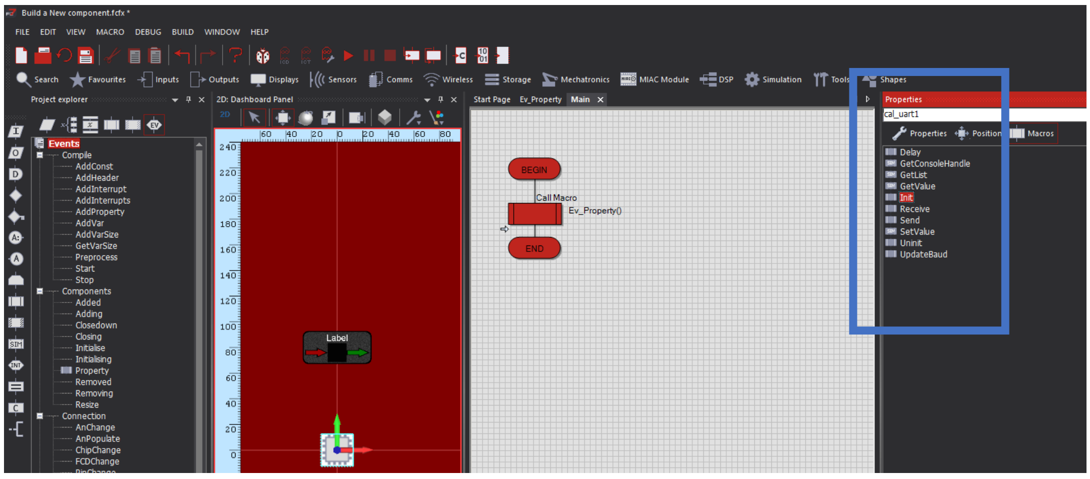
Name	Type	Expression
Handle	HANDLE	this
Property	STRING	"RX"
Value	STRING	.tempstring



Name	Type	Expression
Handle	HANDLE	this
Property	STRING	"RX"
Value	STRING	.tempstring

That completes the most difficult part of the coding for **EV_Property** Macro. All we need to do now is add some other Macros to give our component some useful functionality.

If you click onto the base **CAL – cal_uart1**, the select from the **Properties** explorer – **Macros**, you should see a list of all the macros that this base cal has available.

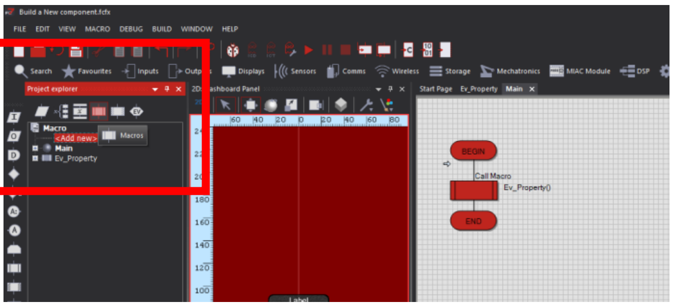


Events
- Compile
  - AddConst
  - AddHeader
  - AddInterrupt
  - AddInterrupts
  - AddProperty
  - AddVar
  - AddVarSize
  - GetVarSize
  - Preprocess
  - Start
  - Stop
- Components
  - Added
  - Adding
  - Closedown
  - Closing
  - Initialise
  - Initialising
  - Property
  - Removed
  - Removing
  - Resize
- Connection
  - AnChange
  - AnPopulate
  - ChipChange
  - FCDChange

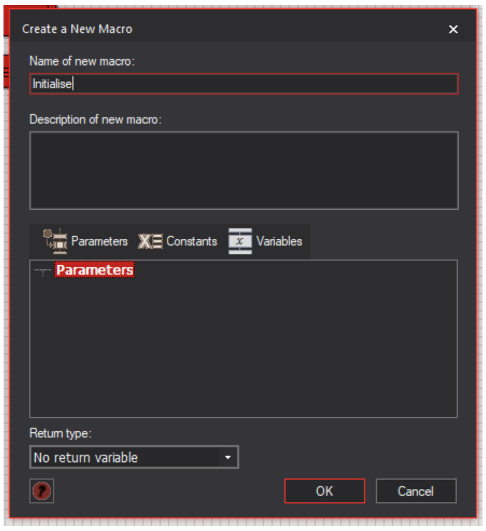
Macros:
- Delay
- GetConsoleHandle
- GetList
- GetValue
- Init
- Receive
- Send
- SetValue
- Uninit
- UpdateBaud

STEP 9 – Building the Interface for this Component

The **EV_Property** Marco will be invisible to the user, however, we need to build some Macros for the component to be downloadable for use in your projects, such as to enable us to **Initialise; Send data; Receive Data** and finally for this example **a TX_Sync** Macro. To include Macros for the component, select the Macro TAB from the Project Explorer and select **ADD New**.


Build a New component.fcfx
FILE EDIT VIEW MACRO DEBUG BUILD WINDOW HELP
Search Favourites Inputs Outputs Displays Sensors Comms Wireless Storage Mechatronics MIAC Module DSP
Project explorer
2D Dashboard Panel
Start Page Ev_Property Main
Macro
<Add new>
Macros
Main
Ev_Property
BEGIN
Call Macro
Ev_Property()
END
Label

The fist Macro to add is – **Initialise**
Select - add new macro
Name of Macro - Initialise
This does not have a Return value.


Create a New Macro

Name of new macro: Initialise

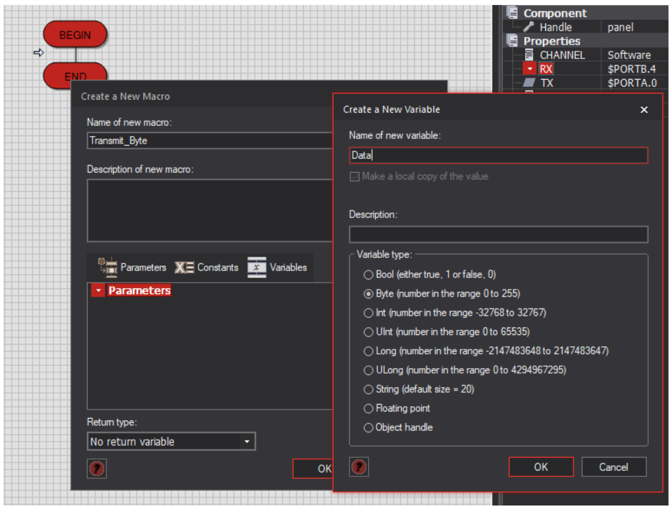
Description of new macro:

Return type: No return variable

Transmit_Byte

We do need a Parameter for this Macro.
To do this we select add new.
Name of Macro – Transmit_Byte
Call this Parameter – Data and set it to – **Byte**

Click OK
There is no return value from a send
We could add a description of what it does later!



Create a New Macro

Name of new macro: Transmit_Byte

Description of new macro:

Return type: No return variable

Create a New Variable

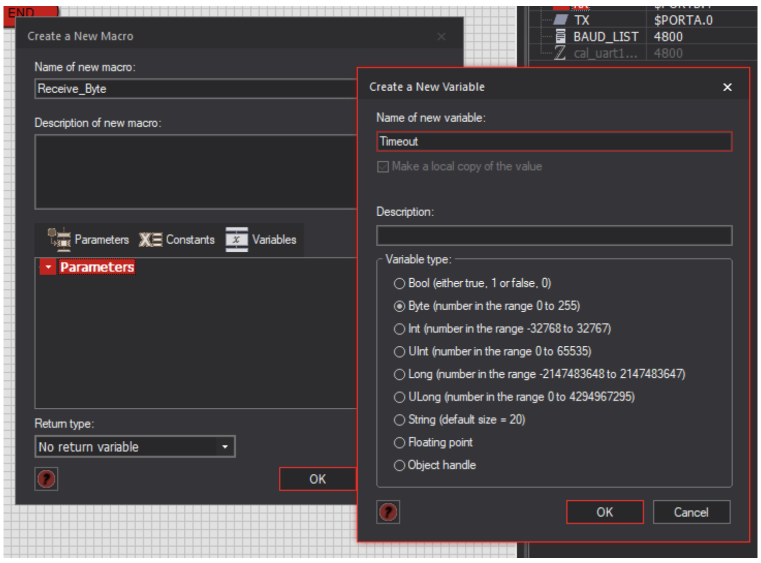
Name of new variable: Data

☐ Make a local copy of the value

Description:

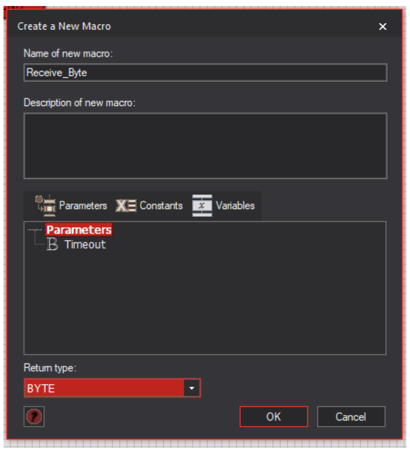
Variable type:
- ○ Bool (either true, 1 or false, 0)
- [✓ ● Byte (number in the range 0 to 255)]
- ○ Int (number in the range -32768 to 32767)
- ○ UInt (number in the range 0 to 65535)
- ○ Long (number in the range -2147483648 to 2147483647)
- ○ ULong (number in the range 0 to 4294967295)
- ○ String (default size = 20)
- ○ Floating point
- ○ Object handle

Receive_Byte
Macro name – Receive_Byte
Parameter - Timeout
To set up this Macro we need a Parameter called Timeout. This is so the base **CAL** will wait for incoming data.
Select add new from Parameter and name it **Timeout** with a byte variable.
Click ok


Create a New Macro
Name of new macro: Receive_Byte
Description of new macro:
Parameters
Return type: No return variable

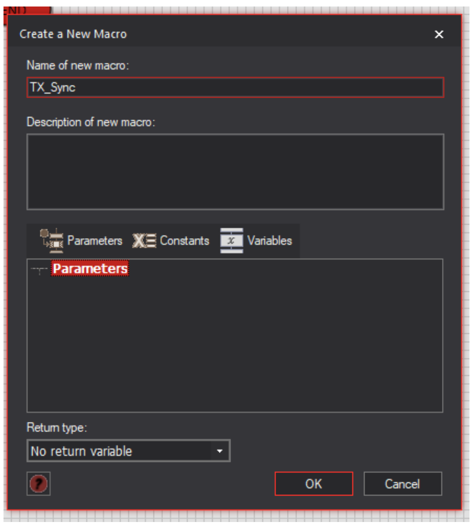
TX $PORTA.0
BAUD_LIST 4800
cal_uart1... 4800

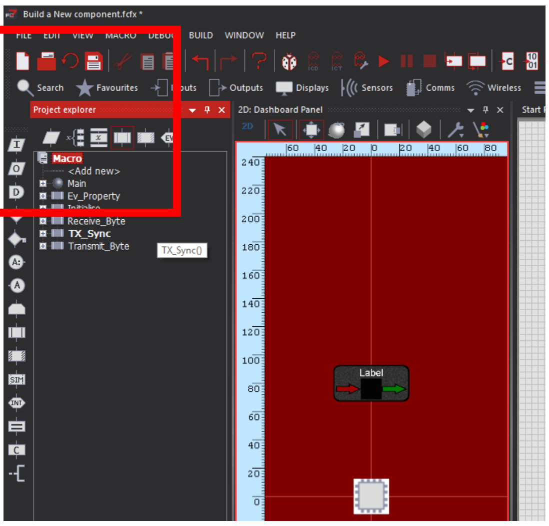
Create a New Variable
Name of new variable: Timeout
Make a local copy of the value
Description:
Variable type:
- Bool (either true, 1 or false, 0)
- [✓ Byte (number in the range 0 to 255)]
- Int (number in the range -32768 to 32767)
- UInt (number in the range 0 to 65535)
- Long (number in the range -2147483648 to 2147483647)
- ULong (number in the range 0 to 4294967295)
- String (default size = 20)
- Floating point
- Object handle


Create a New Macro
Name of new macro: Receive_Byte
Description of new macro:
Parameters
B Timeout
Return type: BYTE

Then we need somewhere to store the Returned data select Return Type and select **BYTE**

The last Macro I'm going to add is the transmit synchronisation pulses that are needed to get the receiver to latch onto the transmitter. Add a new Macro called **TX_Sync**
There are no Parameters needed for this Macro, but we do need to add some functionality as I will show you how.


Create a New Macro
Name of new macro: TX_Sync
Description of new macro:
Parameters
Return type: No return variable

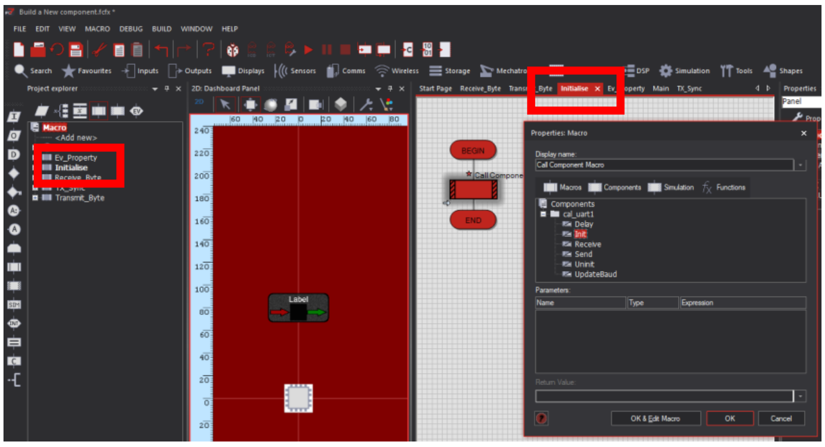


What we should end up with for our Macros.
EV_Property
Initialise
Receive_Byte
TX_Sync
Transmit_Byte

For each of the Macros, to add that important functionality, simply click onto each Macro and insert the **base components CAL macro** you want i.e.

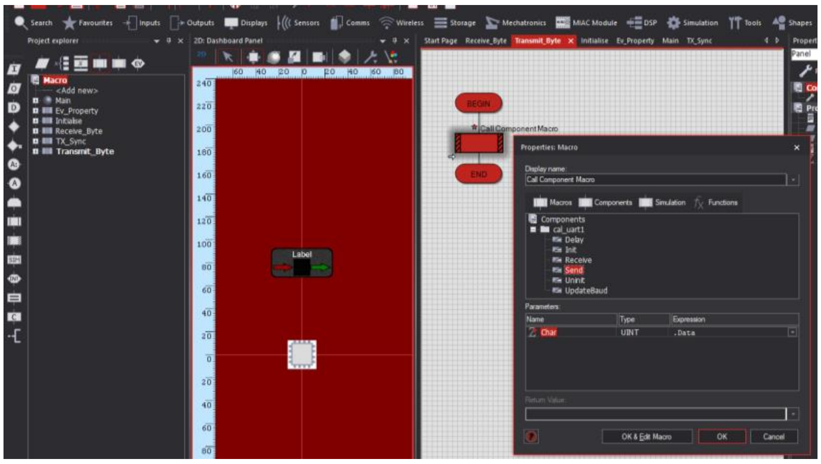
Initialise –
insert the CAL initialise macro
Int

click ok



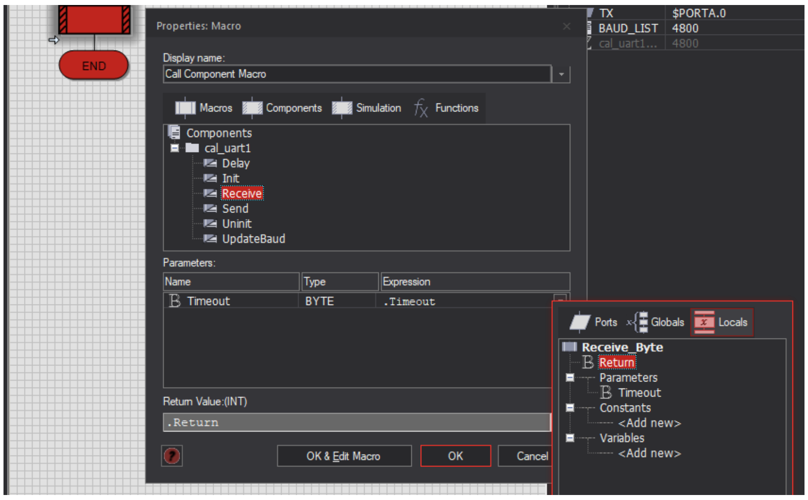
Transmit_Byte
Insert - send
The Parameter is set to
.Data

Click ok



Receive_Byte Macro
Insert - Receive
Parameter – **.Timeout**
Use the locals to find your variables

The Return is –
.Return

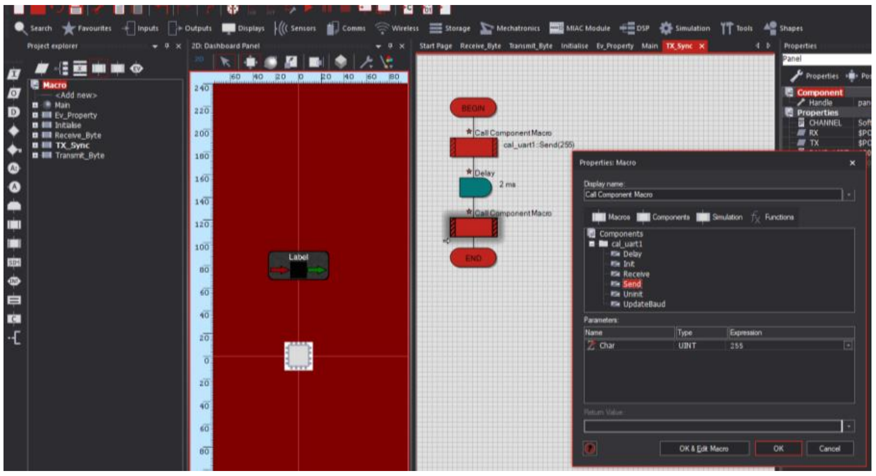


TX_Sync

Use the Macro **Cal_uart**
Insert - Send

And add the decimal value 240 into the Char field

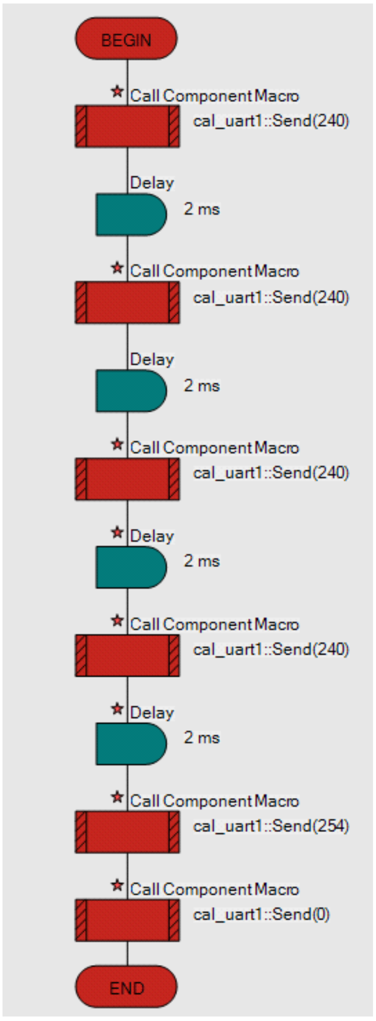
You can also add some delays directly to build the sync pattern you want.



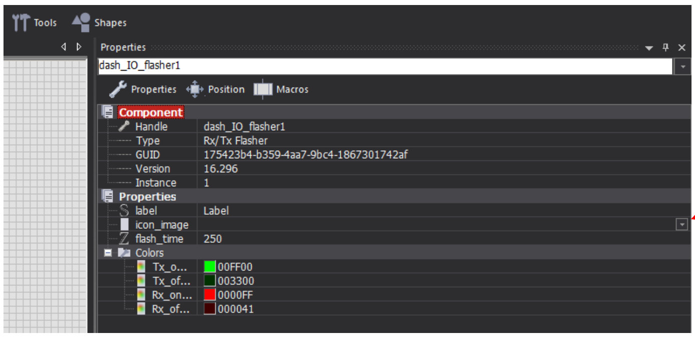
I add the following send macros with the values shown.

Send	**- 240**
Delay	**- 2ms**
Send	**- 240**
Delay	**- 2ms**
Send	**- 240**
Delay	**- 2ms**
Send	**- 240**
Delay	**- 2ms**
Send	**- 254**
Send	**- 0**

You can experiment with whatever sync pattern you think will work best.

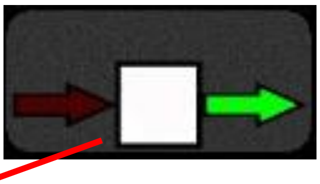


<u>STEP 10</u> – Your component Image



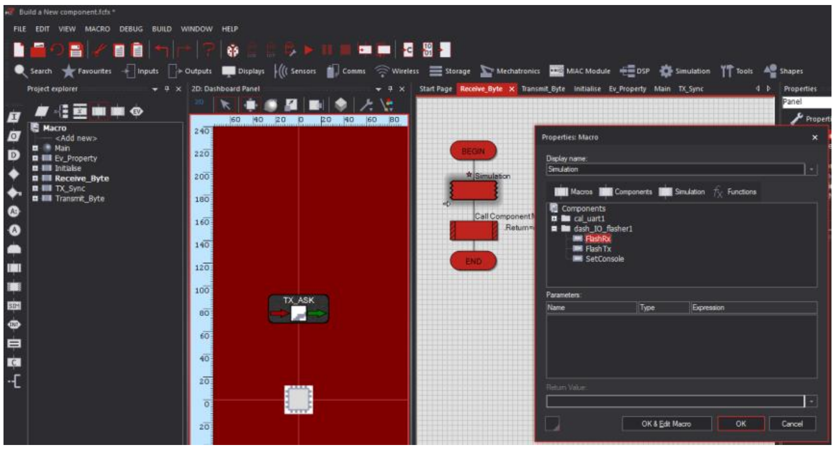
dash_IO_flasher1

Component	
Handle	dash_IO_flasher1
Type	Rx/Tx Flasher
GUID	175423b4-b359-4aa7-9bc4-1867301742af
Version	16.296
Instance	1
Properties	
label	Label
icon_image	
flash_time	250
Colors	
Tx_o...	🟩 00FF00
Tx_of...	■ 003300
Rx_on...	🟥 0000FF
Rx_of...	■ 000041



Choose a suitable icon image to be placed in the black square and set the flash rate to something suitable – just so it stays on long enough when activated in your macros. Give the Flasher a suitable label. Now you are ready to insert it into your component.

The Macro – **Receive_Byte**, insert a simulation macro icon at the top, select the flasher from the list and set it to flash the Receive LED. That's all you need to do.
Do the same for the Transmit Macro to flash the transmit LED.


Build a New component.fcfx *
FILE EDIT VIEW MACRO DEBUG BUILD WINDOW HELP
Search Favourites Inputs Outputs Displays Sensors Comms Wireless Storage Mechatronics MIAC Module DSP Simulation Tools Shapes
Project explorer
2D: Dashboard Panel
Start Page Receive_Byte Transmit_Byte Initialise Ev_Property Main TX_Sync
Macro
<Add new>
Main
Ev_Property
Initialise
Receive_Byte
TX_Sync
Transmit_Byte
TX_ASK
BEGIN
Simulation
Call Component Macro
END
Properties
Panel

Properties: Macro

Display name: Simulation

Macros | Components | Simulation | Functions

- Components
  - cal_uart1
  - dash_IO_flasher1
    - FlashRx
    - FlashTx
    - SetConsole

Parameters:

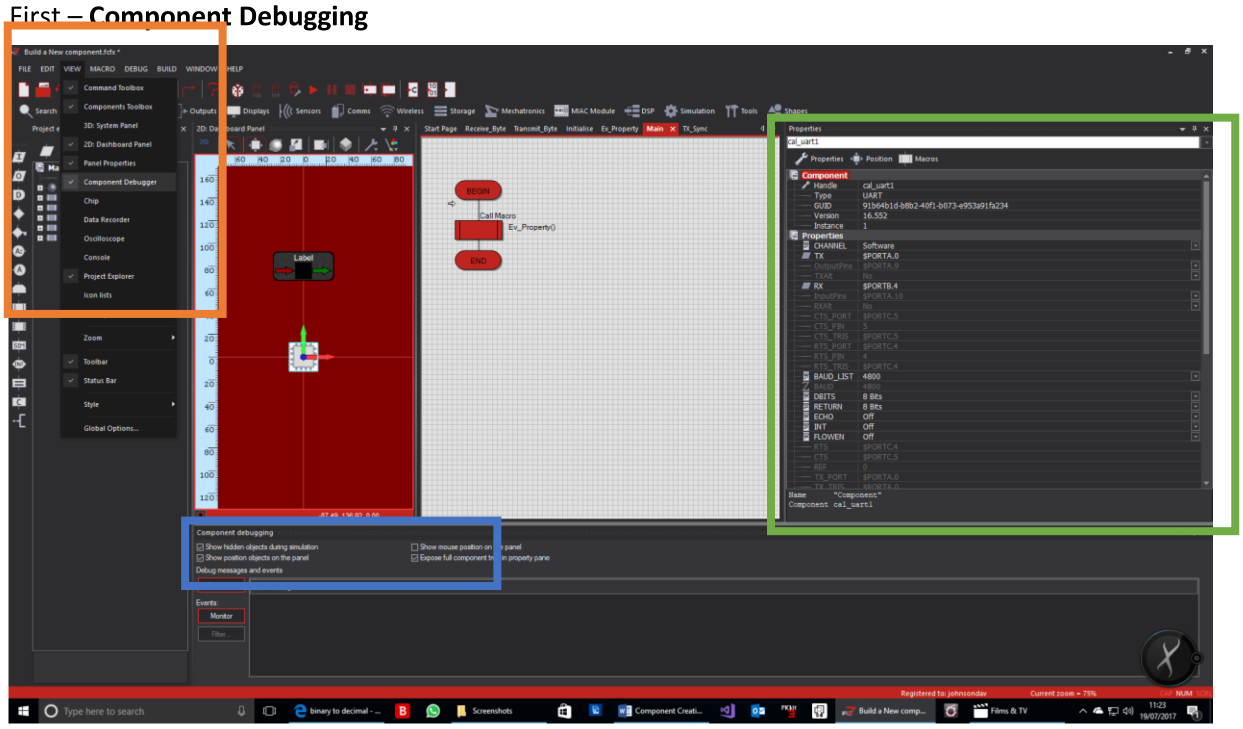
Name	Type	Expression

Return Value:

OK & Edit Macro | OK | Cancel

STEP 11 – Useful tools included in Flowcode 7

First – **Component Debugging**



Properties: cal_uart1

Properties | Position | Macros

Component	
Handle	cal_uart1
Type	UART
GUID	91b64b1d-b8b2-40f1-b073-e953a91fa234
Version	16.552
Instance	1
Properties	
CHANNEL	Software
TX	$PORTA.0
OutputPins	$PORTA.9
TXAlt	No
RX	$PORTB.4
InputPins	$PORTA.10
RXAlt	No
CTS_PORT	$PORTC.5
CTS_PIN	5
CTS_TRIS	$PORTC.5
RTS_PORT	$PORTC.4
RTS_PIN	4
RTS_TRIS	$PORTC.4
BAUD_LIST	4800
BAUD	4800
DBITS	8 Bits
RETURN	8 Bits
ECHO	Off
INT	Off
FLOWEN	Off
RTS	$PORTC.4
CTS	$PORTC.5
REF	0
TX_PORT	$PORTA.0
TX_TRIS	$PORTA.0

```
Name      "Component"
Component cal_uart1
```

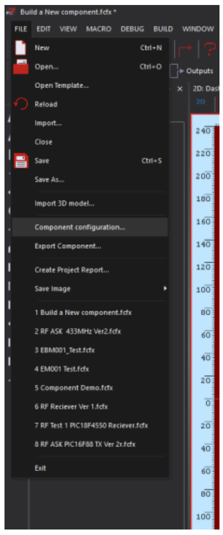
Component debugging

- ☑ Show hidden objects during simulation
- ☑ Show position objects on the panel
- ☐ Show mouse position on the panel
- ☑ Expose full component tree in property pane

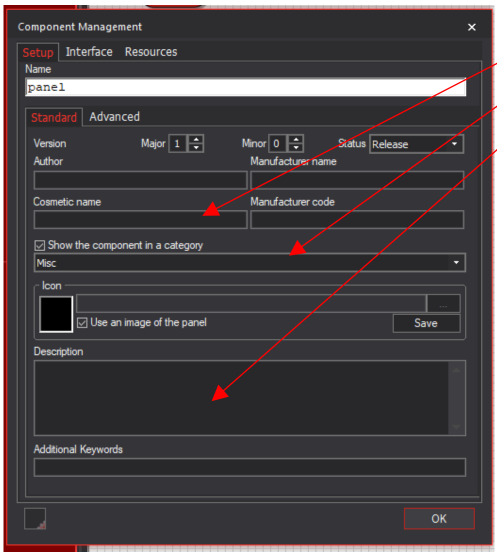
Debug messages and events

You will find this under View
When ticked, this will allow you to inspect any of the component properties, just make sure the appropriate check boxes are ticked as shown.

Now we need to make sure our component is configured correctly. The component configuration tool can be found here under File:



Here you need to enter all of your own details such as Author



Cosmetic Name

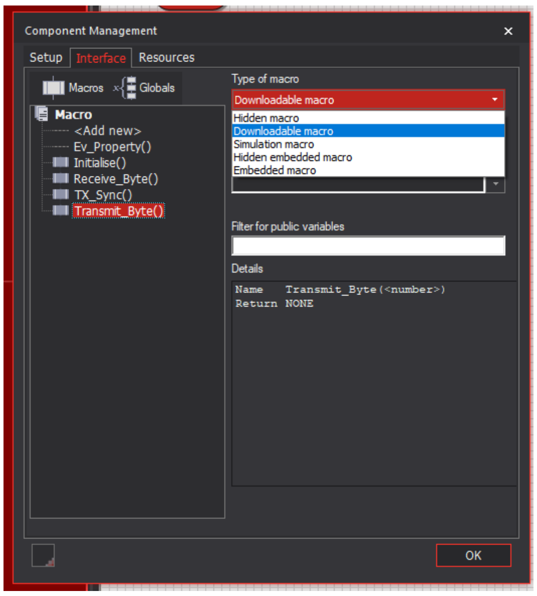
Location to be display (choose from list)

Description

Manufacturer Number

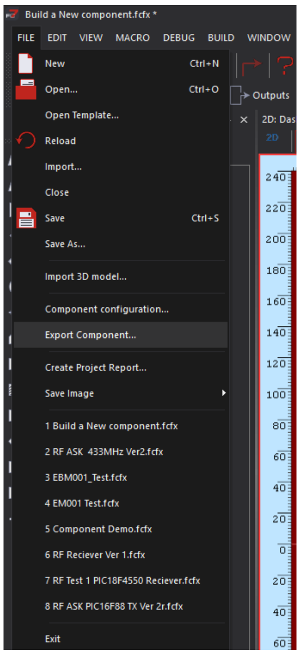
The **Interface** tool allows you to set which macros are downloadable or hidden as shown.

Make the Ev_Property macro HIDDEN

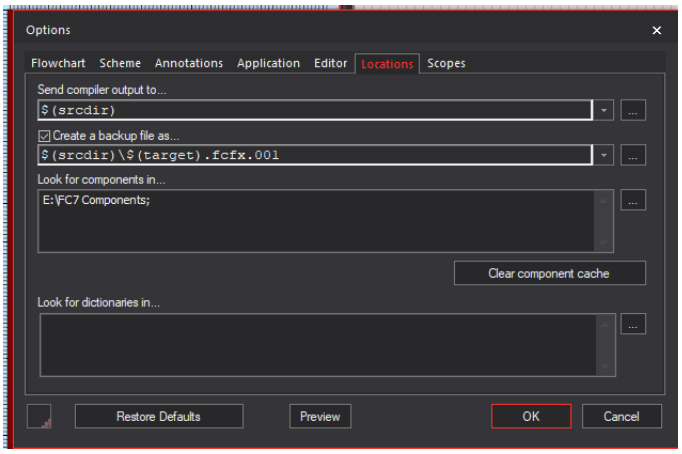


```
Name   Transmit_Byte(<number>)
Return NONE
```

All that is required now is to export the component you have just finished. Give it a suitable name and location where FC7 can find it as shown here.



View – Global Options


Options

Flowchart Scheme Annotations Application Editor Locations Scopes

Send compiler output to...
$(srcdir)

☑ Create a backup file as...
$(srcdir)\$(target).fcfx.001

Look for components in...
E:\FC7 Components;

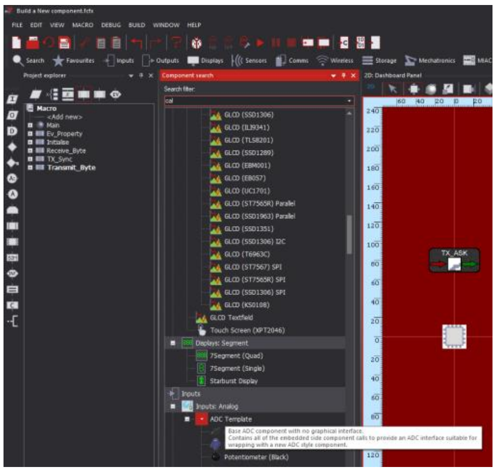
Clear component cache

Look for dictionaries in...

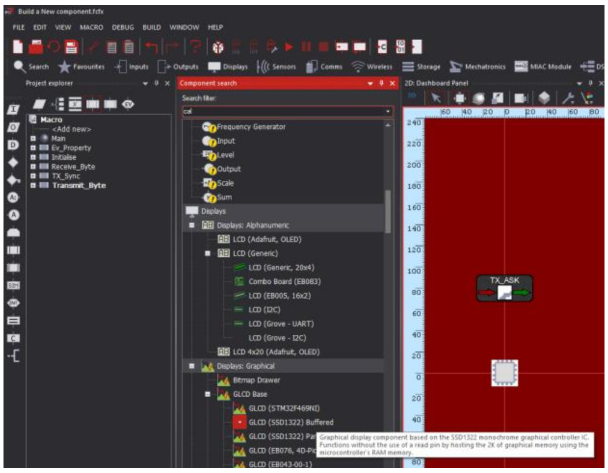
Restore Defaults Preview OK Cancel

STEP 11 – Other CAL's available

Analogue to Digital



gLCD



There are other available too

- 1 CAL Components
  - 1.1 CAL ADC
  - 1.2 CAL CAN
  - 1.3 CAL EEPROM
  - 1.4 CAL I2C
  - 1.5 CAL PWM
  - 1.6 CAL SPI
  - 1.7 CAL UART