M1 Build

TRAINING CONFERENCE 2015

Published July 2015

www.motec.com

Race smart.

INDEX 2015 Training Reference

M1 Build

TRAINING CONFERENCE 2015

M1 Build

Content and Purpose

This training course is intended to give users some experience in the process of using M1 Build to modify M1 ECU firmwares.

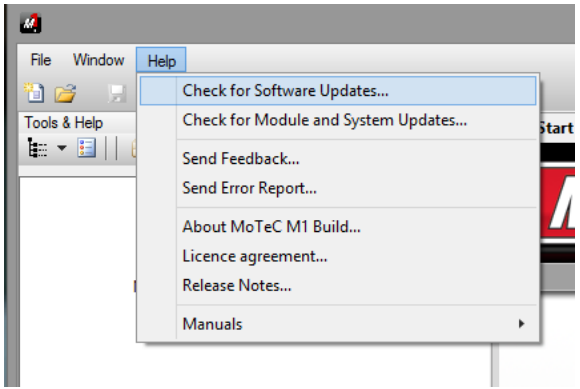
The information we are presenting here is designed to work in conjunction with the M1 Build User Manual, which can be downloaded when you install M1 Build.

These training notes will walk users through the M1 Build process, while the M1 Build User Manual is a detailed reference guide to the entire product and its features.

You should refer to the reference guide for additional information if you are having trouble.

M1 Build Software

- The M1 Build software is available on MoTeC online.
- The latest version of M1 build can be found here:
  - https://moteconline.motec.com.au/Home/Downloads
- Download and install the software
- Once you have the software installed it is always worthwhile to keep it up to date. With M1 Build, just click Help/Check for Software Updates when online to ensure that you are using the latest version.



Overview

- Build is a powerful program used to generate firmware for M1 series ECUs. It includes a sophisticated software editor with predefined and simplified possibilities to provide a user-friendly, timesaving and elaborated programming experience.

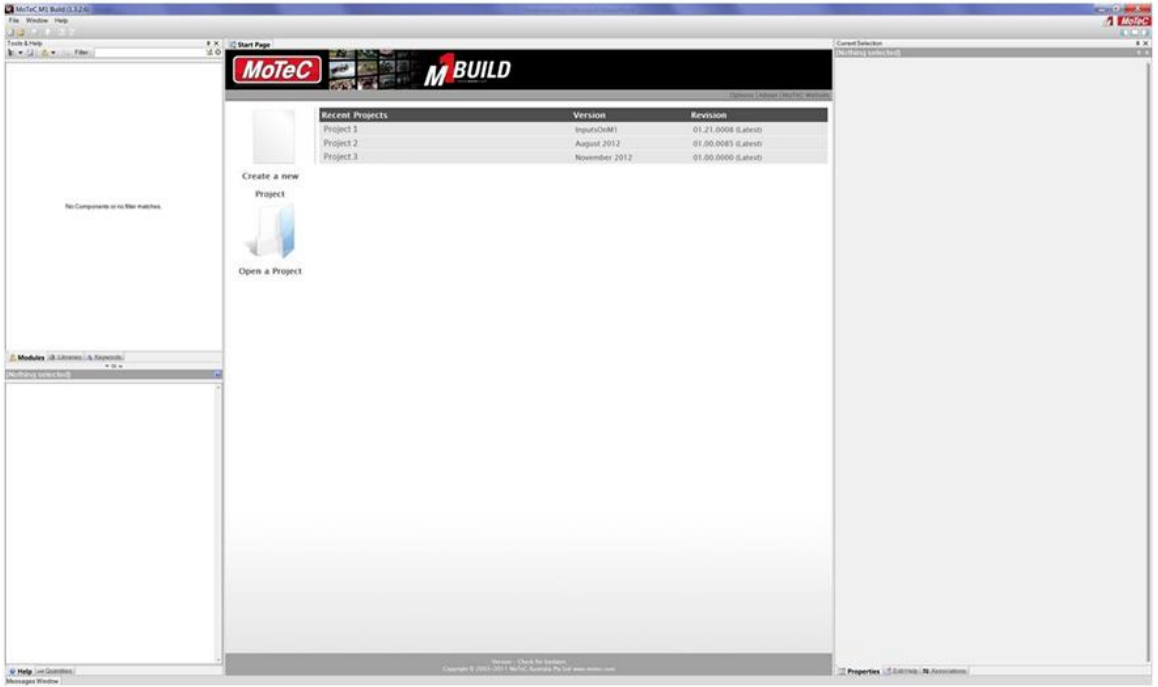
- The integrated compiler ensures automatic firmware integration into the M1.

Starting M1 Build

After starting M1 Build, a screen similar to this displays.

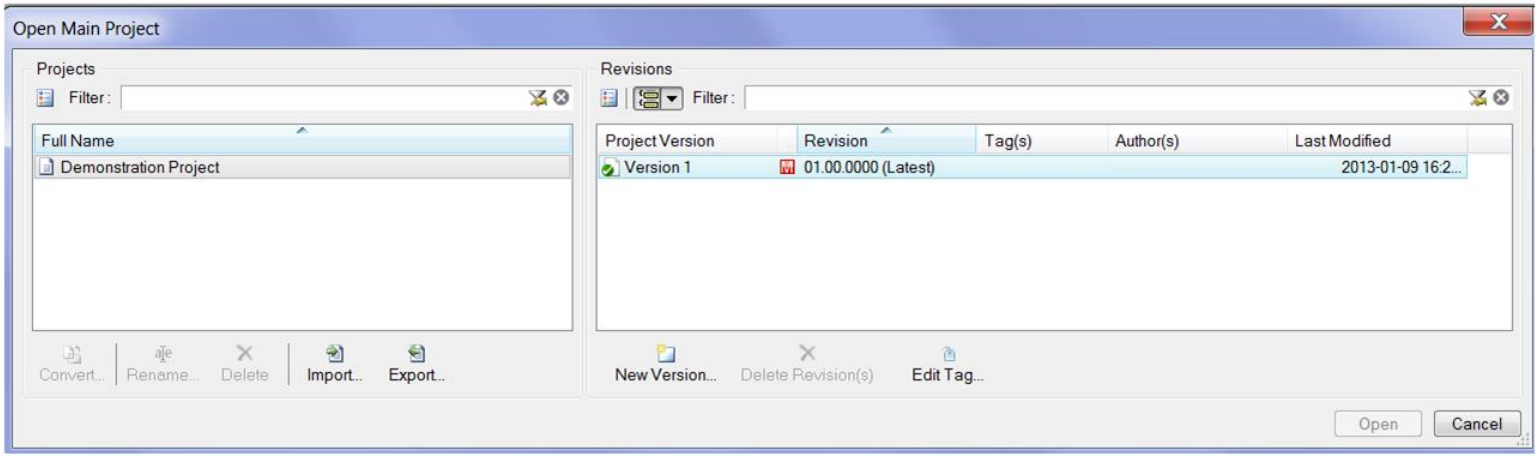
The screen layout consists of three windows:

- Tools & Help window on the left
- Main working window in the middle (this cannot be hidden)
- Properties window on the right
- Messages window along the bottom



Open a Project

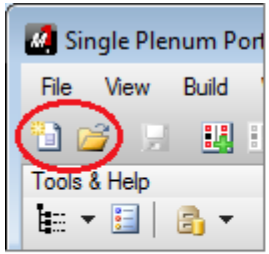
- Choosing to open a Project lists the available Projects in the Project folder, together with their versions and revisions.

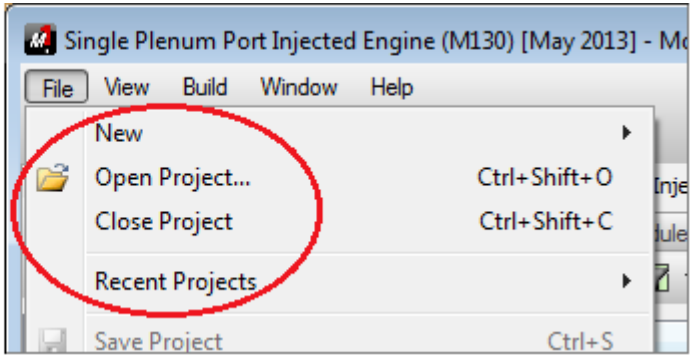


The Projects can be filtered by name. The Project versions and revisions can be filtered by latest revision from all versions, or by latest overall revision.
Additionally, this window contains functions to manage Projects, see Managing Projects and Revisions.

Start a Project

- A Project can be created or an existing Project can be opened by using the toolbar icons in the upper left section of the screen, by using the menu options in the File menu, or by using the options in the main window.





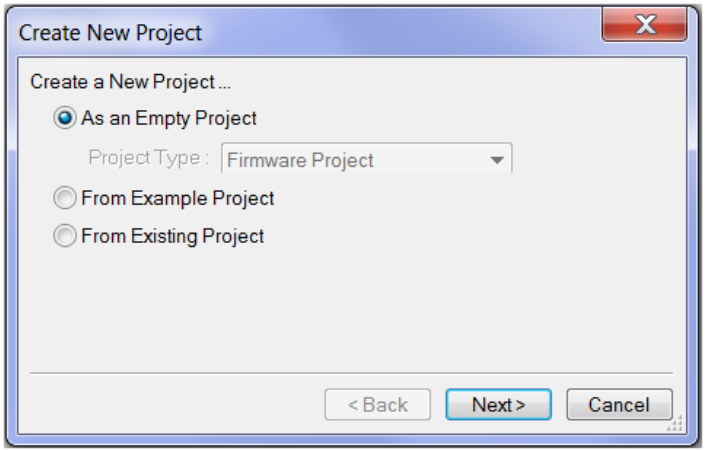
Create a Project

Three methods exist for creating a new Project:

- As an empty (blank) Project
- Using an example Project as a reference
- Using an existing Project as a reference

– Select **File>New** to display the **Create New Project** window

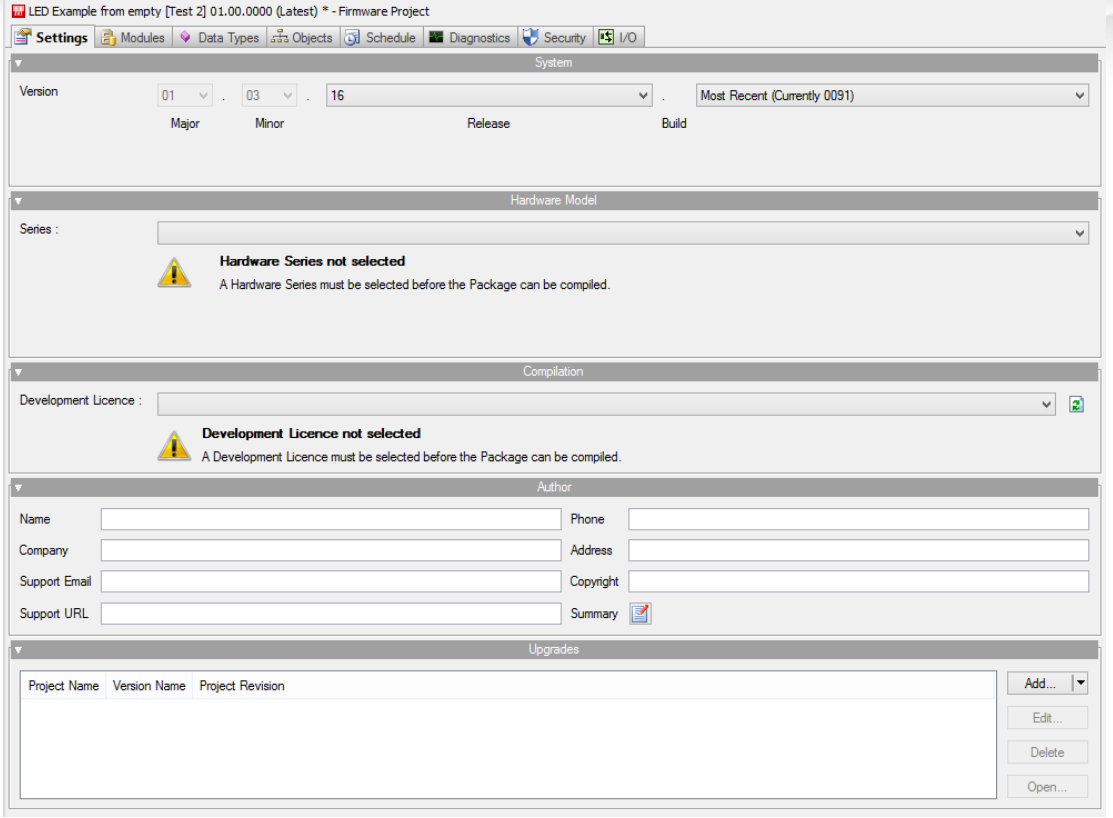
– **Select**



Setting Up a New Project

When you start an empty Project, you need to configure its properties:

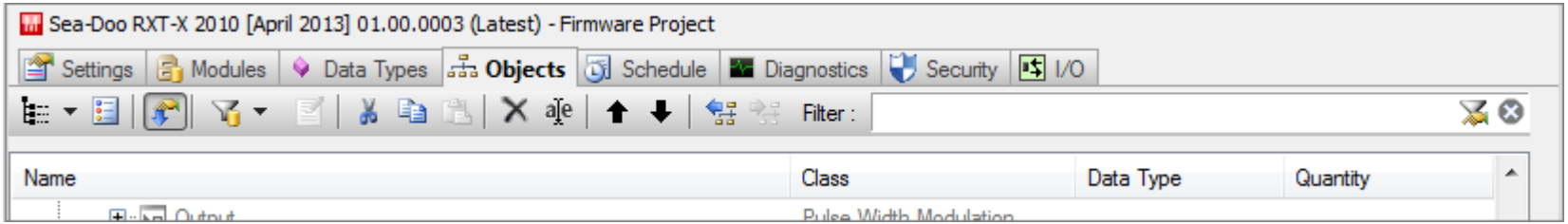
1. Select target Hardware
2. Select Development Licence (this needs to match your target ECU Development Licence)
3. Set the Version to be the most recent (if it is not already)
4. Fill in Name and Company details
5. Save Project



Getting Around M1 Build

Using the Main Window

- At the top of the main window, the name of the current Project, the Project version and revision is displayed.
- Below that name, a number of tabs divide the main window.



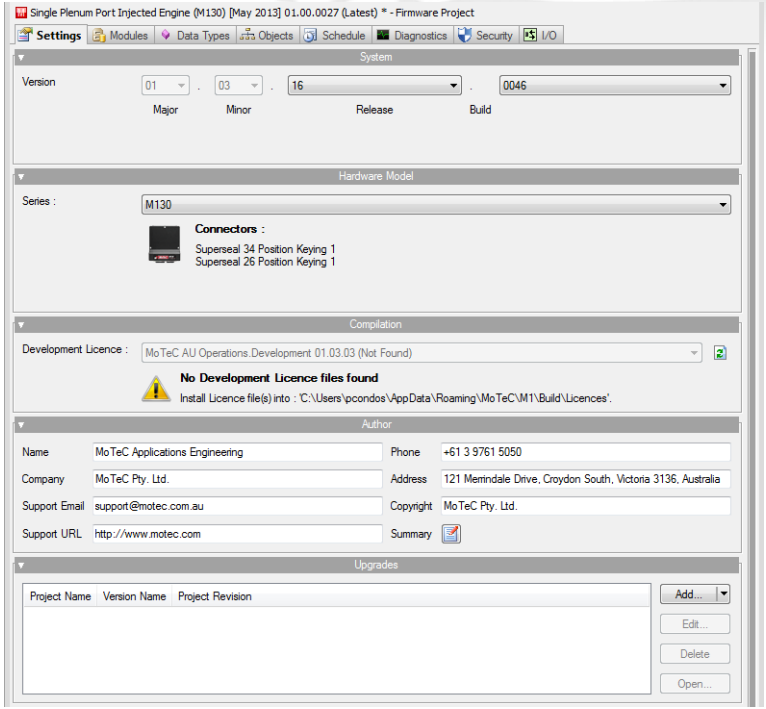
Settings Tab (Page 20)

This tab includes basic information about the Project.

The settings can be defined or adjusted at any time. However, it is recommended to do so at the start of each Project.

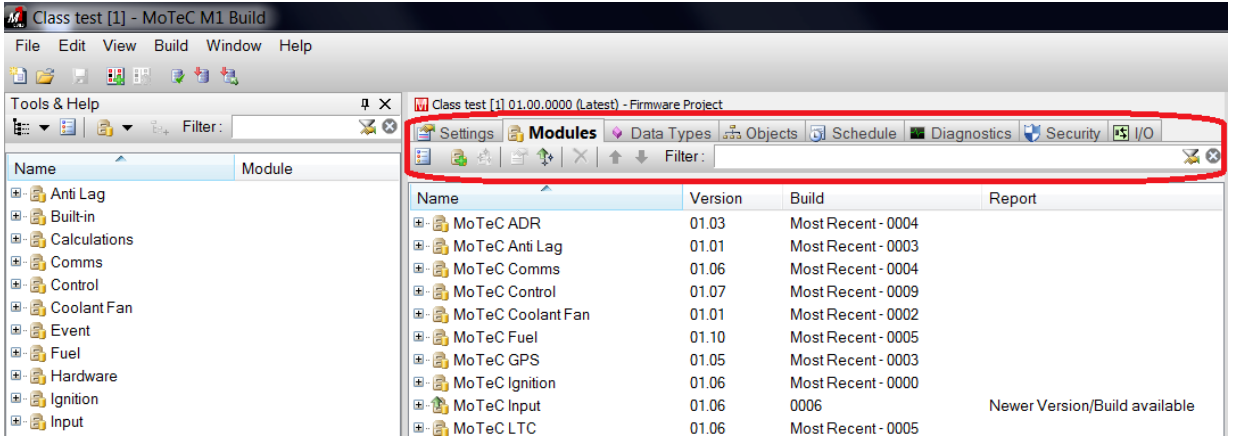
This is because:

- The chosen System Version influences the hardware classes that are available.
- The chosen Hardware Model defines the input and output pins that are made available in the Project.



Modules Tab (Page 22)

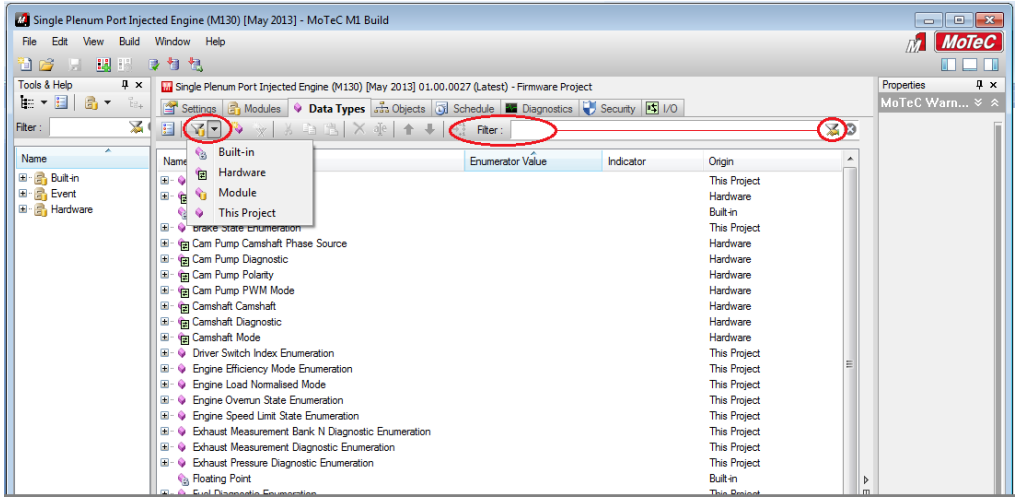
- Modules are collections of fixed, predefined classes and/or data types that can be embedded in the Project.
- A common use of classes is to provide for tasks that are needed often, or to facilitate inclusion of complex tasks.
- The user is able to set the boundary conditions by adjusting the properties. M1 Build comes with various classes to simplify the building of a Project.



Data Types Tab (Page 24)

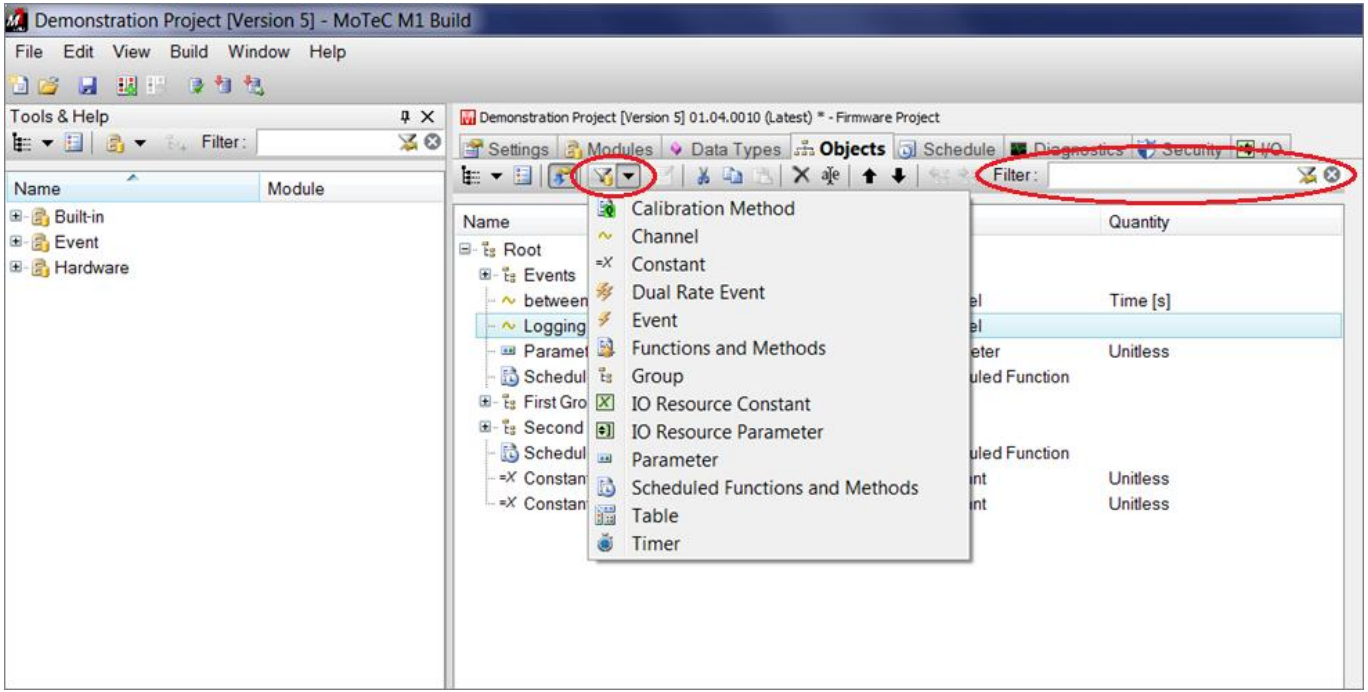
This tab provides for the administration of data types, enumerated data types and their enumerators.

- Data types define characteristics of values allocated to an object. See the M1 Development Manual for a detailed description of data types.
- M1 Build comes with a set of predefined data types, but you can add your own.



Objects Tab (Page 30)

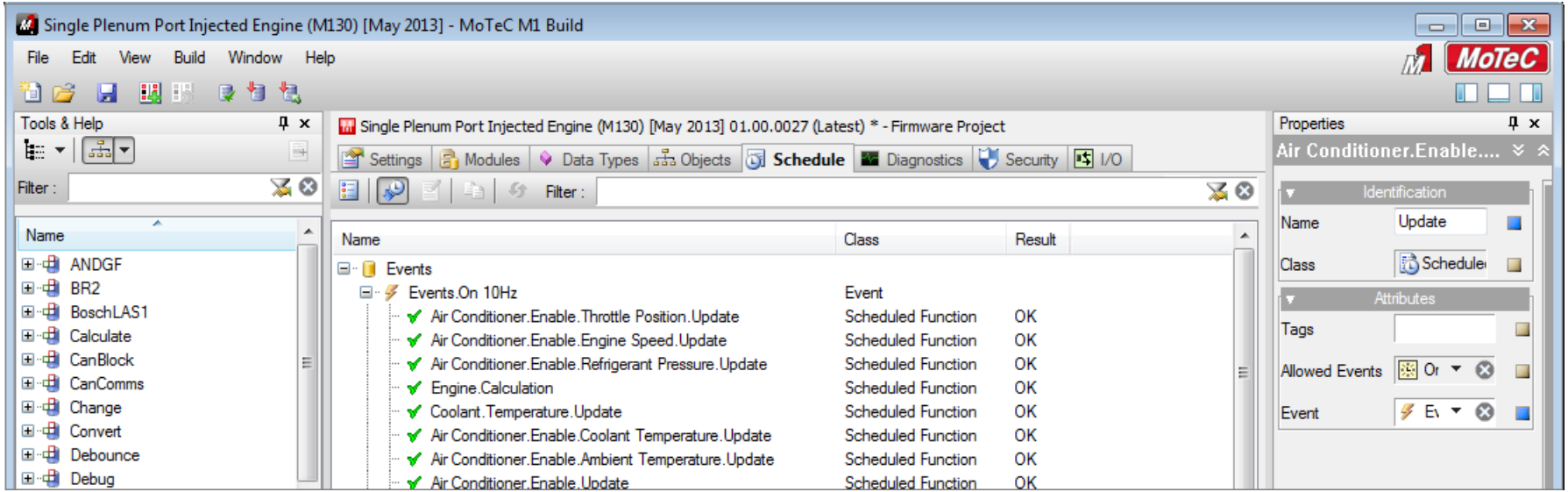
This tab provides for the management of objects in the M1 Project.



Schedule Tab (Page 41)

The Schedule Tab displays the scheduling rate and order of each process in the Project.

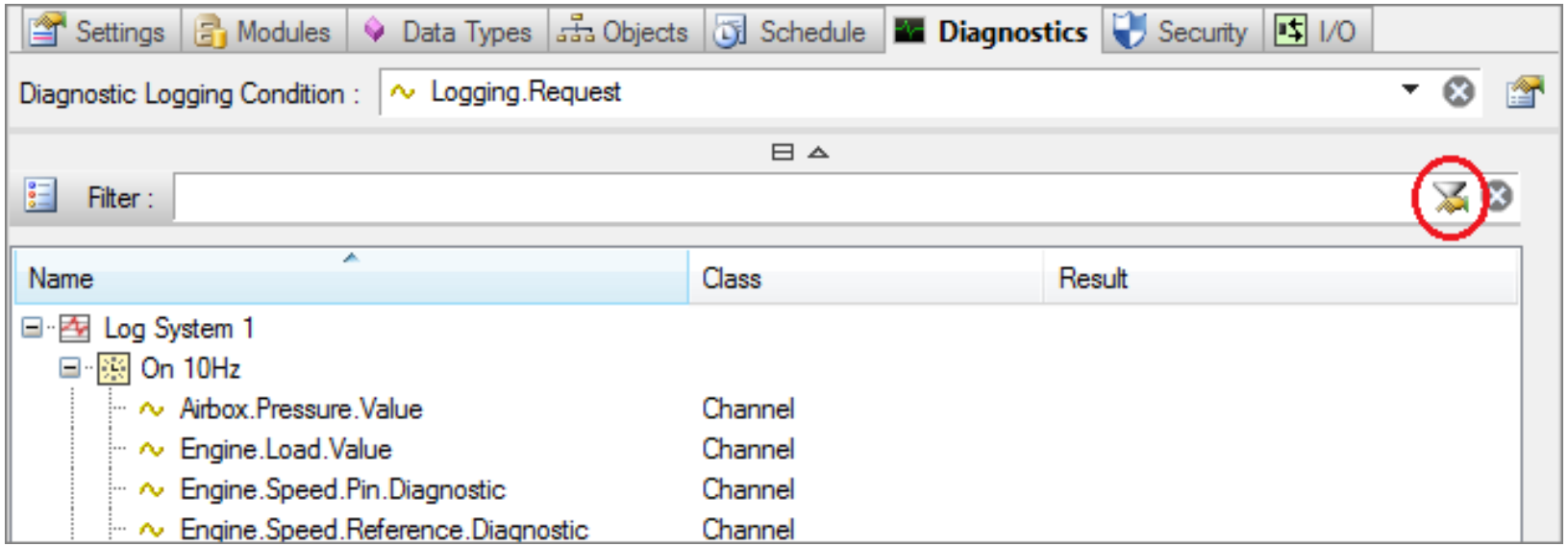
- As M1 Build automatically determines the scheduling of all tasks that need to be scheduled, the Schedule window is mainly for information purposes.



Diagnostics Tab (Page 43)

This tab is used to manage diagnostic setup.

- Channels that are selected for diagnostic logging are mandatorily logged in M1 Tune when logging is active.



Security Tab (Page 45)

The security settings can be set to enforce restricted access (by user) to certain Project objects and configurations in M1 Tune.

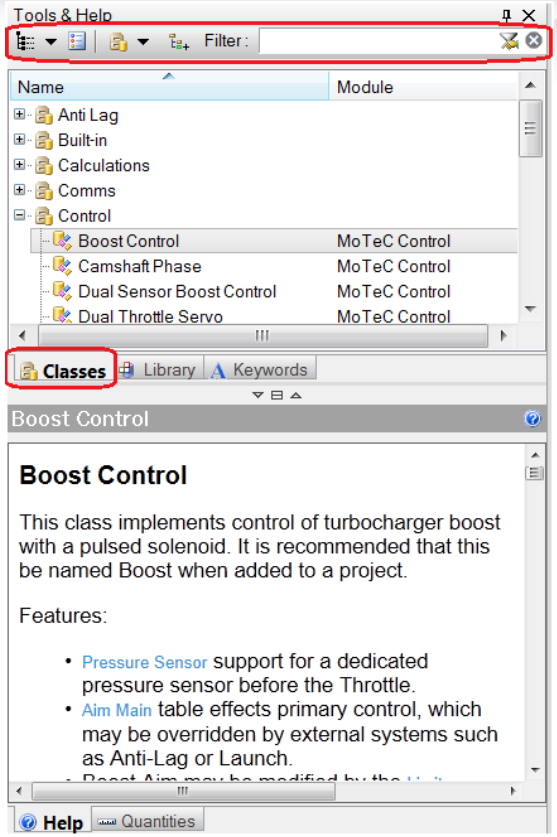
There are three security levels available:

- **Off**
  - No restrictions apply to any user in M1 Tune and no further restrictions can be defined in the Package by M1 Tune.
  - In the M1 Build Project, no further configuration is necessary for this security setting.
- **Basic**
  - As a default, no restrictions apply to any user in M1 Tune, but it is possible in M1 Tune to set up security permissions for different users (however it is not possible in M1 Tune to group channels into different permission groups).
  - In M1 Tune, defined security permissions will be saved with the Package. A change of the security permissions require only a change in the Package and not in the M1 Build Project.
  - In the M1 Build Project, no further configurations are necessary for this security setting.
- **Advanced**
  - In the M1 Build Project, security permissions are predefined.
  - Any change in these security permissions therefore requires a change in the M1 Build Project and the generation of a new firmware version.

Classes Tab (Page 51)

This tab displays all classes currently available in the Project.

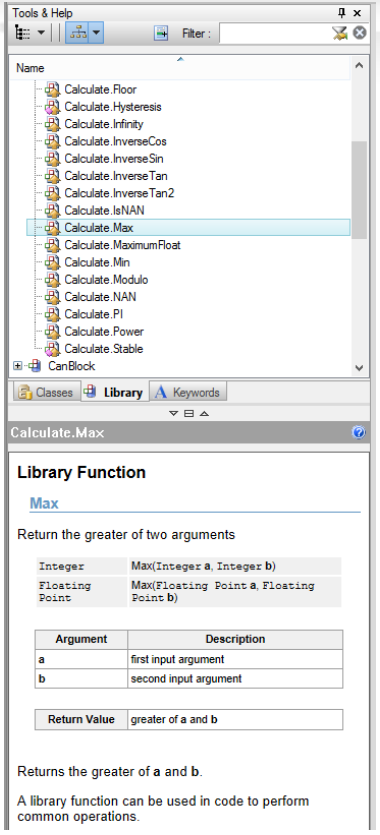
- A class is a construct that is used to create instances of itself called objects.
- As an example, from within Build, you can copy an instance of the Boost control class into your Package.
- You can then use the boost control object to act as a single boost controller by configuring it to work within your Project.



Library Tab (Page 52)

This tab shows the available libraries and the included functions that come with M1 Build.

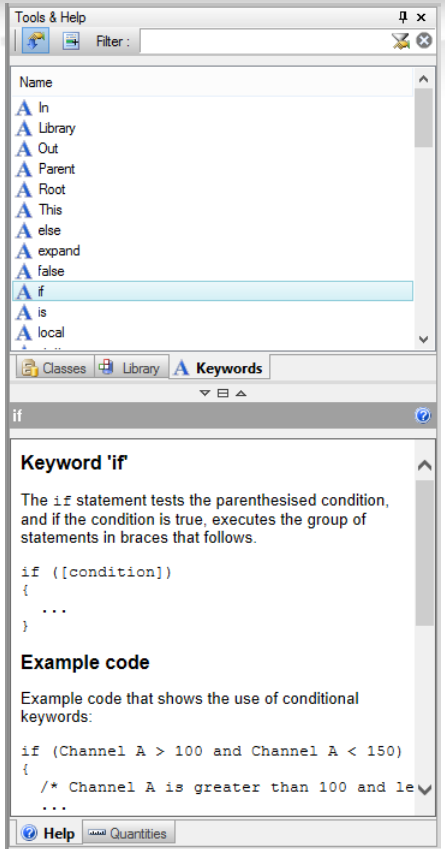
- These library functions can be seen as an extension of the programming language, as they provide program structures or calculations that are often used but do not exist as a single command in the programming language.



Keywords Tab (Page 53)

This tab shows all available keywords and operators used by the M1 Programming Language.

- By selecting an item, additional information concerning this item will be displayed in the 'Help'-section in the upper part of the window.



Working Example

In this example, the Project will create a simple LED light control.
The LED shall be driven with a selectable time-based light pattern.

Light patterns to be chosen from are:

- Sawtooth
- Square
- Sine
- A pattern that can be defined freely by the user
- A square pattern based on an input knob signal

Coding in M1 Build

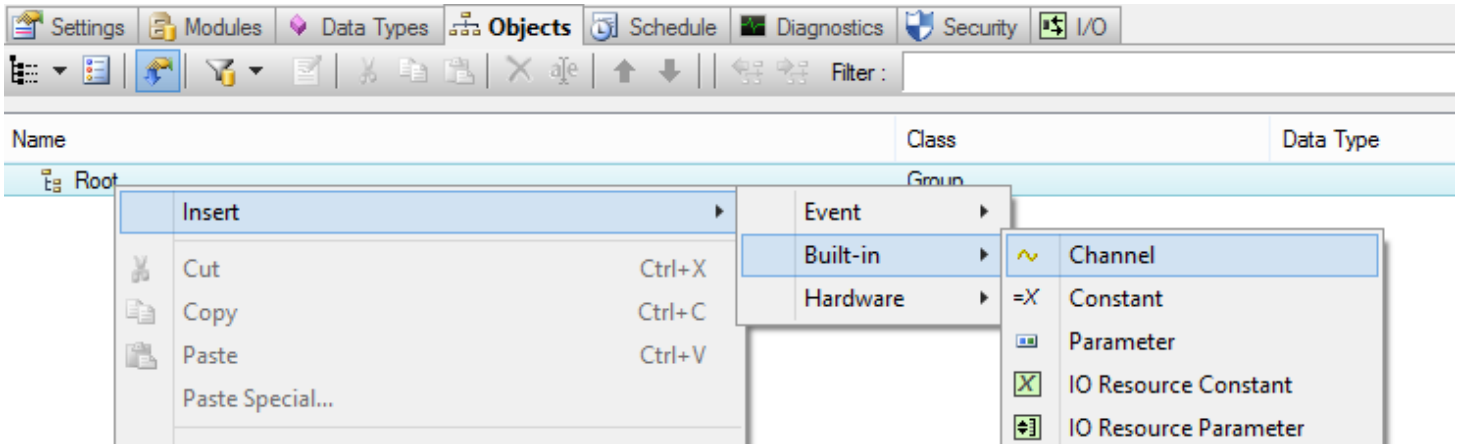
To complete this task, you should break down the one large task into smaller simple tasks like this:

1) Generate a time counter to provide the time base for the light pattern
2) a) Introduce the option for the user to select the pattern
   b) Define the sawtooth, square and sine pattern based on the counter and the user selection
3) Add a possibility to generate a user defined pattern
4) Integrate a knob voltage input
5) Specify the output to the LED

Creating a Time Counter

Step 1 is to create the counter channel:

1) Select the Objects Tab within M1 Build
2) Create a channel for the counter
   - Right click on the root of the Project and select **Insert/Built-in/Channel**
   - Before you choose a name for your channel read the next slide...



Naming Conventions

The naming of an object is restricted as follows:

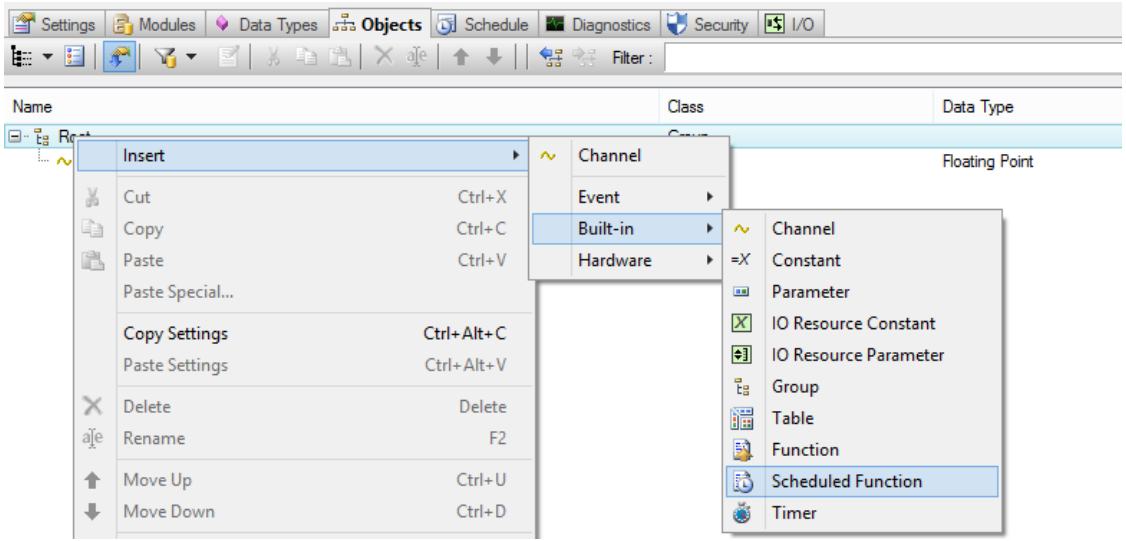
- Must begin with a character
- Must contain only characters, digits or spaces
- Two consecutive spaces are not allowed
- Characters used must not be a keyword of the programming language; such as 'if', 'and', 'false', etc.

Now name this channel **Counter**

Create a Function

Step 2 is to create a scheduled function to calculate and assign the counter channel value:

1) Right click on the root and insert a Built in Scheduled Function
2) Name this function ***Counter Operation***



Create the Code to Run the Counter

Now you have a channel called Counter, and a scheduled function called Counter Operation. What do we do? Create the code to run the counter:

1) Double click on Counter Operation to bring up a window at the bottom half of the centre pane.
2) This window is where you read/edit/create the code for your M1 Project. In this instance, we are going to create a counter that counts up to 360, then resets back to 0, and counts up again in a continuous loop. The value is going to be stored in the channel called Counter. Here is the code.

```
1 if (Counter < 360)
2 {
3     Counter = Counter + 1.0
4 }
5 else
6 {
7     Counter = 0.0;
8 }
```

Writing the Code

The M1 Build software prompts you for suggestions on what you might be wanting to do. Use these prompts to find what you need, because if it isn't offered it probably won't work!

So first line of the code is to create a condition to check if the counter is less than 360.

1) Type in the letter ***i*** – the word ***if*** is now highlighted. Select it by double clicking or pressing Enter.
2) Open normal brackets ***(*** and type in ***C*** – we are looking for the word Counter, so you can either search the list or keep typing to search further into the list. When you find ***~Counter***, select it.
3) Type in the ***Less than symbol (<)*** and then the number 360 then close brackets ***)***

End result: ***if (Counter < 360)***

Brackets

When you create a condition such as **If counter < 360**, you need to encapsulate all of the things that you want to do for that condition in one place.

This is done by putting curly brackets around the list of things that need to be done when the **If** statement is true.

If ....

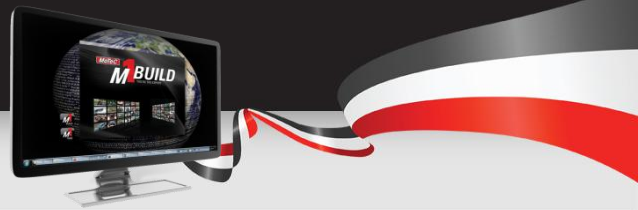
{

Do this

Do this also

And this

}



Incrementing a Value

Now we want to increment the counter. This is done by adding one to the counter each time this function is called. The speed that this counts up is controlled by how regularly we schedule this function to run. This will be done later.

Now we need to code the addition:

Type in **{** to open up the list of items to be done when this condition is true.

On the next line type this ***Counter = Counter + 1.0;***

This line makes the ECU look at the value of the Counter channel, and add one to it.

It can also be written in shorthand like this:

Counter += 1.0;

***REMEMBER that all lines of code not part of a condition need to end with a semicolon ';'**

Close the list of statements for this condition by putting ***}*** on the next line

Formatting the Code

When writing code, there are few rules to force a programmer to make it easy to read. But you should consider how easy it is going to be to debug later. Given this, we generally space out the code to make it as easy as possible to understand.

This code:

```
if (counter<360){counter+=counter;}else{counter=0;}
```

works exactly the same as this code:

```
if (Counter < 360)
{
        Counter = Counter + 1.0;
}
else
{
Counter = 0.0;
}
```

If Else Statements

Now we have written an **if** statement that says **if** counter is less than 360, we should increment the counter.

What do we do when the counter is greater than 360?

In programming terms, we use an **else** statement. It is basically written like this:

If ...

do this

else

do this instead

So for our code, we have written the ***if*** part, time to do the ***else***.

Type in the word ***else*** on the line by itself.

On the next line open the curly brackets ***{*** to start the statements for the ***else***

Setting a Channel to a Value

Set the Counter Value

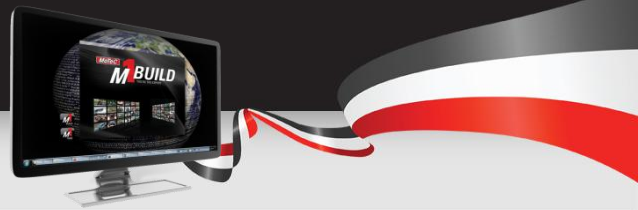
Now we are in the statement group for the ***Else***, we need to set the value back to 0.

This is done with the statement:

Counter = 0;

When written like this the statement assigns the left hand side channel with the value from the right hand side of the ***=*** symbol.

Finally close the Else with a close curly bracket ***}*** on the next line



Using Comments

One of the most undervalued parts of programming is commenting your code regularly.

When you come back to your Project in 6 months, will you remember why you wrote the code in the way that you did?

To place a comment in the code, type in a double slash **//** *then write the comment afterwards.*

Everything on the same line written after a **//** is ignored by M1 Build, but saved with the code.

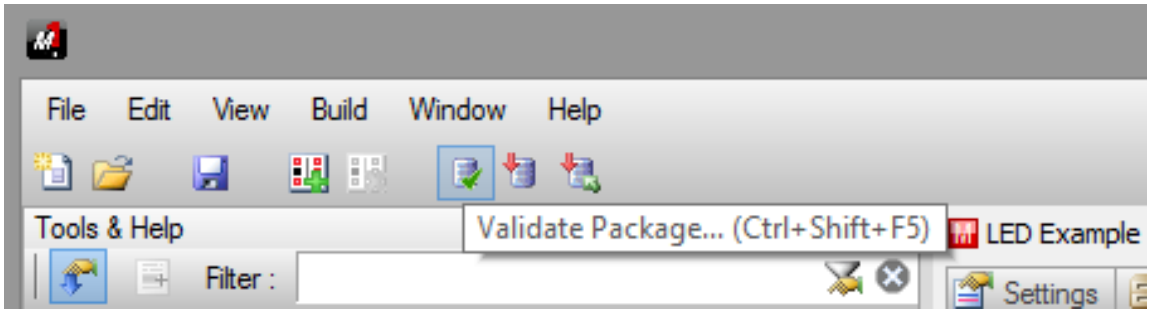
Here is an example of how it is used:

Counter = 0.0; *//This line resets the counter back to zero once we exceed 360*

Validating Your Code

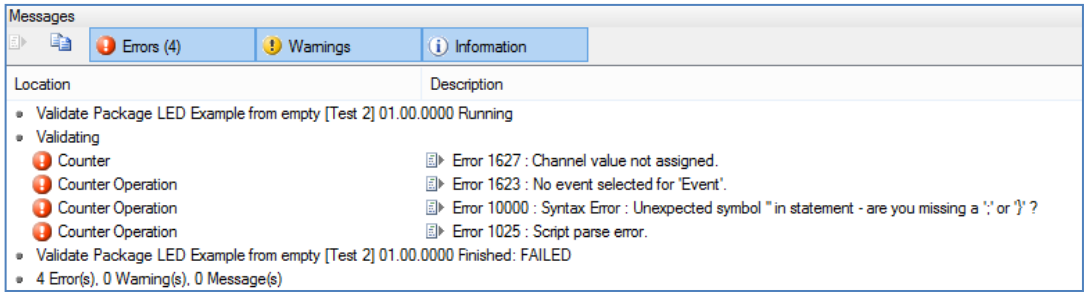
Now you have created your code, it is time to see if it is syntactically valid.

This can be done by pressing the Validate Package button (shown here).

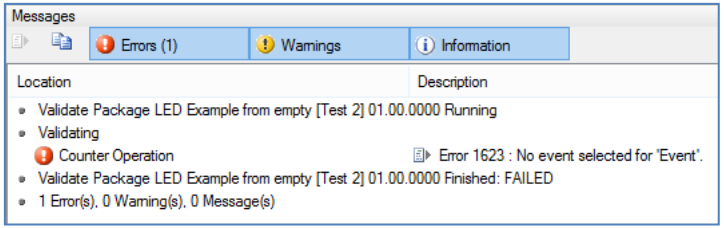


Validation Errors

There is a large number of possible validation errors. One I have shown here initially is the error you will get if you forget to close a bracket. See the third error in the list below. Keep in mind that the first error in the list may be a consequence of an error further down.



If you have written the code correctly, you should get this response.



Scheduling the Event

- So we have completed writing the code to create the counter, but it still doesn't work. Why is this?
- For ***Scheduled Function*** to work, we need to schedule it to run at a particular rate.
- Right click on the root and ***Insert*** an ***Event*** - ***on 100 Hz***
- This event will run at 100hz, so you can assign a scheduled function to be run by it.

- Next, select the ***Counter Operation Scheduled Function***
- When it is selected, on the right is the Properties pane, you can see its properties.
- Later, we will go through the Properties pane more, but for now, select the event drop down, and choose the only coloured item ***Events On 100 Hz.***

Grouping

To keep the Project manageable, it is worthwhile to keep similar items within one group heading.

Examples of this are having a group called ***Brake***, where it may contain items such as

- Brake Pressure
- Brake Temperature
- Brake Switch
- Brake Pedal Position

For our Project, we want to keep all Events in one group (through we have just one currently)

Right click on the root, and ***insert*** a ***Built in – Group***

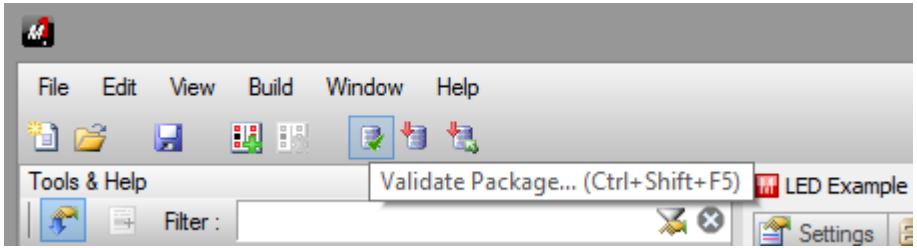
Call this group ***Events***.

Drag and drop the ***On 100 Hz Event*** into this group

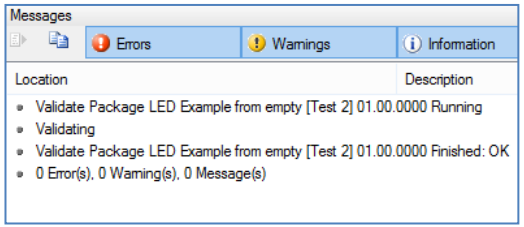
Validating Your Code 2

Now you have fixed your validation error, it is time to see if it is syntactically valid again

This can be done by pressing the Validate Package button (shown here)



You have completed your first valid Project when you get this response. 0 Errors, 0 Warnings



Sending the Package to an ECU

To send a Package to an ECU, you need to have an ECU with a Development Licence.
The Development Licence is the key with which you can control who gets to use your Project.

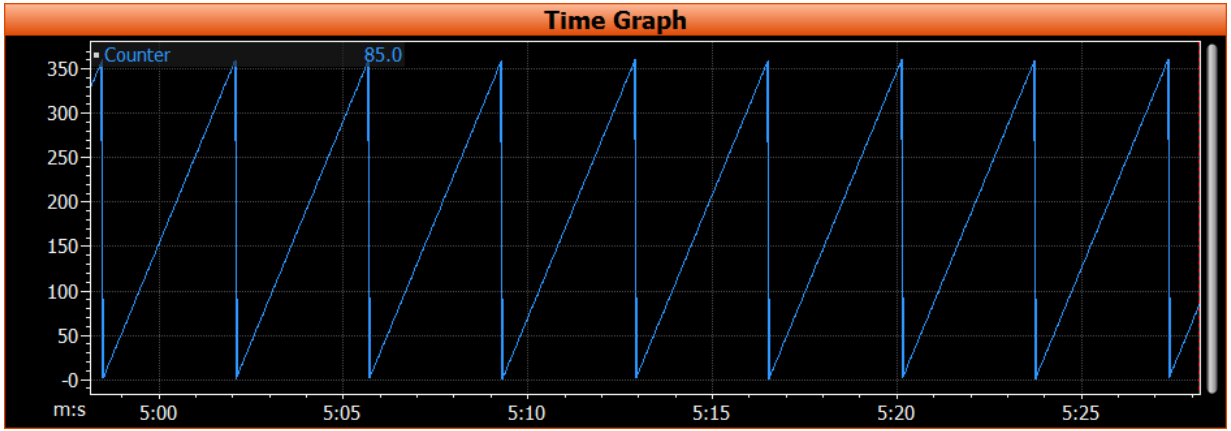
To send your custom firmware into a Development ECU, the Project will need to be built against a Development Licence (e.g. "***AvioRace Development Licence***"). Each developer has their own unique Development Licence.

So if you build your firmware against an ***AvioRace Development Licence*** on your PC, you will only be able to load it into an ECU that has been loaded with the ***AvioRace Development Licence***.

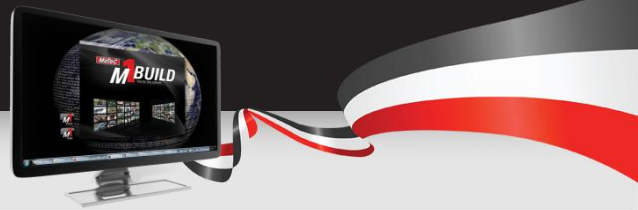
The only person who can order an ECU with your Development Licence in it is you. This system ensures that only you, the owner of a Development Licence, gets to sell or use your firmware.
All ECU Development Licence orders must come through you for your Licence.

Result

Once this code is built and pushed into an ECU, you can view the value in a number of ways, but the best way to view the value of the Counter channel is as a time graph. Now we can see the result of our code, the Counter increments up to 360, then resets to 0 and starts again.



Coding in M1 Build Part 2



Introduce a User Pattern Selection

The Project is extended with:

- An enumerated data type with the values of the patterns
- A parameter that allows the user to choose the pattern

Data Types (Page 20)

Developers can define values to be one of the following data types.

- **Floating Point** Floating point represents real numbers in a way that can support a wide range of values. Numbers are represented by a fixed number of significant digits and scaled using an exponent.
- **Enumeration** Enumerations are used where it makes more sense to use a textual description rather than a numeric value.
- **Integer** An integer is a whole number that is positive, negative or zero.
- **Unsigned Integer** An unsigned integer is a non-negative whole number.
- **Boolean** (Boolean data types are restricted to local variables only). M1 Build supports use of enumerated data types for channels and parameters, as they provide more information than a Boolean data type.
- **String** (String data types are restricted to local variables only). They can be used to show text in information windows that can open up in M1 Tune.

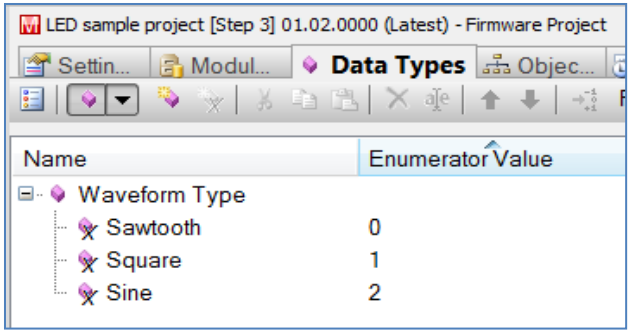
All data types in the M1 ECU are 32 bits in width.

Create a Custom Enumeration

Select the Data Types Tab

Right click in the open space and select ***New Enumeration***

Call it ***Waveform Type***



Right click on Waveform Type and select ***New Enumerator***

Create Sawtooth Enumerator

Repeat to add in Square and Sine

Right click on Sawtooth and select ***Default Enumerator***

You have now created some Types that can be selected from in Code and within Tune later.

Create a Parameter

What is a Parameter, and how does it compare to a Channel?

A ***Parameter*** is an item that can have its value set by the user within Tune.

A ***Channel*** is very similar, but it can only have its value set by the firmware itself.

An example of a ***Channel*** is Engine RPM. The Firmware sets this channel value based on the speed measured by the reference sensor.

An example of a ***Parameter*** is a Reference mode, where the user in Tune can select from a list of available modes.

So, now you can create a parameter by right clicking on the root, then choose ***Insert /Built-In / Parameter.***

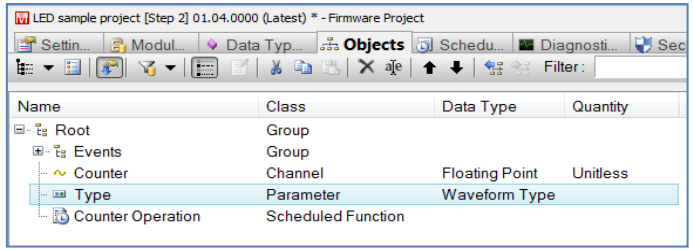
Label the Parameter ***Type.***

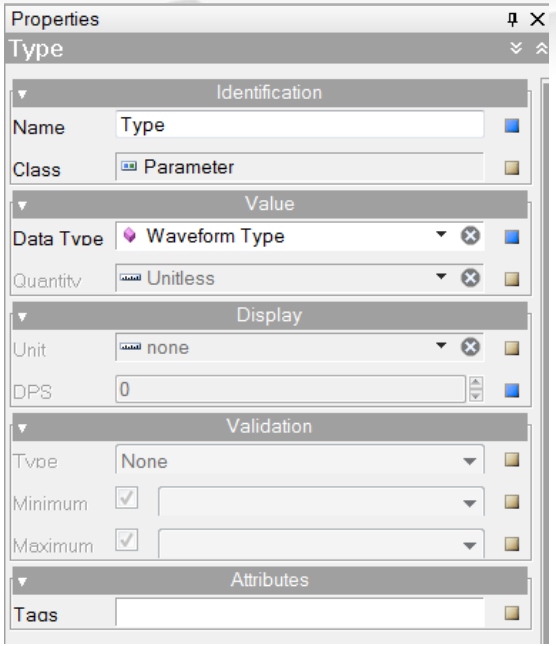
Set the Data Type

Now that you have created a Parameter, we need to tell M1 what type of Parameter it is.

Select the Parameter item from the root list. When you have done this, you will see the items in the right hand pane – ***Properties*** – showing you the properties of this new parameter.

Is this Parameter a **Floating Point Number**, an **Integer,** or a **Enumerated Data Type**? In this instance the Data Type is going to the **enumerated Data Type** that we just created called ***Waveform Type.*** You will find this type at the bottom of the Data Type list.





Coding in M1 Build Part 3

Define the sawtooth, square and sine pattern based on the counter

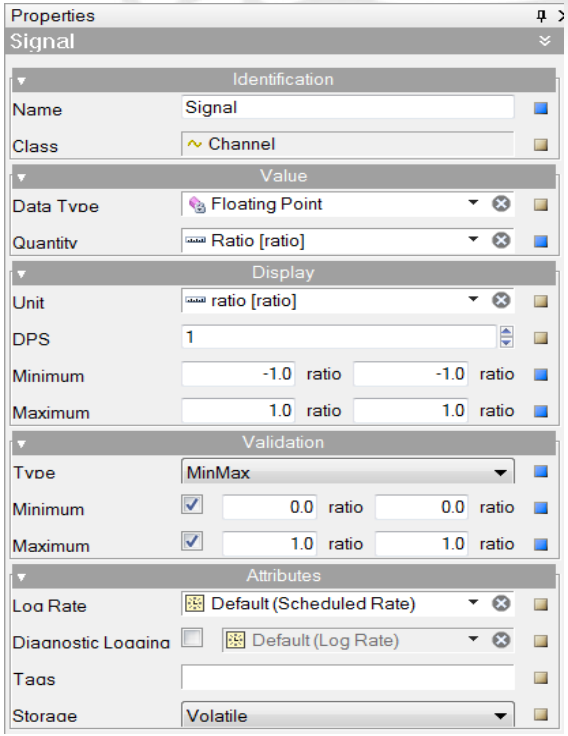
In the Project, the following objects are added and configured:

- A channel to represent the selected Signal
- A new group where the counter calculation is put into
- A scheduled function to calculate the pattern based on the selection from the user, and assign the pattern to the signal channel

Implement User Defined Patterns

Create a ***Channel*** called ***Signal*** using the technique described on Page 22 .

Change its properties on the right hand pane so that it has a data type of ***Floating Point*** with units of ***Ratio***.



Signal Generator

Create a ***Scheduled Function*** called ***Signal Generator*** using the technique described on page 24 .

Once you have created this function, add in this code:

The first item you will come across is the statement

When

```
when (Type)
{
    is (Sawtooth)
    {
        Signal = counter / 360;
    }
    is (Square)
    {
        if (Counter> 180)
        {
            Signal = 1.0;
        }
        else
        {
            Signal = 0.0;
        }
    }
    is (Sine)
    {
        Signal = ((Calculate.FastSin(Counter)+1.0)*0.5);
    }
}
```

Keyword When

The ***when*** keyword begins a ***when/is*** construct.

when (Enumerated Data Type)

{

is ([enumerator])

{ do this }

is ([enumerator] or [enumerator])

{ do this }

}

The [argument] used in the when statement **must be of an Enumerated Data Type**. Each ***[enumerator]*** must be one of the enumerators of the enumeration, and all of the enumerators of the enumeration must be covered by the when/is construct.

The **'or'** keyword can optionally be used to specify multiple enumerators to match an **'is'** statement

Calculate Library Functions

A library function can be used in code to perform common operations more simply.

Examples of this are:

Calculate.Max (a,b) which returns the bigger value out of a and b

Calculate.Average (a,b) returns the average of the two values

Calculate.Hysteresis (arg, High, Low, Filter) returns true false hysteresis calculation of values

So the Calculate Library Functions are helpers to streamline your coding.

In the instance of our sample program we have used the ***Calculate.FastSin*** library function which calculates the sine of the current ***Counter*** channel.

The end result of the ***FastSin*** in this instance will be a sine wave output

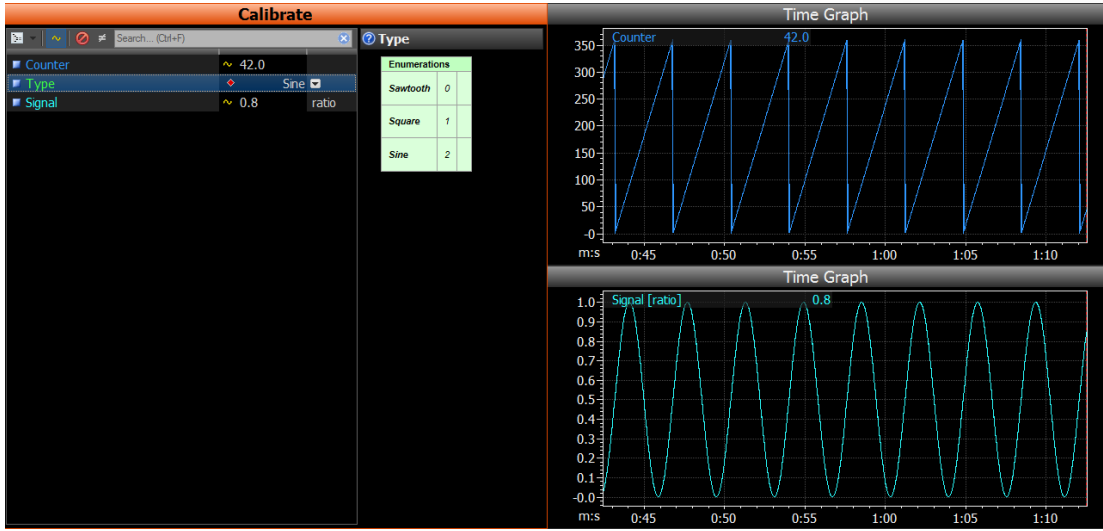
End of Next Stage

You have now completed the next stage of your M1 Build program.

You should be able to build your code.

Spend the time to go through your validation errors until you get a clean build.

I will build the Project and send it to the ECU. This is what it should look like in the ECU.



Coding in M1 Build Part 4

The Project is modified to:

- The signal generation is moved into a new group named Signal
- A table for calibration of the user defined pattern is added
- The data type representing the selectable pattern options is extended to cover the table option
- The signal generation code is complemented to allow the use of the created table

Creating a Group

As discussed earlier, it makes sense to keep your programming tidy. One way to do this is to group related items together. We will now create a group called Signal, and place within it all the items related to the signal generation.

1) Right click on the Channel called Signal, and rename it as Value.
2) Right click on Root and ***Insert / Built-In / Group***
3) Name your new group Signal
4) Drag and Drop the channel ***Value*** into the group ***Signal***
5) Select the group called ***Signal*** and tick the ***Default Value*** check box in the Properties window
6) Click the down arrow at the right hand side of the ***Default Value*** drop down and select ***Signal.Value***

You have now created a Group called ***Signal***. The group ***Signal*** has a default value assigned to it from the channel called ***Signal.Value***.

Groups

Now that you have created this group, you can add in all of the related items.

Drag in the ***Signal Generator*** Scheduled Function

Drag in the ***Type*** Parameter

You should now see your Project look something like this

Name	Class	Data Type	Quantity
Root	Group		
Counter	Channel	Floating Point	Unitless
Counter Operation	Scheduled Function		
Events	Group		
Signal	Group	Floating Point	Ratio [ratio]
Value	Channel	Floating Point	Ratio [ratio]
Signal Generator	Scheduled Function		
Type	Parameter	Waveform Type	

Tables

Now we can add a table into our Group called ***Profile***

Right click on our group ***Signal***, then ***Insert /Built-in /Table***

Change the ***Value Quantity*** to Ratio in the Properties window.

Set the Display and Validation Min Max's 0 to 1

Set the Update event to 100 Hz so the table value updates 100 times per second.

Towards the bottom of the properties window, you will find the Table heading, which lets you select the number of axes on the table, the channel for each axis and the maximum number of sites.

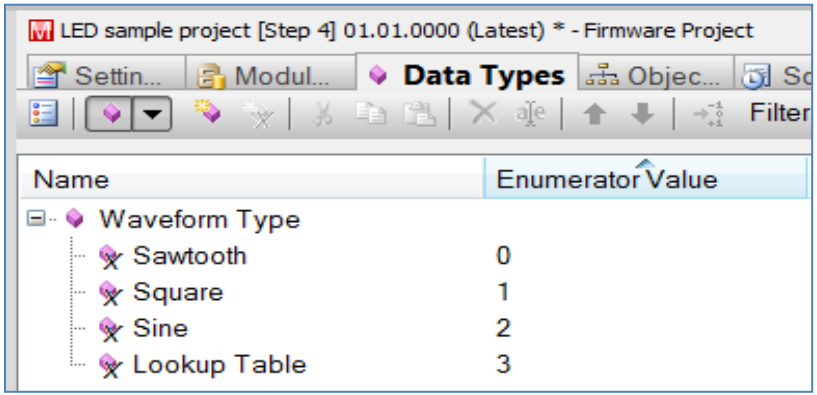
Set the table up with ***1 axis***, set the Object for the X axis as ***Counter***, and the Maximum sites to ***21.***

Update the Data Types

Find your ***Waveform Type*** Data Type within the Data Type tab.

Right click on the Waveform Type label and select ***New Enumerator***

Call this new Enumerator ***Lookup Table***

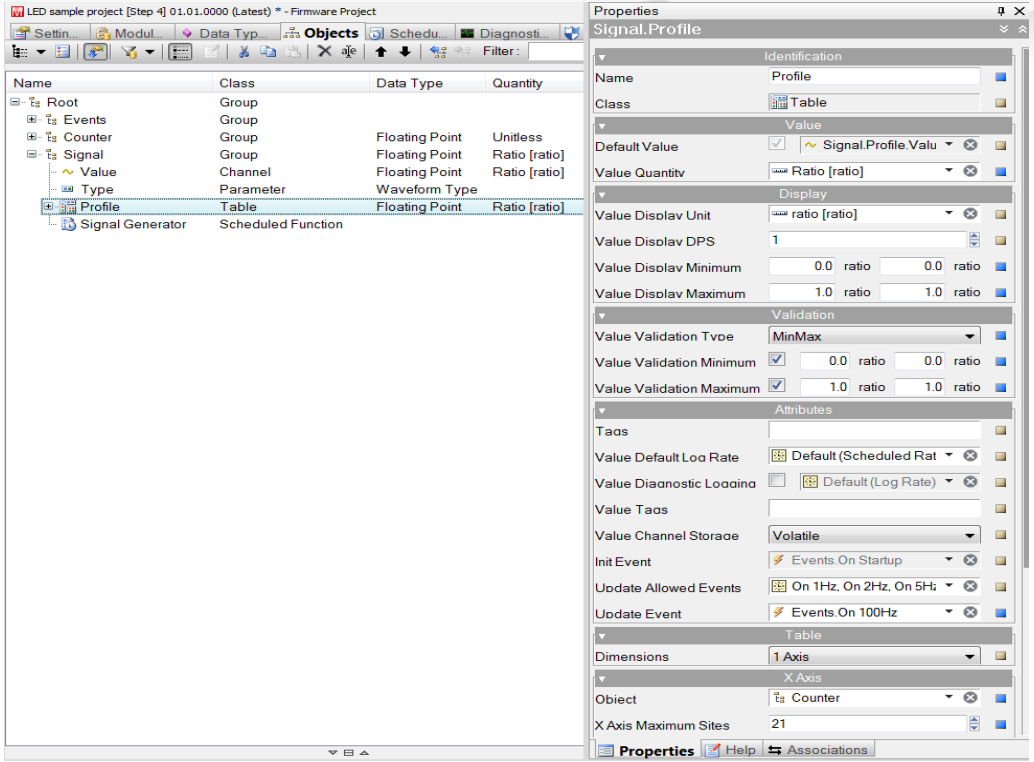


Start Up Event

Some objects need to be associated with a startup event to set their value on start of the firmware.

The Table that we have just added in needs to have its value set on firmware startup, but we currently don't have an event running on firmware startup.

Right click on the Events Group, ***Insert/ Events/On Startup.***



Update to Signal Generator Code

Select your signal generator code.

Update your code to match the sample on the right, adding in the Lookup Table enumerator, and setting the ***Counter.Value*** to the Profile Table output.

Yo.u should now be able to build your Project again

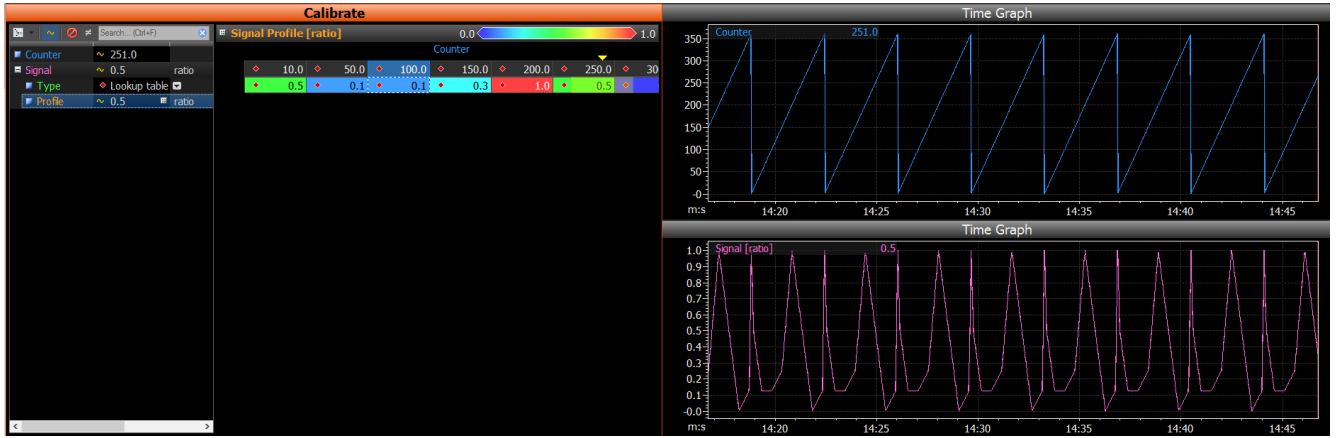
```
when (Type)
{
    is (Sawtooth)
    {
        Value = Counter / 360;
    }
    is (Square)
    {
        if (Counter> 180)
        {
            Value = 1.0;
        }
        else
        {
            Value = 0.0;
        }
    }
    is (Sine)
    {
        Value = ((Calculate.FastSin(Counter)+1.0)*0.5);
    }
    is (Lookup table)
    {
        Value = Profile;
    }
}
```

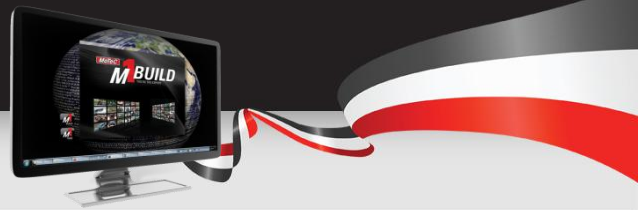
Testing the Project in Tune

Now that we have built the Project, we can send it to the ECU.

We can now select the Lookup Table signal generator.

To make the signal generator produce a wave, you need to update the table to have sites for the whole 360 value range of the counter. When you have updated the axis to have the sites you need, put in values between 0 and 1 to represent what you want the ***signal.value*** to be when the counter is at specific values. An example is shown below:





Add Input to Control Waveform

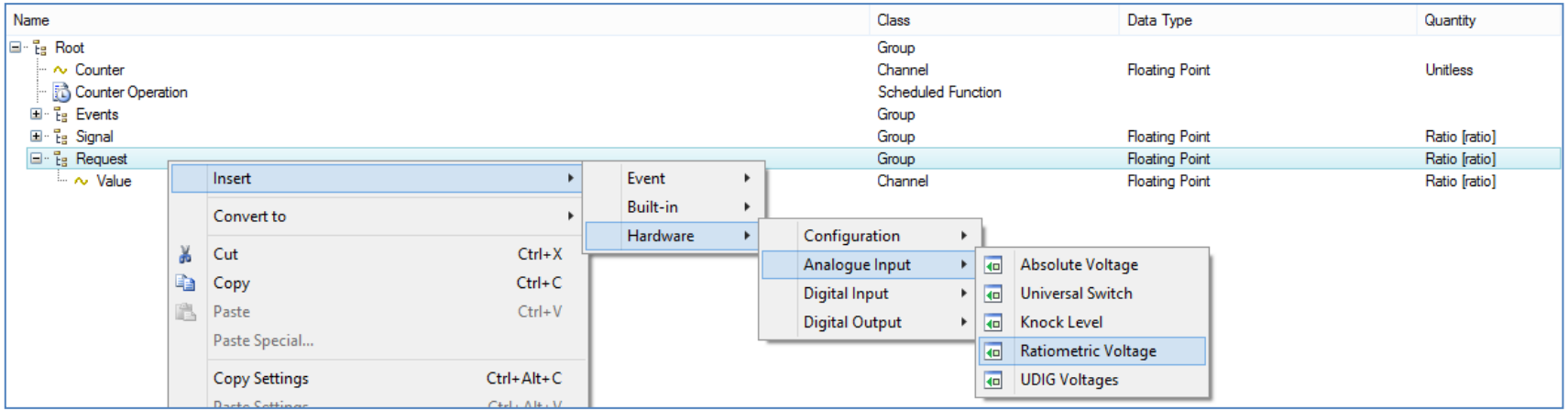
In this section of the training, we will look at how to add an input into the Package from a pin on the device, and have it control the waveform.

1) Add a new group called ***Request***.
2) Insert a ***Channel*** called ***Value*** within the Request group
   - Select a ***Quantity*** of Ratio
   - Set the ***Display units*** to be percentage %
   - Set the ***Value Display Min and Max*** properties to be a ratio with 0 to 1 limits
   - Set the ***Validation Type*** to be MinMax and set the limits as 0 – 1
3) Select the ***Request*** Group, and change the ***Default Value*** to be ***Request.Value***

Add an Input Pin

Select the ***Request*** Group

1. Right click on the Name and Insert a Hardware / Analogue Input / Ratiometric Voltage
2. Call this Analogue input ***Input***



Configuring an Input Pin

Select the Ratiometric Voltage called ***Input.*** To use this input, it needs to be associated with a pin on the device. This can be done as either a ***Resource Constant*** or ***Resource Parameter.***

Resource Constant:

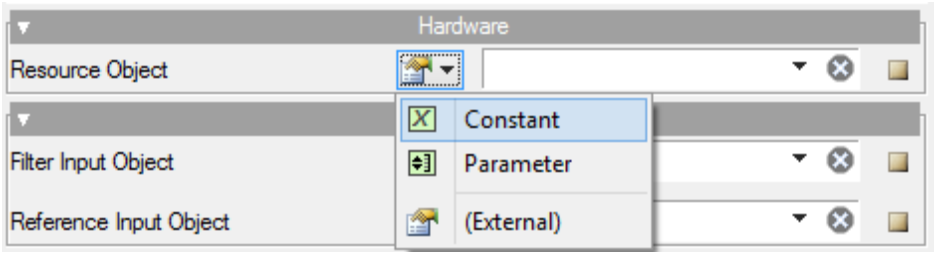
This type of resource is allocated to a specific input pin by the developer during the coding of the Project.

Resource Parameter:

This type of resource is left unallocated to a specific pin on the device by the developer, and the end user can allocate a resource from within Tune.

To set the Resource type, selecting the Small icon next to the Resource Object drop down.

For our Project, select ***Constant.***



Setting up the Input

Now you have set up the input as a constant, we need to allocate it a pin to use.

From the ***Value IO Resource*** drop down, select Analogue Voltage Input 2

Select the ***Filter Input*** object to be a parameter

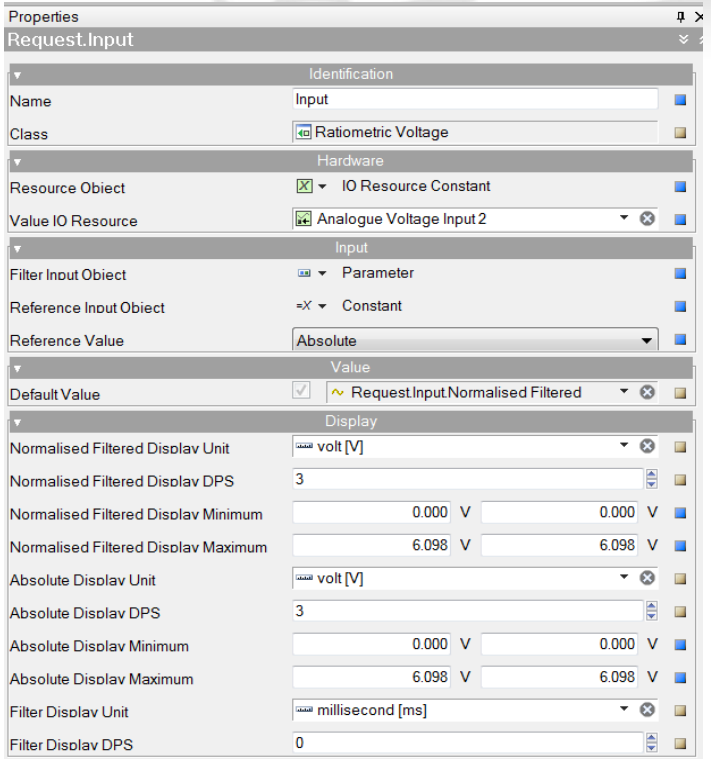
– This allows the end user to set a Filter on the input

Set the ***Reference input*** object to Constant

Finally set the ***Reference Value*** to Absolute.

– This has set the input pin to be read as a reference value rather than ratiometric to one of the 5v rails.

We have now set up our input pin. Time to make it work.



Reading the Input into a Value

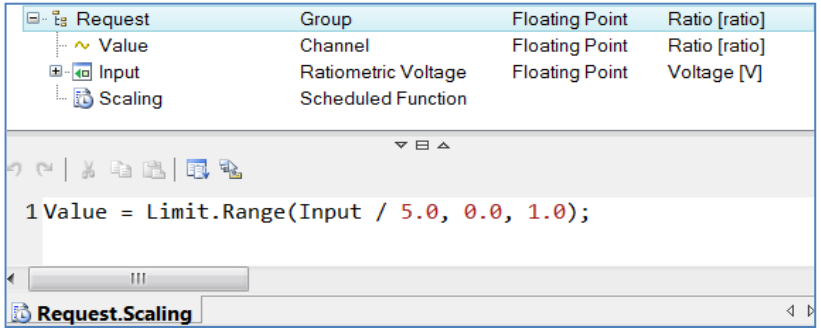
Now that we have setup an input, we need to do something with the input voltage.

1. Right click on the Request group and ***Insert*** a ***Scheduled Function*** Called ***Scaling.***
2. Double click on the Scaling Function to open up the code window
3. Add in this one line of code:

   Value = Limit.Range(Input / 5.0, 0.0, 1.0);

This line of code sets Requst.Value to be equal to ***Request.Input / 5***, with the resultant value limited to a value between 0 and 1

Select the word Range from ***Limit.Range*** to see the help for the ***Limit.Range function***.



Scheduling

Don't forget to schedule your new items.

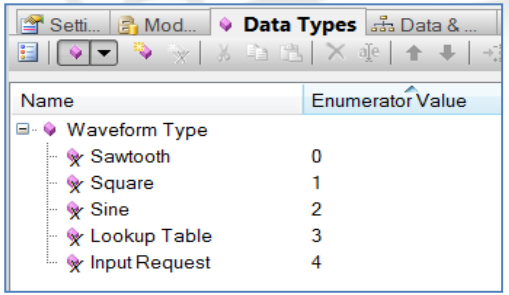
You need to tell the M1 how often to check the value of the voltage at the input pin.

You also need to tell the M1 how often to run the ***Scaling*** Scheduled function.

Select each of these two items and set their Events to 100 Hz.

Add the Next Enumerated Value

1. Select the ***Data Types*** Tab at the top of the window
2. Select the ***Waveform Type*** Enumeration
3. Right click on the ***Waveform Type*** and add a new enumerator.
4. Call this Enumeration ***Input Request***.



Select the Objects tab again.

Open up the Signal group and double click on the Signal Generator Scheduled Function

Add the Code in the box on the right into this function at the bottom.

```
26      is (Input Request)
27      {
28          Value = Request;
29      }
30  }
31
```

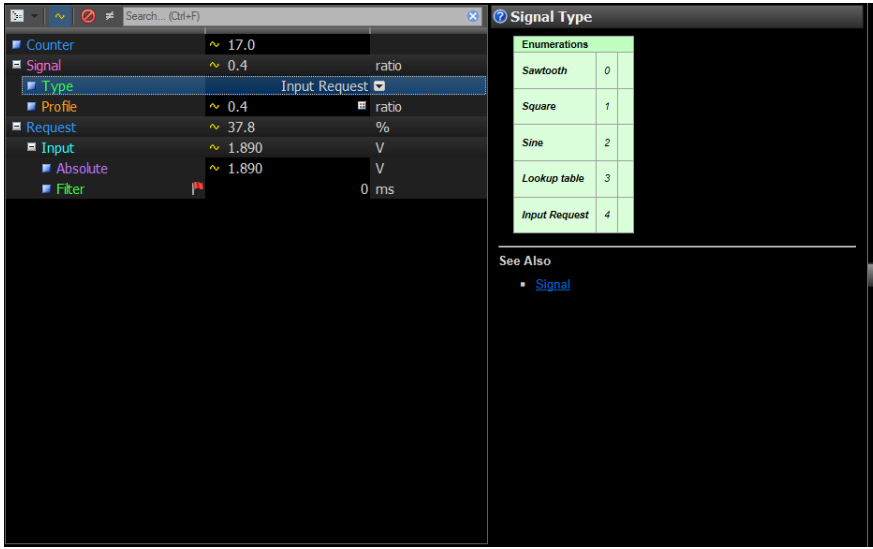
Signal.Signal Generator *

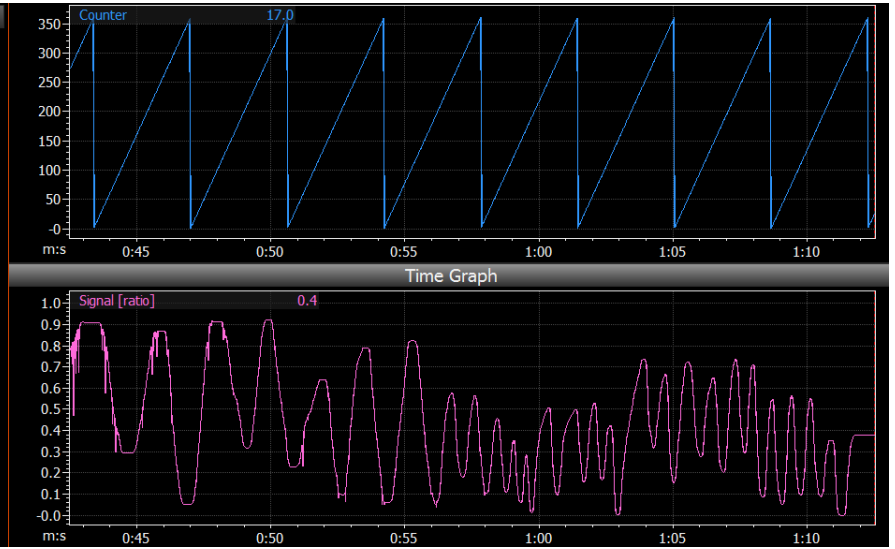
Testing the Input

Build your Project, fix any errors, then send your Project to the ECU.

Select the Type drop down and change it to out new Enumeration, ***Input Request***

Move input dial up and down, and you will get the signal below



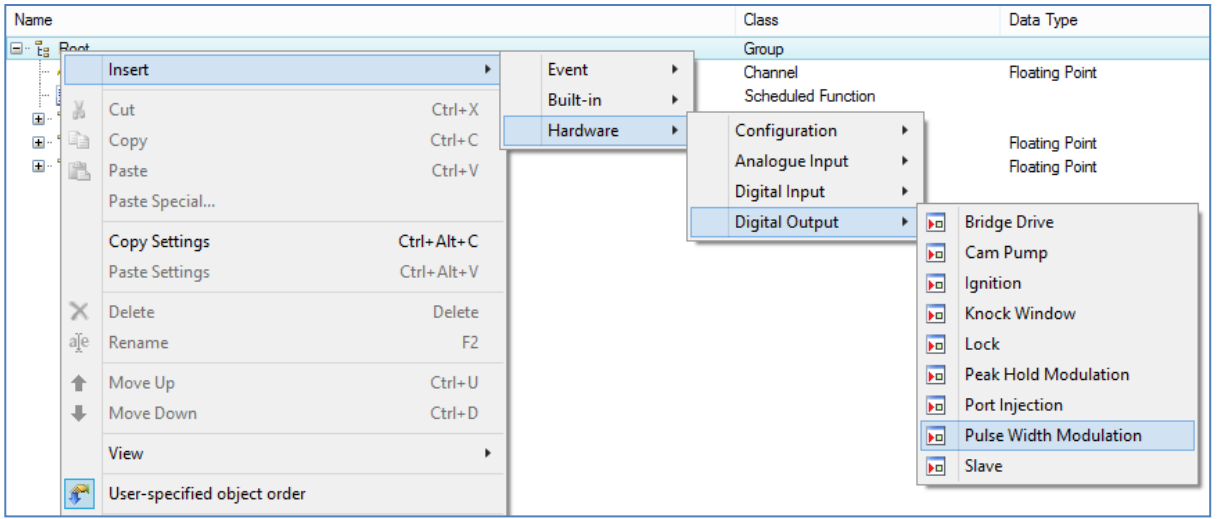


Final part of the Test Project - Output

In this final increment to the test Project, we will control an output with the pattern.

To do this we will Insert a ***Hardware/Digital Output/Pulse Width Modulation*** object into the root.

Once added,call this object LED.



Configure the Output

Once you have added in the LED PWM output, we need to configure its properties.

Select the ***LED output***.

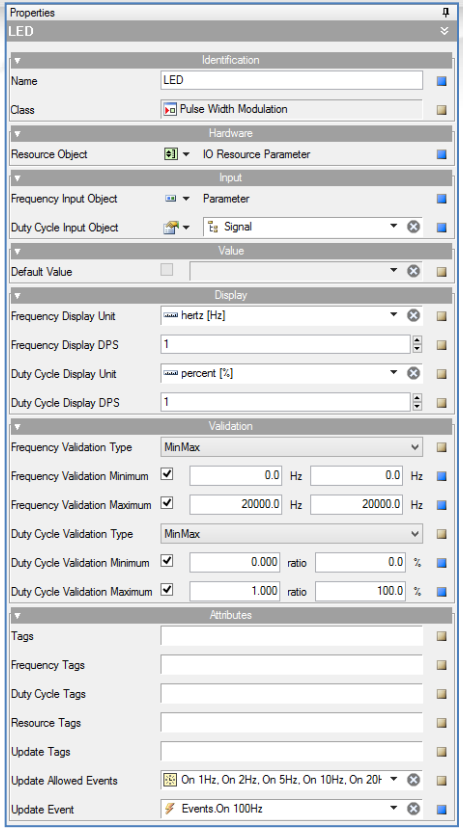
Within the ***LED*** properties window, setup it up to match the properties shown here on the right.

Set the ***Resource Object*** up as ***IO Resource Parameter***. This will let the user in Tune select which output to use.

Set the ***Frequency*** as a Parameter, and the ***Duty Cycle*** to our Group called ***Signal***

Set the Validation properties as shown.

Set the Update event at the bottom to update the state of the output on the 100 Hz event.



Test Project Complete

With those changes, we can now send our complete test Project to the ECU.

When it is sent, you can set the ***LED output resource*** to any available resource for that hardware type.

Set the frequency to 0 (to make a switched output) or to 100 Hz to turn it into a PWM LED which varies brightness in line with the Signal value.

Demonstration of the LED control working on the Simulator

Save your work we have now completed that Project.

What to Do Now?

You have just run through many of the skills that you will need to use when creating your own Packages using M1 Build.

How do you now use those skills to create your own unique firmware?

It is not expected that anyone will be start a new Project from scratch, so if you want to make your own special firmware, what is the process?

Via MoTeC online, access to our Public Projects is provided for developers to use as a starting point. It is most common that a developer will take a copy of the latest GPR Package and use that as the starting point for their customisations.

MoTeC Online

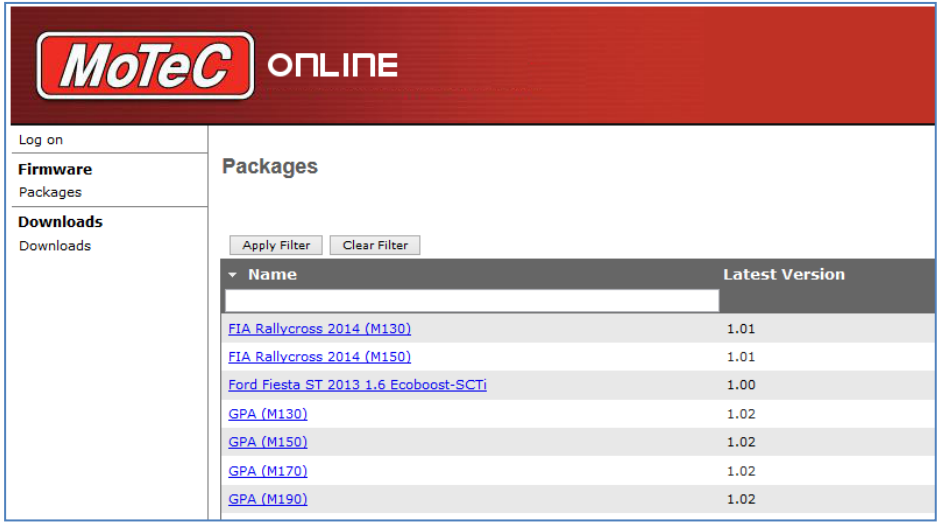
MoTeC Online is the new repository for MoTeC products. Amongst other things, it holds within it all of the packages, projects, licences and other items needed for working with an M1.

https://moteconline.motec.com.au/

When you first open MoTeC Online, you will have access to download any of the publically available packages.

There is also access to the M1 software.

For any additional access you need to register for an account.



MoTeC Online Registration

MoTeC Online is available for any MoTeC dealer to have access to. To gain access, email to dispatch@motec.com.au with your dealer details and ask for access.

If you are not a dealer, then access to MoTeC Online is still available, but the application for an account must come through your local dealer.

Contact your local dealer, and let them know about your interest in M1 Development ECUs. Your dealer will send in an application on your behalf to the same email address.

In general, access to MoTeC Online is limited to dealers, M1 developers and customers genuinely interested in MoTeC Development ECUs.

Motec Online – Logged In

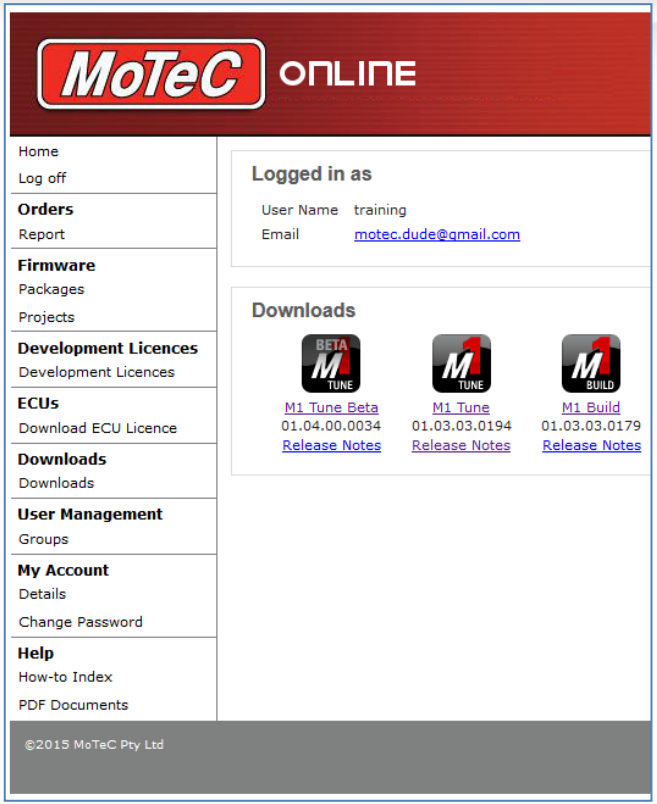
Once you have logged into MoTeC Online, there is a menu down the left hand side of the screen. Your menu options will vary depending on the access level of your account.

Most commonly, MoTeC Online will be used to view and download Packages, Projects and software.

NOTE: A ***Project can be opened in M1 Build* and contains all objects, information and logic of a Package.

During compilation, firmware is generated, definitions of data, security groups, data logging, Worksheets and calibrations are added which results in a Package.

A Package can be opened in M1 Tune. In general, properties that have been defined in M1 Build cannot be changed in M1 Tune.



Downloading a Project

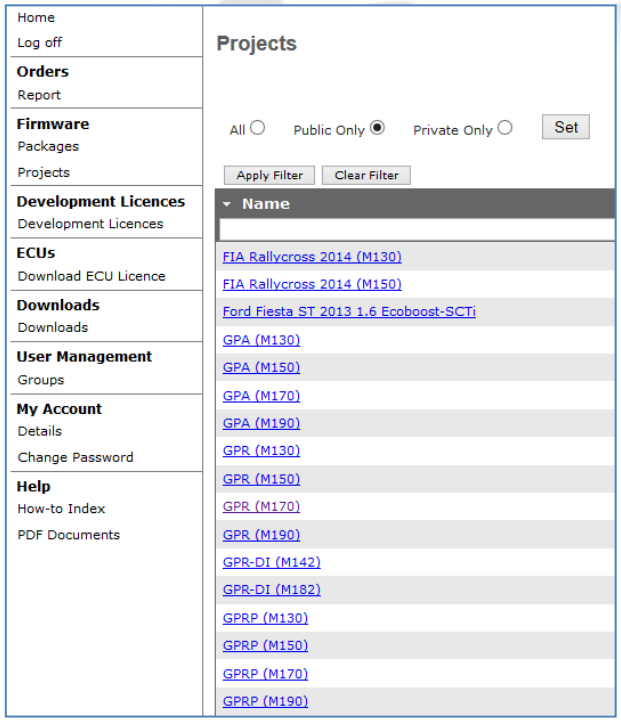
From the **Firmware** heading, select the **Projects** option.

The list shown are all of the publicly available builds that can be downloaded.

For our course, we are going to have a play with GPR (M170) so for those with access to MoTeC Online, select that one.

For those without a MoTeC Online account, a copy of the Project is on your USB.

Select Open to open the Project file, then install the archive.



Opening the GPR (M170) in Build

Go back to M1 Build

Select File/New Project

Select From Existing Project, then Next

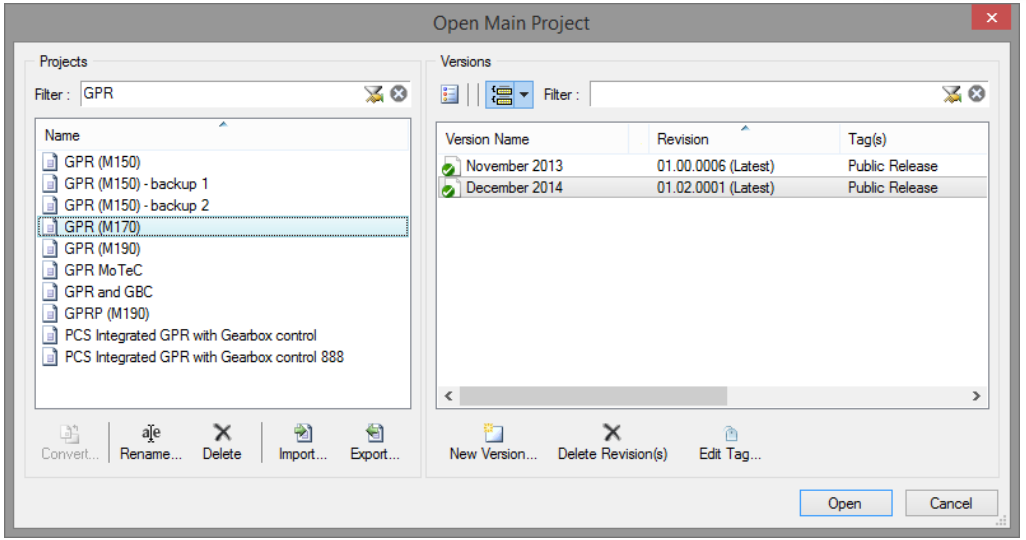
Within the Filter area, type in GPR.

You should be able to find GPR(M170)

When you select it, the right hand pane will show versions you can select from.

Select the latest revision, then click Next

Fill in Project and Version Names - Finish



GPR M170

Given the Project that we have just finished, you can now understand what each of these groups will contain.

It will be a combination of channels, tables, inputs, outputs and functions just to name a few.

To look at how we can modify this Package we will look at the traction control system.

Name	Class	Data Type	Quantity
Root	Group		
ADR	ADR		
Airbox	Group		
Air Conditioner	Group		
Ambient	Group		
Anti Lag	Throttle Based Anti Lag		
Auxiliary	Group		
Boost	Boost Control		
Brake	Group		
CAN	Group		
Clutch	Group		
Constants	Group		
Coolant	Group		
Driver	Group		
E8XX	E8XX		
ECU	Group		
Engine	Group		
Events	Group		
Exhaust	Group		
Fuel	Group		
Gear	Group	Gear Enumeration	
GPS	GPS		
Idle	Group		
Ignition	Group		
Inlet	Group		
Intercooler	Group		
Knock	Group		
Lap	Lap Time		
Launch	Throttle Based Launch Control		
Logging	Group		
PDM	PDM		
Race Time	Group	Floating Point	Time [s]
RS232	RS232 Port		
SLM	Staged SLM		
Steering	Group		
Tachometer	Group		
Throttle	Group		
Traction	Group		
Transmission	Group		
Turbocharger	Group		
Vehicle	Group		
Warning	Group		

Modifying Traction Control

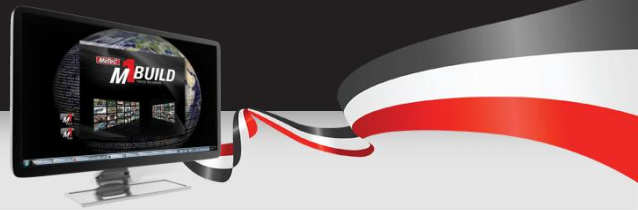
The M1 traction control system operates by limiting the actual ***Engine Speed*** to the ***Traction Engine Speed Limit*** which is calculated from the ***Traction Aim Speed*** value.

The ***Traction Aim speed*** is the current Vehicle Speed + Traction Aim Slip.

The ***Traction Engine Speed limit*** is calculated from tyre circumference and gear/diff ratios.

The ***Traction Engine Speed Limit Ignition Range*** is calculated as Traction Range multiplied by Traction Engine Speed Limit. It defines the engine speed range above Traction Engine Speed Limit where ignition cut is progressively applied for engine speed limiting.

To view how this is coded, expand the ***Traction*** Group, and double click on the ***Update*** scheduled function.



Allowing for Wet Weather Tyres with TC

Sometimes a wet weather tyre for a race car is a different rolling circumference to the dry weather tyres. In our current TC method, you can change the rolling circumference of the tyre in software, but there is no fast way to adjust the TC to allow for the changed tyre diameter.

As an example of how we can modify an existing Package to customise it, we will modify GPR to have the tyre circumference change between a Dry and Wet tyre circumference.

To do this, I am going to assume that we use a switch to change between wet and dry tyres.

Starting your New Group

Add a new group to the ***Root*** by right clicking on the Root/Insert/Built-in /Group.

Call this group **Tyres**.

Now insert a sub group to the ***Tyres*** group called **Circumference**.

Within the ***Circumference*** group, add a sub group called ***Front***.

Within the ***Front*** group, create two parameters, ***Wet*** and ***Dry***.

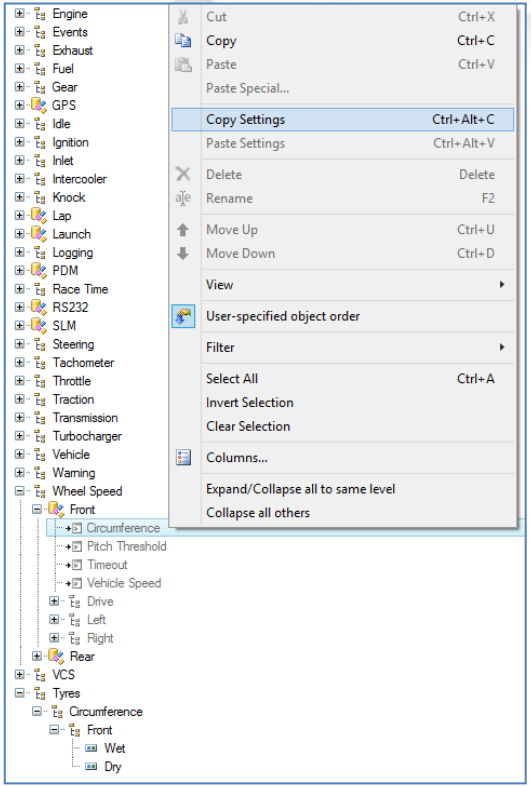
Tyres	Group		
Circumference	Group		
Front	Group		
Wet	Parameter	Floating Point	Unitless
Dry	Parameter	Floating Point	Unitless

Copy Settings

When you are creating a new function, it is often faster to copy a similar item or its settings.

To configure the settings of the dry and wet tyre circumference, ***copy settings*** from an existing circumference channel. In this instance you can use the ***Wheel Speed Front Circumference.***

Select your ***Tyres Circumference Front Wet*** and paste the settings. Do this to the Dry parameters also.



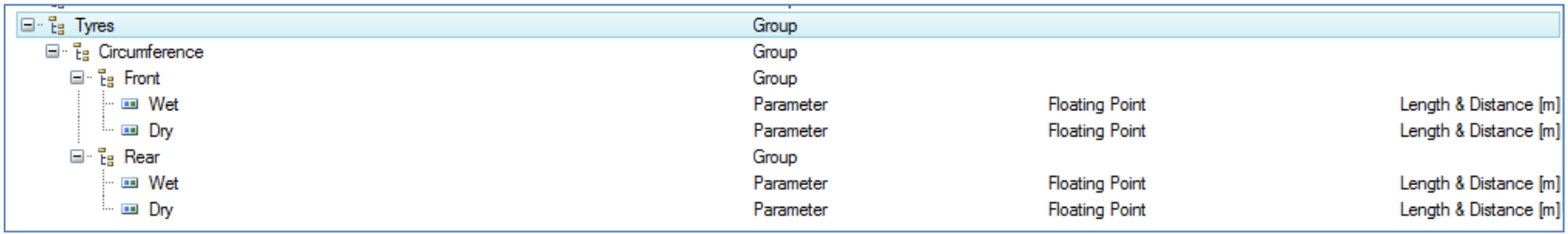
Copy and Paste a Group

To create the ***Rear*** circumference group, we will again take a short cut.

Copy not the settings this time, but the entire ***Front*** group.

Paste this group into the ***circumference*** group.

Rename ***Front 1*** to ***Rear***.



Copy and Paste an Input

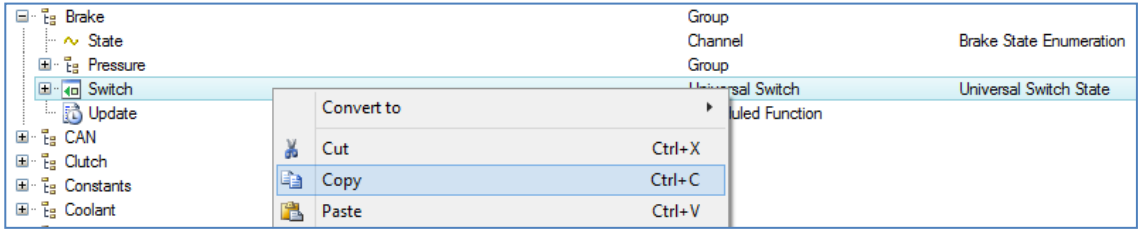
To create most items, it is generally faster to copy a similar item as the starting point.

An example of this is if you want a new temperature sensor, copy the coolant temperature group and paste it into your new temperature group.

In this example, we need a Wet switch. We could create one from scratch, but to create this more quickly, lets just copy an existing switch, in this instance, we can copy the brake switch.

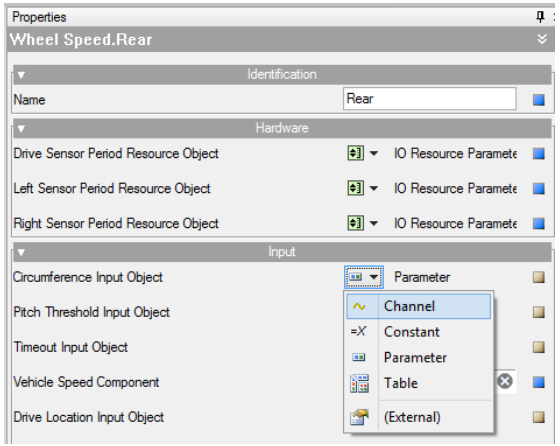
Paste this into your ***Tyres*** Group.

Rename it ***Wet Switch.***



Change Where Speeds Get Circumference

For our example, we want to change the source of the circumference used in the wheel speed calculation to use our new wet and dry values.

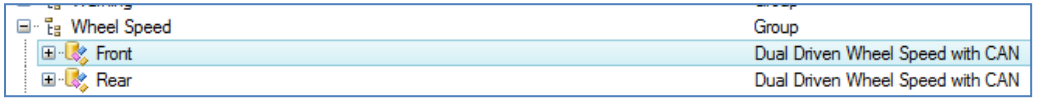


Expand out the ***Wheel Speed*** group and then select ***Front.***

Select the ***Circumference*** input object and change it from ***Parameter*** to ***Channel.*** Make the same changes to the ***Rear*** group.

What we have done here is to change the source of the ***Circumference Front*** and ***Rear*** from a single value the user inputs to a channel that we can define in a scheduled function.

Now we need to create that function.



Create the Function

Right click on the ***Tyres*** group an Insert/Built-in/scheduled function.

Call this scheduled function ***Wet Dry Calculation***

Write this code into your scheduled function:

```
if (Wet Switch eq Wet Switch.On)
{
    Wheel Speed.Front.Circumference = Circumference.Front.Wet;
    Wheel Speed.Rear.Circumference = Circumference.Rear.Wet;
}
else
{
    Wheel Speed.Front.Circumference = Circumference.Front.Dry;
    Wheel Speed.Rear.Circumference = Circumference.Rear.Dry;
}
```

Finally schedule it to run at 100 Hz and build your Project.

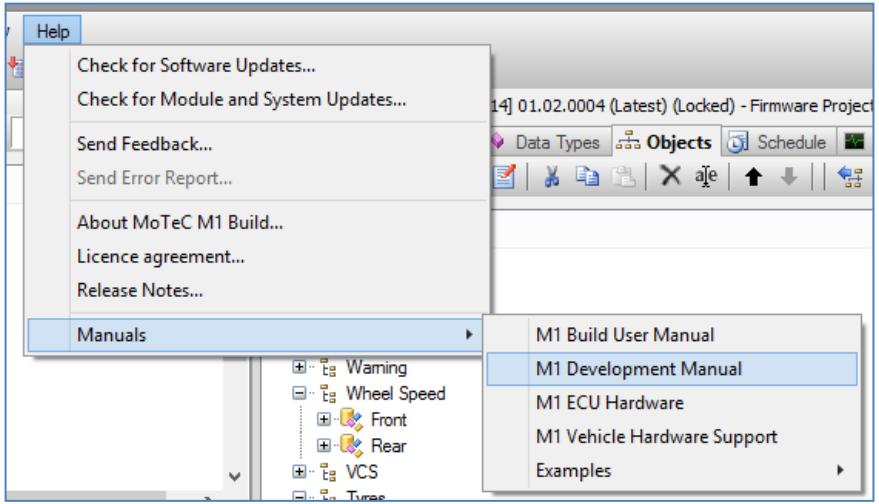
GPR Update Complete

You have now completed a modification to the GPR Package.

This example has shown how you can use the skills from the earlier LED example to modify the GPR Package.

More detailed help can be found from within the build Package under the Help Menu

The Build and Development manuals are located there.



HELP, I'm Stuck!

If you are working on your Project and you get stuck and don't know how to fix your issue:

1) Read the manuals provided with M1 Build.
2) Look through other examples of what you are trying to do in GPR
3) Try the MoTeC M1 Build Forum, where you may get some support from other Build users

From this point, all other support is charged due to the amount of support that may be required. Your options for chargeable support are:

1) **Email** – we offer charged email support to assist to fix your issues.
2) **MoTeC Development Services** – By the hour coding services offered by MoTeC.
3) **External Developers** – Contact MoTeC to be put in touch with an external developer to assist you with your application.

M1 Development Services

The M1 Development Service is a business initiative to support the sale of M1 products. The primary purpose of this service is to provide a means of customising the M1 product to meet individual needs.

What the service provides:

- Development of M1 Projects/Packages based on customer and customer requirements.
- Writing specific software module(s) for a Project based on customers' requirements.
- Reviewing code written by the customer writing their own projects.

There is a complete document which explains more details of the service. You can get a copy if you are thinking of using the service.

M1 Development Model

How does the M1 Development model work?

We need to consider what you might want to do with it. Here are some options:

- Create a one off specialised Package for a customer's car
  - Add in special function to specifically suit their needs
- Create a Package to suit a vehicle modification Package to sell to the public (see example on the right from UGR)
- Create a custom Package to act as a road car ECU replacement to sell to the public

It is with great pleasure that Underground Racing introduces to you the very first Twin Turbo Lamborghini Huracán LP610-4.
Preliminary testing has been a great success. Driving an Underground Racing TT Huracán is something that can only be experienced, not explained.
Driving away on launch control with the help of some boost, that is controlled via our proprietary JRR MoTeC M1 electronics and firmware, and snapping through the gears of the DCT transmission with the additional horsepower is just that, an experience.

The Process

To be able to use M1 Build to produce your own customisations, you merely need to purchase an M1 Development ECU.

An M1 Development ECU is any ECU in the MoTeC M1 range that has a Development Licence loaded into it.

Each Developer's Licence is individually locked to the original purchaser. This means that no one else is able to ever purchase an ECU with your Development Licence. This ensures that anything that you produce in M1 Build is always under only your control. Other people will be able to tune your Package if it is left unlocked, but only you can sell an ECU that can use it.

So, buy your Development ECU and you will be provided by MoTeC with all of the documentation, Licences and access to MoTeC online that you need to get developing.

Selling Your Package Dev ECU

MoTeC have kept in mind your business, and realise that when you create your own custom Package, you will probably want to sell it to your customers.

You can do this any time by purchasing another Development ECU, like your own, with your Licence in it. With that ECU, you can load into it anything built with your Licence.

The Development Licence method of selling your Package gives the most flexibility in how quickly you can make a change in a customer's ECU. That method also means that you can change your Package at any time.

Selling Your Package Partner

The Development ECU method of selling your Package is the most flexible. But it is also means that the customer is paying the cost of a Development ECU each time, plus the cost of your Package to you.

We understand that this may not be a cost effective way of getting your Project out to customers. In this instance, we offer a Partner Package option.

A Partner Package is your own Package that we have rolled up into a finished version that can be purchased only by you at a lower price than a Development Licence.

Partner Package Pricing

The price we charge for Partner Packages is controlled by a set of guidelines. For full details of the guidelines contact MoTeC.

In summary it works like this:

- If the Package has GPA features, you can buy it at your GPA price
- If the Package has GPRP features, you can buy it at your GPRP price

We only charge you for the features in your Package that we supply in the Public Build Projects.

Whatever you add to these Packages yourself on top of this is yours to add your own charges.

So to make a Project worthwhile, you need to add additional features to GPRP to be able to charge more for it and make it worthwhile for the customer.