An introduction to the Module Management Console (MMC)

For MMC version 0.10.x

As MMC evolves with new versions, features will change, and this document is not intended as a detailed manual to how everything works (as it would quickly be out of date), but a general introduction to the basic concepts behind MMC .

Equally, any screenshots presented here may not exactly reflect any specific version of MMC, but will still show the general concept.

Highlights

- Cross-platform – works on windows, linux & MAC-OS
- Friendly user interface to allow management of CBUS and VLCB modules.
- Uses Module Descriptor Files to create the displays for configuring modules
- Support for programming modules using PIC devices including the new PIC18F27Q83 devices
- Faster programming with newer bootloaders
- Allows naming of module ‘channels’
- Supports spreadsheet import & export of node, event & channel names
- Also includes import of node & event names from FCU
- Allows multiple backups of nodes – and selection of a backup to restore
- Support for extended features that VLCB adds to CBUS

Overview

MMC started out primarily as a multi-platform application, and very quickly the problems associated with supporting new designs of modules (as well as all the legacy modules) became apparent. The ability for anyone to load files that enabled support of new designs was obviously attractive, and the idea of ‘Module Descriptor Files’ was born. As these can be used by any application, not just MMC, the definition & working examples reside in their own project.

MMC is intended to be an alternative to other tools like FCU, not a replacement, and is an opportunity to explore different ways of achieving the same goals. This means that some things work differently, and no apology is made for this, this is a deliberate design choice.

The project resides in github, and is publicly accessible.

There is a MERG® Knowledgebase entry here with more links, and a collection of short videos on selected aspects of MMC here.

Installation

At the time of writing, there is a windows installer that runs and creates a desktop shortcut, this is available here with more information.

There is a linux install script being worked on.

There is a short video that explains how to do a manual clone from github onto windows – but now the installer script is available, the script is recommended.

Startup

On starting MMC, a startup dialog appears which allows the user to select or create a layout which to connect to.

With a fresh installation, just the ‘default layout’ will be present, which can be used, although a new layout with a suitable name is recommended.

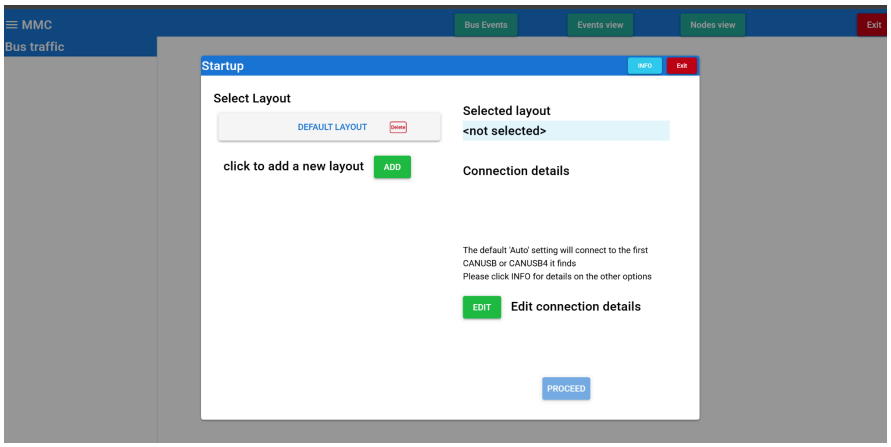
Selecting a layout from the left-hand list will populate the connection details & enable the ‘PROCEED’ button.

Each ‘named’ layout holds all the information about that specific layout, including the connection details used, nodes, events, channels etc..

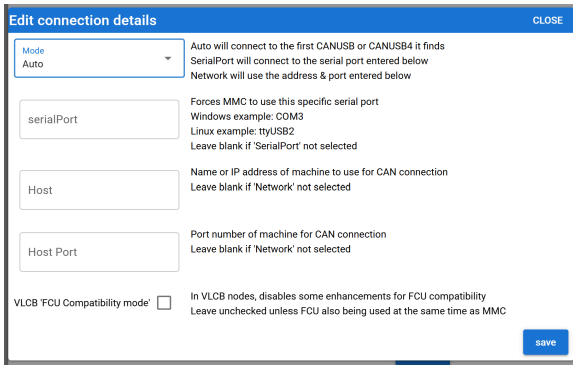
So to swap from one layout to another, you just need to exit & restart MMC, and you can then pick a different layout.

By default, the ‘auto’ connection setting will find the first CANUSB or CANUSB4 and then will use that for communications.

The INFO button at the top of the startup dialog gives more information:

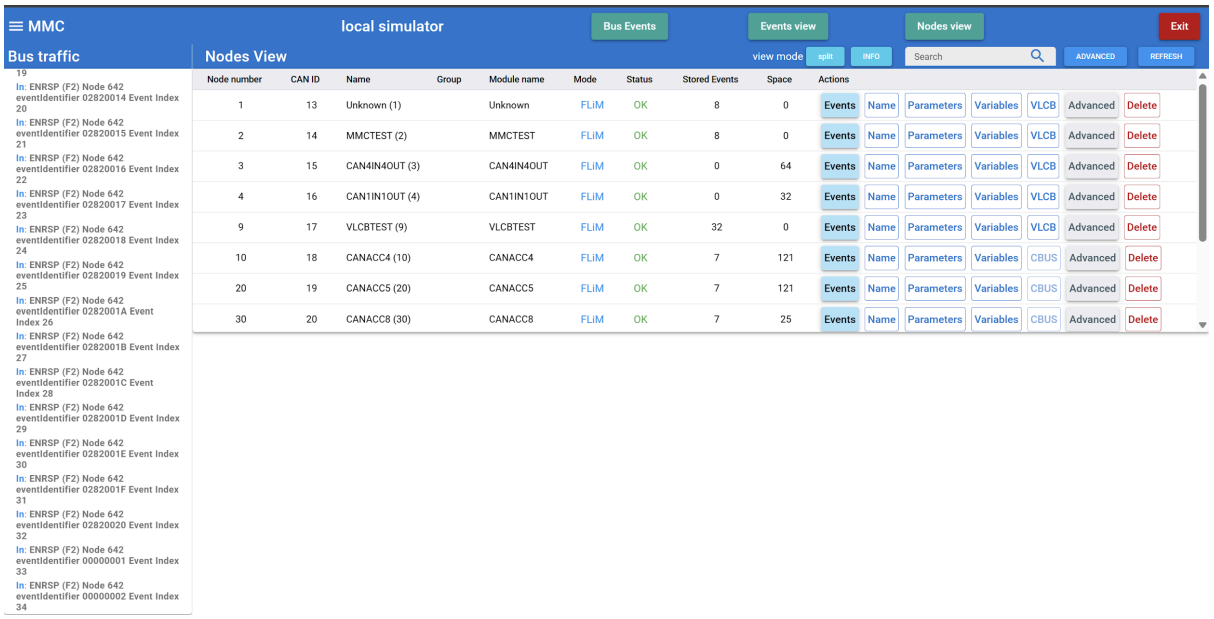


The ‘EDIT’ button will bring up the ‘edit connection details’ dialog, allowing advanced configuration of the connection for the selected layout.



Main Screen

Once PROCEED is selected on the startup dialog, the main screen will be displayed:



There are three main views available, ‘Bus Events’, ‘Events view’ and ‘Nodes view’, selectable by the three buttons across the top of the window, and at startup the ‘Nodes view’ is shown by default.

These offer different views onto the same set of data held byMMC and the differences in these three views is described in the following sections.

Down the left-hand side is the ‘Bus traffic’ column which shows the messages seen on the CANBUS. Its main purpose is to show that activity is in progress, as some operations can take a significant number of messages, so it’s useful to see that activity is still taking place.

At the very top left-hand side is a three-bar icon which opens a side menu with system wide operations such as export & import.

Events

At this point it may be beneficial to those not familiar with CBUS to describe ‘events’.

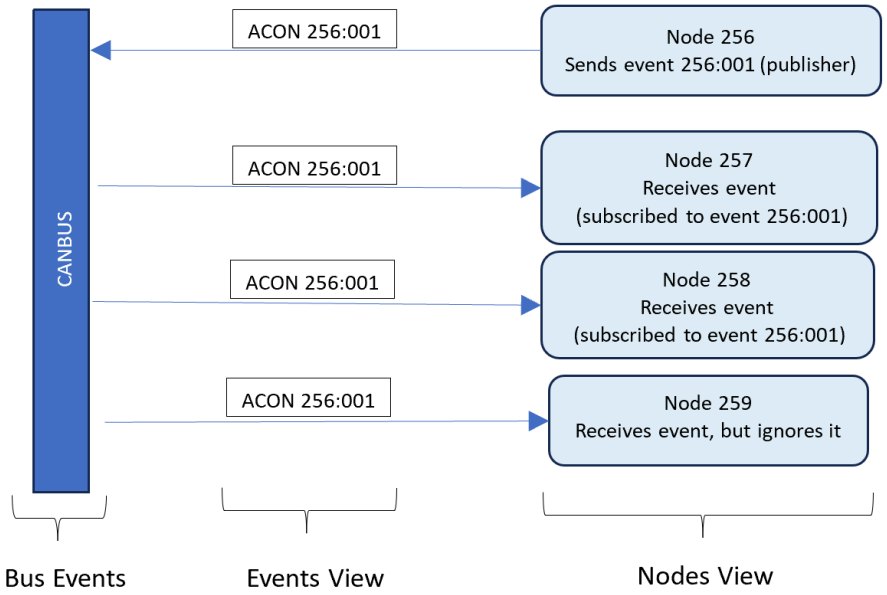
The following descriptions are from the CBUS™ developers guide:

“a ‘producer’ creates an ‘event’ that is sent onto the bus. Strictly, this is just a unique number. Any ‘consumer’ needing to act on this event is ‘taught’ it so it recognises that event in future.
An event is sent in response to a change in state of a layout device and will result in an action on the layout of other devices.”

An event is contained in a message sent across the CANBUS from one node which all other nodes will receive, and these other nodes may then take action on receipt of this event, depending on their programming, or just ignore it.

E.g. a switch connected to one node sends an event when the switch is moved to ‘ON’, and another node sets a relay ‘ON’ when it receives this event.

An example is an ‘accessory on’ event (ACON), which has a unique identity made up of the sending node number and an event number. In this example, event number 1 is sent from node 256, hence this event uses an identity of 256:001.



Every node that uses a particular event will have that event identity programmed (as shown in the three nodes in the diagram), whether it’s as a publisher, or subscriber (or in some cases, both). In certain scenarios, there can be multiple publishers as well as multiple subscribers.

So, an event is an entity in its own right, and can be programmed into as many nodes as desired, so it’s not ‘owned’ by any specific node.

Note that the programming (‘teaching’) of an event into a node will be different depending on what type of module the node is, and the configuration choices it offers.

Nodes View

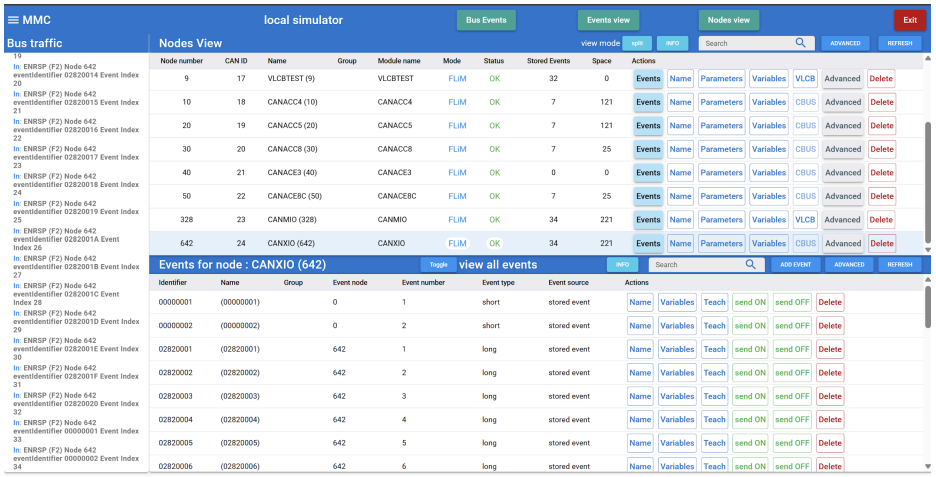
So, how do I program these events into nodes?

The ‘Nodes View’ allows the viewing & editing of events (and other things) for each node, and is probably the view that is used most.

At startup, just the top half will show a list of nodes, with the lower half empty, but once the ‘Events’ button is pressed for a specific node, the lower half will show the events for that node (as shown in the screenshot below).

Information is read directly from each node where possible, so reflects the up-to-date situation of each node, it doesn’t rely on ‘stored’ data which may get ‘out of sync’ with the node (an issue with modules that can dynamically alter their variables). This is automatic, it occurs when necessary for any user selection.

Information that isn’t stored in the node, like node, event & channel names, is stored for this specific layout, so will be reloaded when this layout is selected again at startup.



If it’s preferable to view more nodes on the screen, then by changing the view mode in the ‘Nodes View’ header bar, then the node will occupy the whole screen, and events will appear in a pop-up dialog.

There is an ‘Info’ button on the ‘Nodes View’ header bar, which will show information similar to that presented here.

There is the option to add a name (and/or group) and this can also be used with the search option.

Any name or group added to a node, event or channel is stored on the computer, for this individual layout.

The ‘Search’ option will search the whole of each row for any instance of the text entered – irrespective of case, so entering ‘point’ will find ‘Point1’ as well as ‘point2’.

The ‘Advanced’ button on the node row provides a list of other options for that node, and this list is specific to each node, but may include options like Node backup, Node restore and Node programming.

The ‘Variables’ button on the row for a node will display a dialog allowing the editing of the Node Variables – the exact display layout will differ depending on the MDF being used for that node – see the ‘Module Descriptor File’ section for more information.

As with nodes, it’s possible to add a name and/or group to each event.

From the events list in the lower half, it’s possible to ‘Teach’ an existing event from this node to another node, or ‘Add’ a new event to this node.

There is an ‘Info’ button on the ‘Events for Node’ header bar, which will show information similar to that presented here.

The ‘Variables’ button on the row for an event will display a dialog allowing the editing of the Event Variables for that event – the exact display layout will differ depending on the MDF being used for that node – see the ‘Module Descriptor File’ section for more information.

There are ‘send ON’ and ‘send OFF’ options which allows you to send the event on the CANBUS from MMC, duplicating how the node would send it.

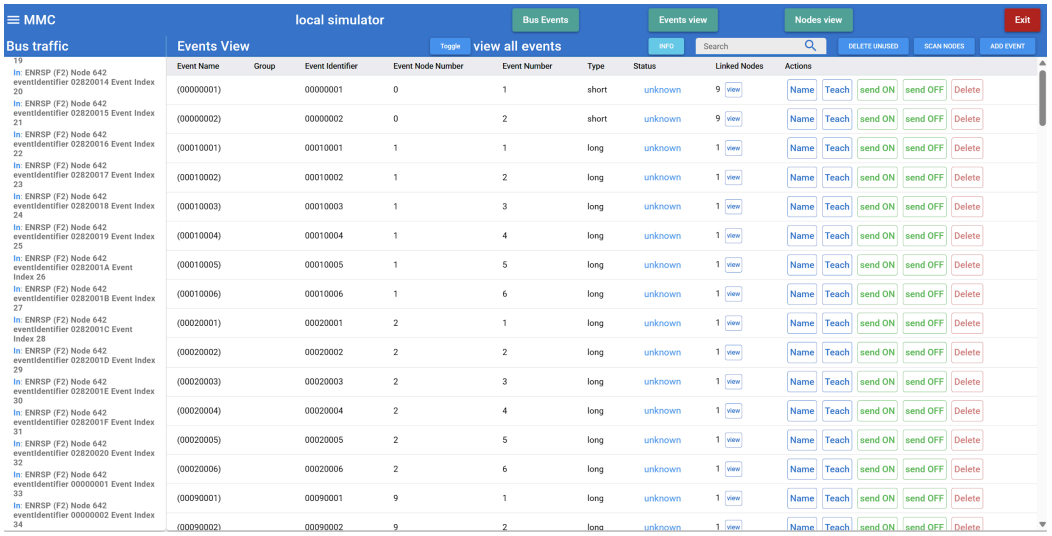
Many modules have ‘default’ events programmed in from the ‘factory’, mainly produced events. Later modules report these events when reading stored events from the node, so will show up by default. Many earlier modules don’t report these when stored events are read from the node, so will only appear once the node has sent them on the CAN bus – this is the difference between events shown as ‘stored event’ or ‘bus event’.

Some modules have a large number of ‘default’ events already programmed, which can make the list appear cluttered. So, if this is an issue, it’s recommended that any events in use are given names, then the ‘toggle’ button can be used to change to ‘view named events’, which will only show the ones that have been given names.

Events View

How do I see which nodes have a specific event, without having to go into each node and checking?

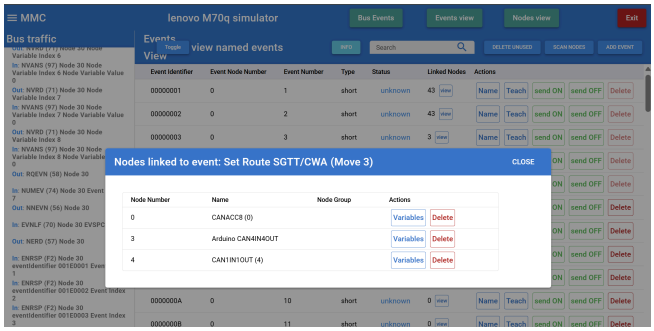
This is what the ‘Events View’ provides, and gives a view of all the events that the system knows about.



There is an ‘Info’ button on the ‘Events view’ header bar, which will show information similar to this presented here.

Each event will appear once, irrespective of how many nodes that event is programmed into.

The ‘linked nodes’ column shows exactly how many nodes are programmed with this event, and the view button alongside this will pop-up a window showing the linked nodes.



Any of these events can be ‘taught’ (or programmed) into any node from here.

It’s also possible to add an event to the system from here, without initially programming it into any node, and that event is then available to be programmed into one or more nodes at any future point.

Some modules have a large number of ‘default’ events already programmed, which can make the list appear ‘cluttered’. So, if this is an issue, it’s recommended that any events in use are given names, then the ‘toggle’ button can be used to change to ‘view named events’, which will only show the ones that have been given names.

Bus Events

How can I tell if an event has actually been sent on the CAN bus, and which node sent it?

The Bus Events view shows the events that have actually been sent on the bus, by which node. Obviously, events that haven't yet been sent won't appear. This list is cleared at startup, and begins capturing events after that.

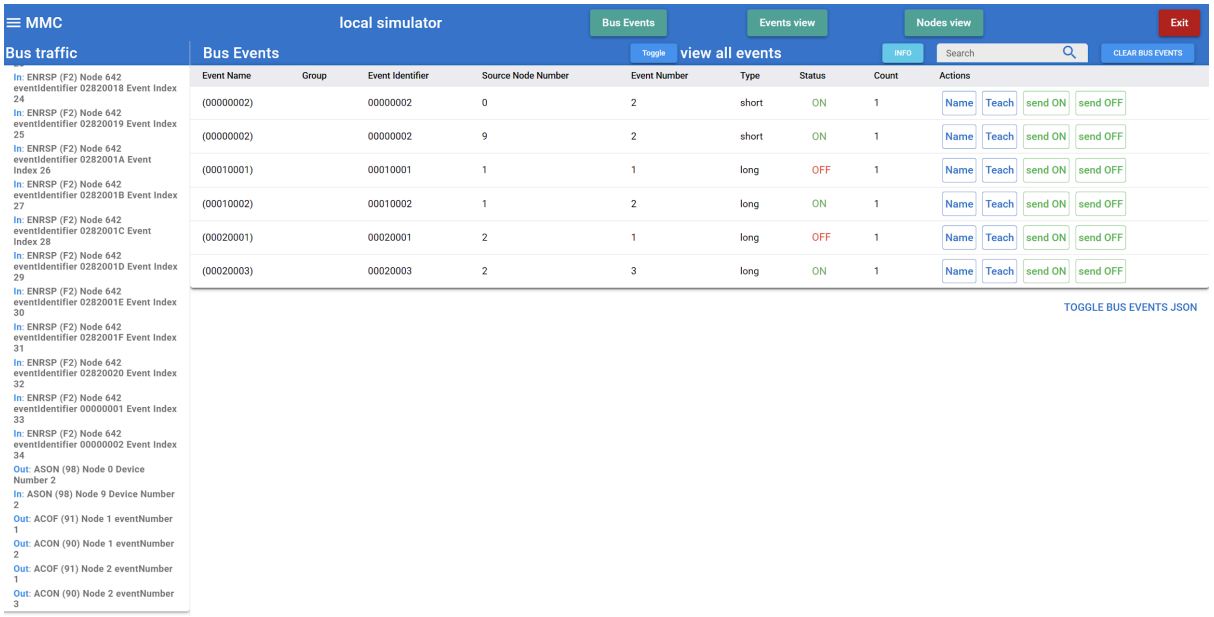
There is an 'Info' button on the 'Bus Events' header bar, which will show information similar to that presented here.

If the same event is sent by more than one node, then there will be multiple entries for that event, one entry for each node, and with a count of how many times that node has sent that event - this is the main difference to the 'Events View', which only ever has a single entry for each event.

This can be quite important for 'short events' which are designed to be able to be sent from more than one producing node (short events replace the node number in the event 'identity' with 0).

Its main use is for diagnostic purposes, and there is the option to clear the current list, so you can just see which nodes send the next events.

For example, you may have more than one switch into different nodes that send the same short event, and this display will show which node sent the short event.



Some modules have a large number of 'default' events already programmed, which can make the list appear 'cluttered'. So, if this is an issue, it's recommended that any events in use are given names, then the 'toggle' button can be used to change to 'view named events', which will only show the ones that have been given names.

Module Descriptor File

Each module type uses the node & event variables (NV's & EV's) in different ways to achieve the aims of that module.

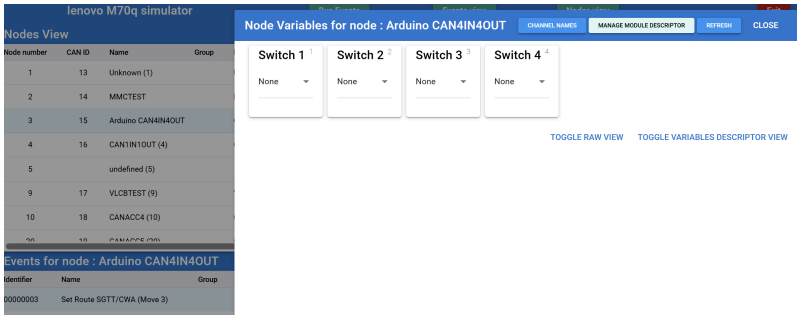
To provide a platform for development of new designs, MMC uses 'module descriptor files' (MDF's) to determine how NV's & EV's should be presented to the user. This means that not only are existing designs catered for, but anyone can upload a new file that determines how a module is configured.

So for anyone developing a new (or modified) module design, they can create and upload an MDF for their new design, and importantly, make their own changes to it as their design evolves, as often as they like, without waiting for new releases of MMC to support it.

The MDF format is intended to be available for any software application using CBUS/VLCB, and so it isn't inherently tied to MMC, it's hoped that other tools will adopt this file format, and so has its own project on github, where full information on the file format can be found, as well as many examples for existing modules.

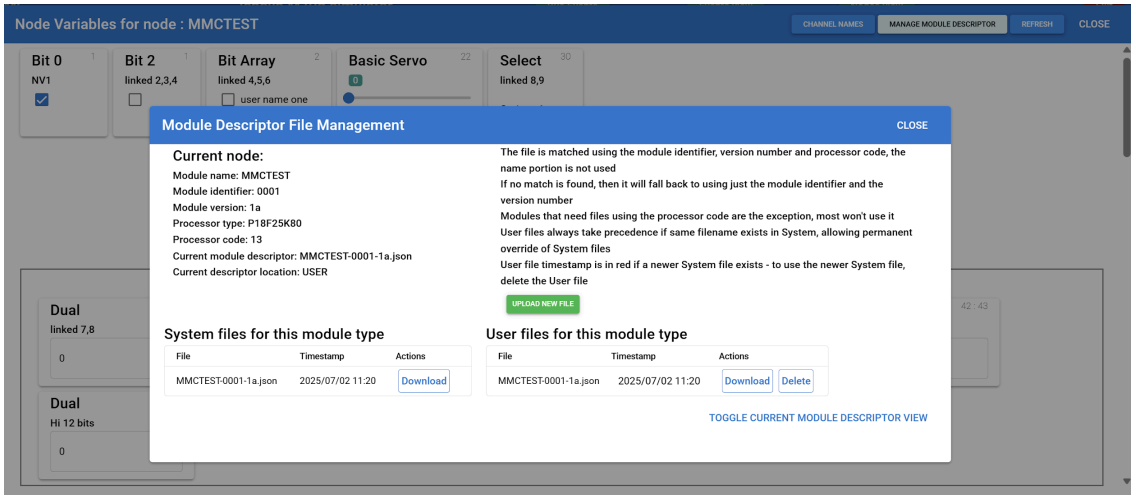
https://github.com/david284/ModuleDescriptor.git

A simple example is the node variables display in MMC for the CAN4IN4OUT Arduino library example:



MMC has preloaded MDF's files for most of the existing CBUS modules, and the 'manage module descriptor' button brings up a management dialog as shown below.

In this example, you can see there is a 'preloaded' system file for this module in MMC, but also a file loaded by the user:



Note that user files always take precedence over system files, thus allowing a user to download a system file, and after amending it and uploading it, MMC will then use the uploaded user file in preference to the system file – the file in use is shown in the detail on the left-hand side.

Whilst preloaded system files may be updated when a new version of MMC is used, the user files are kept in a separate location, and so are never overwritten by a change of version of MMC.