Chip

Chip is an application for controlling a milling machine using TinyG hardware. This is a work in progress. Currently Chip can be used to run a milling machine, including homing, jogging, using a probe and running gcode programs. What is not yet complete, is the macro interpreter and a form editor that is used to add and modify controls (ie. buttons and text boxes).

The macro interpreter language is gcode extended to include IF THEN ELSE, FOR loops and other system functions like opening and saving files and math functions including trig functions.

Complex equations can be used in the macro interpreter. Macros are developed by the user to provide all control functions. No machine control is done directly by the application. Everything is done with macros. This gives total control to the user.

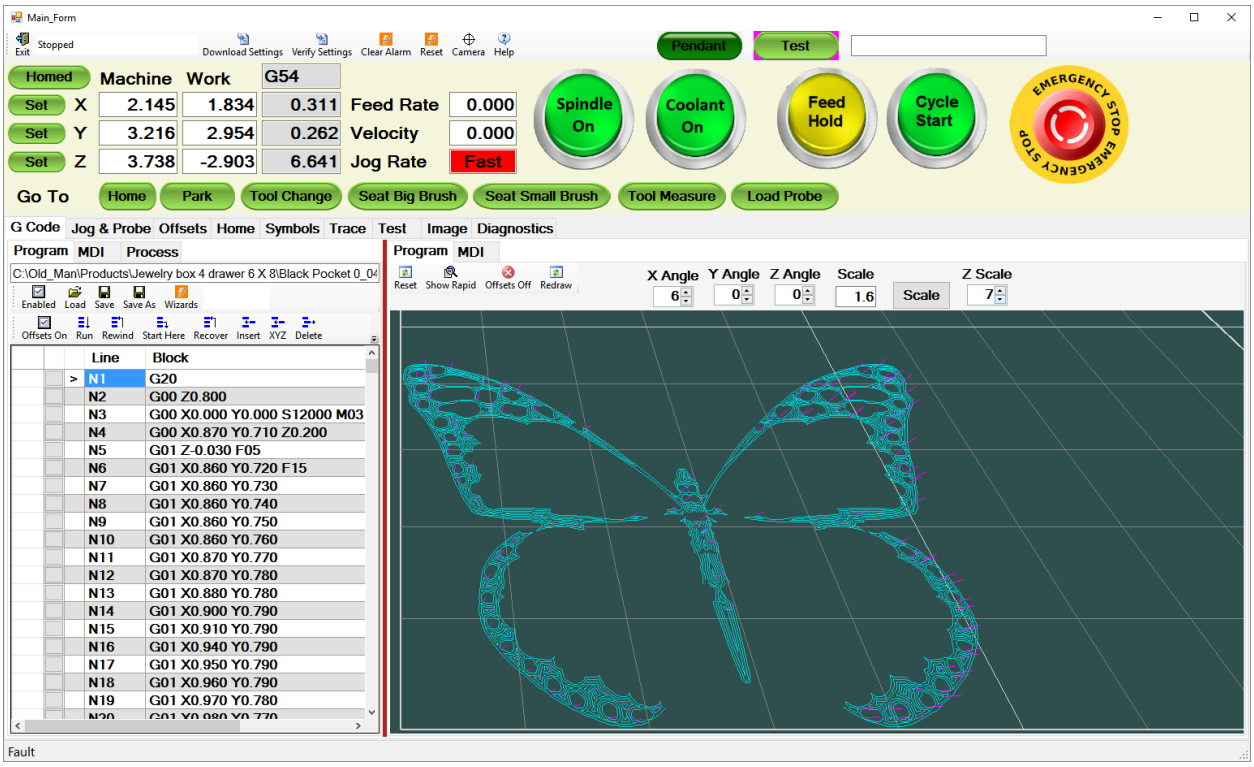
The only control function done directly by the application is downloading the TinyG settings.

The form editor is used to put controls on the form as well as configuring the controls. For example images for buttons can be assigned. Text boxes have font, back color, text color and other properties that can be set.

Note the in the following documentation was written as if the macro interpreter and form editor were done. This is so the documentation does not have to be re-written when they are complete.

This is the main form. With the exception of the tool bar at the top left, all controls are user defined. Most controls are either buttons or text boxes, however there are more complex controls, like the gcode listing in the lower left and the drawing in the lower right.

The form has two main parts, the upper part with the axis positions and some buttons including E-Stop are always visible. The lower part of the form has tabs that contain various functions. The G Code tab is used to run programs, do MDI and Processes.

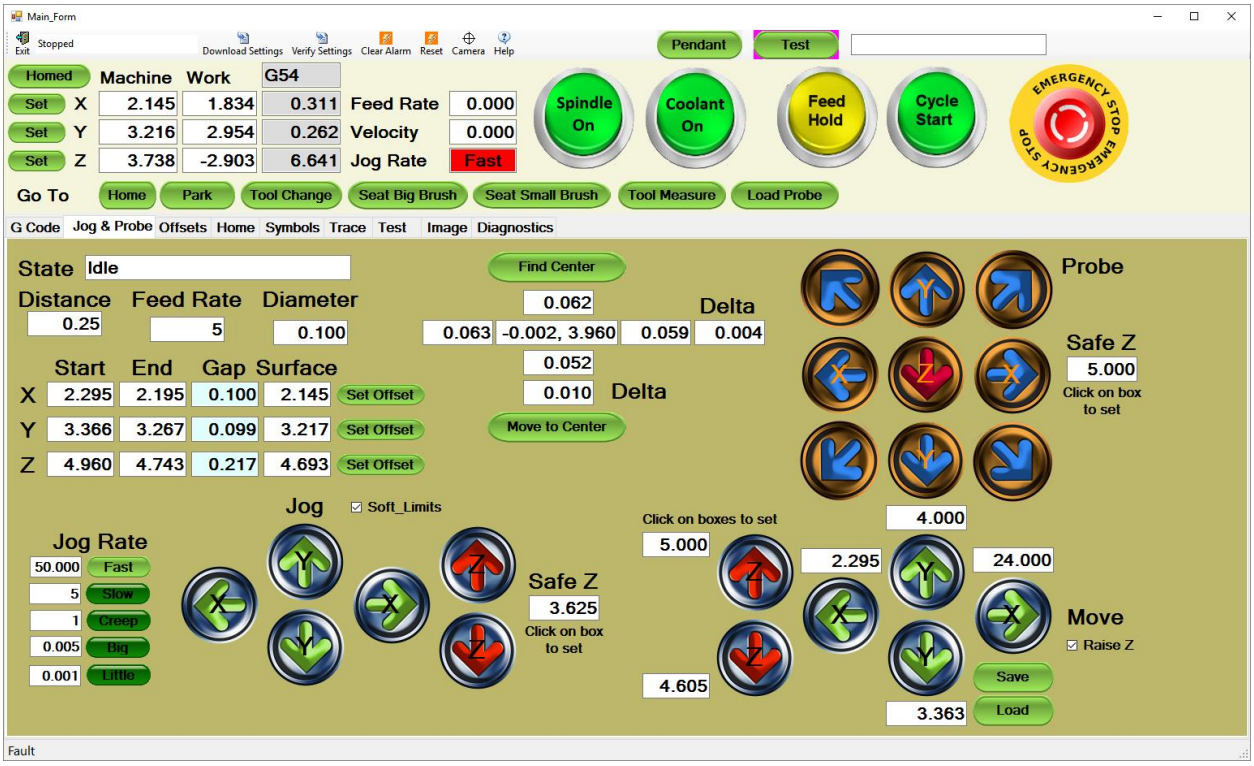


Processes are macros that provide higher level control. For example a process might be used to organize the making of part that has several gcode programs, tool changes and part mountings. The Process macro could display drawings and other documentation like how to mount parts on a fixture, facilitate tool changes, load offsets, load and execute gcode programs. It can also help in product development by organizing CAD drawings, CAM sessions and the like.

More tabs can be added by the user to provide more space for buttons and other controls. Simple controls include buttons, text boxes, labels, check boxes and radio buttons. Complex controls include G Code listing, drawing of G Code, macro editor and symbol table (in tree form).

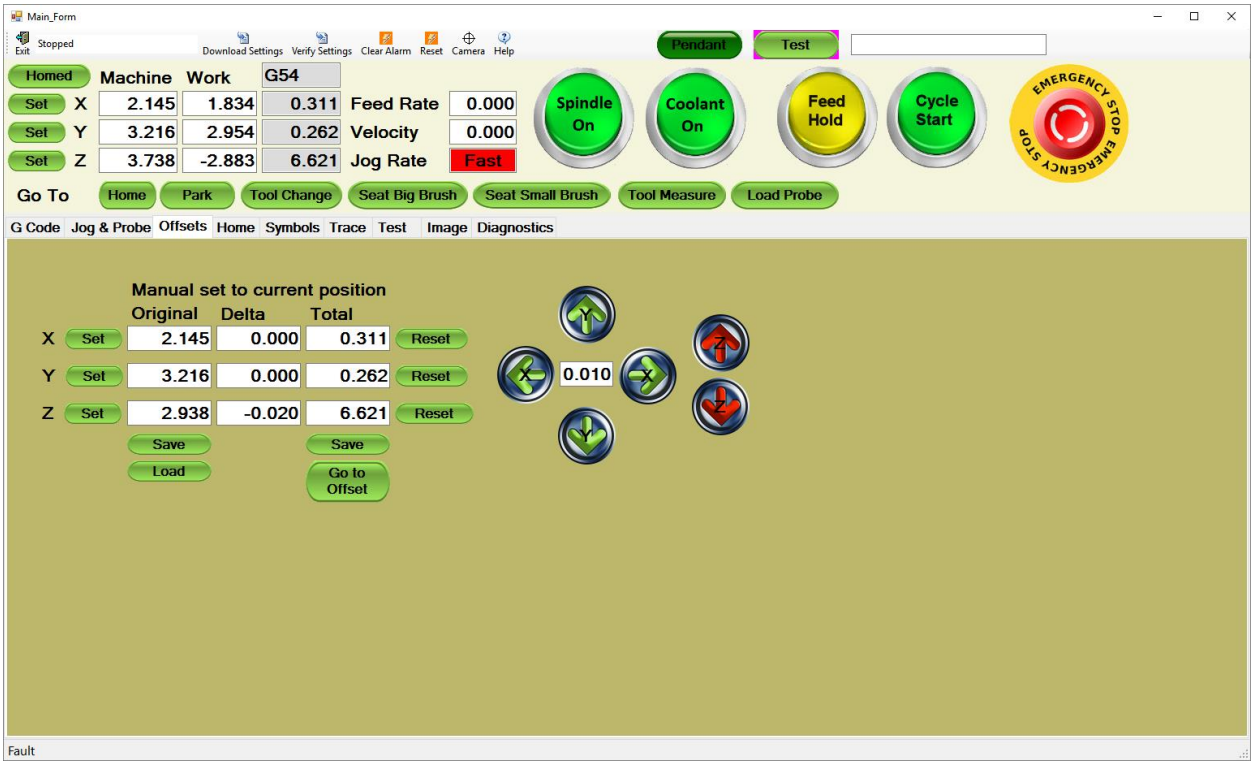
Other forms can be developed that use the same controls and macros. These could be opened by a button for example. The G Code listing control has a ‘Wizard’ button that brings up a form that implements several wizards that do things like simple profiles and drilling.

Shown below is the Jog & Probe tab. The jog functions are in the lower left, move functions in the lower right, and probe functions un the upper half of the tab. Move functions are used for things like adjusting fixtures where probe cycles are run at different positions repeatedly. These positions can be set so that moving between two positions is done with move buttons.



Note that all buttons and text boxes all have macros that do the functions.

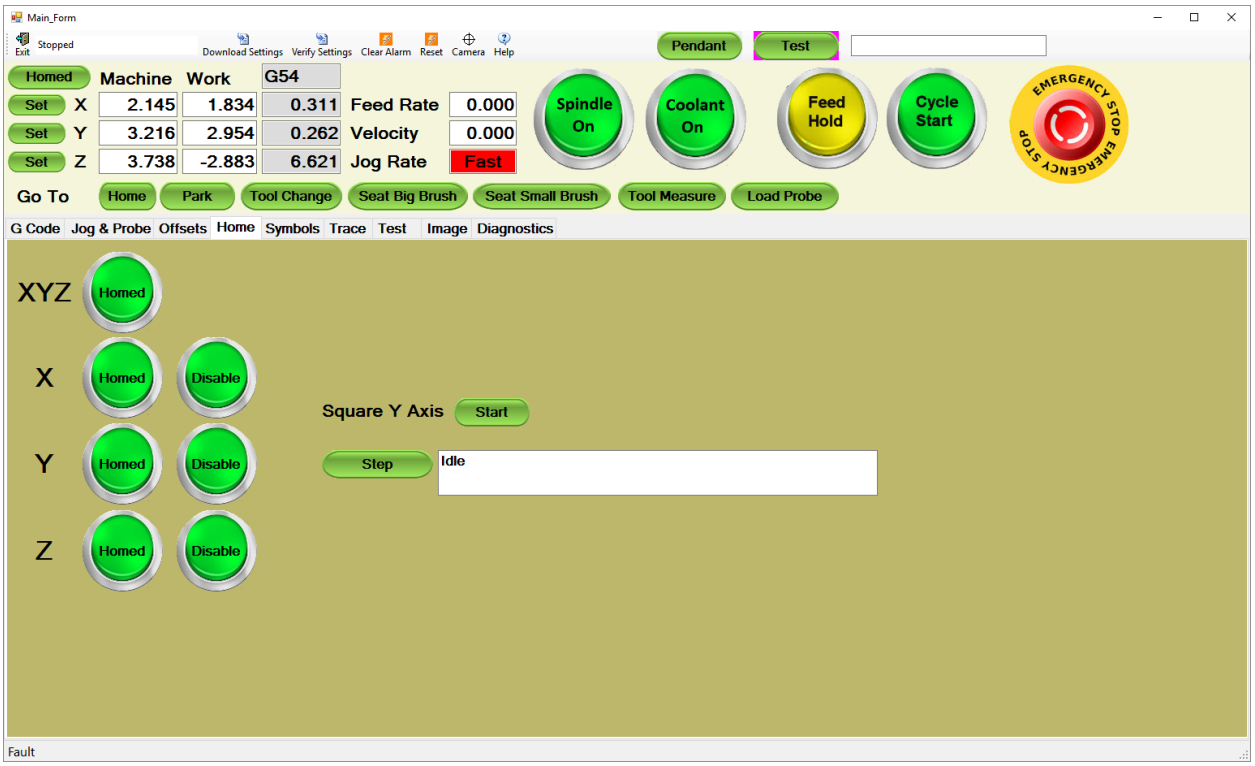
The Offset tab is used to set, load and save offsets. The buttons are used to adjust the offsets. For example a gcode program that flattens a surface may be run several times. The Z offset can be changed each pass by clicking the Z minus button. The Z Original text box has the original offset, the Delta text box shows the adjustment and the Total text box shows the current offset. The text box in the middle of the buttons is the current increment in this case 10 thousandths. So the first pass was at a Z offset of 2.938, then the Z minus button was clicked to subtract 0.010, the program run again, the Z minus clicked to subtract another 0.010. If the program is now run, it will run with a Z offset of 6.621 or -0.020 less that the original 2.938.



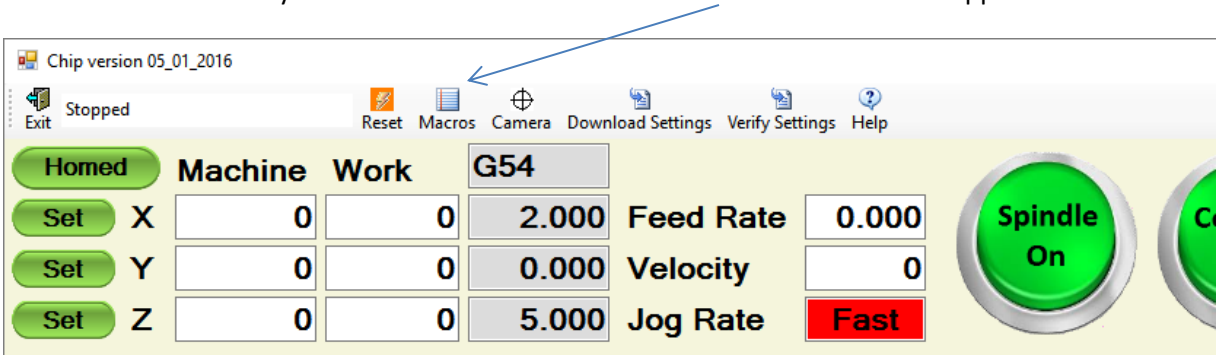
When an adjustment button is clicked, the macro that handles the button adds the adjustment, in this case 0.010 to the Delta, and the Delta is added to the Original to get the Total, then a message is sent to the TinyG to set the new offset. There is another macro that is triggered by the TinyG response to an offset message. This macro updates the Offset text boxes in the upper right of the main form, next to the Machine and Work positions.

This may seem like a round-about way to set the main offset text boxes, but this is a philosophy well worth emulating. The macro that handled the offset adjustment button could have also set the main offset text boxes, but if something happened and the TinyG did not get the message to set offsets, or a macro sends the offset message, but does not set the offset text box, then the application would be out of sync with the TinyG. Status and machine state should always be based on messages from the TinyG. That way the Chip application and the TinyG will always be in sync.

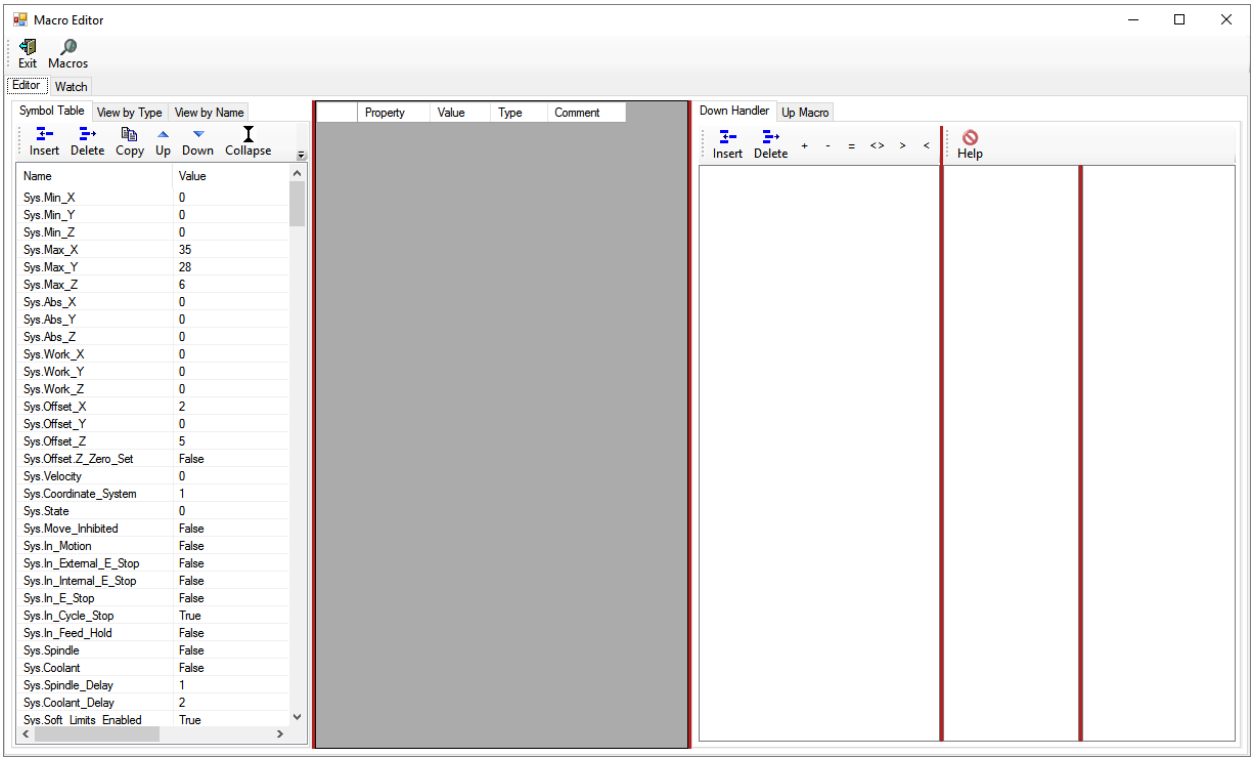
The Home tab is used for homing and enabling axis. Note that this tab has buttons for squaring the Y axis. This is because the Y axis on this particular machine is a gantry with two stepper motors. These buttons lead the operator through a sequence of steps needed to square the axis.



The macro editor and symbol table are accessed with the Macros button in the upper tool bar.

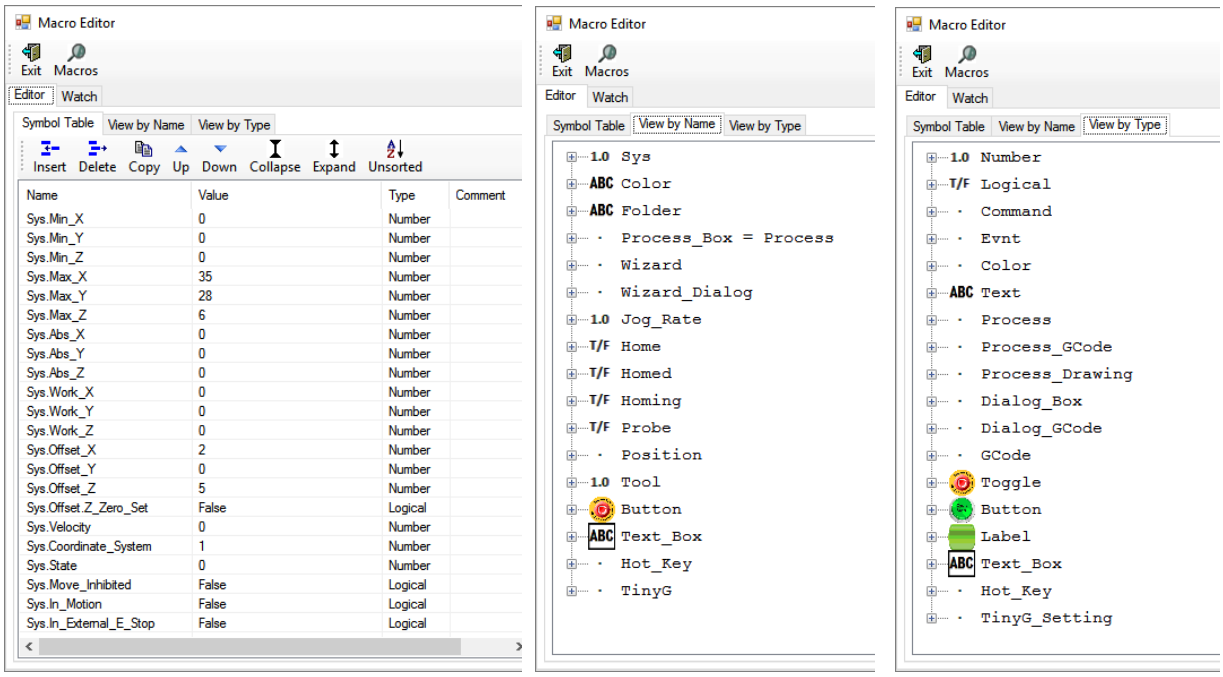


This dialog has three areas, the symbols are shown on the left pane. There are three tabs for showing the symbols in table or tree form. The center pane is for showing and editing the symbol properties and the right pane is for editing macros. It has two tabs to show event handler macros. There is also a help pane that shows context sensitive help, macro errors or values of local macro variables.

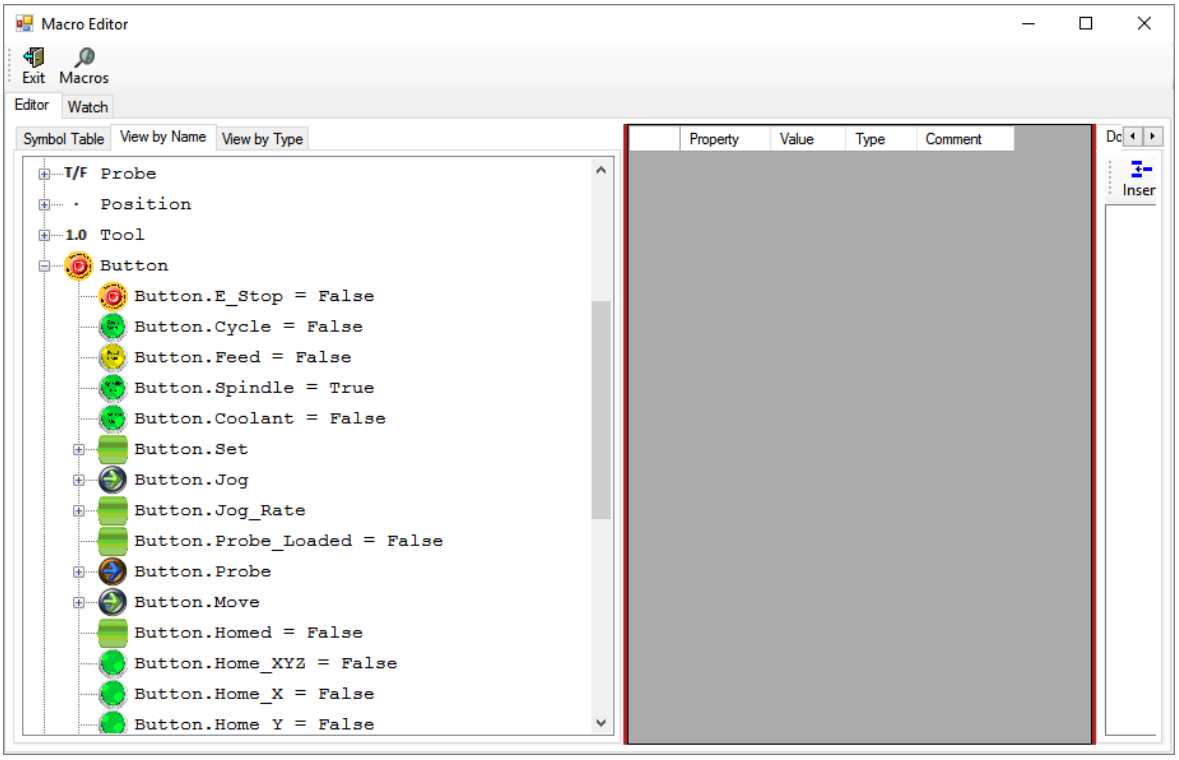


Chip was designed to give total control to the user. All control functions are implemented in macros. All objects (Global Variables, Buttons, Text Boxes, Hot Keys etc.) are in the symbol table. There are currently almost 400 symbols, but the user can add more to implement added macros.

Because there are a lot of symbols, they are shown three different was on the three tabs. The fist is in table format than can be sorted, the second is in tree format with a data type hiarchy and the last is a tree with name hiarchy. To expans a node, just click.



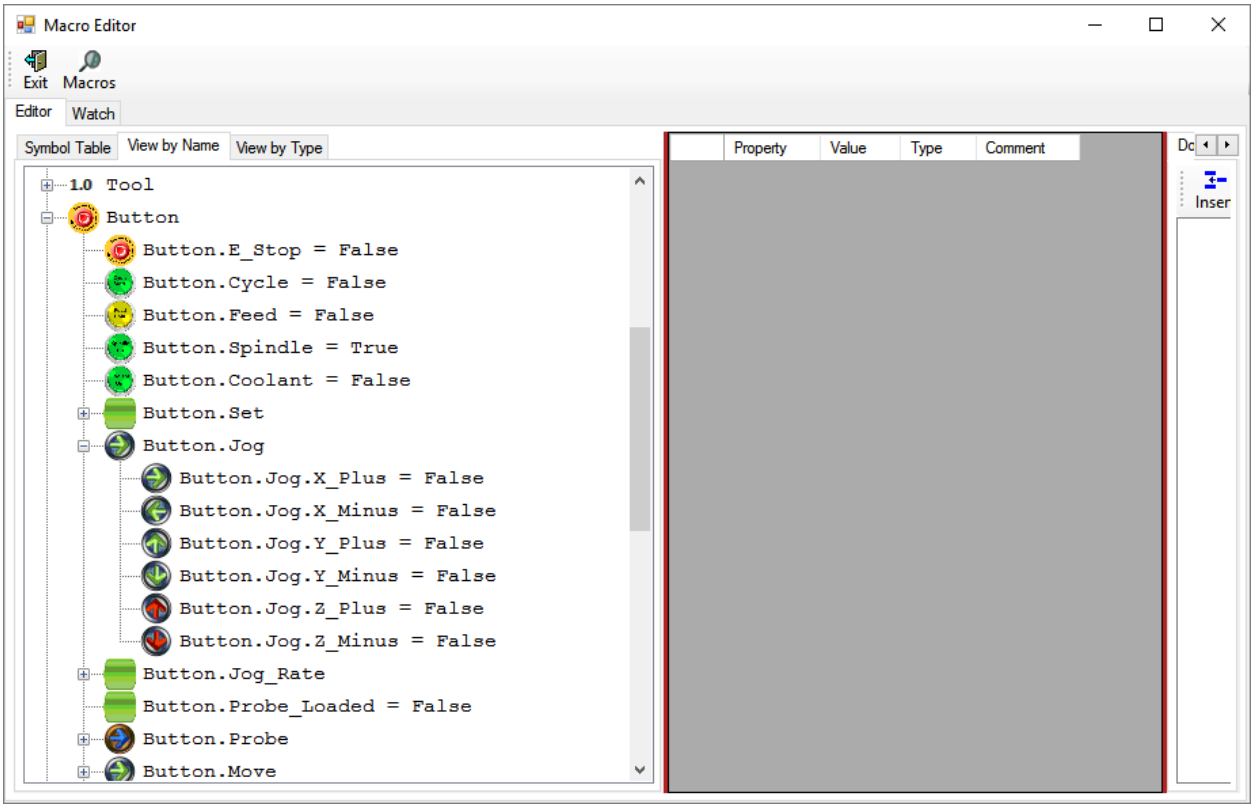
This is the View By Name tab with the button node expanded. The image for the button is shown as an icon. Note the nodes at the lowest level show their current value.



Symbols are shown in object oriented format. The periods in the name separates the different levels.

For example Button.Jog.X_Plus has three levels. When naming symbols take care to make the names so they follow some logical hierarchy. The highest level for this button is ‘Button.’ so all buttons can be in this group and easily found. Similarly ‘Jog’ groups all the jog buttons together.

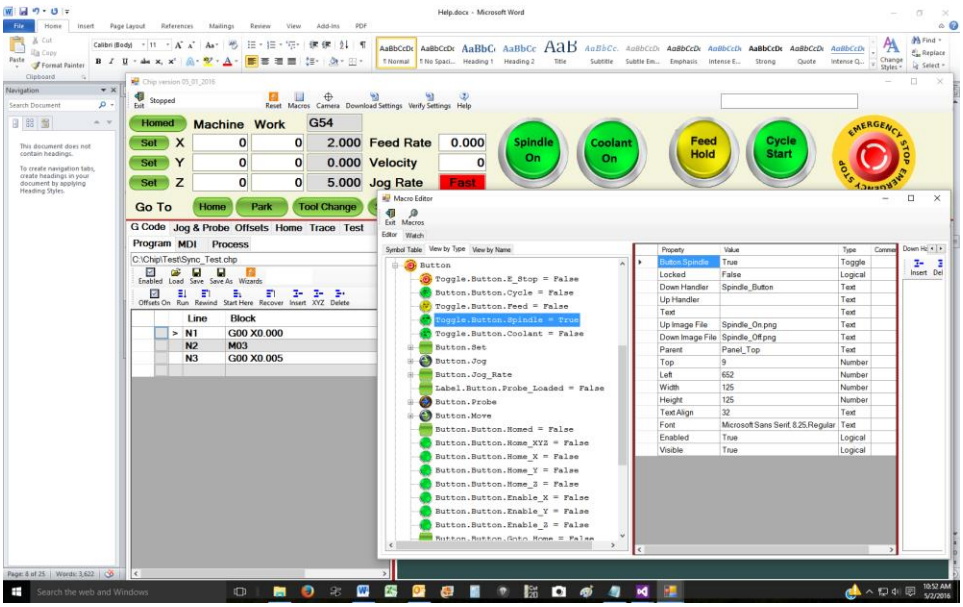
Click the View By Name tab, then click on Button, then click on Button.Jog, then click Button.Jog.X_Plus. When at the lowest level the value is shown in the tree after an ‘=’ sign, if there are comments they will be shown (in this case False because the button in not pressed) . The symbol properties associated with the button are shown pane to the right.



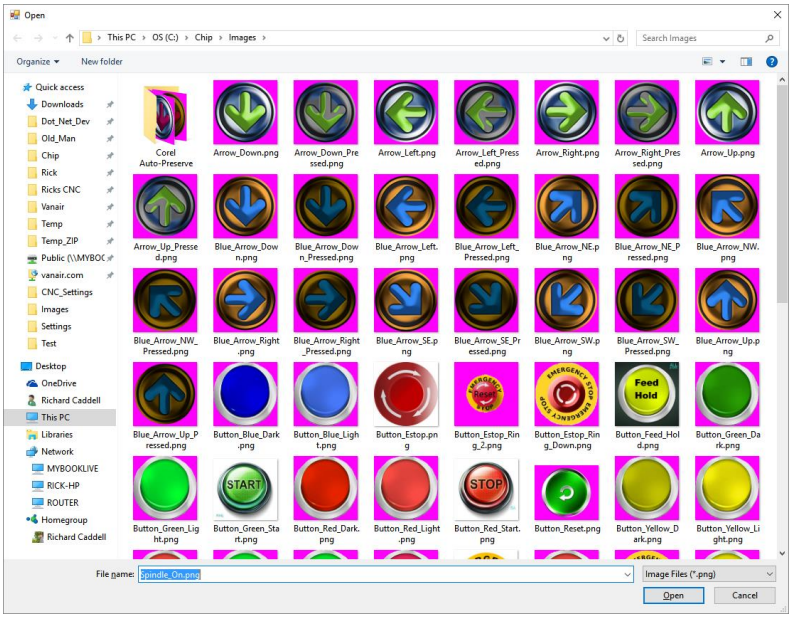
Any of these properties can be changed. The text, images the position, location, size etc. An example follows on the next page.

Open the Macro editor and navigate to the Button.Spindle symbol. Click in the node to show the properties. Position the Macro editor window so the Spindle button is exposed.

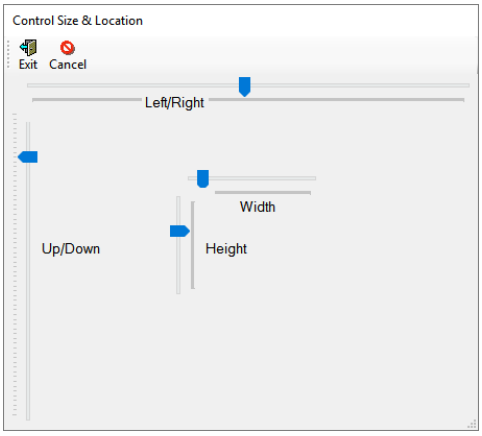
To change a property, click on the property in the properties table. Depending on which property is clicked a dialog box will appear that allows you to change a property.



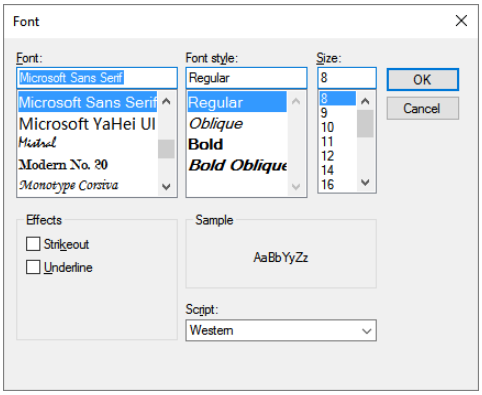
For example, the ‘Up Image’ property is the image shown when the button us up (not held down the mouse or a hot key). This is what is being currently shown. Write down the file name so you can reset it to its original image. Now click the’ Up Image’ property. A file open box will appear. Selecting a file will change the image to that. To reset, click the ‘Up Image’ property and select the original file. These files are stored in the ‘Image’ sub folder of where the .exe file is. Usually ‘C:\Chip\Images’.



When the ‘Top’, ‘Left’, ‘Width’ or ‘Height’ properties are clicked a dialog box appears. Use the track bars to position and size the button image, or text box.



Clicking on the ‘Font’ property allows changing the text font on a button or text box.



Logical (True/Properties) are changed by clicking on the box, text and numbers are just typed in.

Macros

Macros are not available yet, but …

Symbols can be used in macros. If the following line in a macro were executed, the text on the button would be changed to ‘AAA’.

Button.Jog.X_Plus.Text = ‘AAA’

In this manner, by executing macros, controls can be changed to different images, background colors, enabled and disabled, made visible or hidden.

Macros are caused to run by events, like clicking a button, or receiving a message from the TinyG controller. For example, when a status report is received from the TinyG, a macro runs that updates the axis positions (Upper left in form above). When a control is added to the form, macro event handlers can be assigned. For example, buttons have two events, button down and button up. The jog buttons have two handlers for these events, the down event handler sends a gcode command to the TinyG to move the axis and the up event handler sends feed hold.

Macros in combination with the symbol table are used to implement all control and display functions. The macro language is an extension of gcode to include branching capability. It is much like Basic in nature. A Macro editor is used to create the macros. The editor automatically formats the code and provides indentation to make the macro more readable. Errors are also displayed and the lines with errors marks. The editor also has help on things like errors and TinyG settings. TinyG settings are just symbols in the symbol table, so they can be changed and downloaded to the TinyG.

The macros can be stepped through one instruction at a time for debug purposes. The value of the variables used, both local and global are displayed.

When the jog buttons were put onto the form, the two event handlers were assigned, one for the button down event (Jog_Button_Down) and one for the button up event (Jog_Button_Up).

These macros are shown on the next page. The Macro is passed the Symbol information of the Symbol that caused the event, in this case any of the jog buttons. Pushing the button down causes the Jog_Button_Down macro to run, and releasing the button causes the Jog_Button_Up macro to run.

In both macros the first line defines the macro name and a parameter, in this case the Symbol (jog button) that caused the event.

In the Jog_Button_Down macro the next two lines are local variables. Note that they are set to an initial value. This is mandatory, it tells the interpreter what type they are. In this case G_Code = "" is a text string and Rate = Jog_Rate.Selected is a number.

The Jog_Rate is either a step distance like 0.005 or a rate like 50 in/min. The routine either sends a gcode command to the TinyG to step a distance or do a continuous jog depending on the symbol Jog_Rate.Step_Mode which is set by the Jog Rate radio buttons (see Jog Tab above).

The Jog_Button_Up macro is very simple. It does a cycle stop. Macros can also call other macros and 'Cycle_Stop' in the `Jog_Button_Up macro at the very bottom of the` page is an example of this.

The Select Symbol.Name statement is a case statement that determines which button was pressed, so all jog functions can be handled in one macro.

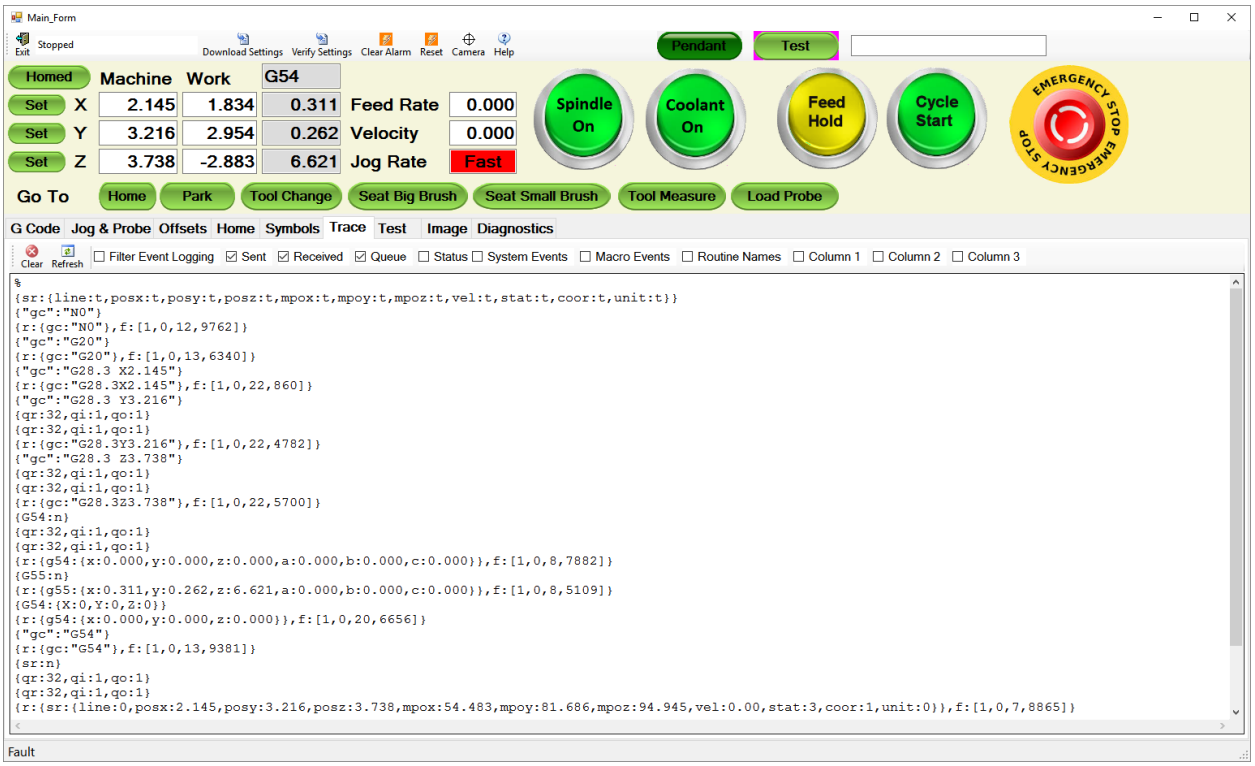
Example Macros For Jog Button Events

```
Macro Jog_Button_Down(Symbol)
  G_Code = ""
  Rate = Jog_Rate.Selected

  If Chip.Inhibited Then Exit

  If Jog_Rate.Step_Mode = True Then 'Step

    Select Symbol.Name
      Case "Button.Jog.X_Plus"
        G_Code = "G01 X" & CNC.Abs_X + Rate
       Case "Button.Jog.X_Minus"
         G_Code = "G01 X" & Sys.Abs_X - Rate
       Case "Button.Jog.Y_Plus"
         G_Code = "G01 Y" & Sys.Abs_Y + Rate
       Case "Button.Jog.Y_Minus"
         G_Code = "G01 Y" & Sys.Abs_Y - Rate
       Case "Button.Jog.Z_Plus"
         G_Code = "G01 Z" & Sys.Abs_Z + Rate
       Case "Button.Jog.Z_Minus"
         G_Code = "G01 Z" & Sys.Abs_Z - Rate
    End Select

    G_Code = G_Code & " F15"

  Else 'Continuous Jog

    Select Case Symbol.Name
      Case "Button.Jog.X_Plus"
        G_Code = "G90 G54 G01 X" & Sys.Max_X
      Case "Button.Jog.X_Minus"
        G_Code = "G90 G54 G01 X" & Sys.Min_X
      Case "Button.Jog.Y_Plus"
        G_Code = "G90 G54 G01 Y" & Sys.Max_Y
      Case "Button.Jog.Y_Minus"
         G_Code = "G90 G54 G01 Y" & Sys Min_Y
      Case "Button.Jog.Z_Plus"
         G_Code = "G90 G54 G01 Z" & Sys Max_Z & " F" & Jog_Rate
      Case "Button.Jog.Z_Minus"
         G_Code = "G90 G54 G01 Z" & Sys.Min_Z
    End Select
    G_Code = G_Code & " F15"

  End If

  Queue_GCode(G_Code)
End Macro


Macro Jog_Button_Up(Symbol)
  Cycle_Stop
End Macro
```

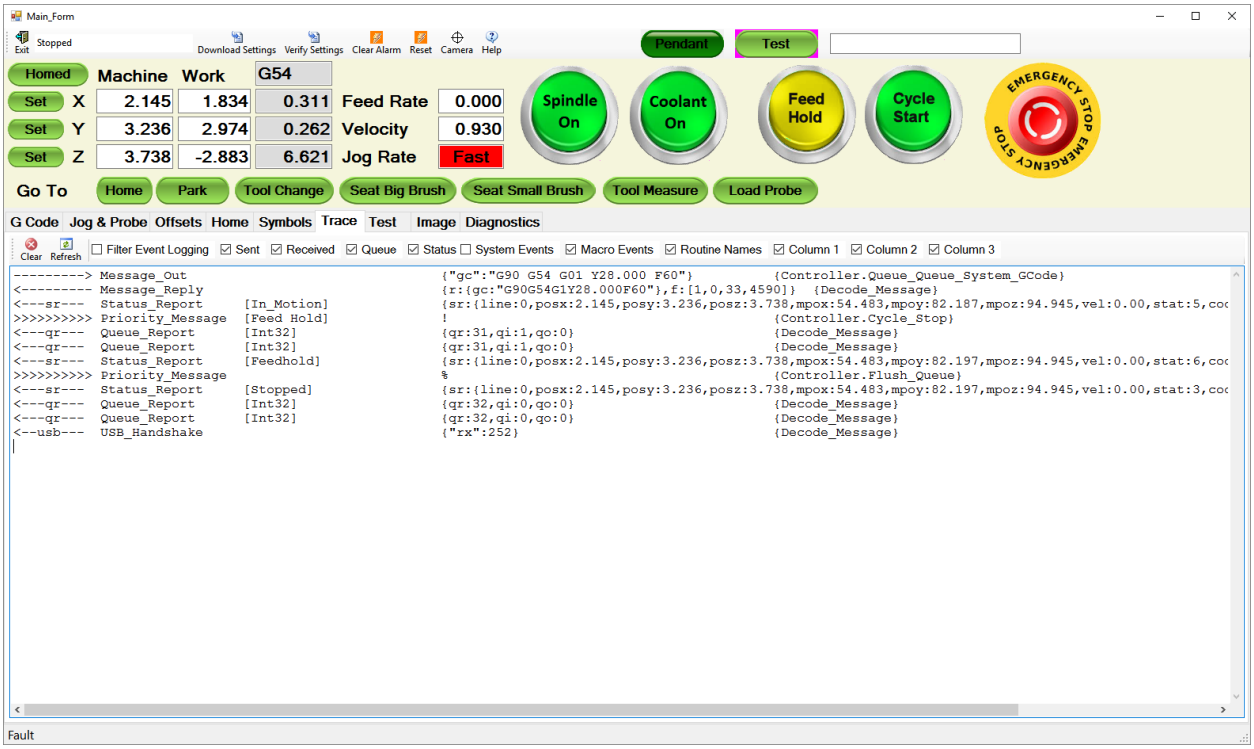
Tracing Messages and Events

The Trace tab is used to debug macros and in application development. All events like messages to and from the TinyG, macros that have run and other information is shown. The last 200 events are saved but the number of events saved is changeable. 200 is usually plenty.

Because there is so much information to see, there are filter check boxes above the trace window. The ones shown below enable the sent, received and queue report messages to/from the TinyG. The filters only affect what is shown, all events are saved in a buffer and are shown in the window depending on the filters. The Clear button can be used to clear the event buffer. This is handy if you want to capture a sequence of events.



Shown below is the capture of messages to/from the TinyG as a result of the Y Plus Jog button being pressed for a short time. The filtering is set to show everything.



The first line shows the message going to the TinyG followed by the name of the macro that sent the message (Controller.Queue_Queue_System_GCode).

---------> Message_Out {"gc":"G90 G54 G01 Y28.000 F60"} {Controller.Queue_Queue_System_GCode}

The second line shows the message coming back from the TinyG, followed by the name of the macro that handled the message event (Decode_Message).

<--------- Message_Reply {r:{gc:"G90G54G1Y28.000F60"},f:[1,0,33,4590]} {Decode_Message}

The third line shows a status report message. Machine state in this case [In_Motion], is shown enclosed in brackets.

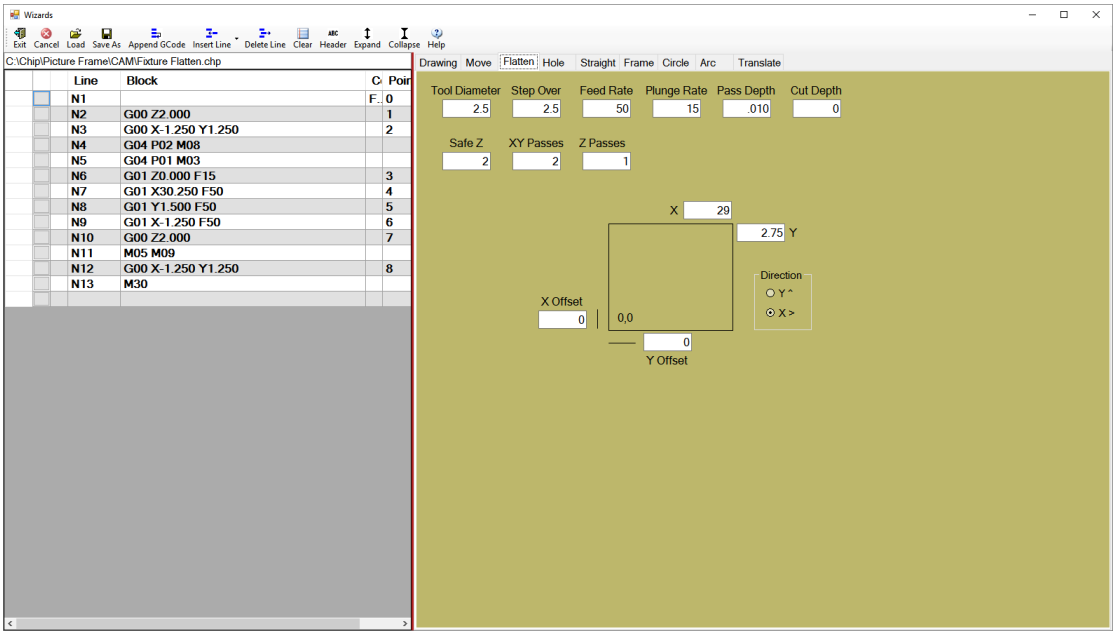
<---sr--- Status_Report [In_Motion] sr:{line:0,posx:2.145,posy:3.236,posz:3.738,mpox:54.483,mpoy:82.187,mpoz:94.945,vel:0.00,stat:5,coor:1,unit:0}} {Decode_Message}

Tracing is invaluable in debugging macros. Macros can also log events into the trace buffer, for example the value of a variable can be put into the buffer.

Wisards

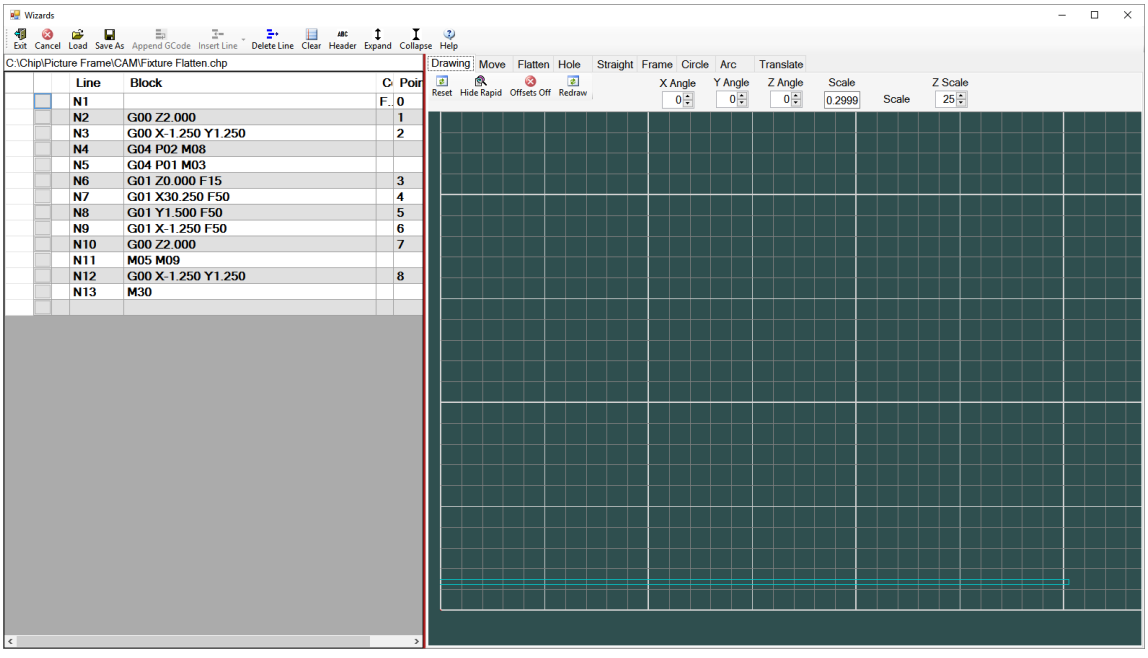
This is the wizard form that is opened whent the ‘Wizard’ button is clicked on the GCode list control

This is used to do common functions. There are several tabs, one for each function. Parameters can be changed and when the Append button is clicked the wizard adds the code to the GCode listing.



NOTE: Only a couple of the wizards work at this time, Flatten and Frame.

The first tab has a drawing control on it so you can see the result of the Wizard.



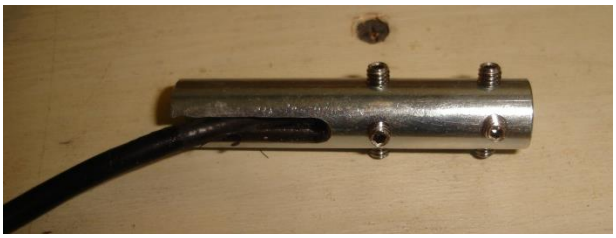
Pendant

I use a Logitech G13 game controller for a pendant. It is cheap, has a joy stick for XY jogging and buttons near the joy stick for Z jogging. I downloaded an SDK for controlling colors and the display and wrote an interface that provides information. The photo below shows the axis position and offset display. I have other displays for other functions like probing. The G7 (upper right) button changes the display as well as the functions of the other buttons. The G14 key shows help for the buttons associated with the display.

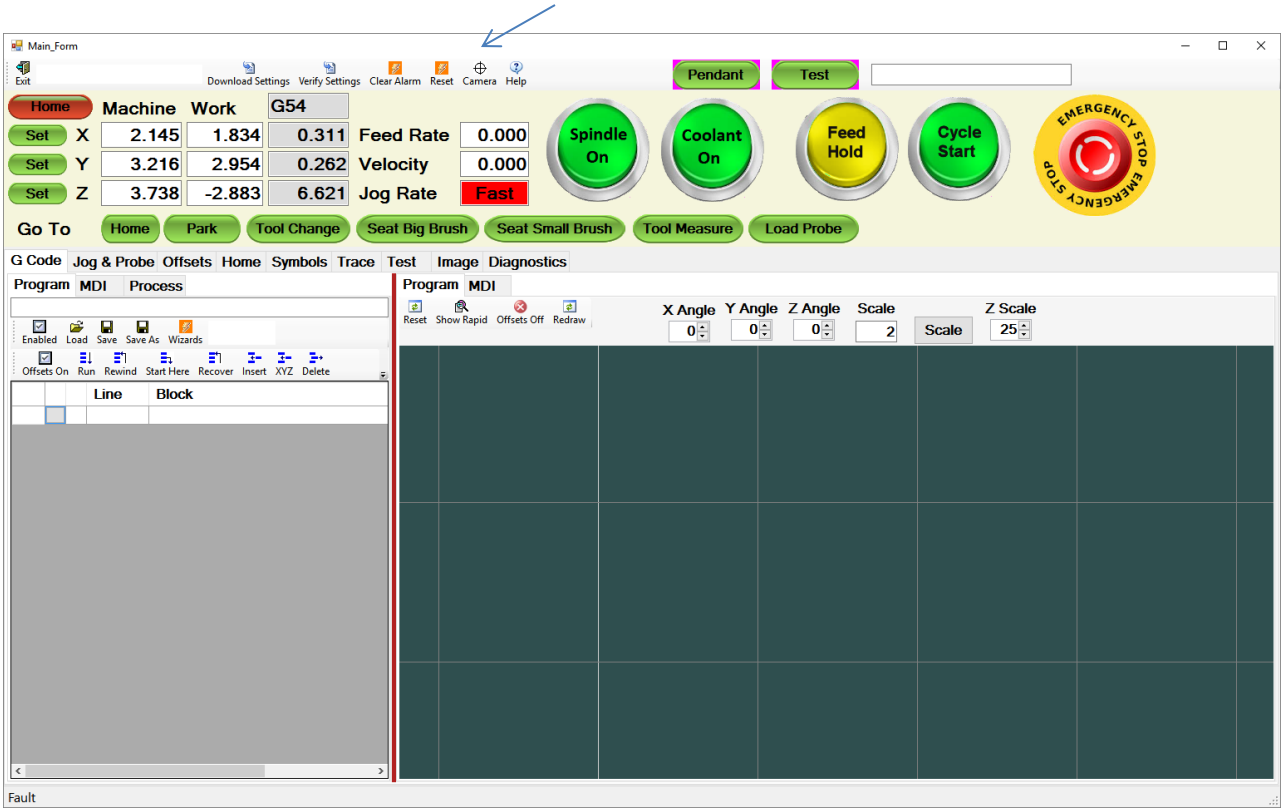
Camera

I built a camera using a piece of ½” outside diameter pipe that fits into my ½” router spindle. The camera is a bore scope with lights that fits into the pipe. I milled a slot for the cable, and drilled and tapped holes for alignment set screws.

I occasionally cut circuit boards. I like to use the blank boards with pre-drilled holes. Lining up the machine to these holes is very difficult without a camera.

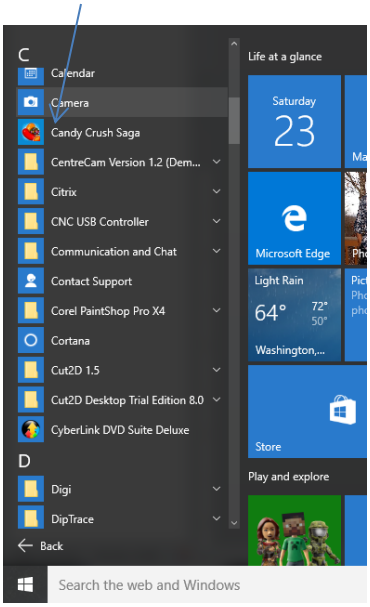


The camera interface is activated by the ‘Camera’ button on the main form tool bar.

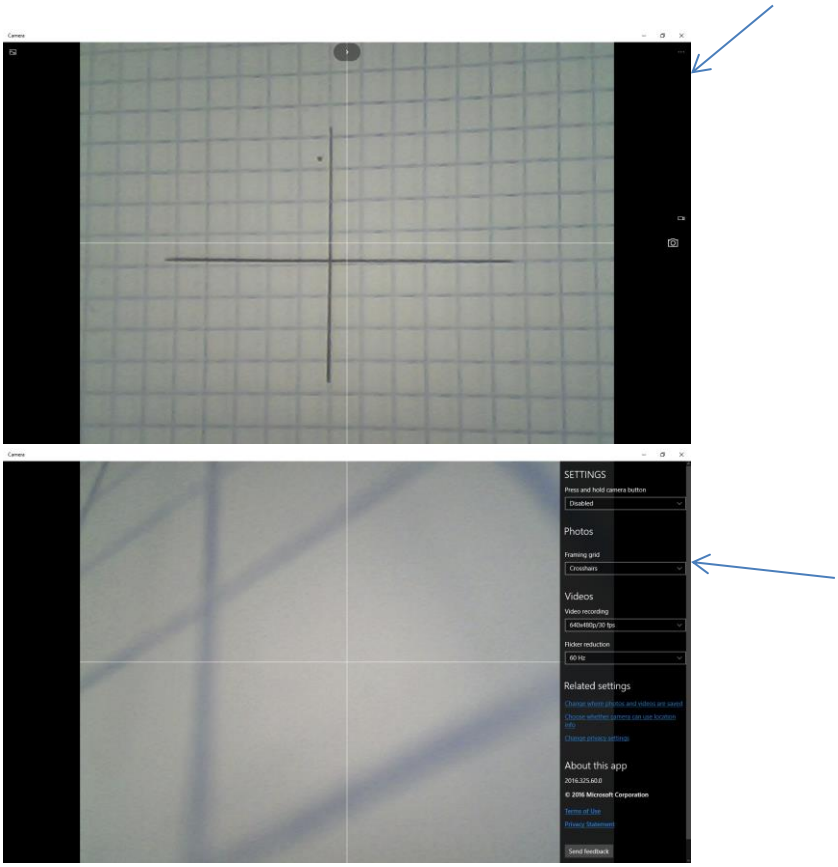


I wanted to get a Visual Studio control that handles cameras so the camera functions would be totally integrated into the application. I found one but it did not work on Windows 10. The only camera program that I found that works is the one that comes with Windows 10. What I did was make a semi-transparent form with crosshairs that overlays the Windows 10 Camera window. To use the camera you must start a camera program and click on the camera button in Chip. The camera program must be full screen so that it can be aligned with the Chip application.

To start the Windows 10 camera program, open the start menu, click the All Apps button then look for Camera.



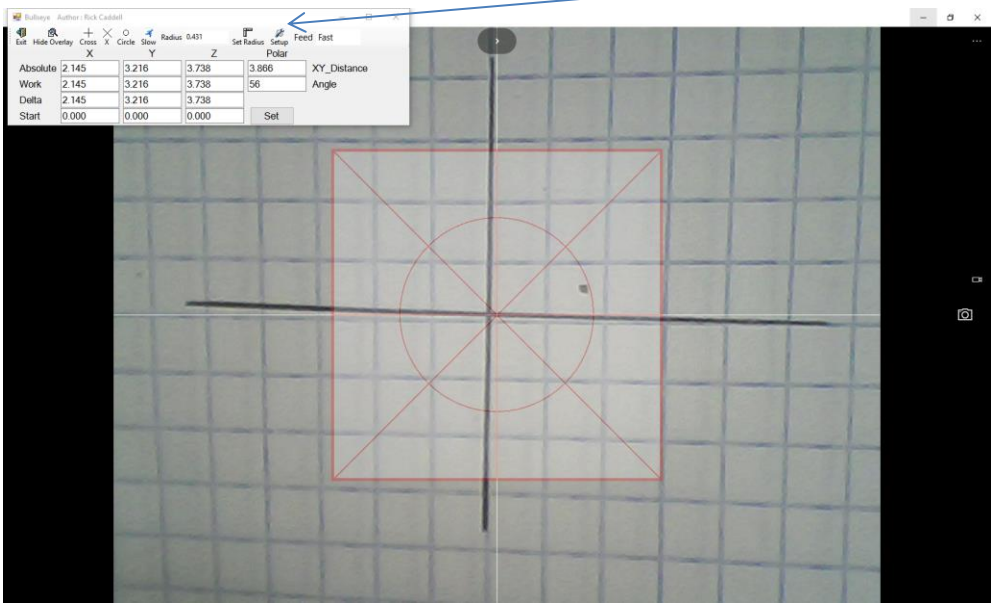
When the application opens, look for three dots in the upper left corner. They are small, dim and hard to see. Click on them, then 'Settings'



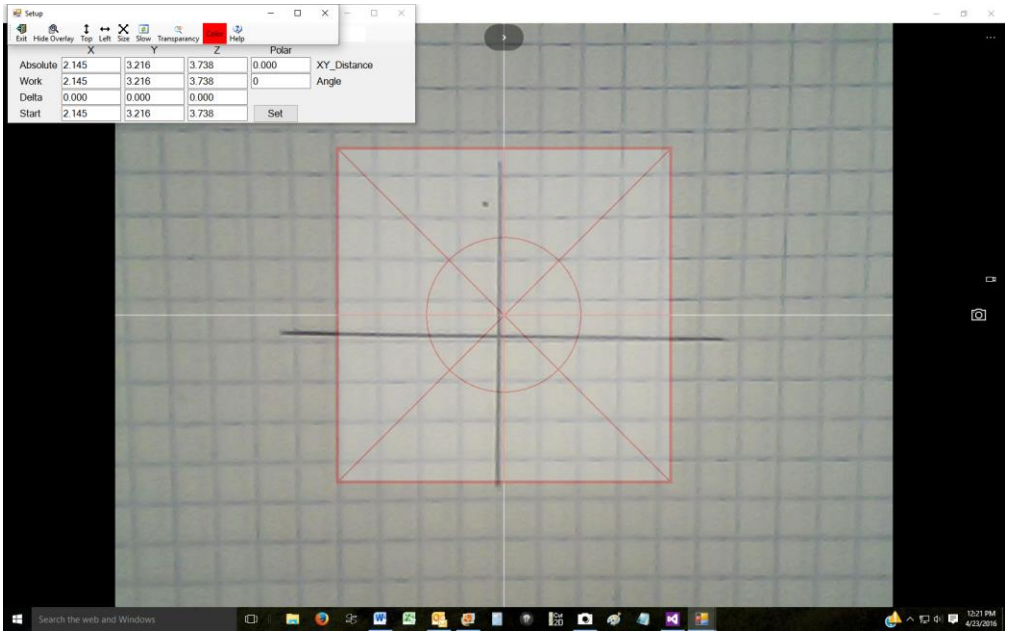
Change the Framing Grid option to Crosshairs.

Click on the Chip camera button and a semi-transparent form will overlay the camera program form.

Another form will appear in the upper left corner. This is controls for the crosshairs. The first thing to do is line the Chip crosshairs with the camera program crosshairs. Click on the ‘Setup’ button.

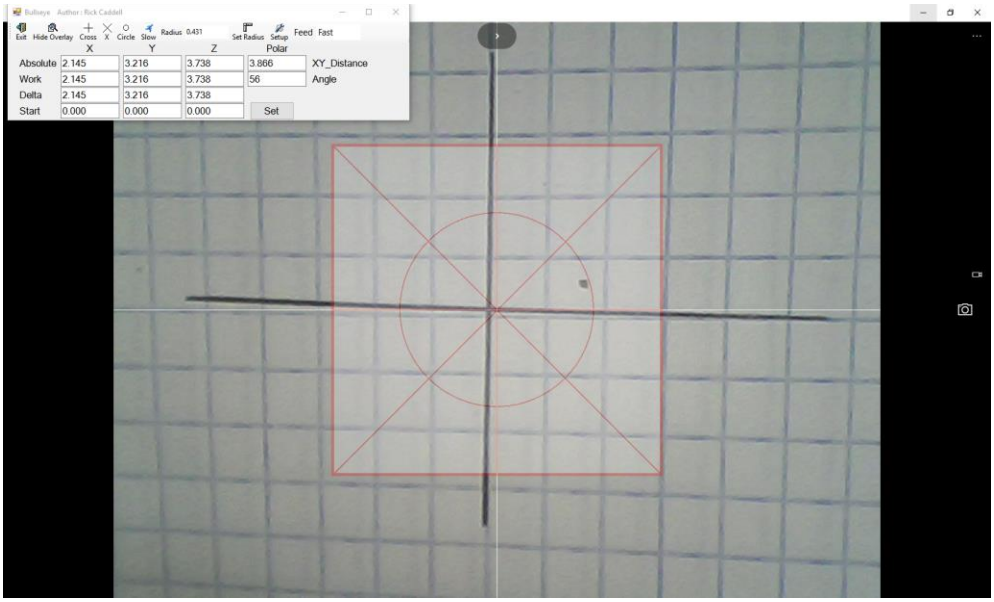


A setup form will appear. Click on the Top button and using the mouse wheel, adjust the box so the horizontal cross hairs line up. Do the same with the Left button. The Size button can be used to make the square bigger or smaller. The Slow button slows down the mouse wheel response for fine positioning. You can also set the color of the crosshairs and the transparency of the form.



While the camera function is active you can only move the machine with the pendant or hot keys because the main form will be hidden. To use the camera, click on the Chip ‘Camera’ button, then start the camera program. Move the machine to a point of interest and click the ‘Set’ button. This sets the Delta boxes to zero. When the machine moves, the Delta shows how much the machine moved.

To measure other features the radius of the circle can be used. Put a circle of known radius under the camera and using the mouse wheel adjust the circle in the crosshairs to match the test circle. Then click the Set Radius button. Type in the nominal value of the test circle. Now when the circle is made larger or smaller the number in the Radius text box will track. The Slow and Fast button can be used to slow down the movement of the mouse wheel for fine adjustment.



Under the Hood

Chip was written in Microsoft Visual Studio using Basic and the source will eventually be turned over to Synthetos. I can hear the groans (why not C). I have done a lot of C programming myself mostly at the microprocessor level, but Visual Studio Basic, besides being a higher level language, has better debugging facilities, and is easier to learn than C to the new programmer. All the same classes are available to both C and Basic.

This work was an almost constant two year effort. The communications between the application and TinyG are very well checked out. The messaging is implemented with two threads besides the main thread. These separate threads make sending and receiving messages very fast and reliable.

Macros are pre-compiled when the application starts, including reverences to symbols, and equations. This makes the macros execute quickly and if there are errors they are detected before the application starts executing macros.

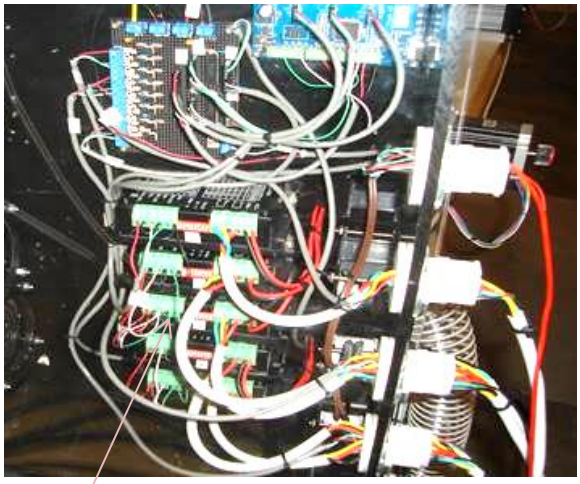
The macros and controls that are provided include many functions that most people probably would not use. When the macro editor and form editor are complete the controls can be deleted or changed as well as adding new ones. Same goes for the macros and symbols.

My Machine

I have designed and built my milling machine from scratch, electronics, software, parts ets.. The only thing I used that I did not do myself was the TinyG. All parts were machined either on a manual milling machine or a Probotix CNC mill. The machine is very rigid and accurate. Has a cutting area of 28” x 35”. It uses five stepping motors two for X, two for Y and one for Z. It is made from aluminum channel and HDPE parts. HDPE is great material (I get most of mine from Zoro Tools). It does not rust, is strong, easy to machine, can be threaded and is fairly inexpensive. All hardware is stainless steel.

The electronics include a buffer board that buffers the TinyG step and direction signals that are used to command external stepper motor drivers, an opto-isolator board to buffer and filter limit switch inputs, and a relay board with high current relays to control the router and dust collection system. I use the coolant commands and output to turn on the dust collection system.

Power supplies on one side



External Stepper Drives on the other

Plexiglass covers allow for viewing of LED’s

Back of machine and back of control cabinet. Front of cabinet has 3 fans mounted with fan guards and filters.

Picture of front of machine showing pendant, E-Stop button and router speed control (under E-Stop button). Hose at the right is connected to Jet Dust collector at one end and a brush at the spindle end.

Picture of Z axis with router, dust shoe and brush.

Note clamp in lower right hand corner.

I machine these myself. They are spiral shaped with a small handle with a counter sunk hole in the center for a ¼-20 hex head cap screw. To mount the clamp, place clamp where desired, drill a hole in the fixture for a ¼-20 tap, tap the hole, then mount the clamp.

To clamp a part, rotate the spiral up against the part then tighten the screw. Holds tight, is compact, and if you accidently machine it, it is made of HDPE so unless you hit the screw you will not break anything. They are cheap too. Works fine, lasts a long time.