



Mach4 CNC Controller

Screen Editing Guide

Version 1.0

Copyright © 2014 Newfangled Solutions, Artsoft USA, All Rights Reserved

The following are registered trademarks of Microsoft Corporation: Microsoft, Windows. Any other trademarks used in this manual are the property of the respective trademark holder.

Table of Contents

Chapter 1 Introduction	4
1.1 About this Manual.....	4
1.2 A Note About Lua	4
1.3 Where is My Screen?	4
Chapter 2 The Editor	5
2.1 Open the Editor.....	5
2.2 Save the Current Screen	5
2.3 Load a New Screen.....	5
2.4 Close the Editor	5
Chapter 3 General Layout	6
3.1 Menu Bar.....	6
3.1.1 Screen Menu	6
3.2 Tool Bar	6
3.3 Controls.....	7
3.4 Screen Tree Manager	9
3.5 Properties and Events	10
3.6 Update and Modify Event Scripts	12
3.6.1 Update Event Script	12
3.6.2 Modify Event Script.....	12
Chapter 4 Create a Basic Screen	13
4.1 Create A New Blank Screen.....	13
4.2 Tab Controls	15
4.3 GCode Window	16
4.4 Digital Readouts (DROs).....	17
4.5 Aligning Controls.....	18

4.6 Buttons.....	20
4.7 LEDs.....	23
4.8 Editing Completed Screen.....	24
4.9 Static Text Controls.....	26
4.10 Sliders.....	27
4.11 Manual Jog Buttons.....	28
4.12 Manual Data Input (MDI).....	29
4.13 Completed Screen.....	31
Chapter 5 Screen Scripts.....	33
5.1 Screen Script Overview.....	33
5.1.1 Screen Load Script.....	33
5.1.2 Screen Unload Script.....	33
5.1.3 PLC Script.....	33
5.1.4 Signal Script.....	33
5.1.5 Timer Script.....	33
5.1.6 Control Event Scripts.....	33
5.2 Debugging.....	34
5.2.1 Debug Preparation.....	34
5.2.2 Starting The Debug Session.....	34
5.2.3 Ending The Debug Session.....	35

Chapter 1 Introduction

1.1 About this Manual

This manual is intended to provide beginners with enough information to start building and editing their own screens and provide a quick reference for advanced users. After a concise description of all the controls available in the hobby and industrial versions of Mach4, there is a tutorial for beginner users. This tutorial will teach you how to use the most common controls and help you build a foundation on which you can construct more complex screens.

1.2 A Note About Lua

Although many, if not all, screen controls can be augmented with the use of Lua scripts, however, Lua programming is outside the scope of this manual. If you are unfamiliar with the Lua scripting language, please reference the Mach4 Scripting manual for further information. When applicable, this manual will indicate where Lua scripts can be used with various controls.

1.3 Where is My Screen?

The screen files (extension .set) are stored in the .../Mach4/Screens/ folder. This is the only place Mach4 will look for the defined screen. Place your screen file in this folder to access it from Mach4.

Chapter 2 The Editor

2.1 Open the Editor

Open the screen editor by navigating to the “Operator” Menu and selecting “Edit Screen”. The screen editor will be launched in the current Mach4 window.

2.2 Save the Current Screen

You can save your current edits by selecting the “Save Screen” option in the “Screen” Menu. Another option is to select the “Save Screen As” option to save your screen with a different name. The screen will also ask to save any changes when you close the editor.

2.3 Load a New Screen

A new screen cannot be loaded from the screen editor interface. Before entering the editor, you can load your preferred screen from Mach4’s “View” menu (See the Mach4 Operation Guide for more information on Mach4’s menu bar and controls).

2.4 Close the Editor

To exit the editor, follow the same procedure as opening it. Unselect the “Edit Screen” option, you can do this by clicking on it in the menu.

Chapter 3 General Layout

3.1 Menu Bar

The menu bar spans the top of the Mach4 window. You will notice that in the editor it is slightly different than the standard Mach4 menu bar. Two items have been added that are specific to the editor, “Screen” and “Format”. You can use these to menu items to do things like add controls or images, or align multiple screen items to each other.

3.1.1 Screen Menu

- Save Screen: Click on this item to save the current screen. If no changes have been made, it will be grayed out.
- Save Screen As: Click on this to save the current screen with a different name.
- Manage Images: Use the manage images tool to add or remove images from the screen set. You can use these images as backgrounds, image buttons or static images.
- Add: You can use the add menu item to add controls to the screen, an alternative to using the tool bar.

3.2 Tool Bar

The tool bar is where you select the controls to add to the screen. There are a variety of controls from buttons to DRO’s and panels. Figures 3.2-1 and 3.2-2 show you the controls available in the Hobby and Industrial versions of Mach4. As you can see Industrial includes a few extra controls. Clicking on a control icon adds it to the currently selected container in the screen.

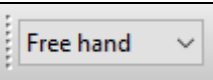














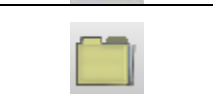












Figure 3-1 Mach4 Hobby Screen Editor Tool Bar



Figure 3-2 Mach4 Industrial Screen Editor Tool Bar

3.3 Controls

	Control Placement Type: Select how you want to place controls on the screen, “Free Hand” or “Snap to Grid”.
	Grid Size: Set the grid size for “Snap to Grid” control placement type.
	Delete: Removes the currently selected control from the screen.
	Add Page: Adds a page to tabs or the screen. Must have a tab control or screen selected to enable this tool.
	Group: Groups are containers for multiple controls, useful for keeping related controls together.
	Button: You can use buttons to trigger events in Mach4 or program them with Lua scripts
	Image Button: Functionally the same as the standard button. Instead of only displaying text, you can use images to create custom buttons. You must first add the images to the screen with the image manager.
	Toggle Button: Toggle buttons trigger events the same as a standard button, but they stay on when clicked and turn off when clicked again. You can use these to turn on and off inputs or outputs, or run custom Lua scripts.
	DRO: Digital Read Outs (DRO) are used to display numerical information from a variety of sources including axis positions and registers.
	LED: You can use LEDs to show the state of inputs, outputs and more.
	G-Code: The currently loaded G-Code file will be shown in this control.
	Toolpath: When you load a program the toolpath is generated and displayed in the toolpath control.
	Static Text: Used to display text, static or from another source, for example Mach status, registers, and other variables.
	Text Box: You can use text boxes to get alphanumerical text input from users.
	Static Image: Static images are pictures on the screen, you can display any image that has been added to the screen with the image manager.
	Tab Control: A tab control can contain one or more tabs, or pages, to display different sets of controls or information to the user. Add tabs to a tab control with the “Add Page” tool button.
	Slider: Sliders provide you with an easy way to adjust a value, for example feedrate or rapid override.
	Gauge: You can use gauges to display the status of variables that have an upper and lower limit, for example spindle override.
	MDI: A Manual Data Input (MDI) control allows you to manually enter GCode to be executed.

	Line: You can use lines to visually separate sections of the screen.
	Tool Offset Table*
	Fixture Offset Table*
	Lua Panel: You can use Lua panels to create more advanced interfaces or self-contained sections of a screen. The contents of a Lua panel must be completely programmed in Lua and cannot be edited in the screen editor.
	Animation Control*: With this you can add an animated GIF (.gif) to the screen.
	Plug-in Panel: Some plug-ins have settings windows or other displays that can be added to the screen with a plug-in panel. Contact the plug-in developer for more information on using their plug-in panels.
	G-Code Editor*: A version of GCEdit built right into the screen.

*Controls only active in Mach4 Industrial

3.4 Screen Tree Manager

On the left side of the screen editor window you will find the “Screen Tree Manager”. All of the screen elements are displayed in a tree according to their hierarchy in the screen. Each page in the screen will be listed by its name, by default there is one page titled “Default”, with all the controls contained within it shown under it in the tree. If you don’t see controls listed under the page name either there are no controls to show or you need to click on the box with the “+” to expand the tree (“-” to collapse it). Containers such as tabs or groups will also have a branch in the tree that shows the controls contained within.

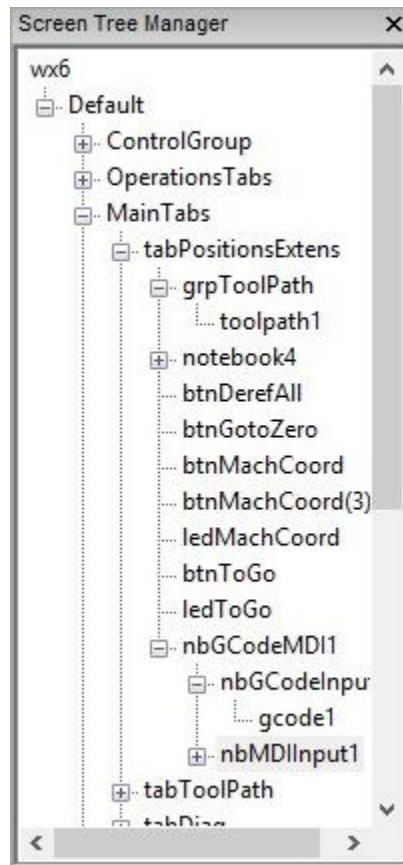


Figure 3-3 Screen Tree Manager

A right click on any control in the screen tree manager displays a menu allowing you to delete, cut, copy or paste the object. For certain containers, there is also the option to import or export. For example, this allows you to easily transfer a tab with all its contents to a new screen.

3.5 Properties and Events

On the left side of the editor, under the screen tree manager, you will find the properties window. This window displays all the editable properties of the currently selected control.

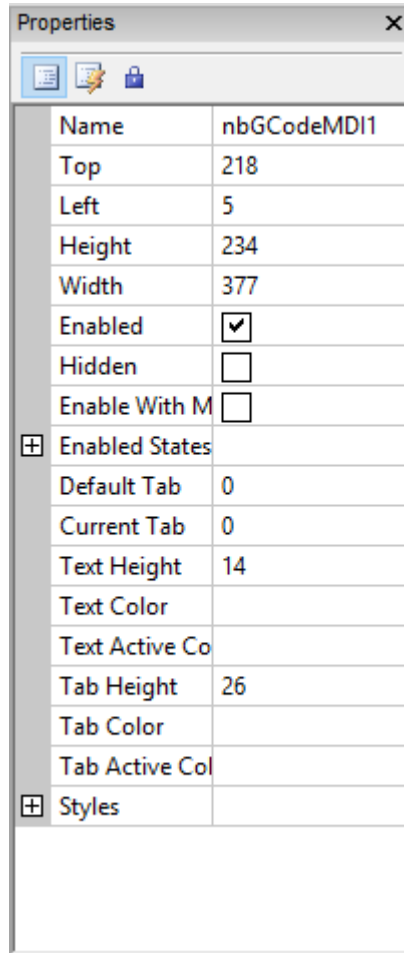


Figure 3-4 Properties

There are two views of properties, selectable at the top of the properties window. The default view, shown in figure 3-4 is the property view. This view shows you basic properties such as name, dimensions, position, enable type, value, label, etc. The other view is the event view, selectable by clicking on the lightning bolt at the top of the window, figure 3-5.

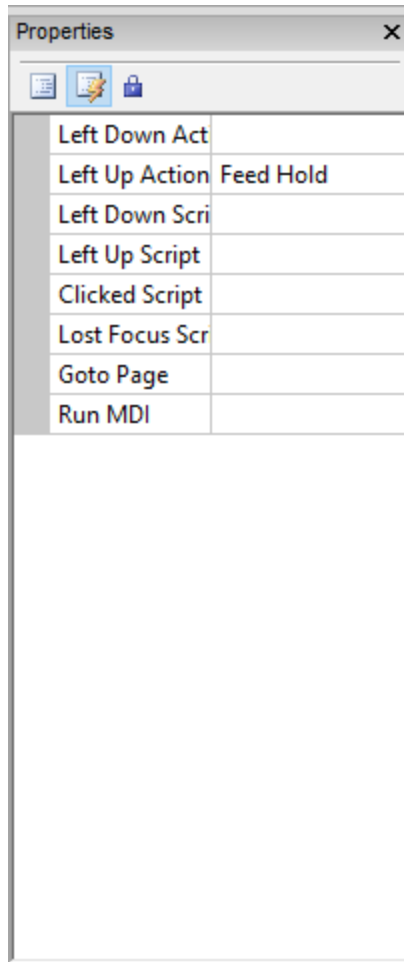


Figure 3-5 Event Properties

The event properties window is where you can define the events, or actions, of buttons and other controls. Figure 3-5 shows the events for a standard button. For the button control, there are multiple events to choose from. The first two options, “Left Down Action” and “Left Up Action”, give you a list of actions to choose from. These are predefined functions in Mach4, opening the drop-down menu will show you a list of all the actions to choose from. A left-down event is triggered when the left mouse button is pushed on the button, when the left mouse button is released the left-up action is triggered. In Figure 3-5 you can see the action “Feed Hold” is assigned to the “Left Up Action”. So, when the button is released, feed hold will be activated.

If you desire a more complex action, the “Left Up Script”, “Left Down Script” and “Clicked Script” options allow you to add a Lua script that will be executed on that event. For more information on Lua scripting please see the Lua Scripting Manual.

Many controls have events that can increase the functionality of your screen. Explore each control and see what can be done.

3.6 Update and Modify Event Scripts

Some controls, such as DROs, static text labels, and text entry boxes have update and (possibly) modify event script(s) associated with them. These allow for the contents of the control to differ from the value that drives them. For example, a DRO may be driven from an encoder register. But the register's count value may not be in a form that is presentable to the operator as they may wish to see a measurement unit.

3.6.1 Update Event Script

The update event script is run whenever the system updates the control's value. If the update script exists, the script must return the (possibly modified) value. The value from the system is passed as the first argument to the script and the control's screen name is passed as the second argument.

```
-- Example encoder update script.  
local inst = mc.mcGetInstance()  
local val = select(1,...) -- Get the system value.  
val = tonumber(val) -- The value may be a number or a string. Convert as needed.  
local name = select(2, ...) -- Get the control name. It is always a string.  
val = val / 1024 -- convert the encoder value to a decimal (assumes 1024 counts per unit).  
  
return val -- the script MUST return a value, otherwise, the control will not be updated.
```

3.6.2 Modify Event Script

The modify event scripts is run whenever the user modifies the control's contents. If the modify script exists, the script must return the (possibly modified) value. The value that the user put in the control is passed as the first argument to the script and the control's screen name is passed as the second argument.

```
-- Example encoder modify/edit script.  
local inst = mc.mcGetInstance()  
local val = select(1,...) -- Get the user supplied value.  
val = tonumber(val) -- The value may be a number or a string. Convert as needed.  
local name = select(2, ...) -- Get the control name. It is always a string.  
val = val * 1024 -- convert the decimal value to encoder counts (assumes 1024 counts per unit).  
  
return val -- the script MUST return a value, otherwise, the encoder register will not be updated.
```

Chapter 4 Create a Basic Screen

Sometimes it makes sense to start with an existing screen, like wx4.set, where all the controls are already placed. Sometimes a screen should be created from scratch, if it is going to be something completely different. For clarity, this manual will be showing you examples on a blank screen. There are a few screens included with Mach4, including a blank one. To start building your own screen copy one of these stock screens and start building.

NOTE! If you modify one of the stock screens and don't save it under a different name, it will be overwritten the next time you update Mach4 and any edits you've made will be lost. Always create your screens with a different file name.

4.1 Create A New Blank Screen

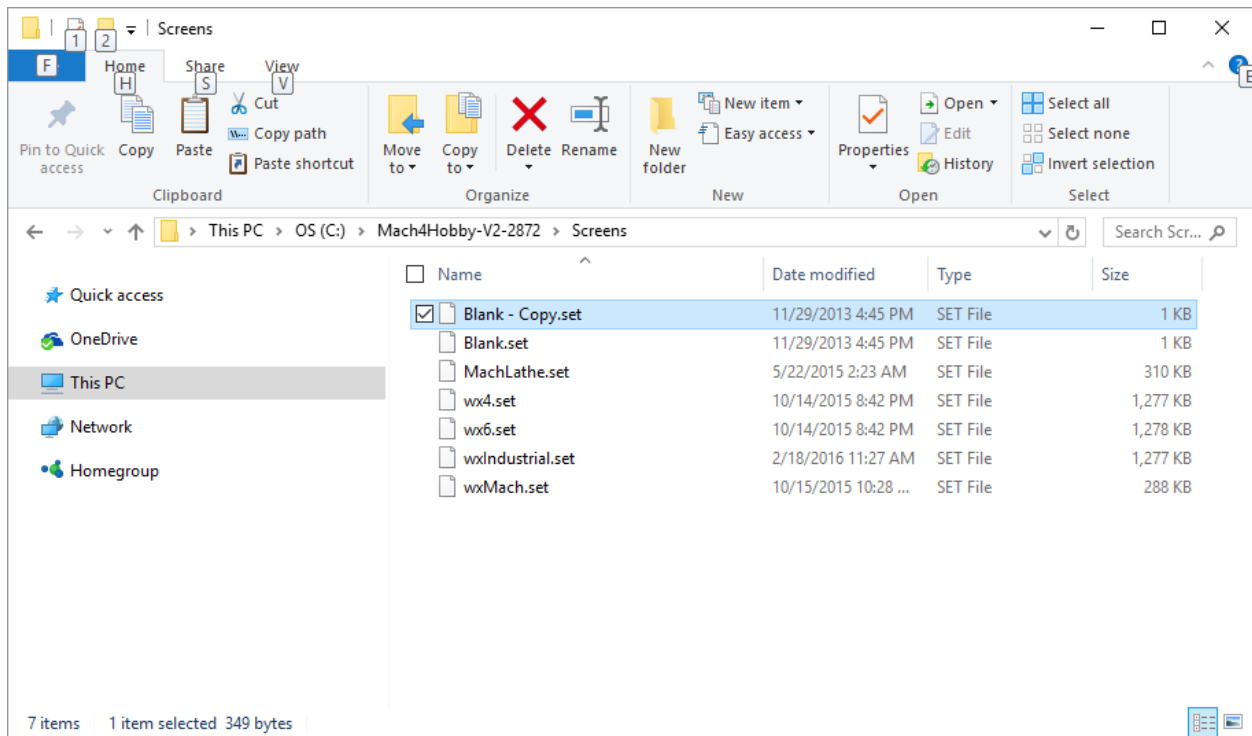


Figure 4-1 Mach4 Screen Folder

To Follow along with this manual, create a copy of the Blank.set screen set, see figure 4-1. Name it something memorable (for this manual I will be using EditorManual.set) and then start up Mach4 with the profile you want to use. When Mach4 finishes loading go to the "View" menu and select "Load Screen". Pick your new blank screen from the folder and open it. The new screen will load and you will be returned to the Mach4 window. The Mach4 screen will be completely blank, don't panic, this is a blank screen set so that is normal.

Now we can start adding controls! Open the screen editor (go to the "Operator" menu and select "Edit Screen"). With the editor open the first thing to do is set up the screen size. Select the screen name

from the screen tree manager and the screen properties will be displayed in the properties window, figure 4-2.

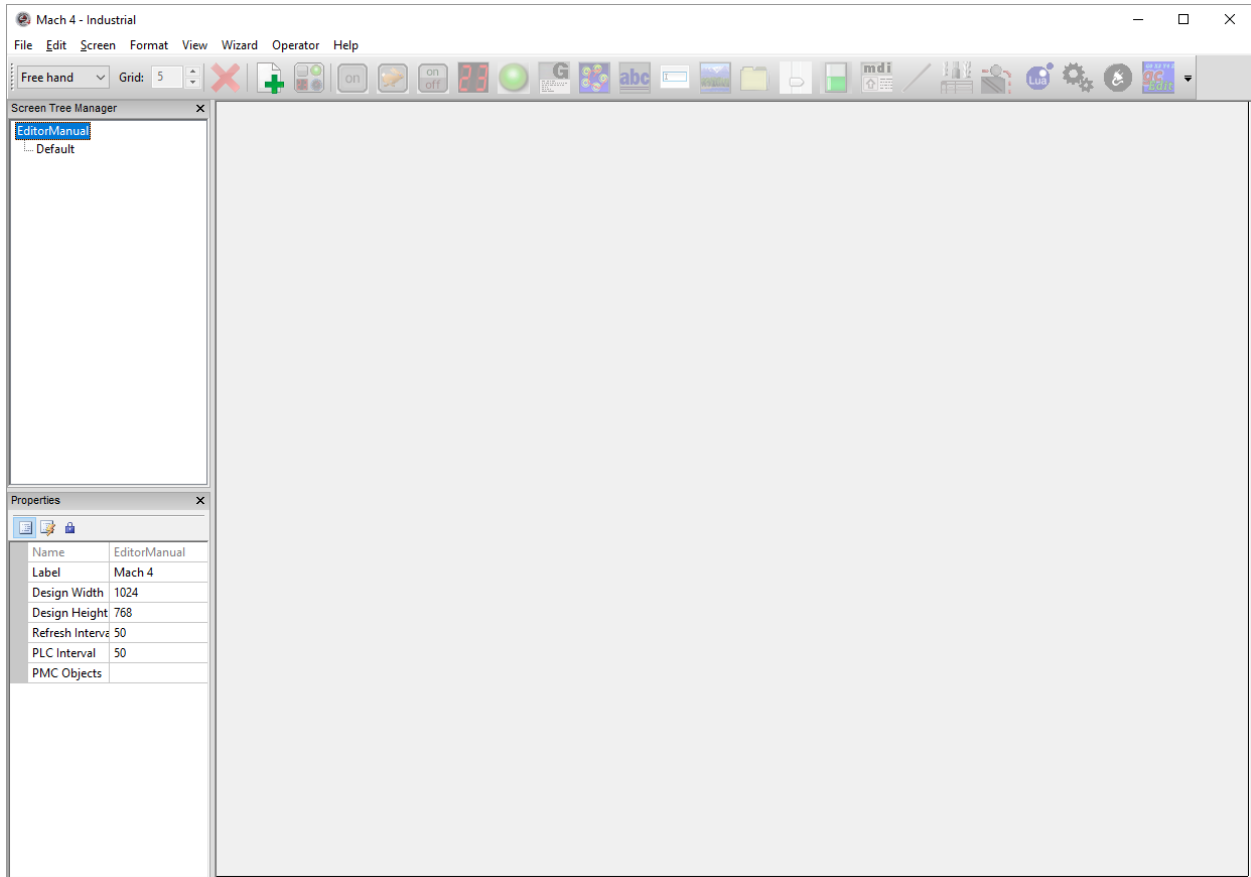


Figure 4-2 Screen Properties

In the "Properties" window you will see two options to set the size, "Design Width" and "Design Height". The blank screen set is set up for a 1024x768 screen. These values are in pixels and should be set to match the resolution of your desired screen. If this height and width does not match the screen then the Mach4 window will be stretched to fit, contents included, so your screen may not display exactly as intended.

Now you can start adding controls to the screen. You'll notice when the screen is selected as in figure 4-2 most of the controls in the tool bar will be unavailable, this is because controls need to be added to containers. There are three container types available, pages, groups and tabs. In the blank screen, there is already a page added for you, named "Default". You can select by clicking on default in the screen tree manager or by clicking on blank space in the main window. When the default page is selected all the controls in the tool bar will activate.

4.2 Tab Controls

You want to have a couple different views in this screen so start by adding a tab control. The new tab control will be added to the top left corner of the page and you'll see the control added to the screen tree manager with a single tab. In figure 4-3 you can see the tab control named "nob" and the single tab named "nbp". These names do have some meaning, "nob" is short for notebook and "nbp" short for notebook page. These are the programming names for this type of control.

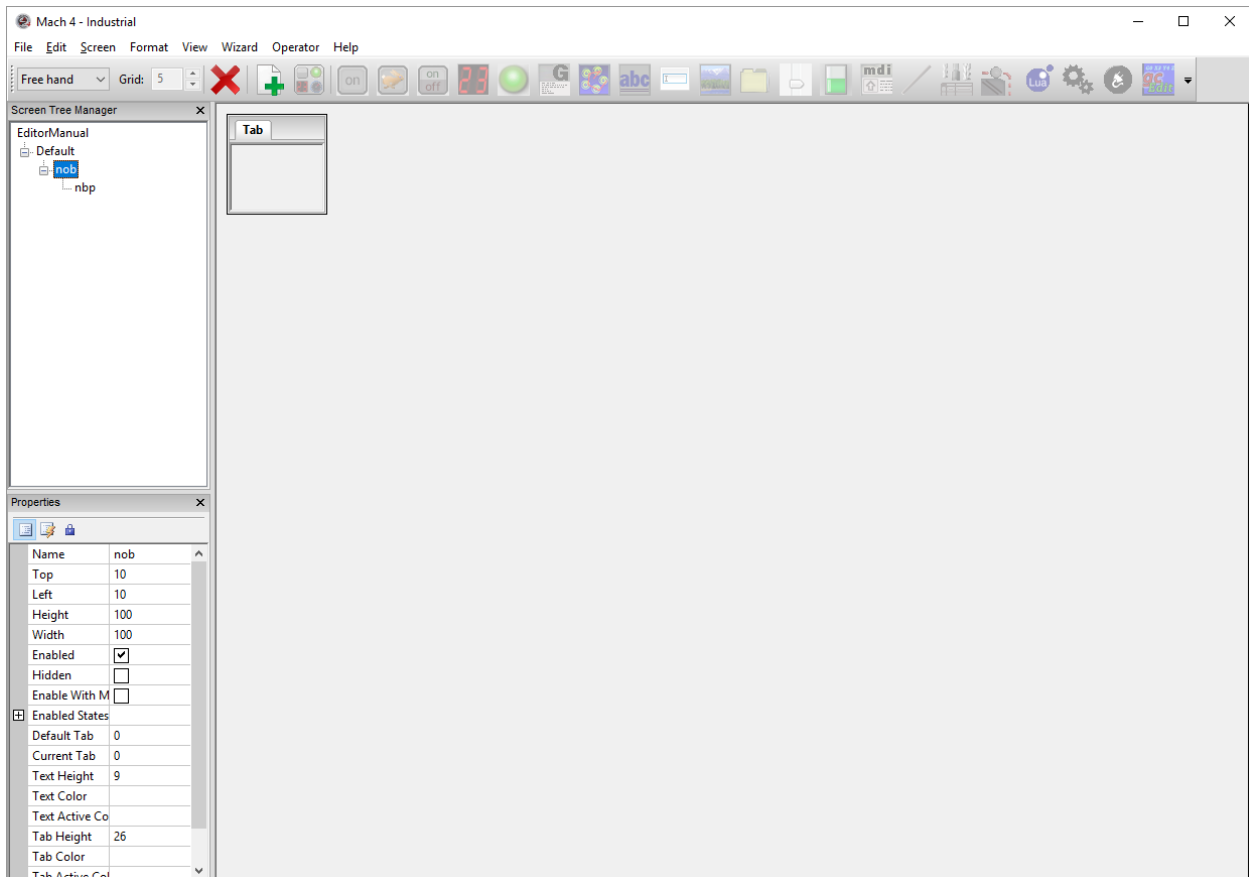


Figure 4-3 Add a Tab Control

Now let's make the tab control fill the top half of the screen, select "nob" on the screen tree manager and set the height to 500 and the width to 1004. Notice the tool bar, there are only two options available, "Delete" and "Add Page". Click the add page button to add another tab to the control. Click on the tabs ("nbp" and "nbp(1)") in the screen tree manager and change the label properties to "GCode" and "Tool Path", see figure 4-4.

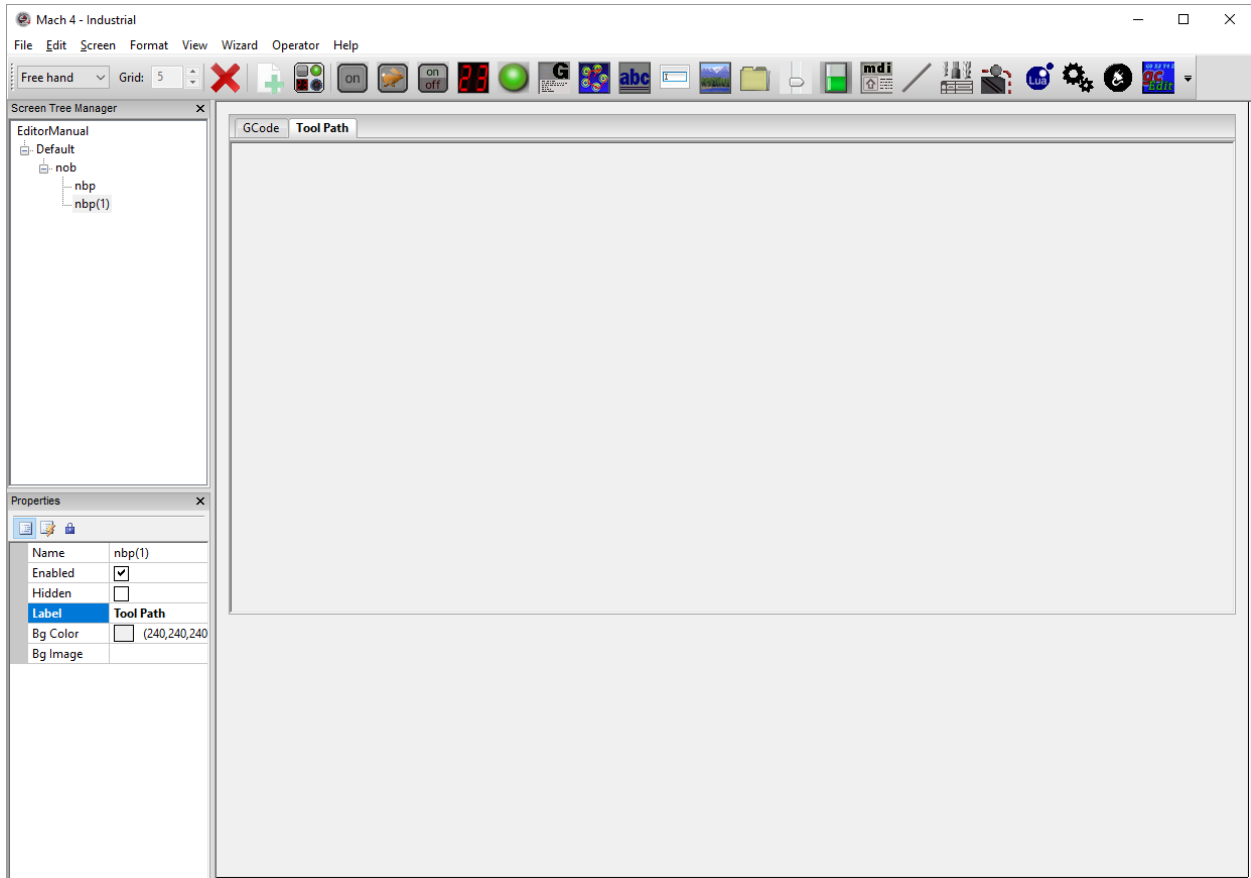


Figure 4-4 Renamed Tabs

The label is the displayed name of the control, this is true for tabs, buttons, static text, etc. The name field is the name that will be displayed in the screen tree manager and the name that can be used to access the control from a Lua script.

4.3 GCode Window

Now let's add a GCode window to the new "GCode" tab, and a tool path display to the "Tool Path" tab. Click on the GCode tab, or select "nbp" from the screen tree manager, then click on the "Add GCode Display" button in the toolbar. A window will appear at the top left corner of the tab. If the new control appears at the top corner of the page, or behind the tab, then you had the page selected and not the desired tab. Now resize the GCode display to fit the tab. Hover your mouse over the button right corner and the pointer should change to a double headed and angled arrow. Click and drag the corner to set the desired size. Repeat these steps with a tool path display on the tool path tab.

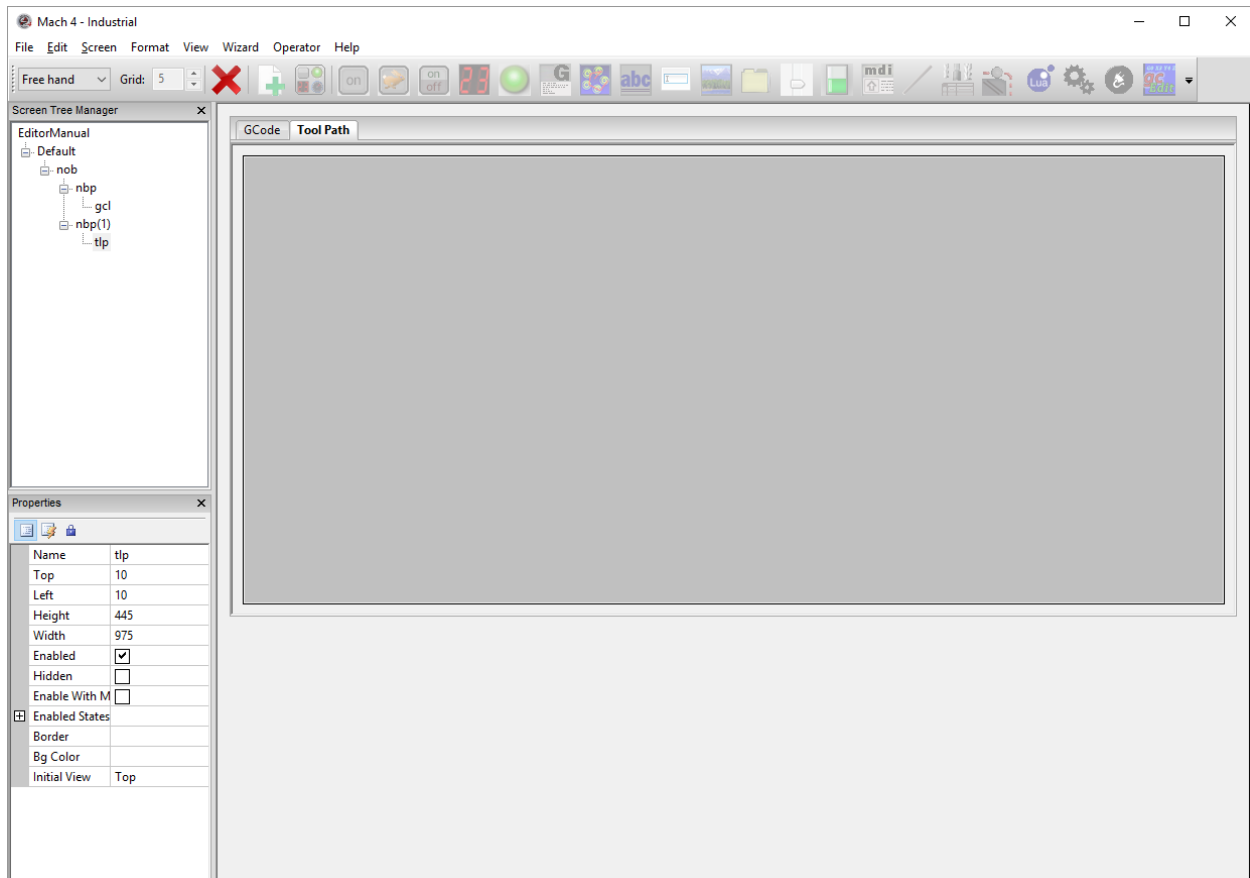


Figure 4-5 Tool Path Display

4.4 Digital Readouts (DROs)

Next let's add some buttons and DROs to the bottom of the screen. Select the page (either in the main window or from the screen tree manager) and add a DRO. It will be added at the top left corner of the page, behind the tab control. The easiest way to move it is to select it in the screen tree manager and type in the desired position or some position (0,0 for example) where you can see the control to click and drag it to position. For this example, we know we want the values we want, Top = 518, Left = 514, Height = 65, Width = 500. At the same time change the border to "Static". Now to add two more. Right click on the DRO and select Copy, the right click again and select Paste. Then right click and select Paste again, there are now 3 DROs all in the same place. You can move them around by clicking and dragging them, by selecting them and then holding control and using the arrow keys on the keyboard, or typing in positions. Just roughly drag the two new ones so they are stacked as in figure 4-6. They do not need to be perfectly aligned, you will fix that in the next step.

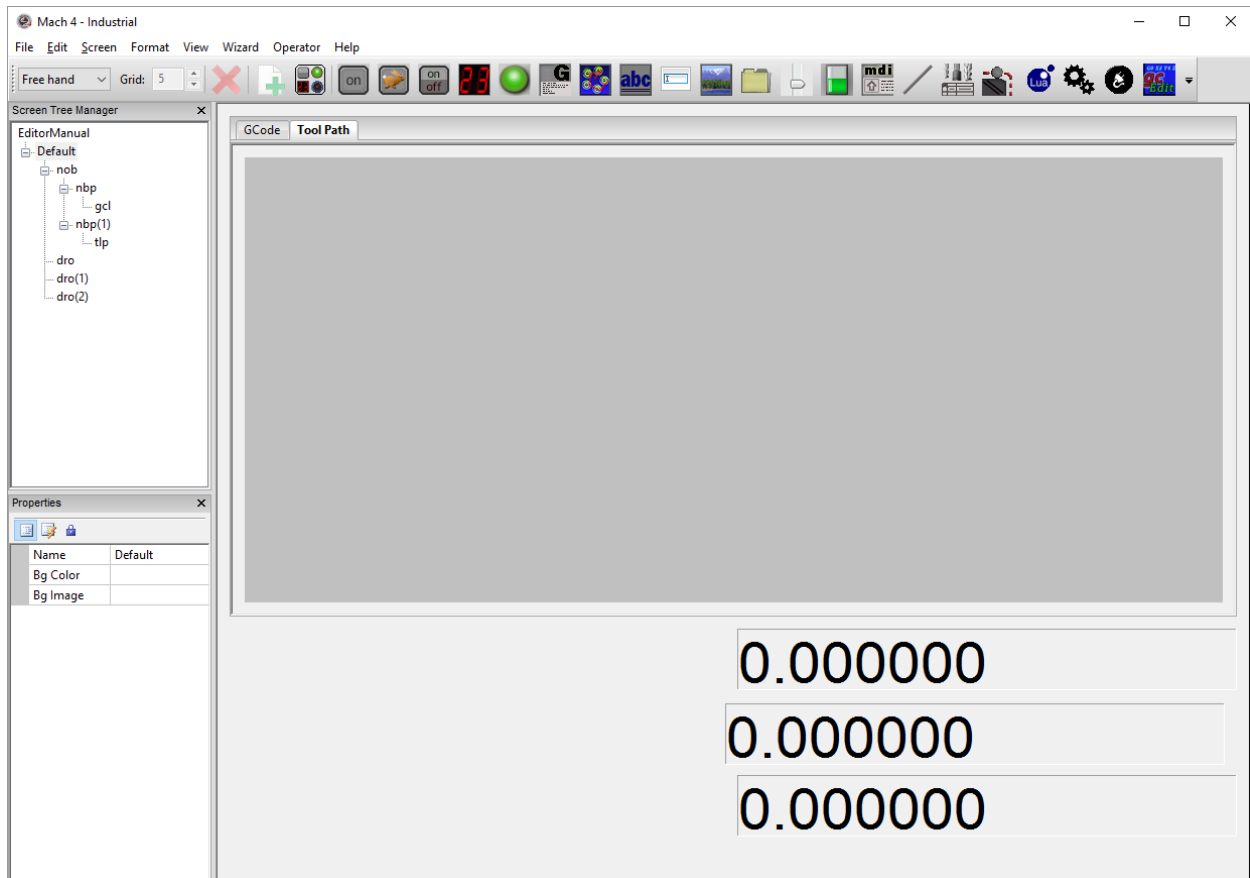


Figure 4-6 DROs Added

4.5 Aligning Controls

Now, you can align them. Move the lowest DRO close to the bottom of the screen and then hold control and select the other two DROs, middle and then top. The order of selection is important! Now go to the "Format" menu and move the mouse over "Spacing..." and then select vertical. The three DROs will be evenly spaced keeping the top and bottom DROs in their current position. Next go back to the "Format" menu and move your mouse over the "Align..." option and select "Rights". This will align the right edges of all the selected control to the LAST selected control, this is the reason the selection order is important. You should be left with a screen that looks like figure 4-7. Next you will add some functionality to the DROs.

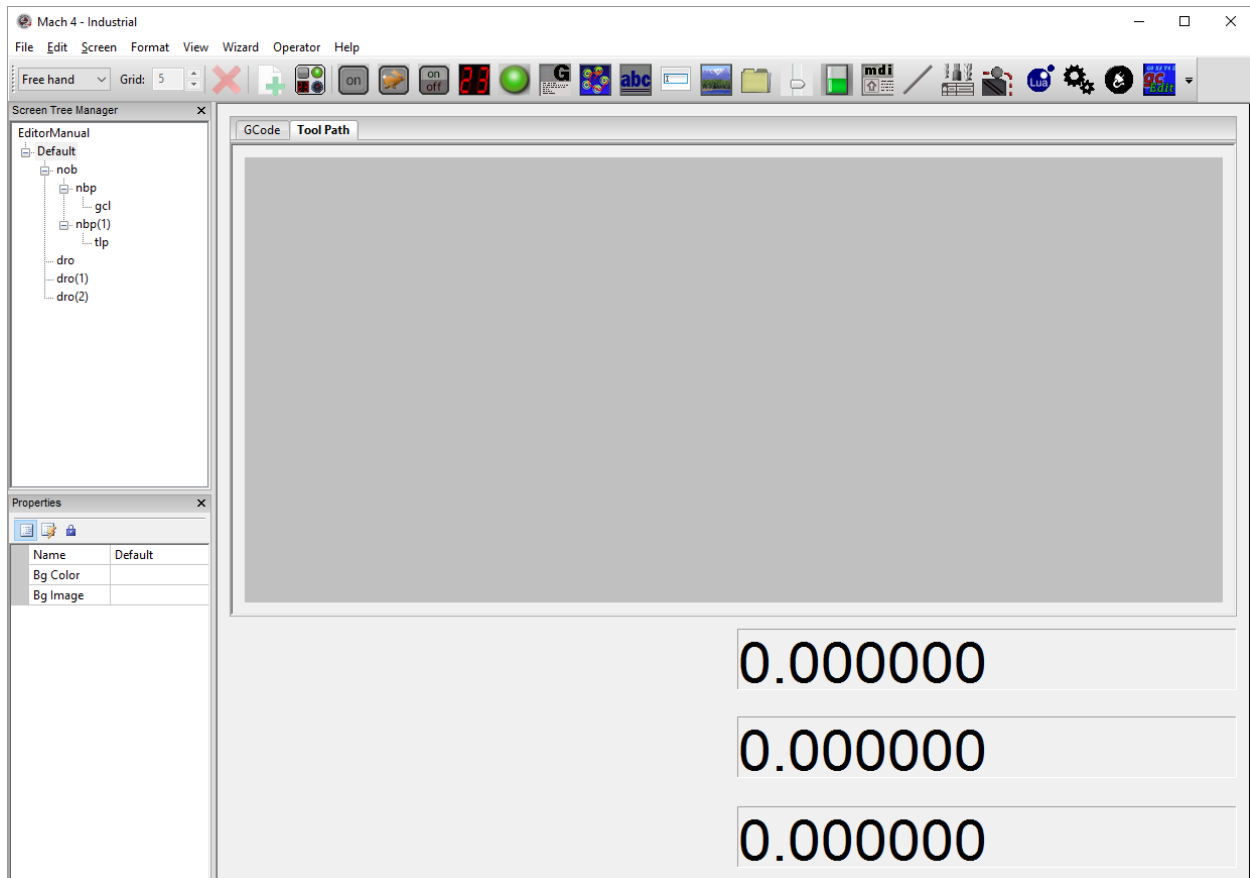


Figure 4-7 Aligned DROs

Click on the first DRO, we will make this the X axis position DRO. In the property window find the item named DRO Code and open the drop-down menu, select "X Multiple Pos". The middle one should be set to "Y Multiple Pos" and the bottom one "Z Multiple Pos". The multiple position option allows you to display absolute, machine and distance-to-go data in the same DRO. Later you will add buttons to toggle between these functions. Right now, let's add some labels so we don't forget which is which. Follow the same basic procedure as for adding the DROs but add static text controls instead. Use the format tools to align the labels with the DROs. You should end up with a screen that looks like figure 4-8.

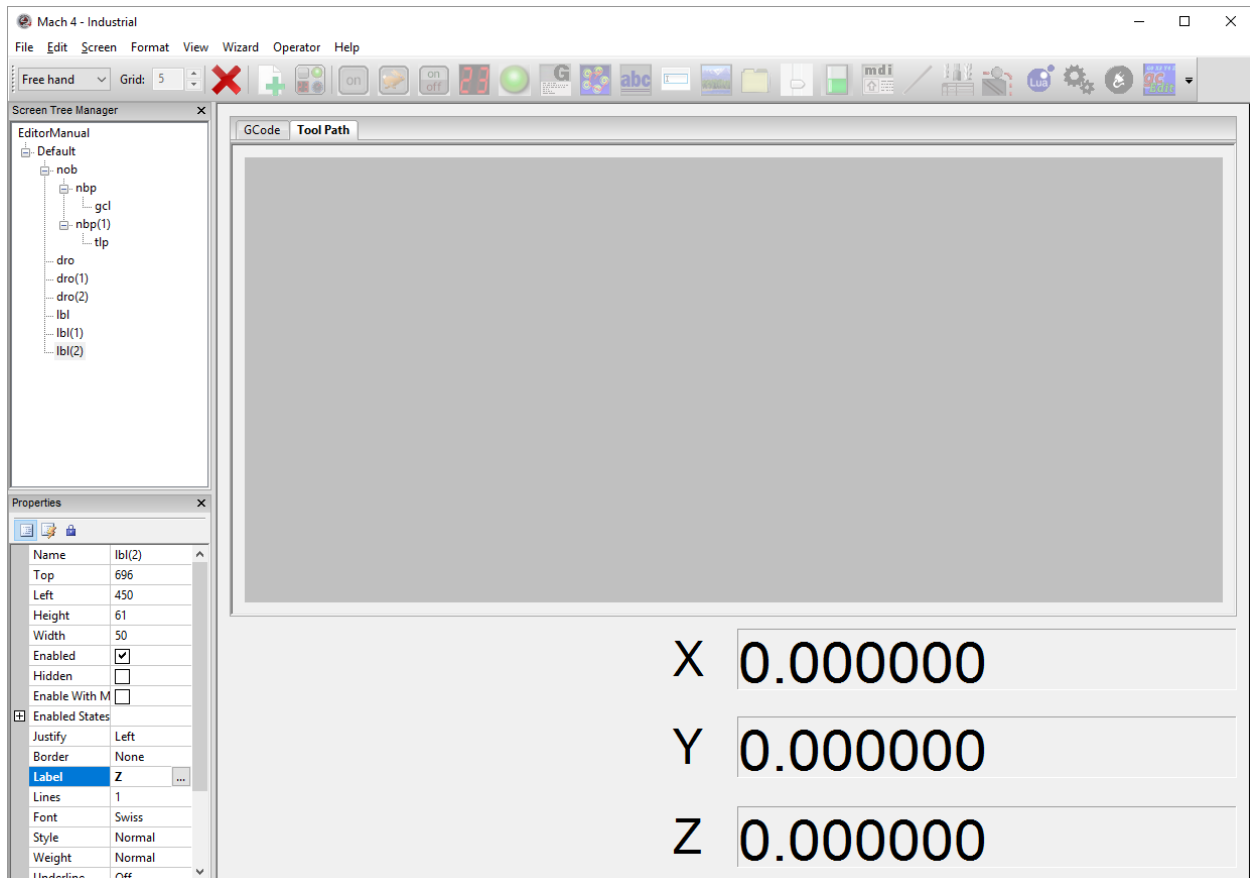


Figure 4-8 Static Text Labels

4.6 Buttons

To make this screen useful you need to add a few buttons to load a program, enable, start and stop a program, etc. Let's start with the enable button. We'll make this a large toggle button on the bottom left side of the screen. Select the default page and click "Add Toggle Button" on the tool bar. Once again it will show up on the screen at the top left corner under our tab control. Move the control to Top = 701, Left = 18 and resize it to Height = 50, Width = 400. You also want to change the text displayed on the button and the color. For toggle buttons, there are two states, up and down. For the enable button you want the "Text Up" to be "Enable" and "Text Dn" to be "Disable". Change the "Button Color Up" property to green and the "Button Color Dn" to red. To change the color, click on the property in the property window and then select the color from the palette. Figure 4-9 shows the properties settings for the new button.

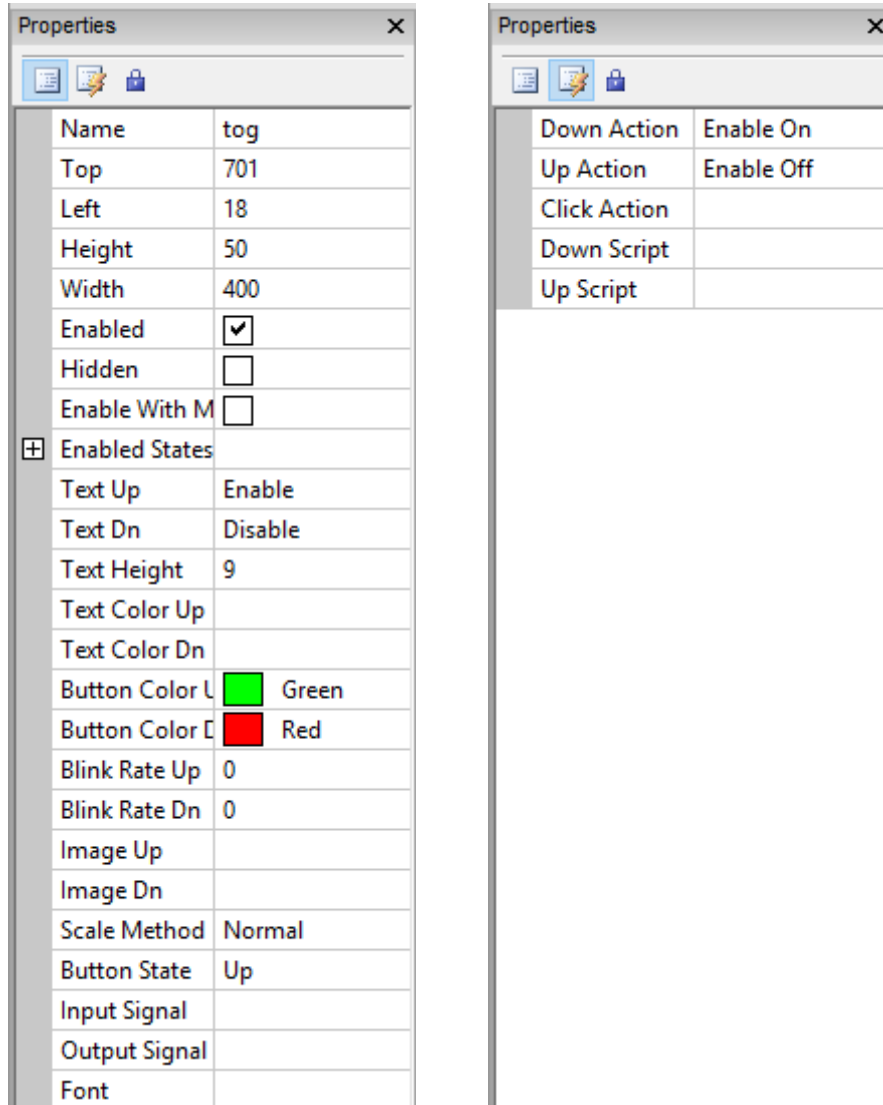


Figure 4-9 Enable Toggle Button Properties

Now you need to add the actions to the button that will enable and disable Mach4. In the properties window click on the lightning bolt and for the “Down Action” select “Enable On” from the drop-down list. Then set the “Up Action” to “Enable Off”, see figure 4-9. You should have a button on the screen that looks like figure 4-10.

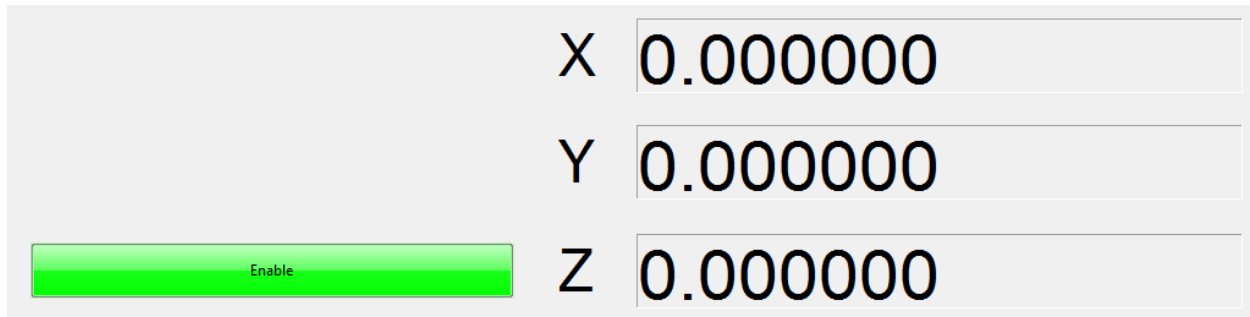


Figure 4-10 Enable Toggle Button

Next you will be adding some standard buttons to the screen and some LEDs to show the home status of the axes. Add 9 standard buttons and set the Height = 40, Width = 90 and label and position them as shown in figure 4-11. Add three LEDs and position them as shown.

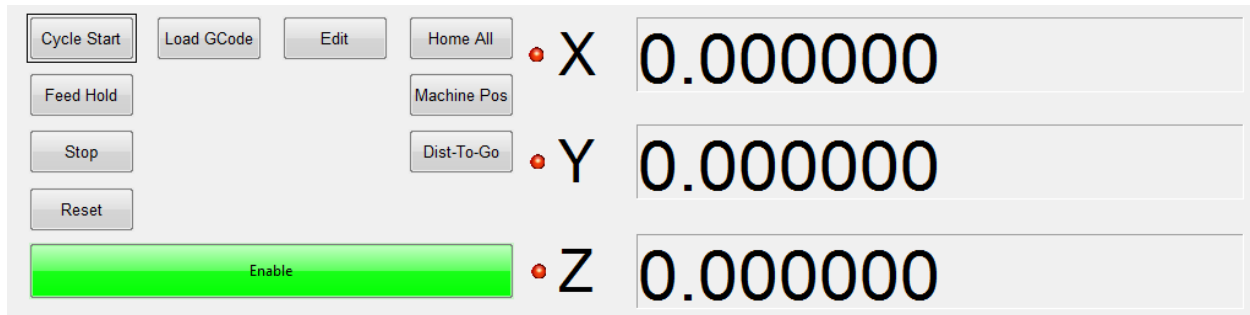


Figure 4-11 Control Buttons and LEDs

At this point you should be comfortable adding the controls, positioning them and changing the displayed text. Now we need to add the actions. All the buttons will have predefined actions for their function, except for the “Home All” button, which we’ll get to later. For example, click on the “Cycle Start” button and then go to the events properties. The first option is “Left Down Action”. Select “Cycle Start” from the drop-down menu to define the function. Repeat for “Feed Hold”, “Stop”, “Reset”, “Load GCode”, and “Edit”, selecting the appropriate function from the drop-down menu.

For the “Machine Pos” and “Dist-To-Go” buttons, select “Machine Coordinates Toggle” (or “Distance To Go Toggle” for both the left down and left up actions, figure 4-12. Now when you push the button down the DROs will display the machine or distance-to-go coordinates, when you release the button they will go back to the absolute coordinates.

The screenshot shows a 'Properties' window with a table of actions. The table has two columns: the first column lists the action type, and the second column shows the assigned action.

Left Down Act	Machine Coordinates Toggle
Left Up Action	Machine Coordinates Toggle
Left Down Scri	
Left Up Script	
Clicked Script	
Lost Focus Scr	
Goto Page	
Run MDI	

Figure 4-12 Machine coordinate button actions

Now for that special “Home All” button. There is no option for home all from the drop-down list. You must add a Lua script to that button. Select the “Left Down Script” action to open the script editor and copy the following text:

```
local inst = mc.mcGetInstance()
local rc = mc.mcAxisHomeAll(inst)
if (rc ~= mc.MERROR_NOERROR) then
    mc.mcCntlSetLastError(inst, "Error starting home all")
end
```

Close the Lua editor and save the changes.

4.7 LEDs

The LEDs also need to be assigned to signals to turn them on and off. In this screen, you want them to display the homed state of each axis. Select the X axis led to display its properties in the properties window, shown in figure 4-13.

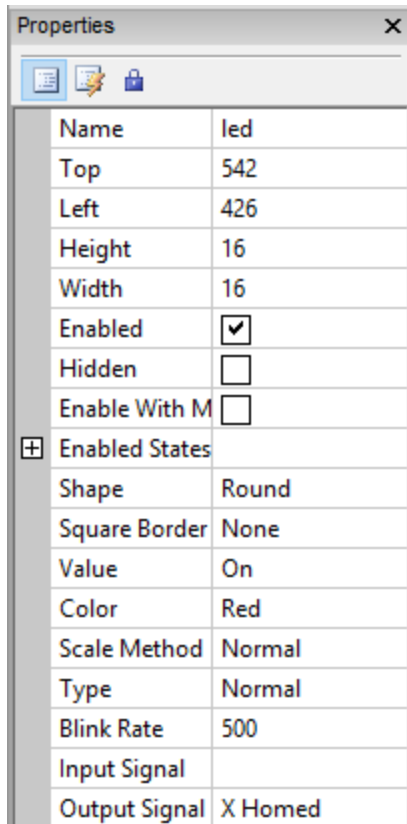


Figure 4-13 LED Properties

Input and output signals can be assigned to an LED, see the bottom two properties in the window. Assign the LED to the “X Homed” signal. Don’t get “X Homed” confused with “X Home”. “X Home” is a different signal and will only display the status of the home switch, not the homed state of the axis. Set the Y and Z LEDs to display those home states as well

4.8 Editing Completed Screen

You have now completed a very basic screen. It should look like figure 4-14. Exit the editor by going to Operator>Edit Screen and save the screen. Try changing tabs, loading a program, etc.

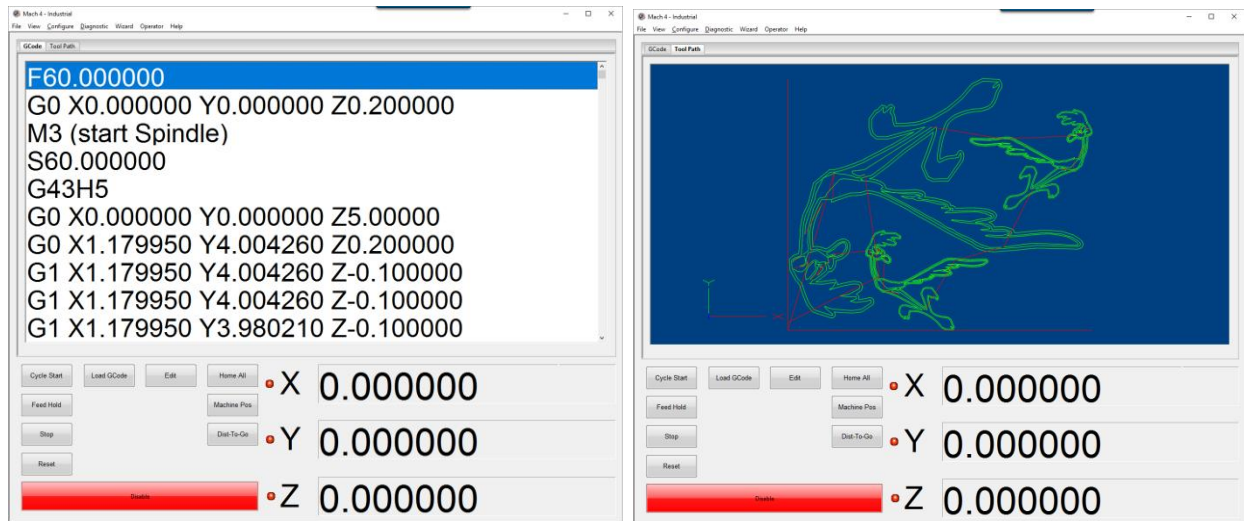


Figure 4-14 Finished Example Screen

Immediately you'll notice this screen works for running a program, but it is missing a couple key features. There are a few things we will want to add, sliders for feed rate and rapid override, a status bar to display messages and some jog controls. Start with the status bar. A good place to put it is across the top. Start by resizing the tab control, select it and drag the top edge down to get enough room to put a static text control across the top, with another for a label and a history button, figure 4-15.

4.9 Static Text Controls

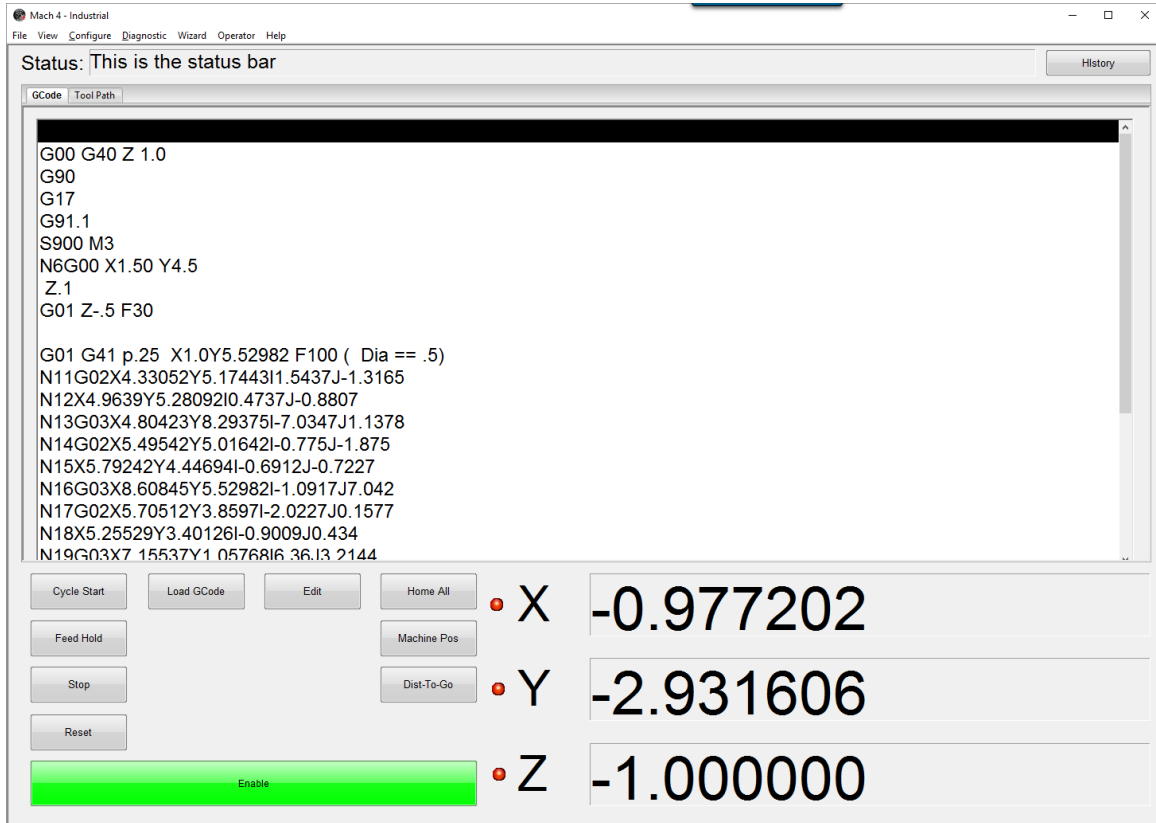


Figure 4-15 Add status bar at the top

Figure 4-15 shows the addition of the status bar at the top. There is one static text control to display the "Status:" label, a second one to display the last or current message and finally a button to show the message history. To set a static text control to display the data like the current message you need to select a label code in the control's properties, figure -16. Select the static text control that will become your status bar and select "Last Gcode Message" from the drop-down menu in the "Label Code" property. Lastly, you need a history button to see previous messages. Add a button, name it "History" and set the left down action to "Show Message History".

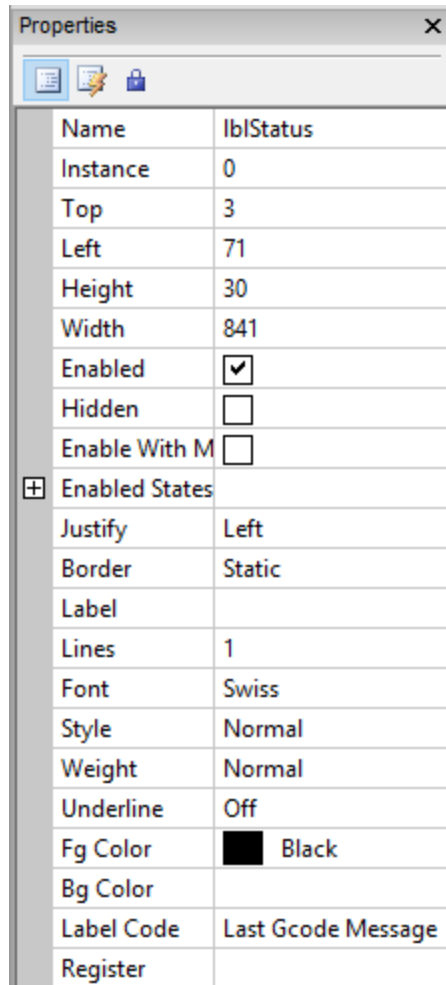


Figure 4-16 Static text properties

4.10 Sliders

Now let's introduce some new controls, sliders. Sliders are useful for dynamically varying the value of something, like feedrate override. Add a couple sliders in the empty space between the buttons. In the properties set the "Code" to "Rapid Rate Override" and "Feed Rate Override". Generally, the upper and lower limits of the rapid override should be 0-100, and the feedrate override 0-200. These settings allow you to stop the motion and increase the feed rate if needed. See the properties in figure 4-17.

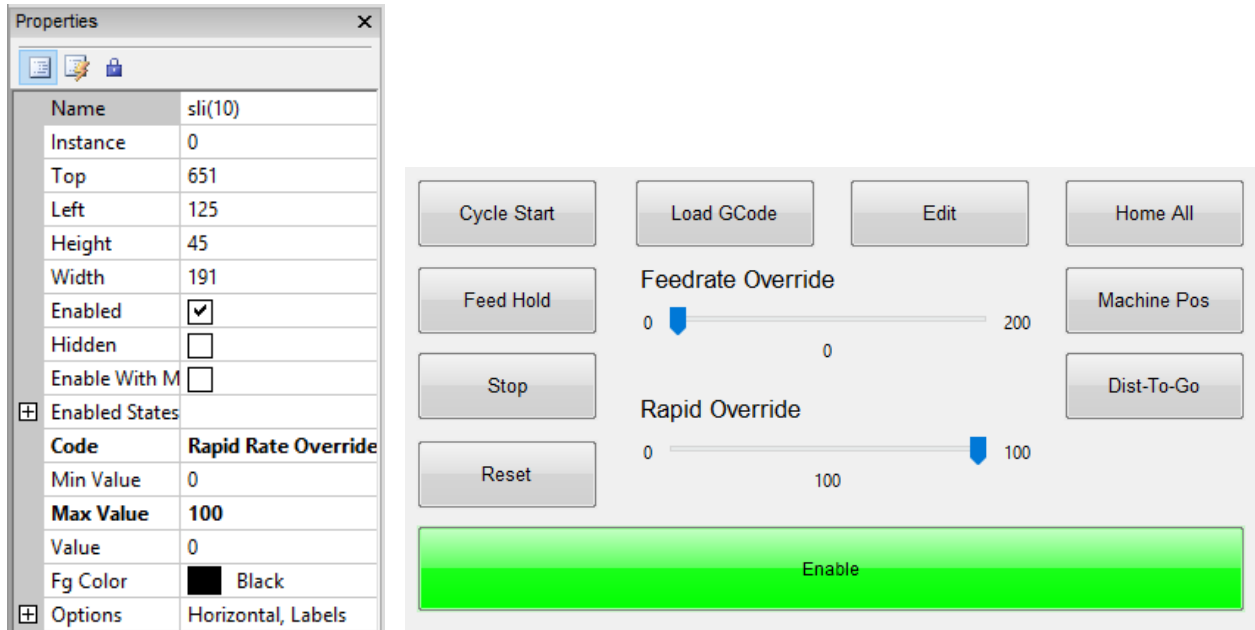


Figure 4-17 Add override sliders

Exit the editor and play with the sliders. Click and drag the pointer to adjust the value anywhere in between the Min and Max value you set in the properties.

4.11 Manual Jog Buttons

Lastly, add some controls for manually jogging the machine. A good place to put these is on another tab. Enter the editor and add a page to the tab control. Next add some standard buttons, two for each axis, and arranged in a way that makes sense to you, see figure 4-18 for an example layout.



Figure 4-18 Jog button tab

Once the buttons are added you need to assign actions to make the machine jog. Jogging is a bit tricky, there are two actions that need to be used to get a button to do what you expect. The first action, which you will assign to the down action, starts the jog command. The second action, which you will assign to the up action will end the jog command. For an example see figure 4-19 for the event properties of the Y+ button.

Properties	
Left Down Act	Jog Y+
Left Up Action	Jog Y Off
Left Down Scri	
Left Up Script	
Clicked Script	
Lost Focus Scr	
Goto Page	
Run MDI	

Figure 4-19 Y + job button actions

If you forget to add the “Jog Off” action, then the axis jog will start and not stop. This is a very important thing to remember. With these actions defined, when the button is pushed the machine will jog until it is released.

4.12 Manual Data Input (MDI)

The last thing that will be added as a part of this manual is an MDI (manual data input) window. Let’s put this on yet another tab, named MDI. An MDI window allows you to manually enter blocks of GCode to be executed. Adding the MDI window is straight forward, executing it is where it gets a bit tricky. So, add a tab and insert an MDI window, size it to fit the tab, figure 4-20.

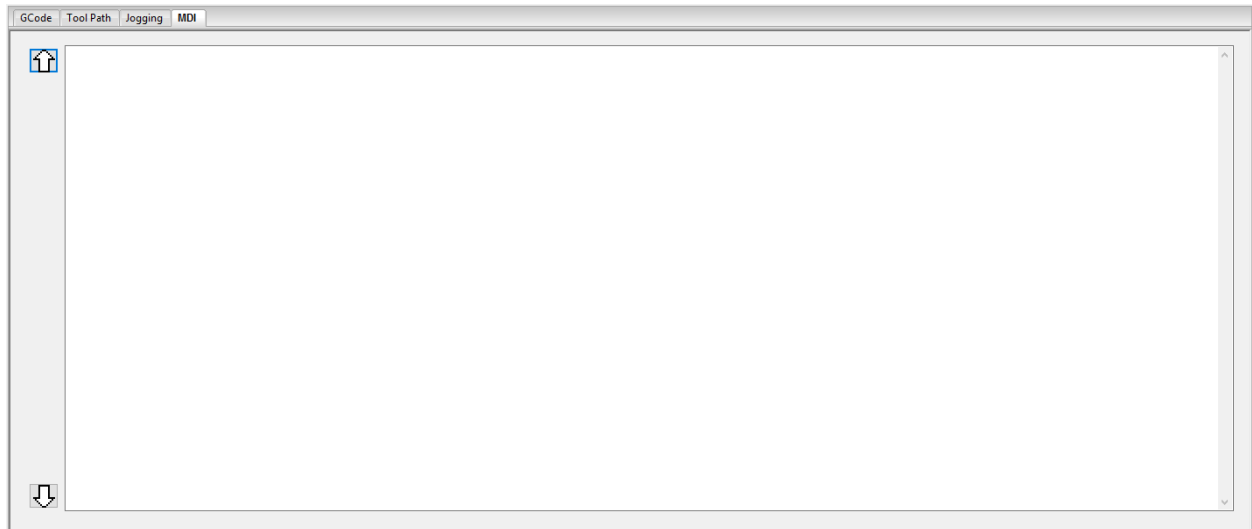


Figure 4-20 MDI tab added

There are a couple options for executing the contents of the MDI window. One is to add a new button to only execute the contents of the MDI window. There is an event property on standard buttons named "Run MDI", select the name of the MDI window you want this button to execute and that is it. The other way, which will be covered in this manual is to have the "Cycle Start" button run either the currently loaded GCode file or the MDI window depending on what the user is looking at. This way the screen is simplified and avoids unintended automatic operation. Start by clicking on the "Cycle Start" button and navigate to the event properties. Remove the "Cycle Start" event from the "Left Down Action". Next you can add a "Left Down Script". The following Lua code can be copied and pasted into the Lua editor:

```

-- Control instance
local inst = mc.mcGetInstance()
-- Get the currently active tab.
-- The current tab starts at 0 and counts up
local tab = scr.GetProperty("nob", "Current Tab")

if (tab == "3") then
  -- If the current tab is 3, the MDI tab
  -- execute MDI
  scr.ExecMdi("mdi")
  mc.mcCntlSetLastError(inst, "Execute MDI")
elseif (tab ~= "2") then
  -- If the current tab is not the jog tab
  -- then cycle start gcode
  mc.mcCntlCycleStart(inst)
  mc.mcCntlSetLastError(inst, "Execute GCode")
else
  mc.mcCntlSetLastError(inst, "Nothing to execute")
end
end

```

Close the Lua editor and save the code. Exit the screen editor and test your new controls. Pay attention to the status bar, the “mc.mcCntlSetLastError()” statements write to the status bar. It’s a good way to know what your custom events are doing. Try some MDI code, try to cycle start the program when the Jogging tab is active.

4.13 Completed Screen

That concludes the creation of a basic example screen. This screen is only meant as an example of how to edit and create screens. There is still plenty to add to this screen, on the Jog screen alone you could add sliders for jog speed, buttons to toggle between continuous and incremental jog and DROs and buttons for the increment amount. There are many functions available in Mach4, far more than fit into the scope of this manual. However, armed with the basics learned in this example you can explore the rest of the controls and their actions to build the rest of your screen. And remember, if a predefined action is not available, nearly anything can be done using the Lua interface, more information on that can be found in the Lua Scripting Manual.

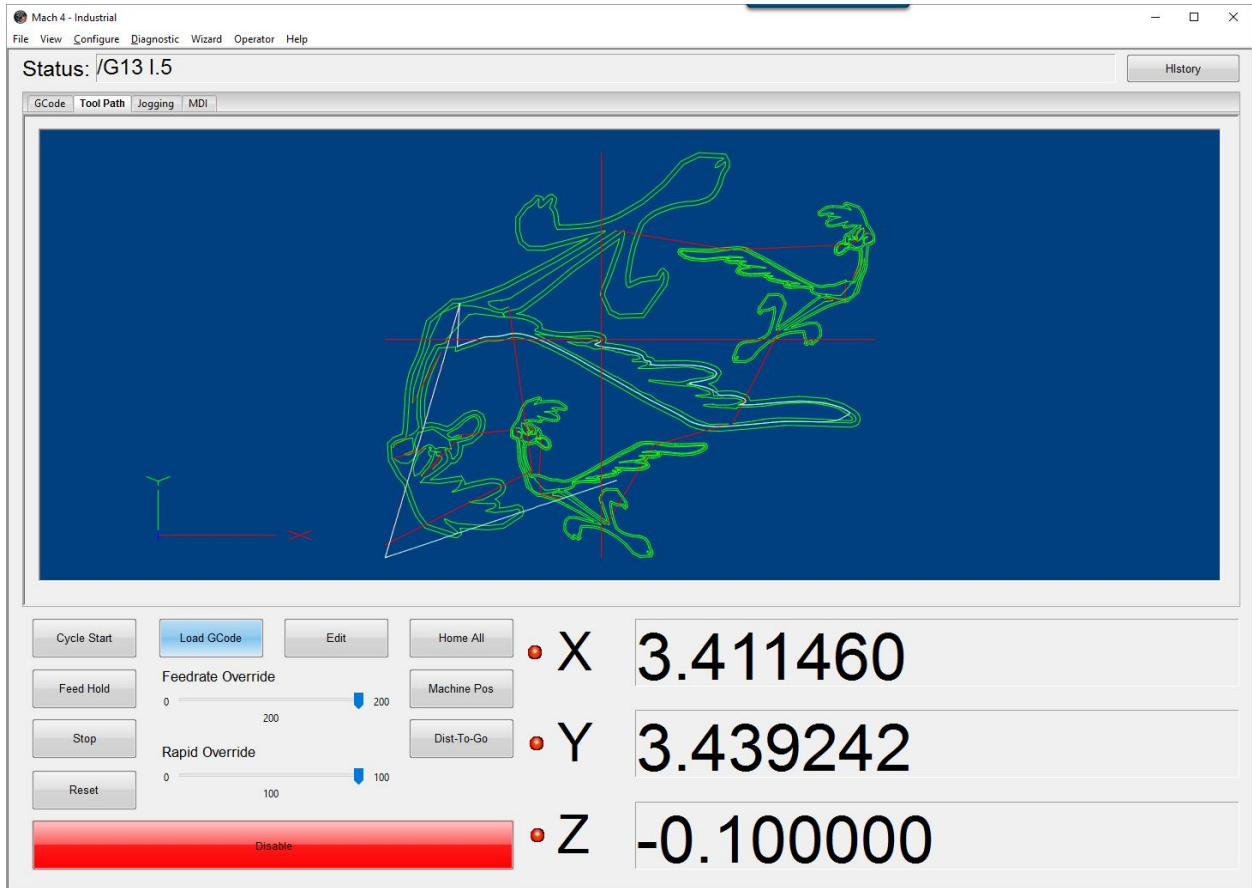


Figure 4-21 Completed example screen

Chapter 5 Screen Scripts

5.1 Screen Script Overview

The GUI has a LUA script that is comprised of several script “snippets”. All of these script snippets are concatenated into one LUA script in the order describe below. This facilitates the use of global variables that all of the script snippets can access.

5.1.1 Screen Load Script

The screen load script is run when the screen is initially shown or after the screen is edited. Global variables, if used, should be defined in the Screen Load Script. These global variables will be available to all of the scripts below the Screen Load Script. Global variables can be used as control variables that dictate what other parts of the script do, for example.

5.1.2 Screen Unload Script

The screen unload script is run when the screen is unloaded, whether by closing the GUI, loading another screen set, or by editing the screen. This script can be used to perform housekeeping duties like cleaning up processes, stop timers, etc.

5.1.3 PLC Script

The PLC script is run on a time interval that is defined by the “PLC Interval” property of the screen. It allows, as the script’s name implies, for Programmable Logic Controller type functionality.

The PLC script snippet is wrapped in a LUA function named Mach_PLC_Script and has no parameters.

5.1.4 Signal Script

The signal script is fired when any core signal is changed.

The signal script is wrapped in a LUA function named Mach_Signal_Script with parameters sig (number) and state (number). The sig parameter contains the signal ID of the changing signal and the state parameter contains the new state of the signal (1 on, 0 off).

5.1.5 Timer Script

The timer script is fired when a screen timer expires. It allows for timers that run independent of the PLC script interval.

The timer script is wrapped in a LUA function named Mach_Timer_Script with a parameter of timer (number). The timer parameter contains the ID of the timer that expired (0-9).

5.1.6 Control Event Scripts

Many controls have event scripts which can be populated with script code to perform actions when their respective events are fired.

5.2 Debugging

Since the screen script is run behind the scenes, and in some cases, as a result of external events, it requires a special debugging technique called Remote Debugging. Remote Debugging is where one process is running the script and another process is debugging it. In the normal case, the GUI will be running the script (client) and the LUA editor will be debugging it (server). The scripts running on the client will need to contain some special commands to initial debugging. This is accomplished with the “mobdebug” LUA module.

5.2.1 Debug Preparation

Screen scripts do not have debugging enabled by default. Since they are expected to run, in the normal case, at full speed, including the debugging module might impact speed in a production environment. Follow the steps below to prepare the screen script for debugging.

- Open the screen editor (Operator->Edit Screen).
- Navigate to the Screen Load Script and edit it by pressing the ellipse button.
- Insert the following lines at the very top of the Screen Load Script:
mobdebug = require('mobdebug')
mobdebug.onexit = mobdebug.done
mobdebug.start()
- Close the editor. This is important, as the script modifications will not take effect unless the editor is closed.
- Save the screen, but do not exit the screen editor. This is also important because exiting the screen editor at this point will start the script!

5.2.2 Starting The Debug Session

The debug session needs to be started before the screen editing is finished for the reason stated above.

- In the GUI menu, open the LUA editor (Operator->Open Script Editor)
- Once the editor is open, in the editor’s menu, start the debug server (Project->Start Debug Server)
- Back to the GUI, exit the screen editor (Operator->Edit Screen).

Once the screen editor is exited, the LUA editor will pop up with a copy of the entire screen script and break right after the line that contains “mobdebug.start()”. The script is read only and cannot be edited.

At this point, you can step through and debug the screen load portion of the script, if needed. Otherwise, you can set break points in other areas of the script that are of interest. Say you want to debug the PLC script. Simply put a breakpoint on a line in the Mach_PLC_Script

function. Then continue the debug session by pressing the play button on the LUA editor's tool bar or by pressing F5.

Once the PLC script processes the line with your break point, the LUA editor will pop back up allowing you to inspect variables and step through the script. This can be done as many times as needed. Keep in mind that strategic placement of breakpoints can help make the debugging process smoother.

If an error is found, you can't really edit it in place. This is because the script that is displayed is not the actual script that is running, but rather a copy of it. You will have to exit the LUA editor and edit the screen to apply changes. Repeat as necessary until the desired functionality is achieved.

5.2.3 Ending The Debug Session

Once you are done debugging the screen script, you simply close the LUA editor. If you do not want to do anymore further debugging, edit the screen load script and either remove the debugger commands or remark them out and save the screen.