# **CAN BUS QT GUI Tool**

This tool is a proof of concept of the CAN BUS GUI Tool.

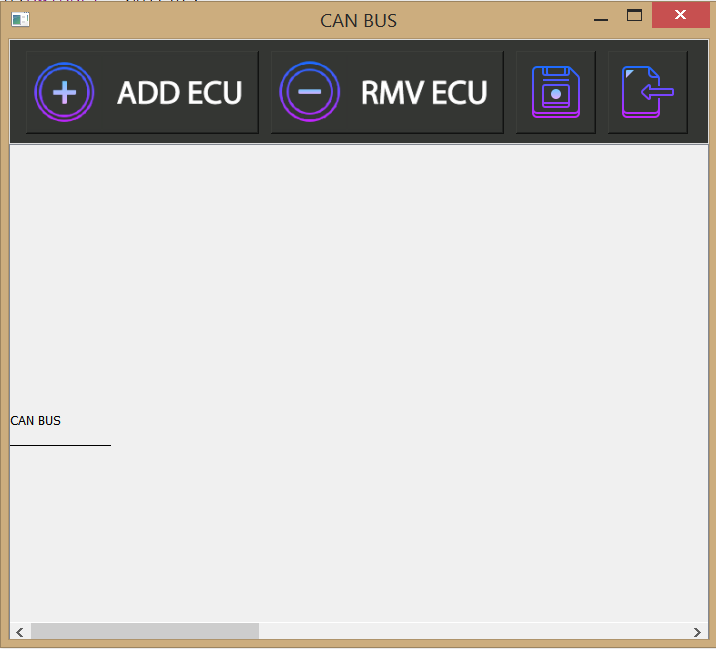
## Main GUI Look:

## Main Technologies used:

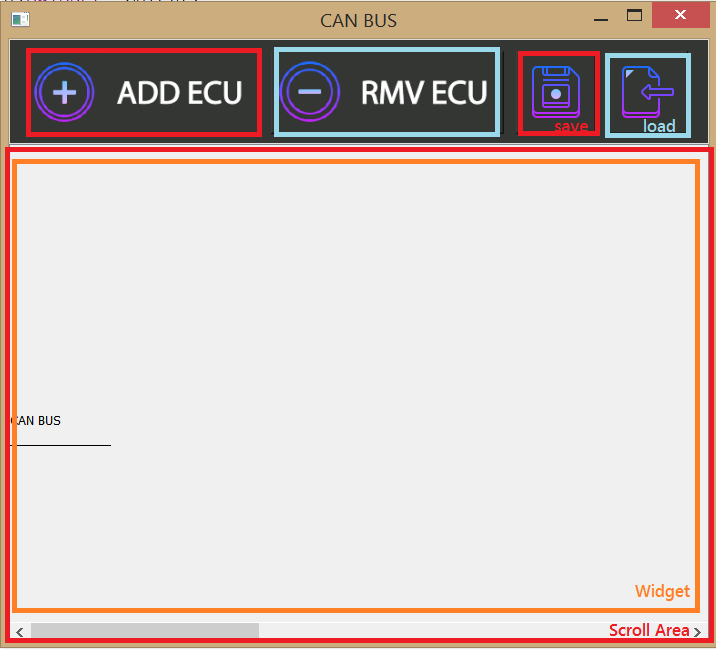
* C++ Qt
* Python BackEnd
* XML

## Main Features:

* Creating New Design
* Loading Existing Design
* Adding new ECU to the Existing Design
* Removing ECU from the Existing Design



## Main Components of the GUI:

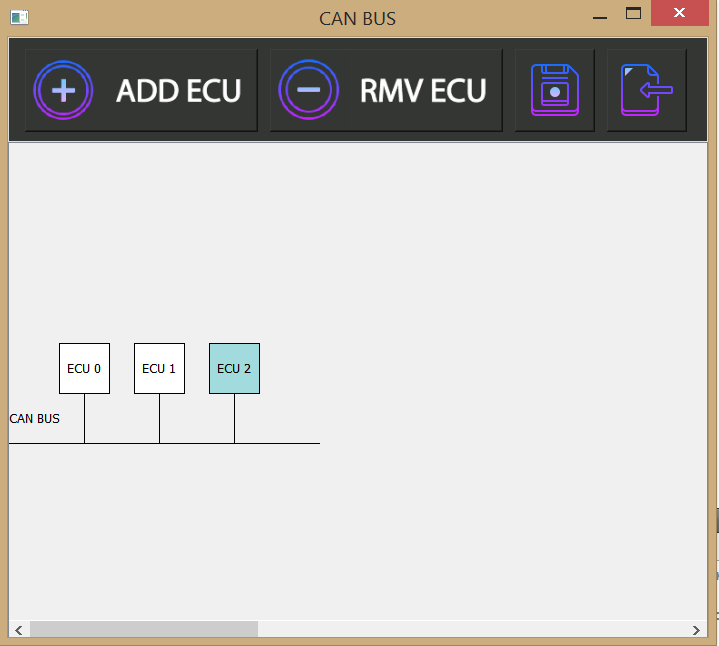


* Buttons GroupBox
  + Add new ECU Button
  + Remove Existing ECU
  + Save Current Design
  + Load Old Design
* Scroll Area
  + Contains QWidget that has an over ridden paint Event

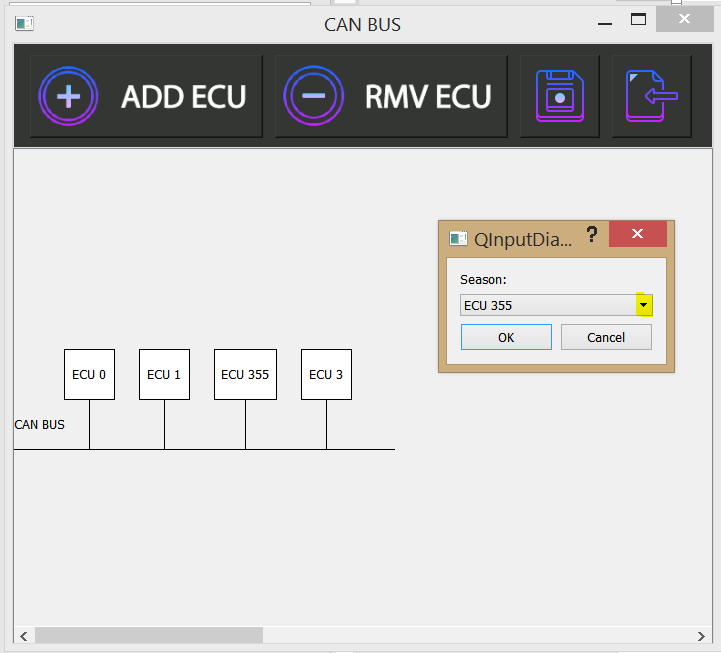
## Adding New ECU:

**Can Bus Grows with the existing ECU Number**

## Removing Selected ECU:



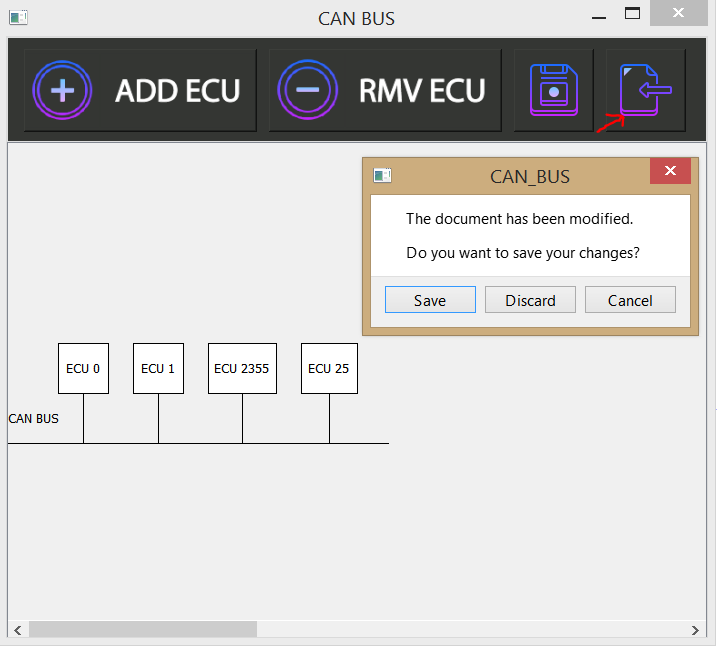
## Removing ECU By Choosing its Name from the existing ECUs Name list:



## Saving Design in the desired Destination:

**CTRL + S Shortcut is supported for saving the Existing Design**

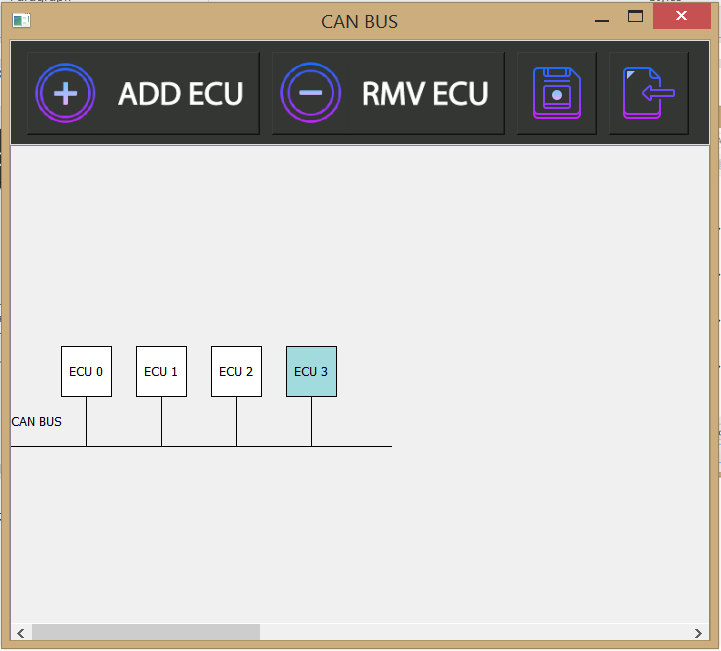
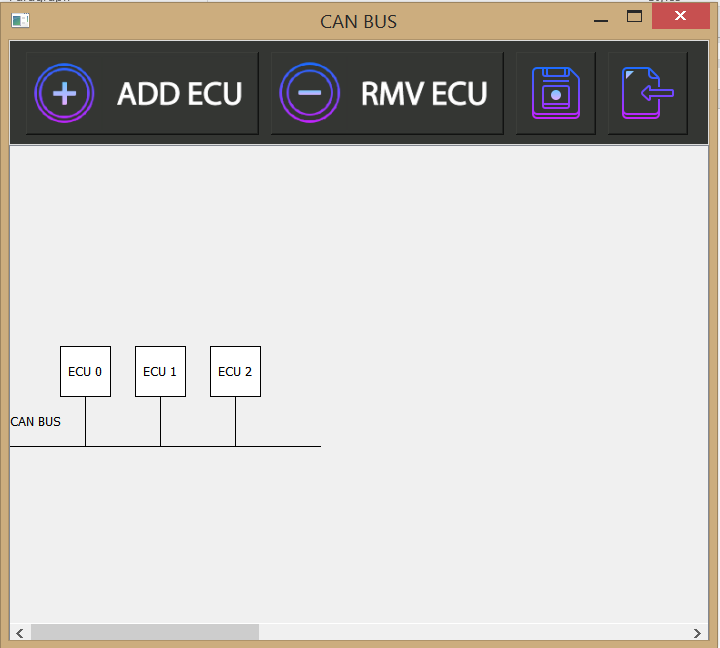
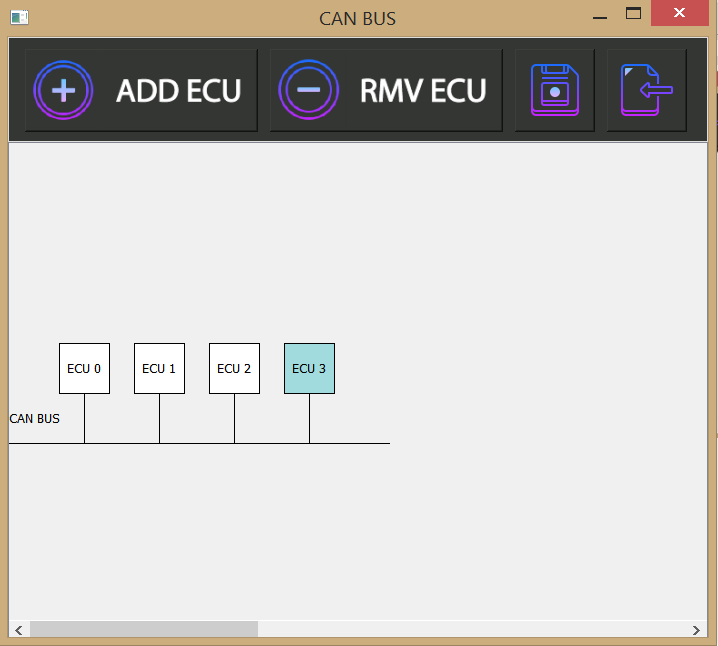
## Loading Old Design checks first if there is an existing design that needs to be saved:



## Loading Existing Design from a file named test.XML:

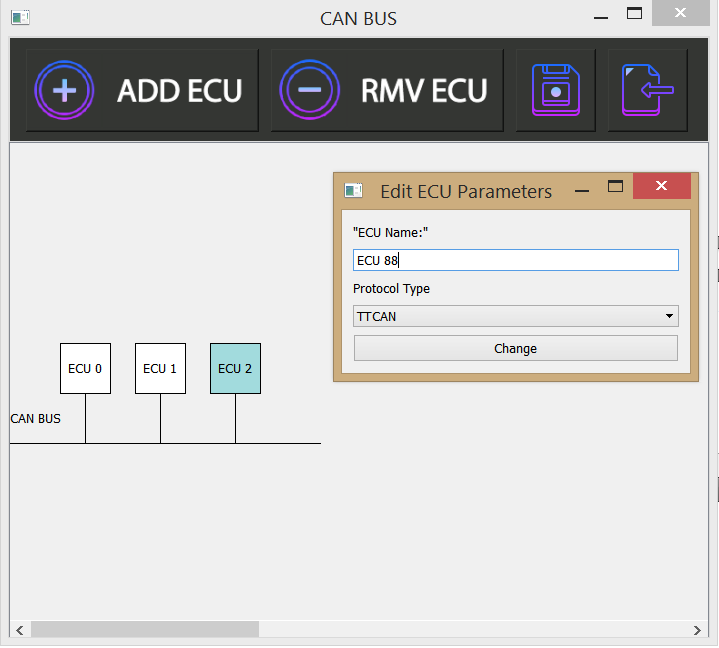
### **CTRL + O Shortcut is supported to load existing Design**

## Undo the last add or remove is also supported via the CTRL+Z shortcut:

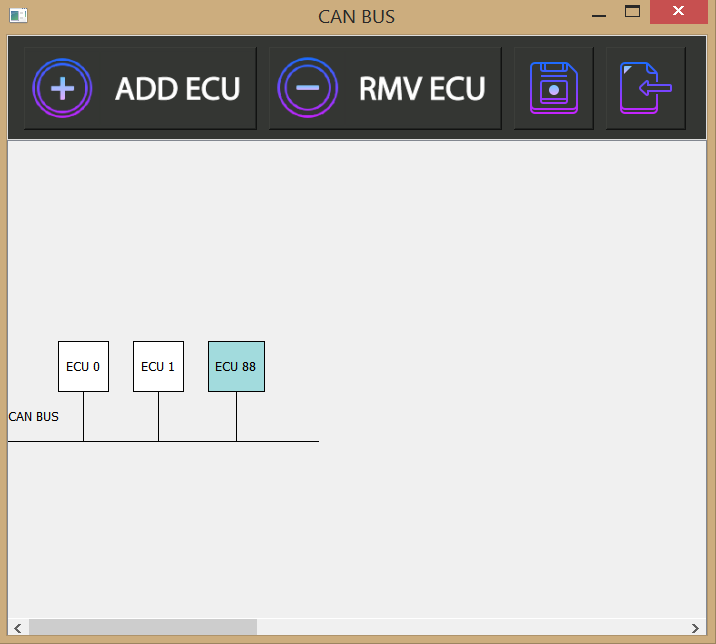


**ECU 3 was deleted using the RMV ECU button then retrieved via the CTRL + Z Shortcut**

## Editing Current ECU by double clicking it:



## After editing ECU2 to be ECU 88 with TTCAN Protocol:



# **Brief info about the implementation**

* The ECUs are drawn inside a class called PaintArea created by subclassing the QWidget class
* Subclassing QWidget was mainly for overriding the PaintEvent to be able to use QPainter Class
* The Main Class used to draw the ECUs is the QPainter Class, The performance of QPainter is pretty high compared to QGraphicsScene and other similar solutions
* QPainter’s main disadvantage is that it requires lots of math to calculate where each element should be
* For Each ECU QPainter draws Rectangle, connector line to the CAN Bus and the Text representing the Name of the ECU
* Then the inter ECU space is calculated and the next ECU in the ECUs Vector is Drawn
* Other Events that were over ridden in the PaintArea class are the mousePressEvent to select the ECU and the mouseDoubleClickEvent to view the edit pop menu
* Selecting the ECU is done by calculating the mouse position and checking among which ECU that position lies then updating the color of the current ECU
* The same goes for the double click event but instead a function that creates the Edit properties pop up window is called
* The next subclass is the PropertiesWindow, it is subclassed from the QMainWindow class
* PropertiesWindow is used to view the pop up window after double clicking the mouse to enable editing the ECU Properties

# Python BackEnd Parsing

**The used parser script is** *“CAN\_BUS\_PARSER.py”*

## Script Sequence:

1. The Script gets the right file path
   * Checks the availability of the given file path and the xml extension

* Using the xml.etree.ElementTree library it parses the XML File and returns the xml tree
* Iterates over the ECU Nodes
* Gathers the Parameters for each ECU in a dictionary
* Creates a text file for each ECU with the name existing in the Name tag, then appends the parameter pairs in the file comma separated lines.

## Running the Script:

The scripts needs to be provided with the file name or path only to generate the ECU Files.

