M1 Build example project

13/6/2019

Note

This project gives an example about possible ways and basic implementations that are required to run an engine.
The main purpose is to show how to generally set up engine projects in M1 Build with the most often used components.
Also, it was intended to include some variety of classes, functions and methods to demonstrate their use.

Although the project is derived from implemented engines and would run an engine, advanced engine projects may need more sophisticated calculations and additional features to get the desired results.

In addition to this presentation the example project is also available as a M1 Build project.

Overview M1 Build example project

- 3 cylinder naturally aspirated engine
- Throttle servo (‘Drive by wire’)
- Basic sensor and hardware component configuration

Content of the presentation:

1) Setup of the hardware and sensors
2) Defining engine conditions and boundary conditions
3) Processes from reading inputs to generating outputs
4) Helpful features for user facilitation

Hardware of the example engine

- Engine speed reference sensor
- Camshaft position sensor
- Inlet manifold pressure sensor
- Inlet manifold temperature sensor
- Throttle servo
- Fuel pump
- Fuel injectors
- Ignition coils
- Coolant temperature sensor

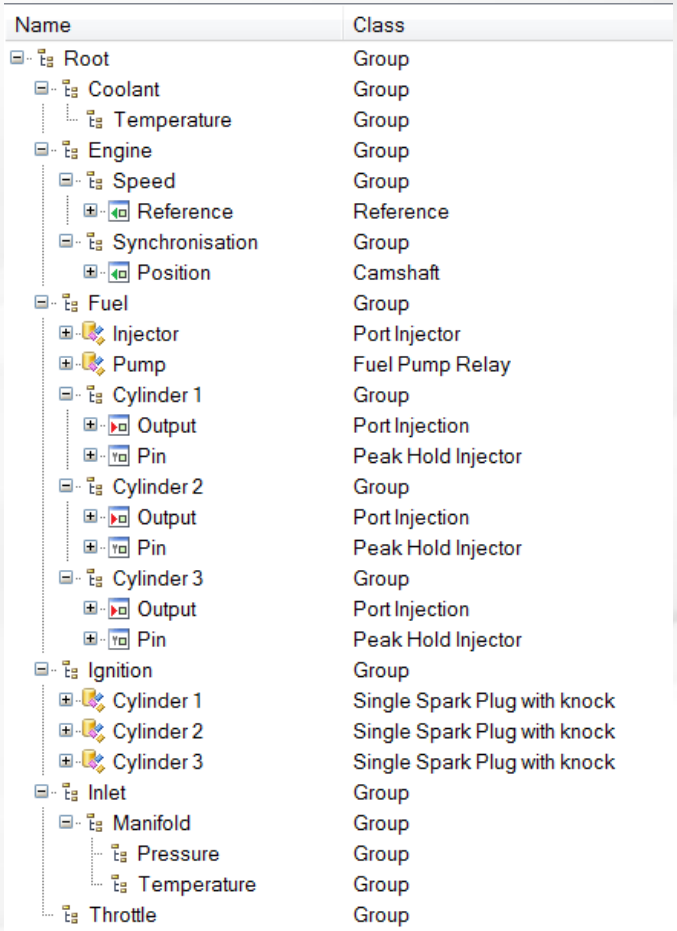
- ECU/Battery Voltage

- (Camshaft phase control)
- (Ambient pressure sensor)
- (Fuel temperature sensor)
- (Fuel pressure sensor)
- (Lambda sensors)
- (Knock sensor)
- (Oil temperature sensor)
- (Oil pressure sensor)

Setup of the hardware and sensors – Getting started

Define the project structure according to the hardware and functional flows.
In the example project, the main groups reflect the given hardware and the engine functions.
In some cases, already respective classes are used (in anticipation of what will be shown later).

- Engine speed reference sensor
- Camshaft position sensor
- Inlet manifold pressure sensor
- Inlet manifold temperature sensor
- Throttle
- Fuel pump
- Fuel injector
- Fuel outputs
- Ignition outputs
- Coolant temperature sensor

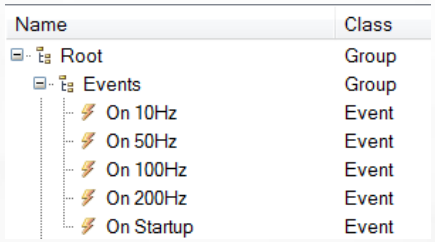


Events

All objects that have to be executed regularly (scheduled objects) need to have an event assigned.

Terminology: an instance of a class implemented in a project is called 'object'

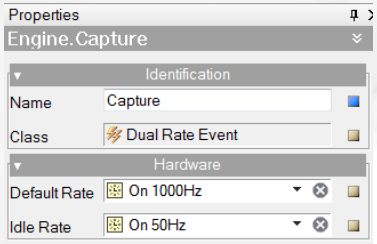
Therefore, an event group is introduced. All events that are needed within the project can be incorporated here.

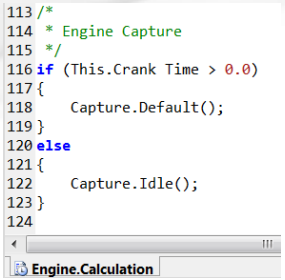


For each scheduled object, the event has to be selected in the Attributes section of the object's Properties window.

In addition, a dual rate event is introduced in the Engine group. This allows updating certain channels or calculations at a higher rate under given conditions.
In the example project, event rates of 1000Hz and 50Hz are selected and the engine speed reference voltage, camshaft position voltage and the ECU Battery voltage are assigned to that event, to be measured at a higher rate during the start process of the engine.
In the Engine Calculation function, the event rate of that object is switched depending on the engine condition.

Note: M1 Tune provides an input capture which logs the engine speed reference inputs at a high rate.
This dual rate event is therefore technically superfluous, however it serves as an example here.





Significance of the Engine speed Reference

To run crank angle based functions like ignition timing, injector timing, camshaft timing, DI fuel pump control and others, an engine reference is needed.
The rotation of the engine is determined by the engine speed reference.
Because a four-stroke engine requires 2 rotations for each power cycle, camshaft position information is needed for synchronisation (except for rare hardware combinations).

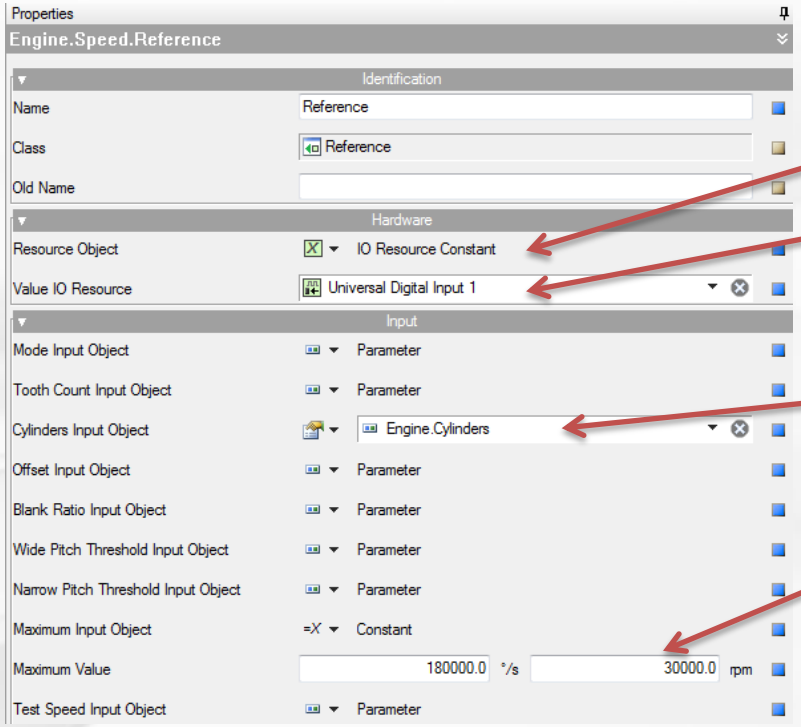
To run an engine, the M1 system in most cases requires that a 'Reference' class for engine speed reference and a 'Camshaft' class for synchronisation are incorporated in the project.
Such classes are included in the modules that come with M1 Build.

The M1 system will not drive injectors or ignition systems until both the reference and camshaft classes have provided synchronised engine timing.
This is reported by the Reference State channel (part of the Reference class).
‘Cycle Lock’ status is required to allow driving of injectors or ignition systems.

Also, injectors will only be driven if an injector configuration (Peak Hold Injector Group class or Direct Injector Group class) and an ignition configuration (Low Side Ignition Group class) is defined.

Engine Speed Reference class

This class is used for provision of the Engine Speed channel.
It must be used in conjunction with a Universal Digital Input class which defines the trigger pattern at the pin.
This class is described on the next slide.
Universal Digital Input 1 is the reserved resource for engine speed measurement in the M1 and must be assigned here.



The Resource Object is set to 'Constant'

Universal Digital Input 1 is selected

Number of cylinders is given as a constant

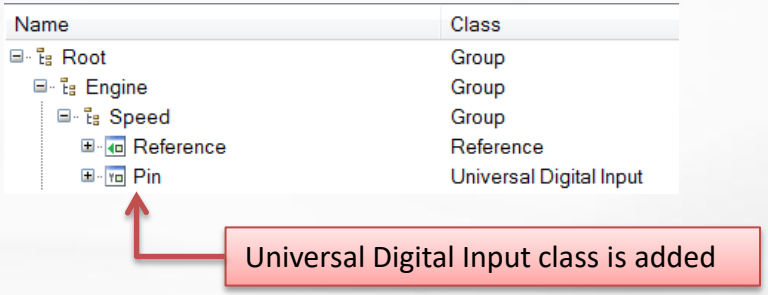
Maximum allowed engine speed is provided as a constant

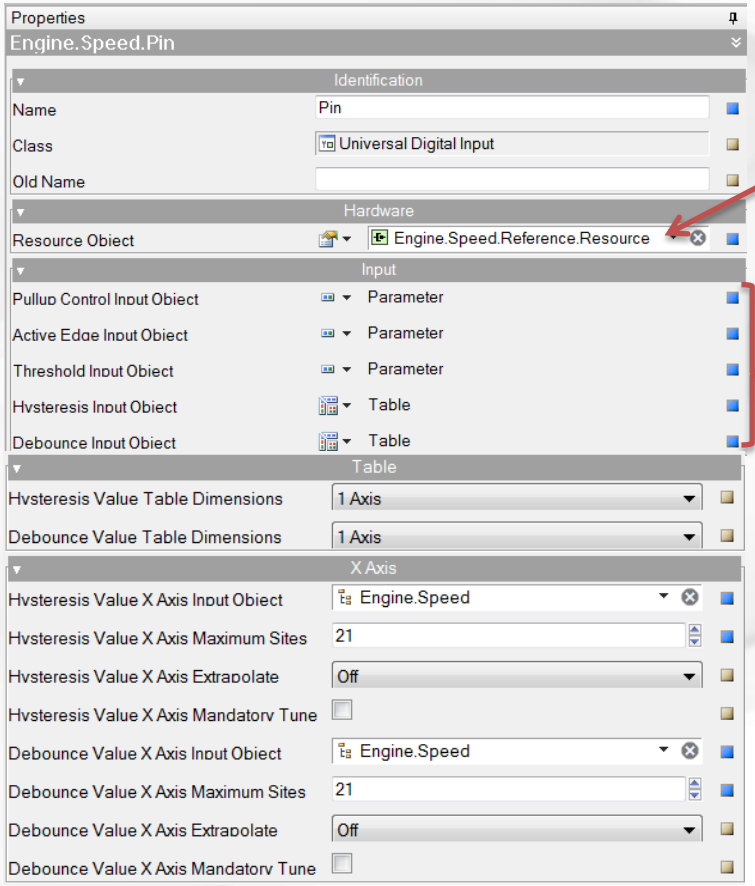
All other Input Objects are defined as Parameters, so they can be defined in M1 Tune

The engine speed is needed with a high accuracy and update frequency to proper calculate the engine conditions.
Therefore, a 200Hz event rate is assigned to the Reference object.
However, Diagnostic channels do not need to be updated so often, so a 10Hz event rate is sufficient there.

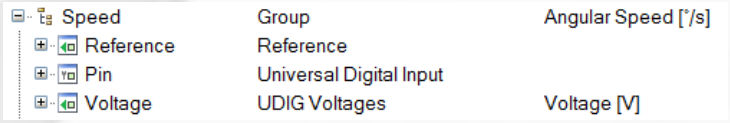
Engine Speed Reference Pin

The Universal Digital Input object for the Speed Reference defines the input characteristic of the crankshaft position sensor, which is translated into a trigger pattern.





To allow a measurement of the voltage on the reference pin, an additional UDIG Voltages class can be used. The resource of that object has to be assigned to the resource of the Engine Speed Reference in the same way as the Pin Resource. Measurement of the voltage is not necessary for the determination of the engine speed, but can be useful for diagnostic purposes.

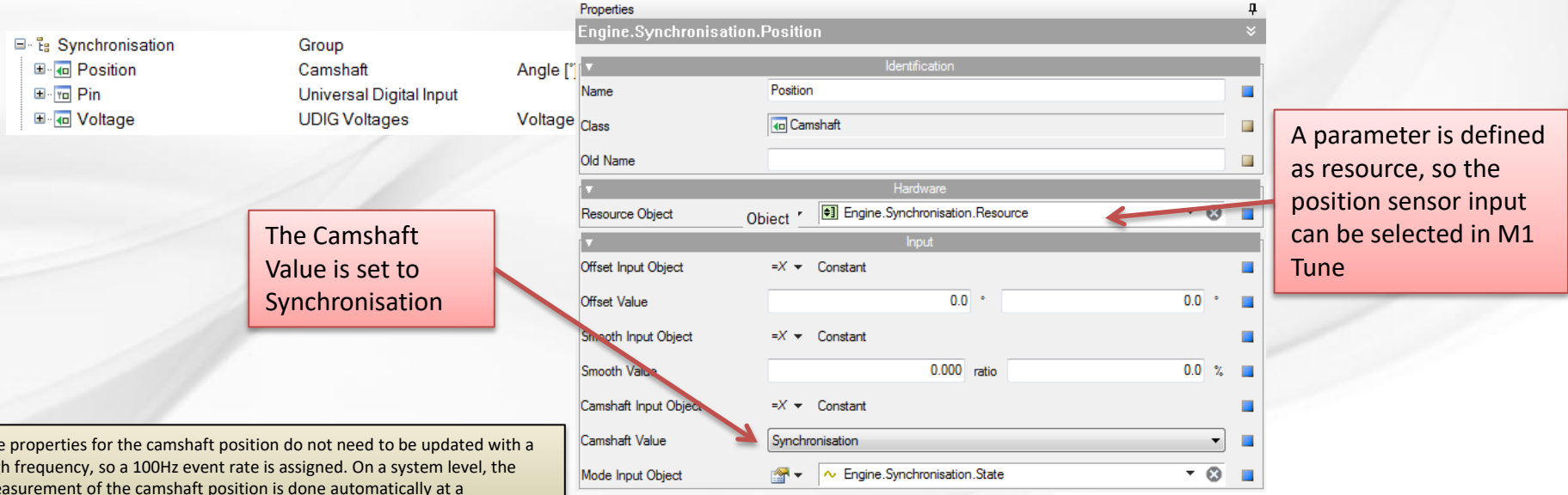


It is not necessary to assign very high event rates to these objects, as on a system level, the measurement of the engine speed is done automatically at a sufficiently fast rate. The event rates of this object only influence the update frequency of its diagnostic information or the reaction of changes in the input values.

Camshaft position

To reach a synchronisation for engine cycle, camshaft position information is needed.
A Camshaft class can be chosen to provide the camshaft position only,
or a Camshaft Phase class can be chosen if the Camshaft is equipped with a phase control actuator.

In this example project, both systems are defined for use.
A Camshaft object is used to deliver the camshaft position for engine speed synchronisation. Comparative to the engine speed reference, also for the camshaft position a Universal Digital Input object is used to define the trigger pattern of the camshaft position sensor, and a UDIG Voltages object is used to measure the sensor signal.



The Camshaft Value is set to Synchronisation

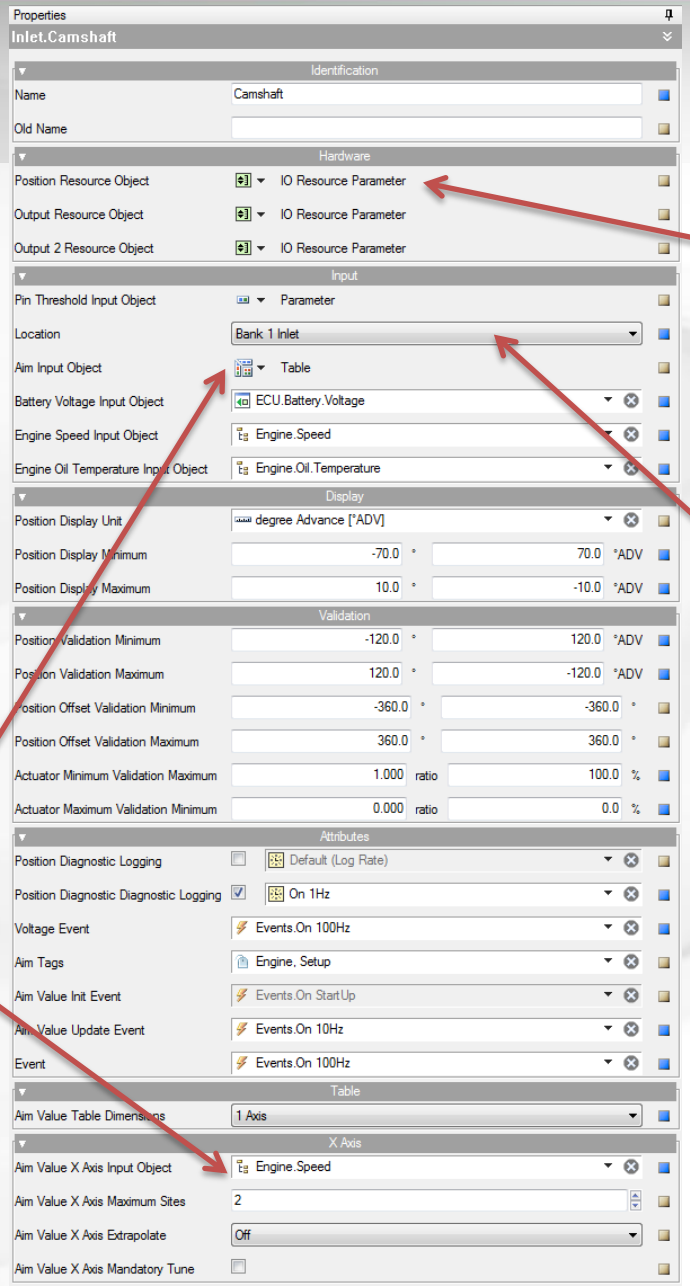
A parameter is defined as resource, so the position sensor input can be selected in M1 Tune

The properties for the camshaft position do not need to be updated with a high frequency, so a 100Hz event rate is assigned. On a system level, the measurement of the camshaft position is done automatically at a sufficiently fast rate.
Diagnostic information does not need to be updated so often, so a 10Hz event rate is sufficient there.

Camshaft phase control

For an inlet camshaft that is equipped with a phase control actuator, a Camshaft Phase class is introduced into the Inlet group.

As already the Camshaft Position object in the Engine Synchronisation group is used to gather the engine speed cycle reference information, the Position Camshaft Value input from the Camshaft Phase object is set to Bank 1 Inlet.
(However, internally the M1 will recognise the 2 camshaft position sensors and use both of them for a faster engine speed synchronisation.)



A parameter is used to define a position resource

The utilisation of the position sensor is defined as Bank 1 Inlet to distinguish it from the synchronisation reference.

The position aim values are realised as a table in dependency of the engine speed

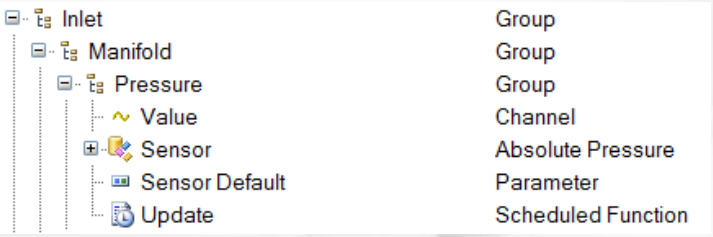
Other necessary settings, for example the output configuration, is provided from the class and is not exposed.

Different event rates are reasonable for this object:
The actuator voltage update and the calculation update should be fast enough to follow engine condition changes. A 100Hz event rate is selected for this.
The Position reading should have the same event rate, as it is used for controlling the actuator.
The aim and diagnostic updates can be at a lower rate.

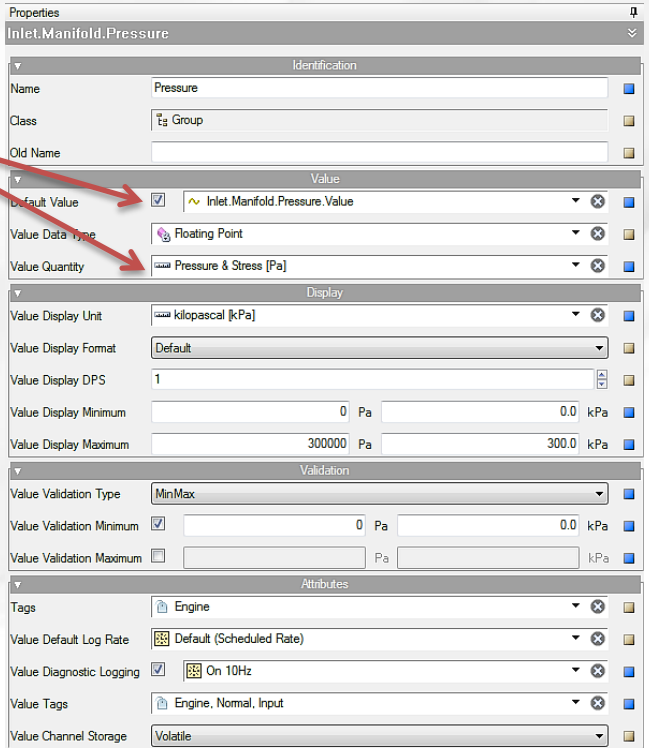
Inlet Manifold Pressure

The inlet manifold pressure is a basic value for many calculations, for example engine load calculation, engine efficiency calibration, fuel and ignition transient trim and fuel mixture aim.
In most cases, a pressure sensor in the inlet manifold will deliver a measured value.

In this example project, a Value channel, an Absolute Pressure class, a Sensor Default parameter and a scheduled function are introduced into the Inlet Manifold Pressure group.
By using the Absolute Pressure class, in M1 Tune the sensor properties can be calibrated manually or the user can select from a comprehensive list of common pressure sensors in the 'Calibration' menu.



The Value channel is assigned as default value to the group. The quantity is set appropriately.



The Value channel is assigned with the measured sensor value in the scheduled function.
If no sensor is assigned or the sensor has a fault, the Sensor value will be NAN (not a number) and Sensor Default is used to provide a limp home functionality.

```
1 if (Calculate.IsNAN(Sensor))
2 {
3     Value = Sensor Default;
4 }
5 else
6 {
7     Value = Sensor;
8 }
```

Inlet.Manifold.Pressure.Update *

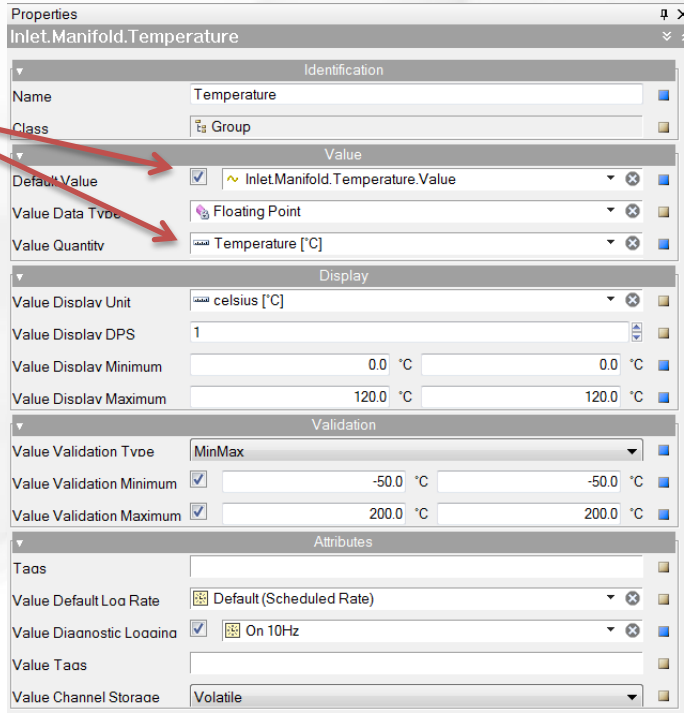
As the inlet manifold pressure is the main value that determines the engine load calculation, an accurate value is necessary that reacts fast to dynamic changes. Therefore, a 200Hz event rate is selected.

Inlet Air Temperature

The inlet air temperature is used for the calculation of the engine load.
Also, this temperature is used for influencing the fuel mixture aim and the ignition timing.

In this example project, a Value channel, a Temperature class, a Sensor Default parameter and a scheduled function are introduced into the Inlet Air Temperature group.
By using the Temperature class, in M1 Tune the sensor properties can be calibrated manually or the user can select from a comprehensive list of common temperature sensors in the 'Calibration' menu.
Additionally, two tables are defined that use the Manifold Temperature as x-axis value. These tables will later be used to compensate the inlet temperature influence on ignition timing and fuel mixture aim.

The Value channel is assigned as default value to the group. The quantity is set appropriately.




```
1 if (Calculate.IsNAN(Sensor))
2 {
3     Value = Sensor Default;
4 }
5 else
6 {
7     Value = Sensor;
8 }
```

Inlet.Manifold.Temperature.Update *

The Value channel is assigned with the measured sensor value in the scheduled function.
If no sensor is assigned or the sensor has a fault, the Sensor value will be NAN (not a number) and Sensor Default is used.

As the inlet air temperature changes slowly, a 10Hz event rate is sufficient.

Throttle - Overview

The throttle position is realising the driver's engine performance desire and influences the engine efficiency. A throttle servo ('drive by wire') requires driving the servo motor and gives a position feedback by a position sensor.

Due to the flow characteristics of a throttle body the effective throttle area is not directly proportional to the position of the butterfly.
Therefore, an Area table is used to describe the correlation between the servo position and the effective throttle area. The resulting value is assigned to Throttle Position.

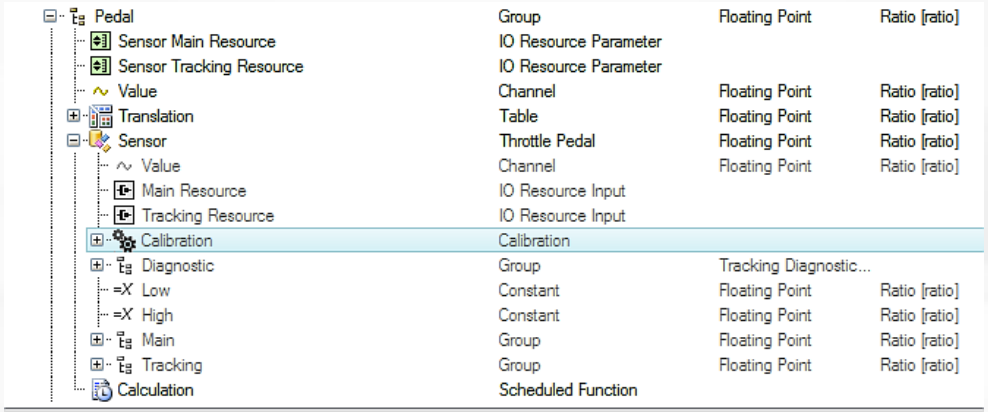
For servo control in this example project a Single Throttle Servo class is chosen, which includes a servo position sensor feedback, a control loop for the servo position, an Output class to drive the motor and diagnostic extents.

A further group to be set up is the throttle pedal which is relative to the driver's request. The pedal position is therefore a main input value for the throttle aim calculation.
Also, a test group is implemented which contains settings for the servo motor step test.

A fast throttle response and control is desired, so the event rate for most of the scheduled objects in this main group is set to 200Hz.

Throttle pedal

For throttle pedals a dedicated class is provided which in M1 Tune allows to select from a list of pedals by the Calibration parameter. Alternatively the throttle pedal position sensor can be configured manually.



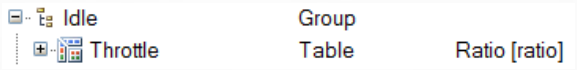

```
1 if (Sensor.Diagnostic.AsInteger() >= Sensor.Diagnostic.OK.AsInteger())
2 {
3     Value = Translation;
4 }
5 else if (Sensor.Diagnostic eq Sensor.Diagnostic.Not in Use) // cable throttle
6 {
7     Value = Throttle.Position;
8 }
9 else    // sensor fault
10 {
11     Value = 0.0;
12 }
13
```

Here we also allow for a cable throttle, where only one position sensor is used.
The Throttle Pedal Translation table can be used to adjust the feel of the throttle pedal. Its input values in the example project are Engine Speed and the Throttle Pedal Sensor value.
The calibrated value is then assigned to the Throttle Pedal value by the Calculation function.

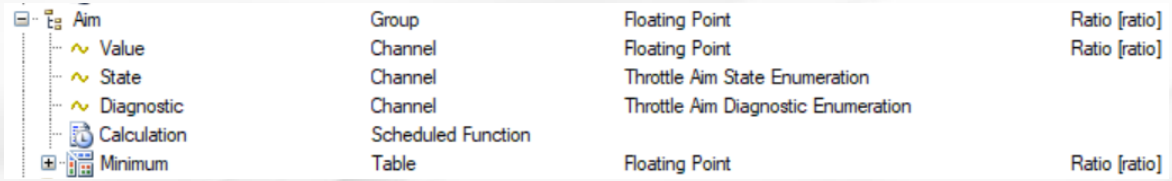
Throttle aim

The throttle position aim is calculated from the driver's request which originates from the pedal value.

Exceptions occur in idle, where the throttle request from the idle group is used.
The Idle Throttle table defines the target idle throttle position in dependency of the coolant temperature.



If the release conditions for the servo motor step test are active, the calibrated values from the Throttle Test group determine the aim.



The Throttle Aim value is then passed through the Throttle Area table and assigned to the servo motor aim channel of the Single Throttle Servo object.
Additionally, a State channel is introduced to report the throttle aim conditions, such as Test, Fault, Idle or Lever.

Fuel pump

A Fuel Pump Relay class is used to drive the fuel pump. This class provides a simple on/off control of the pump.

The pump is activated by the Engine Calculation function using a method called Update inside the Fuel Pump class when the engine starts turning.

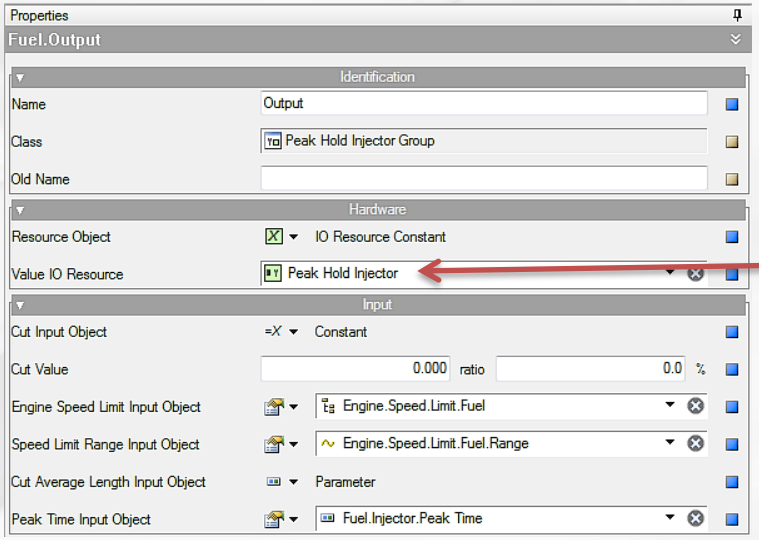
```
/*
 * Fuel pump
 */
Fuel.Pump.Update(
    turning,
    not (Run Switch.Index.AsInteger() > 0) or (Run Switch eq Run Switch.On)
);
```

Note: see the M1 Development Manual for further explanation on syntax and keywords of the M1 programming language and on methods.
See the Build help on a method for details about the method's arguments.

Fuel injector configuration

Some of the necessary objects to drive the injectors have already been introduced, such as Engine Speed and Camshaft references.
The Port Injector class, which is introduced into the project using the name 'Fuel Injector', includes all settings required to drive an injector, such as injector flow characteristics and pin configurations. These values can be set manually to match the injector in use, but the 'Calibration' menu also offers to select from a comprehensive list of common injectors.

However, the injectors will not be driven without a Peak Hold Injector Group class. In this class, basic low level configurations for realising the injector pin output are defined and controlled.
Also, cut level requests and engine speed limits are applied in this class.
Only one instance of this class can be part of the project, providing the configuration for all used injectors.

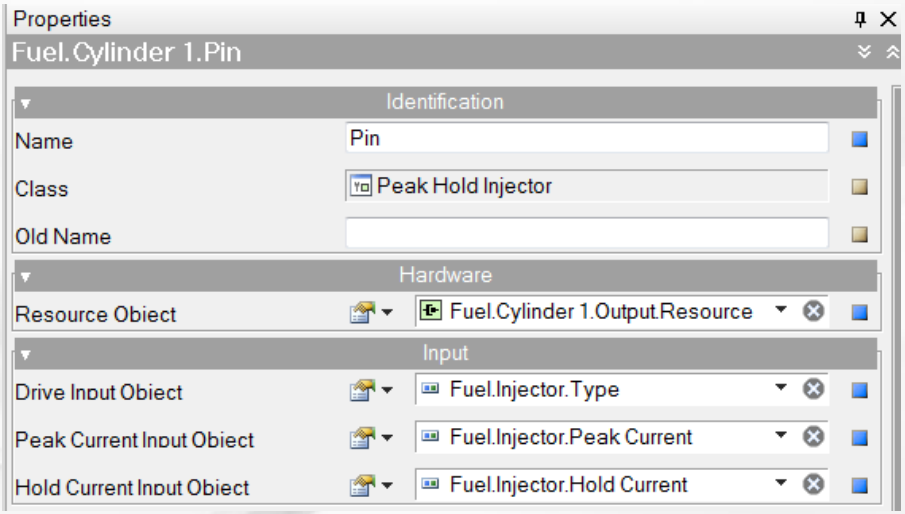


The resource of the Peak Hold Injector Group object has to be assigned to ‘Peak Hold Injector’.

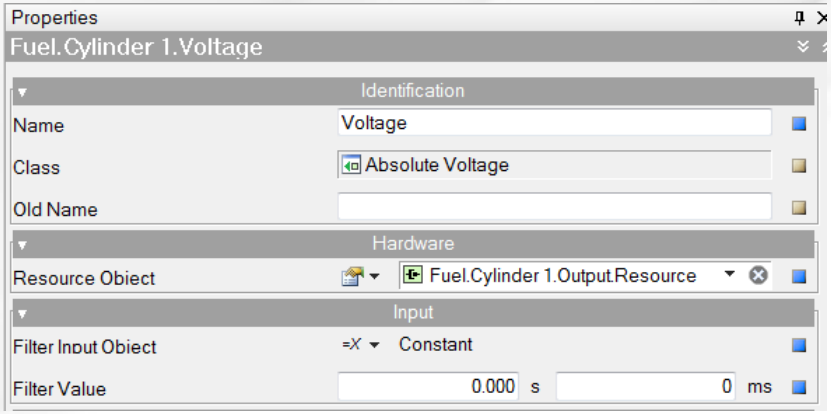
As with other engine combustion related values, also for injector control a fast response is desired, so the event rate for the injector objects is set to 200Hz.

Fuel injector pin settings

Each cylinder has its own Output and Pin classes. The Output class controls the injection lengths and timings. This is described later in this presentation.
The pin settings are provided from the Port Injector class which is already introduced as 'Fuel Injector' into the project. They have to be assigned to the Peak Hold Injector pin classes. The Resource must be linked to the Output Resource.



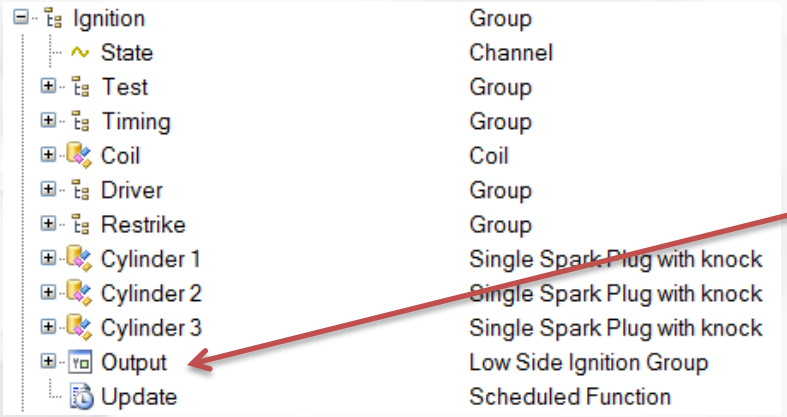
In addition, each injector output gets an Absolute Voltage class assigned. This class provides the voltage at the pin which is used for diagnostic purposes. Also here, the Resource must be linked to the respective Output Resource.



Ignition

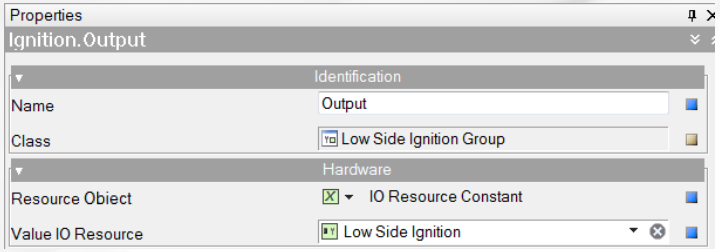
In this example project, Single Spark Plug with knock classes have already been chosen to realise the cylinder individual ignition.
However, ignitions and injections will not be allowed without a Low Side Ignition Group class. In this class, basic low level configurations to configure the ignition pin outputs are defined and controlled, comprising also charging, spark duration(s), cut levels and engine speed limits.
Only one instance of this class can be part of the project, providing the configuration for all ignition objects.

Additionally, in this example project certain groups and objects are introduced to be home of the basic ignition settings. These covers the ignition timing, coil charging, driver and restrike definitions.



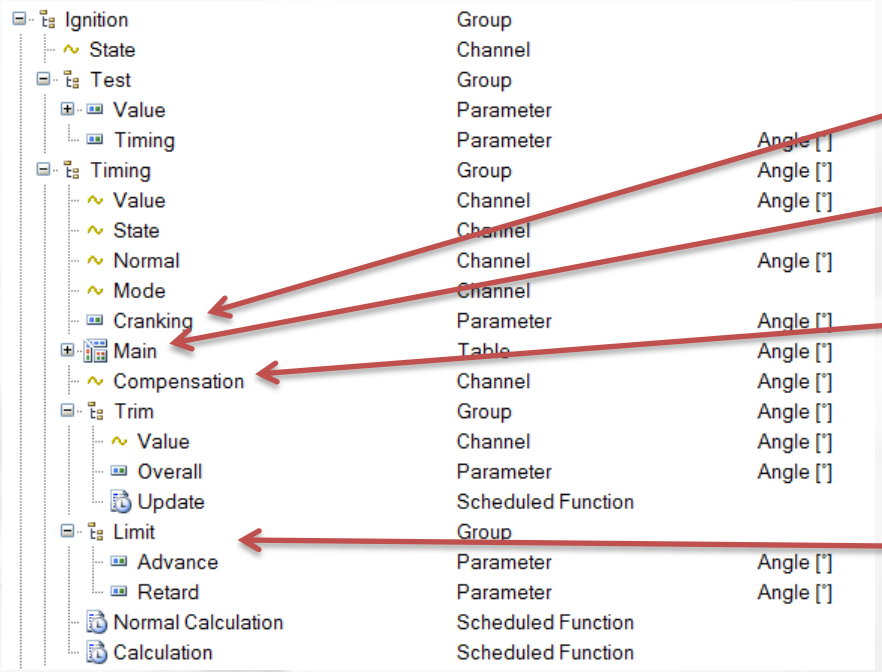
As with other engine combustion related values, also for ignition control a fast response is desired, so the event rate for the ignition objects is set to 200Hz.

The Resource of the Low side Ignition Group has to be assigned to the Low Side Ignition value.



Ignition timing settings

The Timing group contains the sub-groups Trim and Limit and the Main Table for the ignition timing.
The current Ignition Timing Value is determined in the Calculation function.



During cranking of the engine, the separate Cranking parameter is used.

The ignition timing is provided by a table in dependency of engine speed and engine load.

Additionally, the timing is influenced by the coolant temperature and inlet manifold temperature, both of them collected in the Compensation channel.

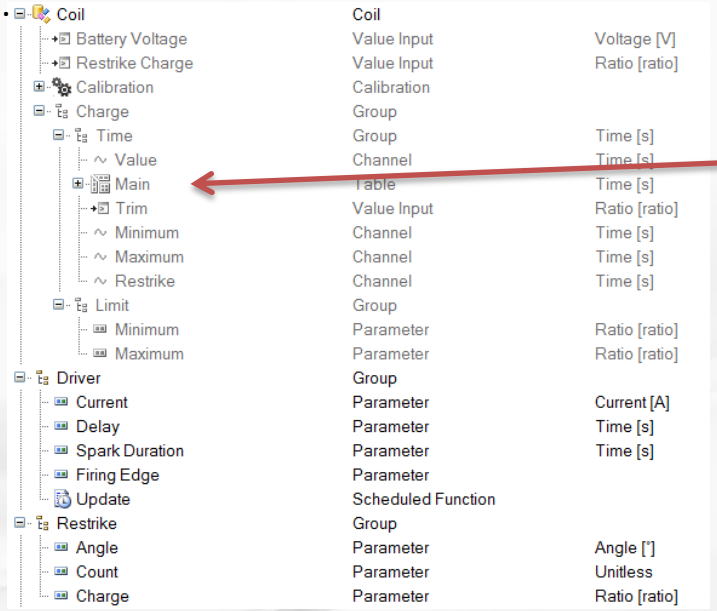
The Timing Value is clamped by the defined Limit parameters.

Mainly to help calibrating the ignition timing, an additive Trim value can be deployed onto the Timing value, and by using the Test parameter, the timing can be specified by the Test Timing parameter.

As with other engine combustion related values, also the ignition timing has to be adjusted fast, so the event rate for the injector objects is set to 200Hz.

Ignition charge, restrike and driver settings

Using the Coil class allows in M1 Tune to define the charge settings manually to match the coil in use or to select from a list of common coils provided by the 'Calibration' menu.



The Main dwell time is defined by a table in dependency of the ECU voltage and engine load.

The Maximum, Minimum and Restrike time channels are defined by multiplication with the corresponding charge limit and restrike ratio parameters.
Additionally, two parameters for Angle and Count complete the restrike strategy.

In the Driver group, the respective parameters specify the settings of the ignition driver.

All values are used in the Low Side Ignition group or on the Single Spark Plug with Knock class, respectively.

Coolant Temperature

The coolant temperature is used in the example project to determine the idle target throttle settings.
Also, this temperature is used for influencing the fuel mixture aim, the fuel volume and the ignition timing.

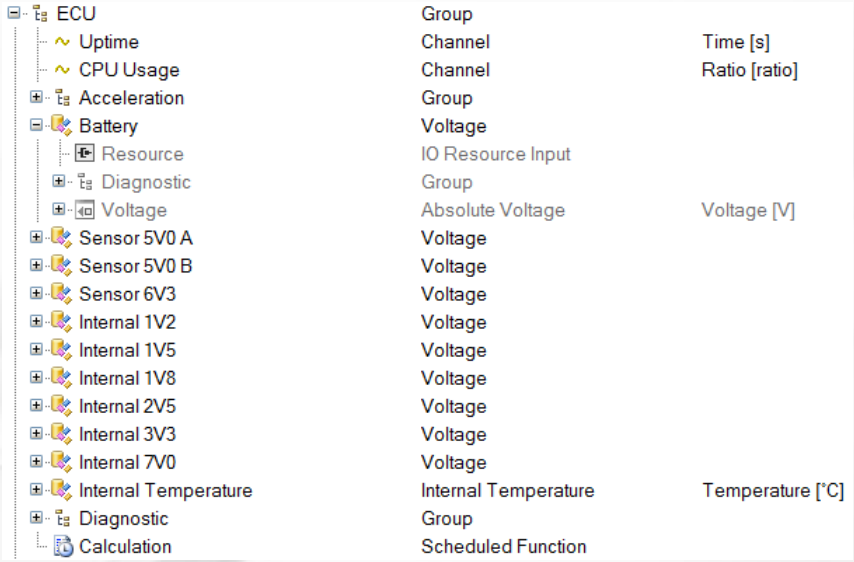
In this example project, a Value channel, a Default parameter, a Temperature class and a scheduled function to assign the sensor measurement to the value are introduced into the Coolant Temperature group.
In M1 Tune, the sensor properties can be set manually to match the sensor in use, but the 'Calibration' menu in the class also allows to select from a comprehensive list of common temperature sensors.
Additionally, four tables (for the above mentioned compensations) are defined that use the Coolant Temperature as x-axis value.

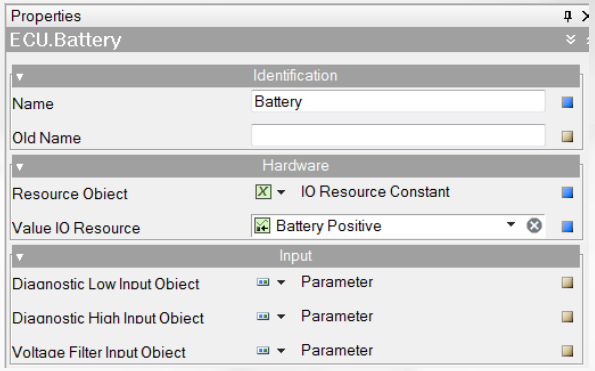
Coolant	Group		
Temperature	Group	Floating Point	Temperature [°C]
Value	Channel	Floating Point	Temperature [°C]
Sensor	Temperature	Floating Point	Temperature [°C]
Default	Parameter	Floating Point	Unitless
Engine Speed Limit	Table	Floating Point	Angular Speed [°/s]
Ignition Timing Compensation	Table	Floating Point	Angle [°]
Fuel Volume Compensation	Table	Floating Point	Ratio [ratio]
Fuel Mixture Aim	Table	Floating Point	Air Fuel Ratio [LA]
Update	Scheduled Function		

As the coolant temperature does change only slowly, a 10Hz event rate is sufficient.

ECU/Battery voltage

The battery voltage is used in the objects that drive the throttle servo, the injectors and the camshaft phase regulator.
Also, the ignition dwell time is dependent on the battery voltage.
The M1 ECU provides an internal voltage resource that is assigned to a Battery object using a Voltage class in the example project.





Diagnostic voltage thresholds can be used to get a warning when the voltage exceeds these values.

In this example project, also other internal ECU voltages and sensor readings are integrated by using Voltage classes. This is for diagnostic purpose only.

For the use of the battery voltage, the dual rate event is selected, so the battery voltage is available at a higher event rate during engine start.
For the other values, event rates of 10Hz to 100Hz are sufficient.

Additional features of the example project

At this point, all hardware is already implemented that is required to run an engine.
However, in the example project some more features are realised that are most commonly used in an engine setup.

These are:

- Lambda measurement
- Knock sensor
- Engine oil temperature sensor
- Engine oil pressure sensor
- Fuel temperature sensor
- Fuel pressure sensor
- Ambient pressure sensor
- Run switch (‘ignition switch’)

Lambda measurement

In this example project, 4 lambda measurement devices are defined for use. That is a separate lambda measurement device for each of the three cylinders as well as one collective measurement device.

The used measurement devices are MoTeC LTC types, which are available for a Bosch LSU or a NTK UEGO sensor. An LTC class is used to run each sensor, which includes all necessary processes to run an LTC.

The scheduled function in the Exhaust Lambda group determines which sensors are active, and collects the respective Lambda value to write it into Value, to filter it and write into Filtered, and to multiply it with the Fuel Closed Loop Control Trim to get the Normalised value. The Diagnostic channel gives the indication about the source of the Lambda value.
Also, in the scheduled function the activation of the sensor heating is done dependent on the engine run time, using predefined library functions from M1 Build tailored for the LTC devices.

The update rate of the sensor signals is limited by the dynamic behaviour of the lambda sensors, therefore an event rate of 50Hz is sufficient for these objects.

Knock control

During Ignition setup, 'Single Spark Plug with knock' classes have been chosen. These objects require the definition of knock information inputs.
For this purpose, a new main group ‘Knock’ is introduced. It contains a Window group where a Knock Window class is implemented. This object generates the knock window output for an SKM device, using the Start, Width and the Frequency A/B/C/D parameters.
The Threshold, Recovery Rate, Trim Gain and Trim Limit tables define the knock settings which are used in the ‘Single Spark Plug with knock’ objects.

Knock	Group		
State	Channel	Knock State Enume...	
Mode	Parameter	Knock Mode Enum...	
Frequency A	Parameter	Floating Point	Frequency [Hz]
Frequency B	Parameter	Floating Point	Frequency [Hz]
Frequency C	Parameter	Floating Point	Frequency [Hz]
Frequency D	Parameter	Floating Point	Frequency [Hz]
Threshold	Table	Floating Point	Ratio [ratio]
Recovery Rate	Table	Floating Point	Angular Speed [°/s]
Activate	Group		
Trim	Group		
Gain	Table	Floating Point	Angle [°]
Limit	Table	Floating Point	Angle [°]
Window	Group		
Start	Parameter	Floating Point	Angle [°]
Width	Table	Floating Point	Angle [°]
DSP	Knock Window		
Warning	Group	Warning Enumeration	
Cylinder 1	Knock Level		
Cylinder 2	Knock Level		
Cylinder 3	Knock Level		
Calculation	Scheduled Function		

The tables provide values in dependency of the engine speed, the knock threshold is additionally dependent from the engine load.

As the values in this group only set the boundary conditions for the knock measurement, an event rate of 50Hz is sufficient.

Although internally the M1 measures knock events with a higher, engine speed correlated rate, the event rate of the knock level objects has to match the adjustment rate of the ignition objects and is therefore set to 200Hz.

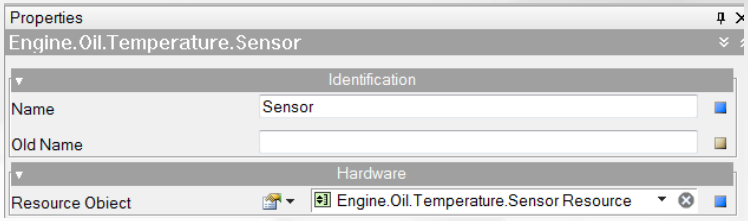
Additionally, for each cylinder a Knock Level class is introduced to deliver information about the knock events for the ‘Single Spark Plug with knock’ objects.

Oil pressure and temperature

These values are suited for monitoring the engine conditions.
The oil temperature is acquired by using a Temperature class and assigning the measured value to the channel. In the absence of valid oil temperature, an Estimate table using Coolant Temperature provides a fallback value.
The oil pressure is obtained by using a Pressure class and assigning the measured value to the respective channel. Additionally, a Universal Switch class is implemented to acquire information about the oil level via a Low Switch.

Oil	Group		
Temperature	Group	Floating Point	Temperature [°C]
Sensor Resource	IO Resource Parameter		
Value	Channel	Floating Point	Temperature [°C]
Estimate	Table	Floating Point	Temperature [°C]
Engine Speed Limit	Table	Floating Point	Angular Speed [°/s]
Sensor	Temperature	Floating Point	Temperature [°C]
Warning	Minimum Maximum Warning	Warning Enumeration	
Update	Scheduled Function		
Pressure	Group	Floating Point	Pressure & Stress [Pa]
Value	Channel	Floating Point	Pressure & Stress [Pa]
Sensor	Pressure	Floating Point	Pressure & Stress [Pa]
Low Switch	Universal Switch	Universal Switch St...	
Warning	Minimum Maximum Warning	Warning Enumeration	
Update Warning	Scheduled Function		
Update	Scheduled Function		

```
if (Sensor.Diagnostic eq Sensor.Diagnostic.Not in Use)
{
    Value = Calculate.NAN();
}
else if (Calculate.IsNAN(Sensor))
{
    Value = Estimate;
}
else
{
    Value = Sensor;
}

local override = false;
This.Warning.Update(
    Sensor,
    override,
    Sensor.Diagnostic eq Sensor.Diagnostic.OK
);
```



The Resource for the engine oil temperature sensor is implemented as an IO Resource Parameter located at the top of the group to make use of an M1 Tune feature: if that resource is set to 'Not In Use', all other objects inside this group are not displayed in M1 Tune.

A Warning class is used to activate a warning and optionally apply an engine speed limit if the oil pressure moves outside the defined Warning Minimum/Warning Maximum range, or when the Low Switch is activated.

Oil pressure and temperature change only slowly, so a 10Hz event rate is sufficient.

Fuel temperature and pressure

The fuel temperature is used for the calculation of the fuel density, which is needed to calculate the fuel volume flow.
It is acquired by using a Temperature class and assigning the measured value to the respective channel.
If no sensor is installed, an estimated value for the fuel temperature will be used.

Temperature	Group	Temperature [°C]
Value	Channel	Temperature [°C]
Estimate	Table	Temperature [°C]
Sensor	Temperature	Temperature [°C]
Update	Scheduled Function	

The fuel temperature changes only slowly, so a 10Hz event rate is sufficient.

The fuel pressure is influencing the injector flow characteristics and is therefore needed as input information to the injector objects.
It is acquired by using a Fuel Pressure class, taking the Sensor type and Regulator Reference into account and assigning the measured value to the respective channel. The Control group may be used to drive a fuel pump.

Pressure	Group	Floating Point	Pressure & Stress [Pa]
Value	Channel	Floating Point	Pressure & Stress [Pa]
Estimate	Channel	Floating Point	Pressure & Stress [Pa]
Default	Parameter	Floating Point	Pressure & Stress [Pa]
Control	Group	Floating Point	Ratio [ratio]
Regulator	Group		
Type	Parameter	Fuel Pressure Regul...	
Ratio	Parameter	Floating Point	Ratio [ratio]
Sensor	Fuel Pressure	Floating Point	Pressure & Stress [Pa]
Warning	Minimum Maximum Warning	Warning Enumeration	
Update	Scheduled Function		
Update Warning	Scheduled Function		

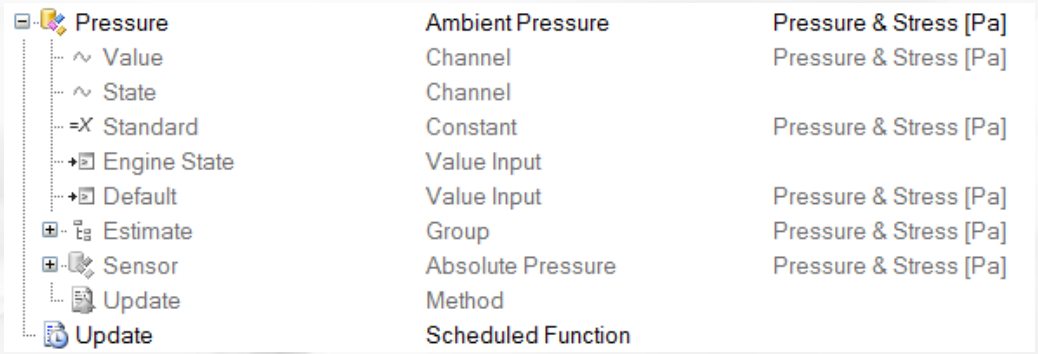
The fuel pressure can have faster changes, so a 100Hz event rate is chosen.

As for the injector characteristics the difference pressure between the fuel rail and the inlet manifold is needed, the Update function also determines the difference pressure. The Update Warning will trigger a warning when the fuel pressure sensor is unavailable or diagnosed as faulty.

Ambient pressure

If the sensor reference of the fuel pressure sensor is not the manifold pressure, the ambient pressure may be needed to calculate the difference pressure between the fuel rail and the inlet manifold.

It is determined by using an Ambient Pressure class.
This class includes a Sensor group to acquire the respective sensor signal. If no ambient pressure sensor is available, an estimated ambient pressure will be calculated within the class, based on the inlet manifold pressure.



The ambient pressure changes only slowly, so a 10Hz event rate is sufficient.

Run switch

The run switch (‘ignition switch’, ‘kill switch’) gives information about the driver’s intention to run the engine.
It is used to enable fuel delivery, fuel injection and ignition.

The switch value is acquired by using a Universal Switch class which can process various input sources.

Engine conditions

At this point, all the hardware of the example engine has been introduced into the M1 Build project.

The next step is to define the engine conditions and the boundary conditions of the engine operation.

In particular, the next section will cover the following items:

- Top Dead Center information
- Engine speed limits
- Engine Overrun definition
- Engine operation conditions

Top Dead Center information

In the Engine group, for each cylinder the top dead center information is provided.
By using this information in the respective subsystems, the correct injection, ignition and knock window timing for each cylinder can be determined.

The TDC of cylinder 1 must be 0°, therefore this value is fixed as a constant.
TDC of cylinder 2 and 3 can be defined in M1 Tune.

Engine speed limits - Overview

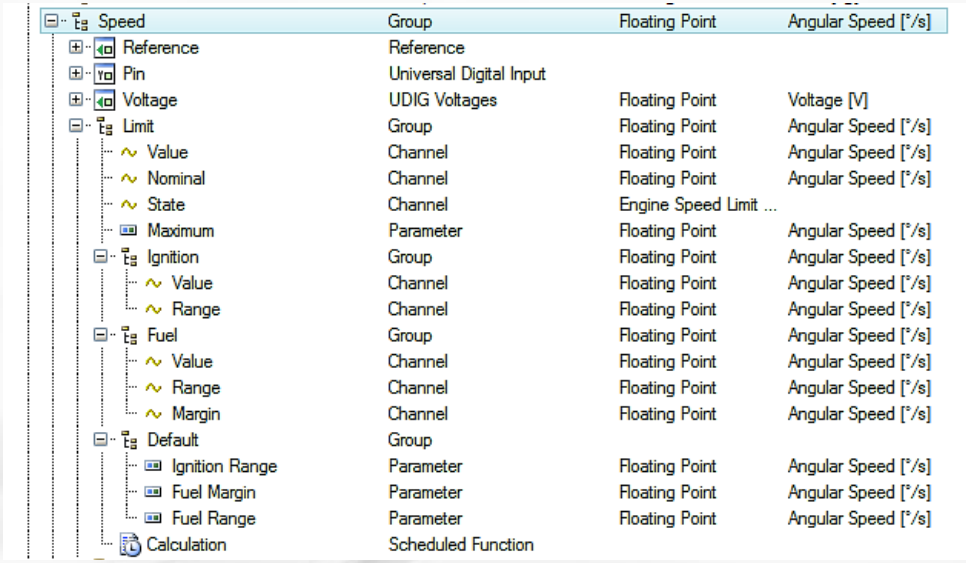
In the classes that were introduced into the example project, engine speed limits are applied at several places:

- Fuel Output: the Peak Hold Injector Group object provides an engine speed threshold and range which allows to gradually switch off the injection.
- Ignition Output: the Low Side Ignition Group object provides an engine speed threshold and range which allows to gradually switch off the ignition.
- Engine Speed Reference: the Engine Speed Reference object will switch both injection and ignition off if a given speed limit is exceeded.
- Throttle: the Single Throttle Servo object contains an engine speed limit in case of servo faults.
- Engine Oil Temperature Warning provides an engine speed limit that becomes relevant when the warning is On.
- Engine Oil Pressure Warning provides an engine speed limit that becomes relevant when the warning is On.

The particular threshold and range inputs in the Fuel Output object and the Ignition Output object allow to implement different cut off strategies.

Engine speed limits - Implementation

In the already existing Engine Speed group of the project, a Limit group is introduced, with the sub-groups Ignition, Fuel and Default.



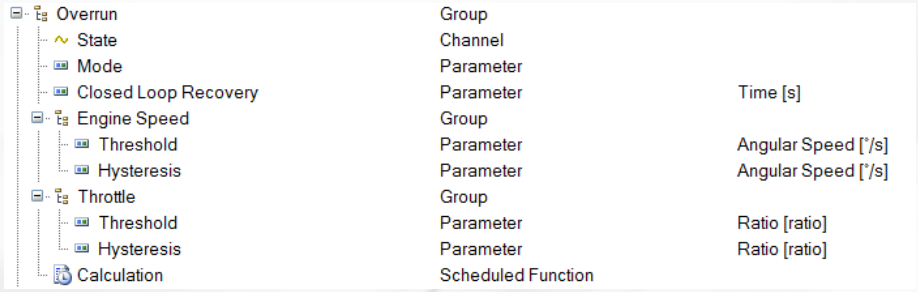
Speed	Group	Floating Point	Angular Speed [°/s]
Reference	Reference		
Pin	Universal Digital Input		
Voltage	UDIG Voltages	Floating Point	Voltage [V]
Limit	Group	Floating Point	Angular Speed [°/s]
Value	Channel	Floating Point	Angular Speed [°/s]
Nominal	Channel	Floating Point	Angular Speed [°/s]
State	Channel	Engine Speed Limit ...	
Maximum	Parameter	Floating Point	Angular Speed [°/s]
Ignition	Group	Floating Point	Angular Speed [°/s]
Value	Channel	Floating Point	Angular Speed [°/s]
Range	Channel	Floating Point	Angular Speed [°/s]
Fuel	Group	Floating Point	Angular Speed [°/s]
Value	Channel	Floating Point	Angular Speed [°/s]
Range	Channel	Floating Point	Angular Speed [°/s]
Margin	Channel	Floating Point	Angular Speed [°/s]
Default	Group		
Ignition Range	Parameter	Floating Point	Angular Speed [°/s]
Fuel Margin	Parameter	Floating Point	Angular Speed [°/s]
Fuel Range	Parameter	Floating Point	Angular Speed [°/s]
Calculation	Scheduled Function		

The Calculation function determines the speed limit settings for the Fuel Output and the Ignition Output objects, defining the respective channels in the Limit Fuel and Limit Ignition groups.
In case of a throttle servo failure, the speed limit which is defined in the Throttle Servo Fault object is used.
In case of an Engine Oil Pressure Warning the speed limit from that subsystem is used.

```
local max = Maximum;
local ir = Default.Ignition Range;
local fm = Default.Fuel Margin;
local fr = Default.Fuel Range;
local s = State.Maximum;

if (Engine.Speed.Reference.State eq Engine.Speed.Reference.State.Test)
{
    max = Engine.Speed.Reference.Maximum;
    s = State.Engine Output Test;
    Nominal = max;
}
else
{
    if (max > Engine.Load.Average.Engine Speed Limit)
    {
        max = Engine.Load.Average.Engine Speed Limit;
        s = State.Engine Load Average;
    }

    if ((Throttle.Servo.Diagnostic.AsInteger() < Throttle.Servo.Diagnostic.Not in Use.AsInteger())
        and max > Throttle.Servo.Fault.Engine Speed Limit
    )
    {
        max = Throttle.Servo.Fault.Engine Speed Limit;
        s = State.Throttle Servo Fault;
    }



    if (max > Root.Fuel.Injector.Duty Cycle.Warning.GetEngineSpeedLimit())
    {
        max = Root.Fuel.Injector.Duty Cycle.Warning.GetEngineSpeedLimit();
        s = State.Fuel Injector Duty Cycle Warning;
    }

    Nominal = max;
}

if (Engine.Speed.Reference.Diagnostic eq Engine.Speed.Reference.Diagnostic.Experimental Mode
     and max > 18000 /* 3000rpm in deg/sec */)
{
    max = 18000;
    ir = 0;
    fm = 0;
    fr = 0;
    s = State.Experimental Reference Mode;
}

This.Value = max;
This.Ignition = max - Calculate.Min(fm, 0.0);   // if fm is negative Ignition is above max
This.Fuel = max + Calculate.Max(fm, 0.0);
This.Ignition.Range = ir;
This.Fuel.Margin = fm;
This.Fuel.Range = fr;
This.State = s;

```

Engine Overrun

Overrun conditions are useful for driveability and cover a deceleration request from the driver at certain engine speeds.
In the example project, an Overrun group is introduced into the Engine group, delivering the Overrun State (which is later used to switch off the fuel injectors and to deactivate the lambda control).



A Threshold and a Hysteresis parameter for engine speed and throttle position define the activation conditions for the overrun.

In the Calculation function, the State channel value is determined.

```
local ensp = Calculate.Hysteresis(Engine.Speed, Engine Speed.Threshold, Engine Speed.Threshold + Engine Speed.Hysteresis, 0.0);
local thrp = Calculate.Hysteresis(Root.Throttle.Position, This.Throttle.Threshold, This.Throttle.Threshold + This.Throttle.Hysteresis, 0.0);

if (ensp and (not thrp) and (Mode eq Mode.Enabled))
{
    State = State.Enabled;
}
else
{
    State =  State.Disabled;
}

```

Engine operation conditions

Engine operation conditions are primarily needed to decide when to enable the active output components like the injectors, the fuel pump, the ignition and the camshaft control.
In the example project, the Engine Calculation function identifies the engine condition.

The Engine Speed Reference State channel is indicating the condition of the engine speed determination.

```
local et = System.ElapsedTime();

/*
 * Determine engine state
 */
local cyclelock = false;
local stopped = false;
local turning = false;
when (This.Speed.Reference.State)
{
    is (Open or Initialise or Stall or Error or Exit)
    {
        stopped = true;
    }
    is (Test)
    {
        stopped = true;
    }
    is (First Edge or First Period or Search or Pattern Lock)
    {
        turning = true;
    }
    is (Cycle Lock)
    {
        turning = true;
        cyclelock = true;
    }
}

```

The M1 Engine Speed Reference class also provides a Test Mode, where the engine speed is set to a calibrated parameter.
State is ‘Test’ then.

Additionally, the Run Threshold is used to differentiate between a cranking and a running engine.
Accordingly, Stall Time, Run Time and Crank Time are calculated.

```

/*
 * Crank Time, Run Time, Stall Time and engine state.
 */
if (stopped or Ignition Switch eq Ignition Switch.Off)
{
    Stall Time = Stall Time + et;
    This.Crank Time = 0.0;
    Run Time = 0.0;
    State = State.Stop;
}
else if (Run Time > 0.0)
{
    Run Time = Run Time + et;
}
else if (This.Speed > This.Run Threshold and cyclelock)
{
    Run Time = et;
    Stall Time = 0.0;
    This.Crank Time = 0.0;
    State = State.Run;
}
else if (This.Crank Time > 0.0 or cyclelock)
{
    This.Crank Time = This.Crank Time + et;
    Stall Time = 0.0;
    State = State.Crank;
}
```

Engine operation strategy based on the engine conditions

The determined engine operation conditions are used to activate the fuel injectors, the fuel pump, the ignition and the camshaft control.

```
local testmode = Engine.Speed.Reference.State eq Engine.Speed.Reference.State.Test;
local off = ((Engine.Run Switch eq Engine.Run Switch.Off)
          or Engine.Overrun.State eq Engine.Overrun.State.Enabled) and not testmode;

if (testmode)
{
    expand (n = 1 to Engine.Maximum Cylinders)
    {
        Cylinder $(n).Output.Test = Cylinder $(n).Output.Test.Enabled;
    }
}
else
{
    expand (n = 1 to Engine.Maximum Cylinders)
    {
        Cylinder $(n).Output.Test = Cylinder $(n).Output.Test.Disabled;
    }
}

local cs = off ? Port Injection Enable.Disabled : Port Injection Enable.Enabled;
local c1 = cs;
local c2 = cs;
local c3 = cs;

when (Test)
{
    is (Disabled)
    {
        if (off)
        {
            State = State.Disabled;
        }
        else
        {
            State = State.Enabled;
        }
    }
    is (Cut Cylinder 1)
    {
        c1 = c1.Disabled;
        State = State.Cut Cylinder 1 Test;
    }
    is (Cut Cylinder 2)
    {
        c2 = c2.Disabled;
        State = State.Cut Cylinder 2 Test;
    }
    is (Cut Cylinder 3)
    {
        c3 = c3.Disabled;
        State = State.Cut Cylinder 3 Test;
    }
}

expand (n = 1 to Engine.Maximum Cylinders)
{
    if ($(n) > Engine.Cylinders)
    {
        c$(n) = c$(n).Disabled;
    }
}

if (Injector.Location eq Injector.Location.Not in Use)
{
    expand (n = 1 to Engine.Maximum Cylinders)
    {
        Cylinder $(n).Output.Enable = Cylinder $(n).Output.Enable.Disabled;
    }
}
else
{
    expand (n = 1 to Engine.Maximum Cylinders)
    {
        Cylinder $(n).Output.Enable = Cylinder $(n).Output.IsActive() ? c$(n) : Cylinder $(n).Output.Enable.Disabled;
    }
}
```

Fuel Update Enables Calculation function:
Fuel injection is allowed when the run switch is on and no overrun conditions apply. An active output test will override these conditions.

The conditions to enable the ignition are very similar, see the Ignition Update function.

Camshaft and fuel pump operations are enabled in the Engine Calculation function:
Camshaft control is activated when the engine is running, Fuel Pump is activated when the engine is turning.

```
/*
 * Cam control
 */
local running = (This.Speed > This.Run Threshold) and cyclelock;
Inlet.Camshaft.SetEnable(running);

/*
 * Fuel pump
 */
Fuel.Pump.Update(
    turning,
    not (Run Switch.Index.AsInteger() > 0) or (Run Switch eq Run Switch.On)
);
```

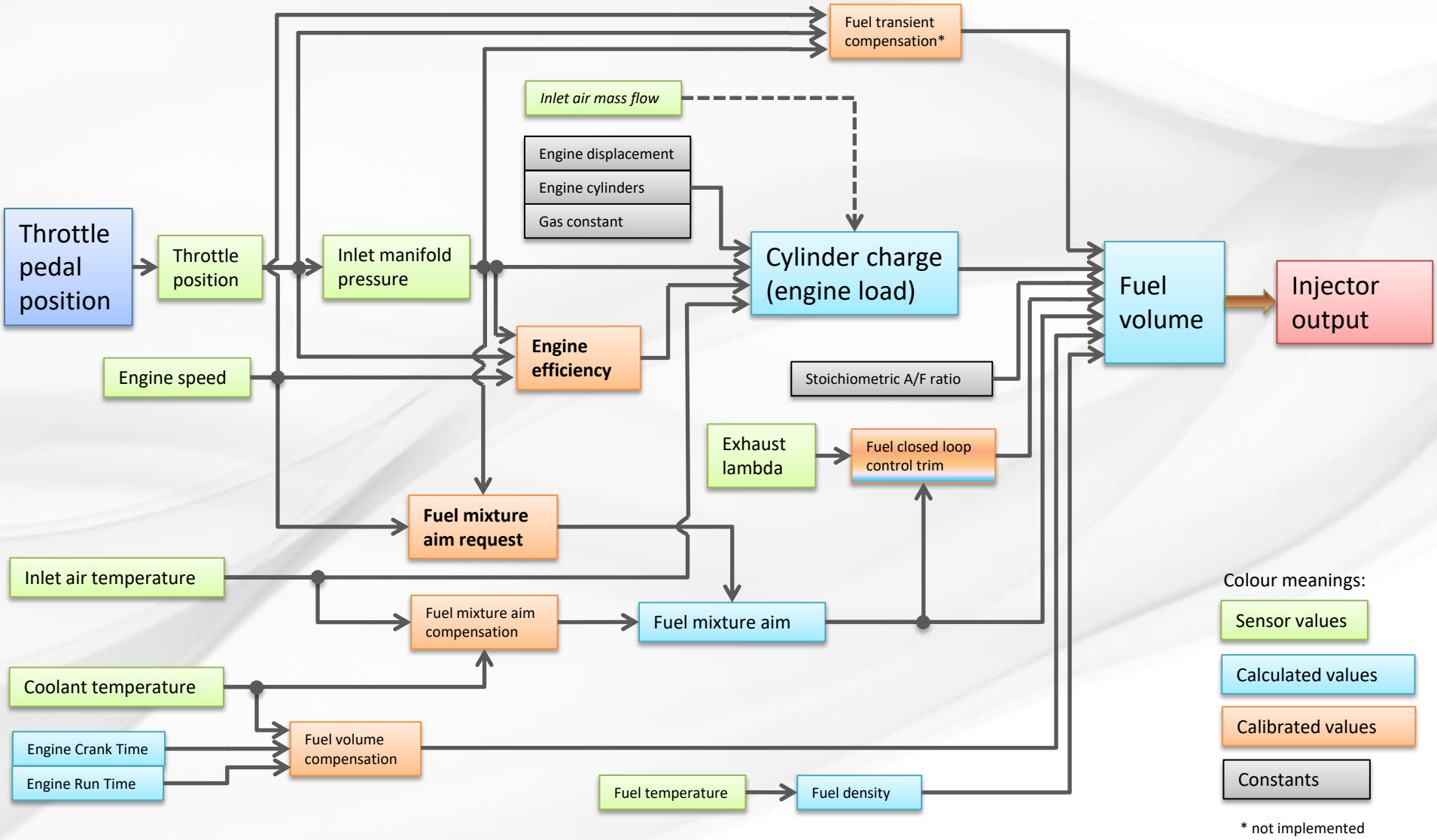
Processes from reading inputs to generating outputs

All hardware is now introduced and the boundary conditions for the engine operation are defined.
The following section describes the process flow and shows how the settings for the fuel injection are calculated.

Topics covered are:

- Overview of the way from sensor readings (input) to injector output
- Calculation of engine load
- Determination of the fuel mixture aim
- Fuel volume calculation
- Fuel closed loop lambda control
- Fuel compensations
- Implementation of the injection event

Fuel delivery process flow



Engine load

The engine load is calculated based on the ideal gas law.

$$m_{Air} = p * \eta * \frac{V_D}{z} * \frac{1}{R * T_{Air}}$$

with

m_{Air}	=	engine load (air mass)
p	=	inlet manifold pressure
η	=	engine efficiency
V_D	=	engine displacement
z	=	cylinder amount
R	=	gas constant for air
T_{Air}	=	inlet manifold temperature

In the example project, the Displacement, Cylinder number and Engine Efficiency are introduced into the Engine group.

Engine Efficiency is implemented as a 3 axis table with engine speed, inlet manifold pressure and throttle position as axis values.

The Calculation function calculates the load value using the above shown formula. Efficiency Mode is used to determine the source of the inlet manifold pressure component in the calculation.

Efficiency	Group	Floating Point	Ratio [ratio]
Value	Table	Floating Point	Ratio [ratio]
Mode	Parameter	Engine Efficiency M...	

```
local p = 0.0;
when (Engine.Efficiency.Mode)
{
    is (Manifold Air Density)
    {
        p = Inlet.Manifold.Pressure;
    }
    is (Ambient Air Density)
    {
        p = Ambient.Pressure;
    }
}

    Value =
        p *
        Engine.Efficiency *
        (Engine.Displacement / Engine.Cylinders) *      // Displacement of one cylinder
        (1 / (Constants.rAir * (Inlet.Air.Temperature + 273.15)));  // Ideal gas law
```

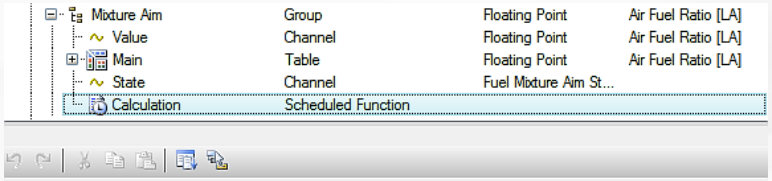
As with other engine combustion related values, the engine load needs to be updated at a high rate, so the event rate for the function is set to 200Hz.

Fuel mixture aim

In the example project, a group named Mixture Aim is introduced into the Fuel group.
The target lambda is provided by a table in dependency of the engine speed and the inlet manifold pressure.

The Calculation function determines the current Fuel Mixture Aim Value by allowing the aim tables from other subsystems to override the Main table value.

In the Properties of the Mixture Aim group, Value is assigned to the group as default value.




```
local v = Main;
local s = State.Normal;

if (Coolant.Temperature.Fuel Mixture Aim < v)
{
    v = Coolant.Temperature.Fuel Mixture Aim;
    s = State.Coolant Temperature;
}

if (Inlet.Air.Temperature.Fuel Mixture Aim < v)
{
    v = Inlet.Air.Temperature.Fuel Mixture Aim;
    s = State.Inlet Air Temperature;
}

if (Engine.Load.Average.Fuel Mixture Aim < v)
{
    v = Engine.Load.Average.Fuel Mixture Aim;
    s = State.Engine Load Average;
}

if (Throttle.Position.Fuel Mixture Aim < v)
{
    v = Throttle.Position.Fuel Mixture Aim;
    s = State.Throttle Position;
}

Value = v;
State = s;

```

As with other engine combustion related values, the mixture aim needs to be updated at a high rate, so the event rate for the function and the table is set to 200Hz.

Fuel volume

Based on the definition of lambda, the needed fuel volume is calculated by the formula :

$$V_{Fuel} = \frac{m_{Air}}{L_{st} * \lambda * \rho_{Fuel}}$$

with

V_{Fuel}	=	fuel volume
m_{Air}	=	engine load
L_{st}	=	stoichiometric A/F ratio
λ	=	target lambda
ρ_{Fuel}	=	fuel density

However, before the target fuel volume can be passed to the injector, some compensations have to be made to cover additional influences onto the needed fuel volume:

- Variance in the used hardware, tolerances in the calibration and the simplification of physical correlations all add up in a deviation between the target mixture lambda and the effective lambda.
  This is compensated by using a lambda closed loop control.

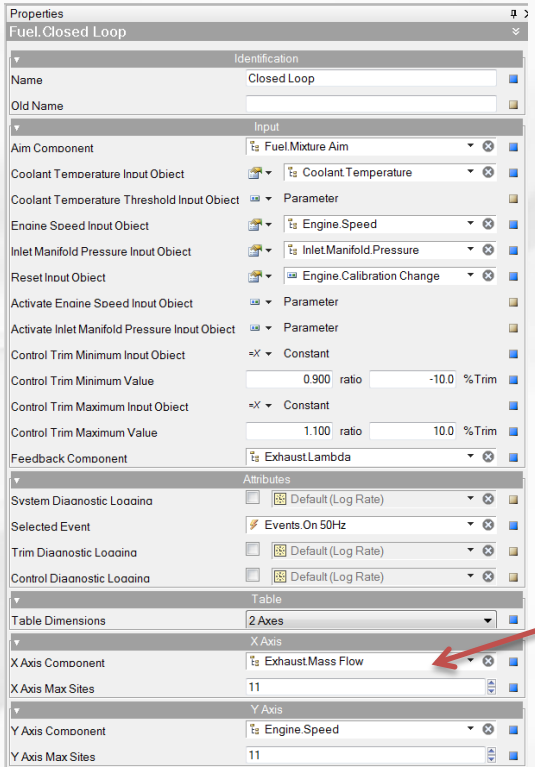
- Sensor reactions may lag behind in dynamic changes of engine conditions.
  This can be compensated by transient corrections.

- Engine temperature effects on fuel vaporisation and combustion efficiency are not neglectable.
  This is compensated by applying temperature compensations.

Lambda control

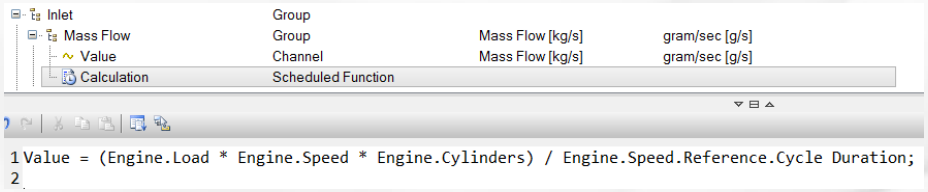
To compensate the deviation between the target mixture lambda and the effective lambda, a closed loop lambda control is used.
In the example project, lambda measurement is already set up using LTC's, and a target lambda has been defined. Therefore, a lambda control can easily be implemented.

For that purpose, a Single Closed Loop Fuel class is introduced into the Fuel group of the project. The Control Trim value of this object is then applied against the calculated fuel volume.
The Fuel Closed Loop Enable Calculation function activates the closed loop control in dependency of the engine conditions, using the provided SetEnable-Method of the object.

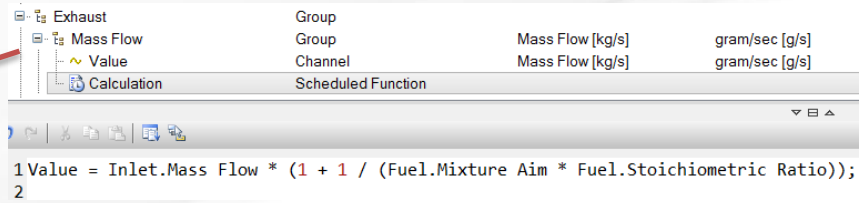


All necessary input objects are already available in the project, except of the Exhaust Mass Flow, which is assigned to the x axis for the system dead time table inside of the class.

To get the exhaust mass flow, a Mass Flow group is introduced into the Inlet group. A function calculates the inlet mass flow value based on the engine speed, the engine load and the number of cylinders.



Then a Mass Flow group is introduced into the Exhaust main group. A function calculates the exhaust mass flow value based on the inlet mass flow, the target lambda and the stoichiometric A/F ratio.



The lambda control is set to the same event rate as the reading of the lambda sensor values (50Hz).

Transient fuel effects

When engine conditions change rapidly, sensor reactions may lag behind and the engine load calculation may be not accurate enough to cover such unsteady conditions.
Also, part of the injected fuel will be sprayed onto the inlet duct wall, remain there in liquid form and not reach the combustion chamber.
On the other side, when the intake valve opens, not only fuel coming from the latest injection will be sucked into the combustion chamber, but also fuel from the wall film will come loose and get into the combustion chamber.

These effects should be are taken into account when the fuel volume for the injection is determined.

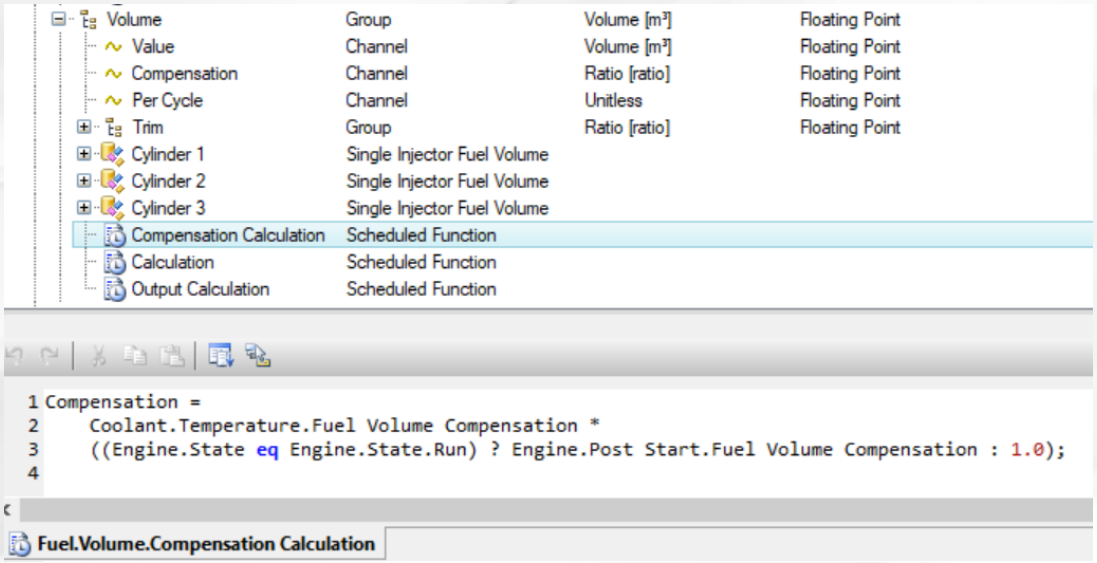
The example project does not include transient fuel compensations.
However, M1 Build provides dedicated classes for modelling of fuel films and determination of fuel volume compensations.

Engine temperature effects on fuel volume compensation

At lower engine temperatures, the fuel vaporisation is poorer, so additional fuel is needed to care for that effect.
In addition, during cranking of the engine the mixing of air and fuel is poor due to the low air speed.

In the example project, a fuel volume compensation table has already been introduced into the Coolant Temperature group, providing a general compensation value in dependency of the coolant temperature.

Now, additional compensation factors in dependency of the crank time (during cranking of the engine) and in dependency of the run time (after start of normal engine operation) are introduced into a Fuel Volume Compensation group.
The function in this group determines the compensation to be used currently. This value is then applied against the calculated fuel volume.

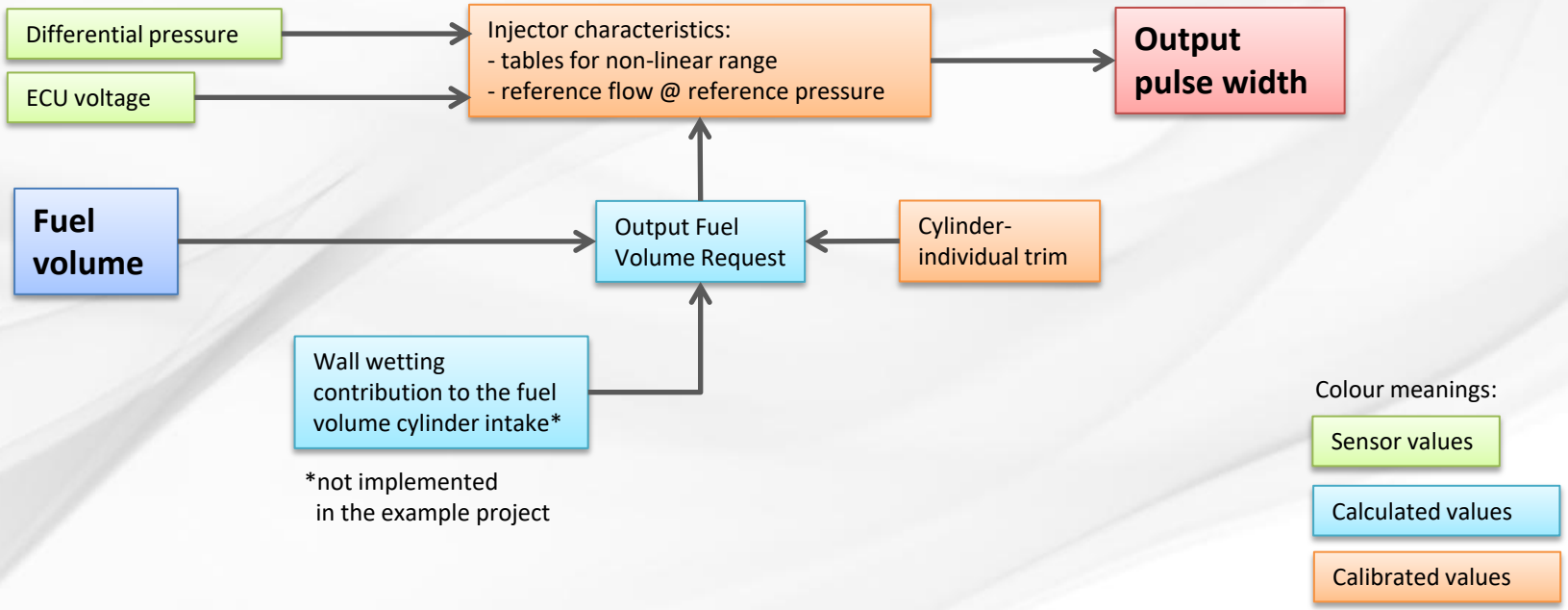


As with other engine combustion related values, the fuel compensation needs to be updated at a high rate, so the event rate for this object is set to 200Hz.

Process for injector pulse width determination

When the desired fuel volume for the current cycle is defined, it must be ensured that this amount of fuel is supplied into the engine.

The following graphic shows the process how a pulse width for the injectors is determined to deliver the desired fuel volume.

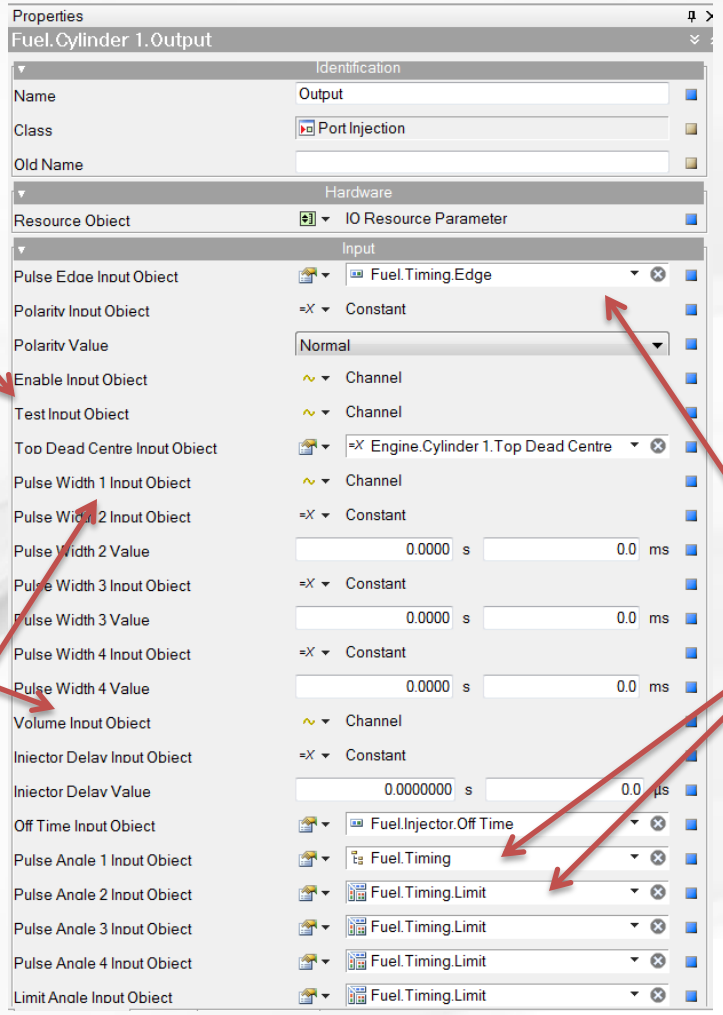


Implementation of the injection event

The already introduced Port Injection classes provide the injector output control.

Fuel State and Fuel Test determine if the injection output is disabled or enabled, this is calculated in the Fuel Update Enables function.

Volume to deliver and the pulse width are calculated in the Fuel Volume Output Calculation function based on the injector specifications provided by the Fuel Injector class.



The injection timing values as defined in the Fuel Timing group

The Main table allows to calibrate the injection moment in dependency of engine speed and engine efficiency.
The Limit table defines the latest stop time of the injection.
The Edge parameter allows to set the end or the start of the injection as reference.

Helpful extensions to facilitate handling of the project

The project now provides the possibility to run the example engine.

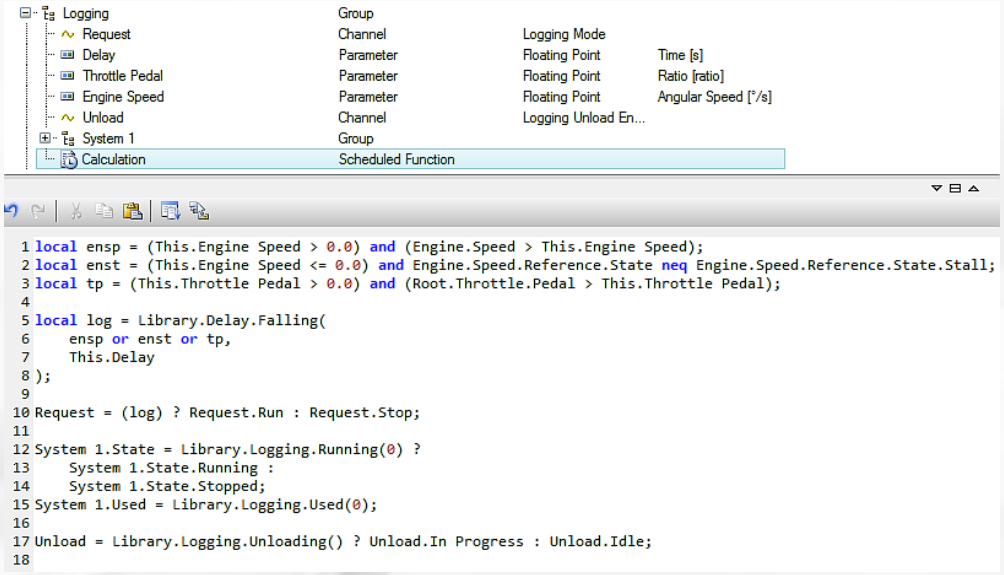
Furthermore, the example project contains some features that are often found very useful.

The following section therefore will discuss

- Logging setup
- Tuning methods
- Fuel consumption information
- Help
- Tags

Logging setup

The M1 provides 8 different logging systems. They can be defined in M1 Tune, however, they can be only activated in dependency of channels using the built-in data type ‘Logging Mode’. These have to be defined in the project.



In the example project, a Logging Request channel is introduced with the data type Logging Mode. In the Calculation function, the Logging Request channel is set to Run if the engine is not stalled and a calibrated throttle position is exceeded.

The State and the using conditions of the first logging system are gathered for information purpose.

Tuning methods

To facilitate calibrating a parameter or table in M1 Tune, tuning methods can be used.
The method will be executed every time the associated parameter or table is selected in M1 Tune and the ‘Q’ key is pressed. Dedicated methods and functions that can only be applied in calibration methods allow the interaction between the method and M1 Tune.

The example project contains a Quick Lambda tuning method for the Engine Efficiency table.

The Quick Lambda method corrects the calibrated value at the respective position of the efficiency table by evaluating the difference between the aim lambda and the currently measured lambda.

```
26      local new = This * Exhaust.Lambda.Normalised / Fuel.Mixture Aim;
27      Fuel.Closed Loop.Reset = Fuel.Closed Loop.Reset + 1;
28
29      if (This.Validate(new))
30      {
31          // new value is valid so set and mark site as adjusted
32          This = new;
33          this.Mark(true);
34      }
```

By using this method, calibration of the table is much easier and faster compared to calculating and typing all values manually.

Fuel consumption information

The Port Injection class provides channels that gather the fuel volume of the current injection, as well as the currently used pulse width.

With this information, the Fuel Calculation function determines
- the injector duty cycle,
- the current fuel volume flow
- the used fuel volume since engine start

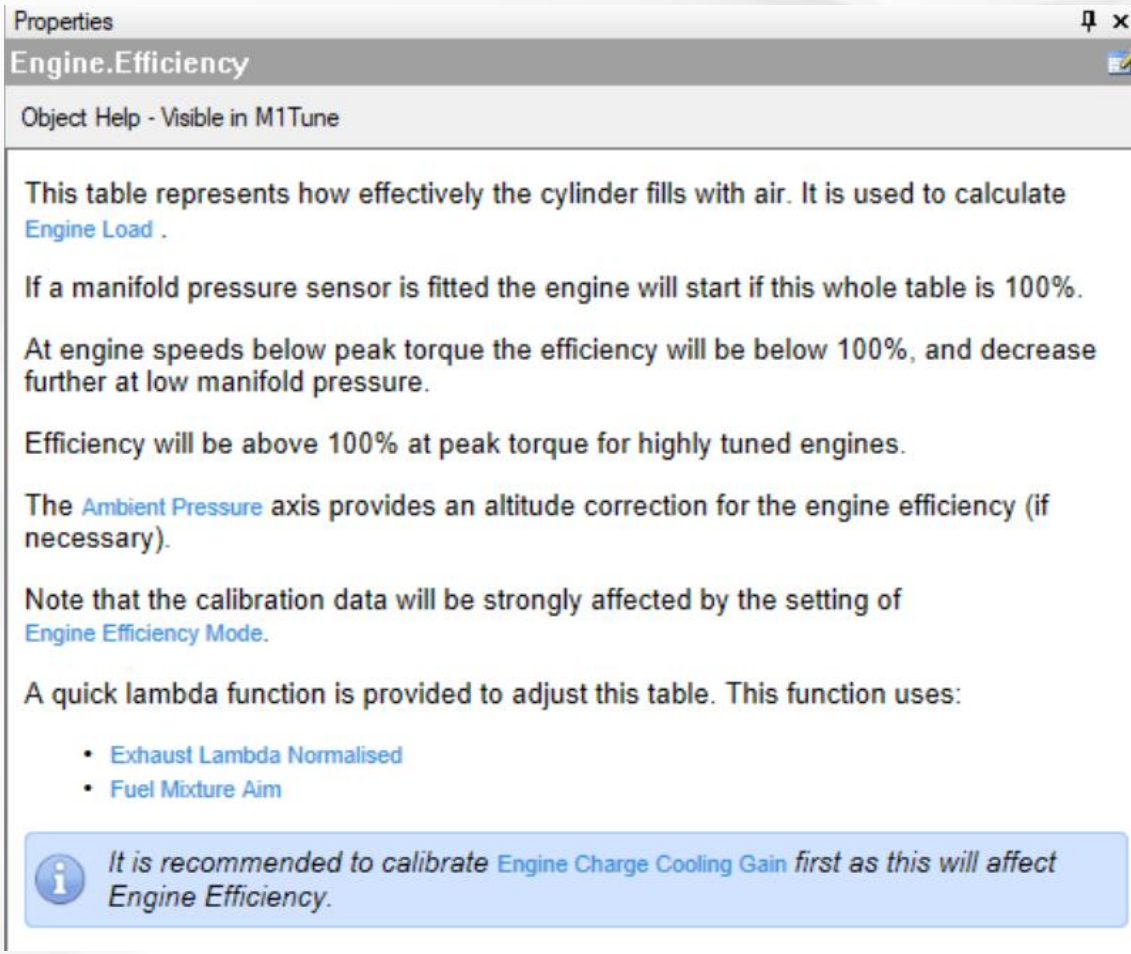
```
local cps = Engine.Speed / Engine.Speed.Reference.Cycle Duration; /* Engine Speed in cycles per second */
local pwPrim = 0.0;
local deliveredPri = 0.0;
local volPrim = 0.0;
local primCorr = 1.0;

expand (n = 1 to Engine.Maximum Cylinders)
{
    if (Cylinder $(n).Output.Enable eq Cylinder $(n).Output.Enable.Enabled)
    {
        volPrim += Cylinder $(n).Output.Volume;
        pwPrim = Calculate.Max(pwPrim, Cylinder $(n).Output.Pulse Width 1);
    }

    deliveredPri += (Calculate.IsNAN(Cylinder $(n).Output.Delivered) ? 0 : Cylinder $(n).Output.Delivered * primCorr);
}

Volume.Per Cycle = volPrim * (1.0 - Output.Cut Average);

Flow = Volume.Per Cycle * cps;
Used = deliveredPri;

if (Injector.Location neq Injector.Location.Not in Use)
{
    Injector.Duty Cycle = pwPrim * cps; /* maximum duty cycle of all cylinders */
}
else
{
    Injector.Duty Cycle = Calculate.NAN();
}
```

Help

The value of a reasonable package help for the user in M1 Tune should not be underestimated.
Therefore the help text on each object should contain content such as:

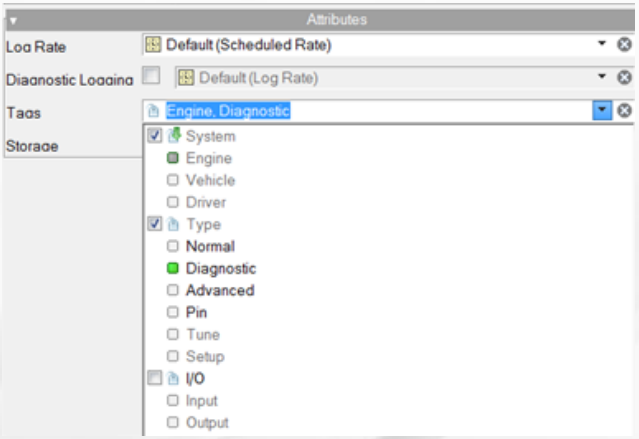
- explanation of the object's function
- list where the object is used
- influences on the object
- calibration procedure
- ...

The integrated help editor allows to format contents for better readability.



Tags

Tags can be used to facilitate the creation of worksheets and the user interaction in M1 Tune as well as the logging setup. M1 Build supports the assignment of tags by reporting warnings during validation if the tags do not match the convention.



Name	Class	Tags	Data Type
Lambda	Group	Engine	Floating Point
Value	Channel	Engine, Normal	Floating Point
Filtered	Channel	Engine, Advanced	Floating Point
Normalised	Channel	Engine, Advanced	Floating Point
Diagnostic	Channel	Engine, Diagnostic	Exhaust Measurement Status Enumeration

Each object has to be categorised to either the Engine, Vehicle or Driver tag.
Dependent on the nature of the object the Type tag needs to be defined. A diagnostic channel for example should be tagged 'Diagnostic'. M1 Build may apply additional checks then, for example a channel tagged 'Diagnostic' is expected to be of an enumerated data type and to have at least one enumerator which has an indicator defined.

Exhaust.Lambda.Diagnostic

Channel

'Exhaust Measurement Status Enumeration' Values		
Unavailable	-1 (Information)	
Not in Use	0	
Collector	1	
Cylinder Average	2	

The end