Skinning Manual v1.0

Introduction

Centroid Skinning, available in CNC11 v3.15 r24+ for Mill and Lathe, allows developers to create their own "front-end" or "skin" for their application. Skinning allows developers to create their own interface for everything from simple, single screen applications to advanced multi-screen applications with 3D graphics. Skinning provides several methods for your applications to send and receive information to and from CNC11. File Based Communication, Packet Based Communication and a combination of the two.

Skinning Example

1. Download skinningbase.zip from www.centroidcnc.com

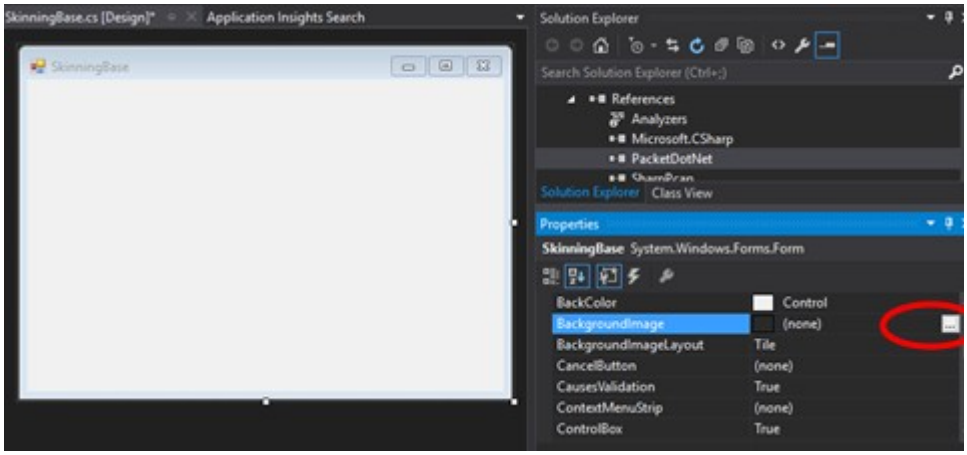
Extract the contents of the zip file to a folder in your Visual Studio 2015 projects folder named skinningbase. This is a blank Winforms project that has all the .dll's necessary to communicate with Cnc11 already included. This project requires Cnc11 v3.15r24+ in order to run

2. Open the SkinningBase project in Visual Studio 2015

In Visual Studio 2015, select File→Open→Project and select the solution in the skinningbase folder.

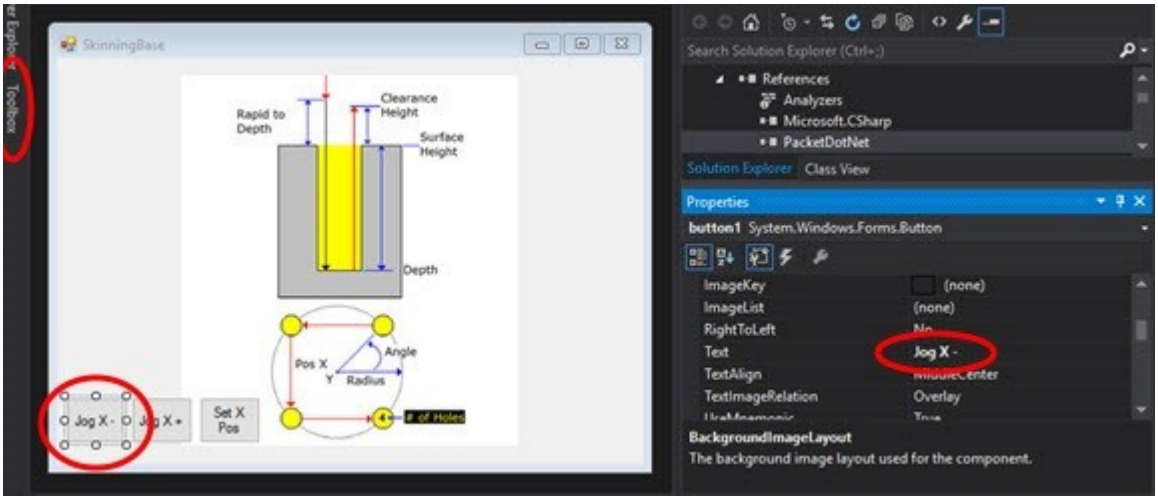
3. Set the background image for Form

When the SkinningBase project is opened, Visual Studio opens the SkinningBase Form in "Design" mode. Design mode means you can open the Toolbox and drag and drop buttons, text boxes, images, panels etc.. onto your form to use in your application. Click on the Form to select it. When a form or control is selected, the available properties and action are displayed for that object in the Properties window. A Form has many different properties available -height, width, background and foreground colors etc.. In this case, add a background image to your form by expanding Appearance tab, clicking on BackgroundImage to highlight it. Once highlighted a box will be displayed to the right with 3 periods (…) click on this, then select "Local Resource" and "Import" to browse to the background image you wish to display. You can use BackgroundImageLayout to tile, center, stretch or zoom the image as desired.



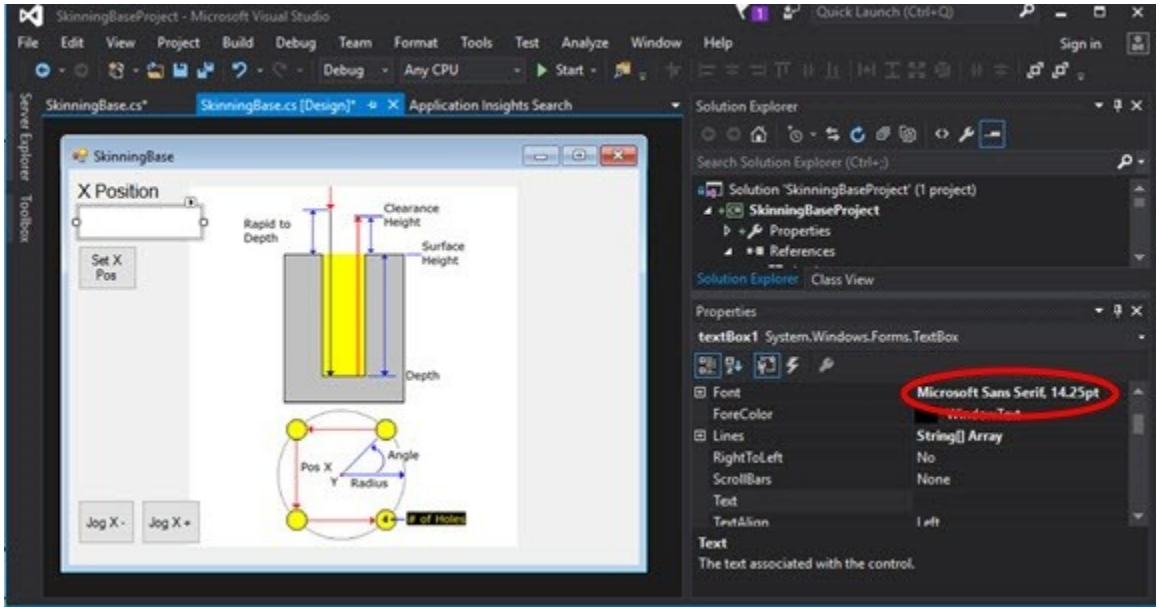
3. Add 3 buttons to Form

Select the Toolbox flyout and select “Button” then move your mouse over the Form and click drag to size and place the buttons on the form. Expand the font properties and change the “Text” property for each button to “Jog X -”, “Jog X +” and “Set X Pos”. You can also experiment with different font styles and sizes.



4. Add a TextBox and a Label

Select the Toolbox flyout and drag a TextBox and Label into the form. Expand the font properties and change the “Text” property for the label to “X Position” and change the font size to 14. For the TextBox, change the font size to 14. Move the Set X Pos up closer to the TextBox as shown below.



5. Make them do something

When a control is double clicked while in the Design view of a form, it opens up the “code behind” function for that control. This is the code that executes if a button is clicked, the text in a TextBox changes etc.. It is in this “code behind” that the Cnc11 interaction is defined.

6. Bind the text in the TextBox to the X axis position reported by Cnc11

In order to bind the content of the TextBox to the X axis position, the code below will use the Cnc11Packets function getAbsPos() was used. This function reads the network traffic between Cnc11 and the MPU11/Oak,Allin1dc to retrieve the positional data. In this example, a DispatchTimer has been defined to perform operation(s) at a defined interval. This timer, updateTimer calls the function updateTimer_Tick() at an interval of 50 milliseconds. This is the function that you would place the code that updates a DRO (like the TextBox.)

Placing the code below in the function updateTimer_Tick will update the TextBox on your form with the position of the 1st (X) axis as reported by the Cnc11Packets.getAbsPos(1) For instance:

```
*if(!textBox1.Focused() && !button3.Focused())
{
  //Get absolute encoder position of 1st axis and convert it to inches.
  double xAxisPosition = (Convert.ToDouble(Cnc11Packets.getAbsPos(1)) / 40000);

  //Create text format specifier for textbox
  string specifier = "0.0000"

  //Convert double to string and format as specified
  textBox1.Text = xAxisPosition.ToString(specifier);
}
```

*By only updating the TextBox if the user is not typing in it, the same TextBox can be used to both report the position and set the position.

6. Wire the Set X Pos button to set the X axis part position in Cnc11.

In binding the content of the TextBox to the absolute position of the X axis as reported by Cnc11, the Cnc11Packets function getAbsPos() was used. This function reads the network traffic between Cnc11 and the MPU11/Oak,Allin1dc to retrieve the positional data. Skinning also has the ability to request action from Cnc11 via file based commands. Further details on file commands and their usage can be found at the end of this document. Double click the Set X Pos button in Design mode to open the click event handler for that button. Note: Because the offset is not being added in the X axis position shown in the TextBox, you need to watch Cnc11 to see that the x position actually changed to value you entered in the TextBox

```
private void button3_Click(object sender, EventArgs e)
{
  if(File.Exist("c:\\cncm\\clientlocked.txt"))
  {
    MessageBox.Show("Cnc11 is not running or is busy");
  }
  else
  {//Tell Cnc11 a file command is coming
    using (File.Create("c:\\cncm\\clientwriting.txt")) { };

   //Create the file command loadcommand.txt and insert desired content
   string command = "G92 X" + textBox1.Text;
   FileStream loadcommand = new FileStream("c:\\cncm\\loadcommand.txt", FileMode.OpenOrCreate,
   FileAccess.Write);
   StreamWriter writer = new StreamWriter(loadcommand);
   writer.WriteLine(command);
   writer.Close();
   writer = null;
   //Tell Cnc11 the file command is ready to be processed
   File.Delete("c:\\cncm\\clientwriting.txt");
 }
}
```

6. Wire the Jog X– and Jog X+ buttons to jog the X axis.

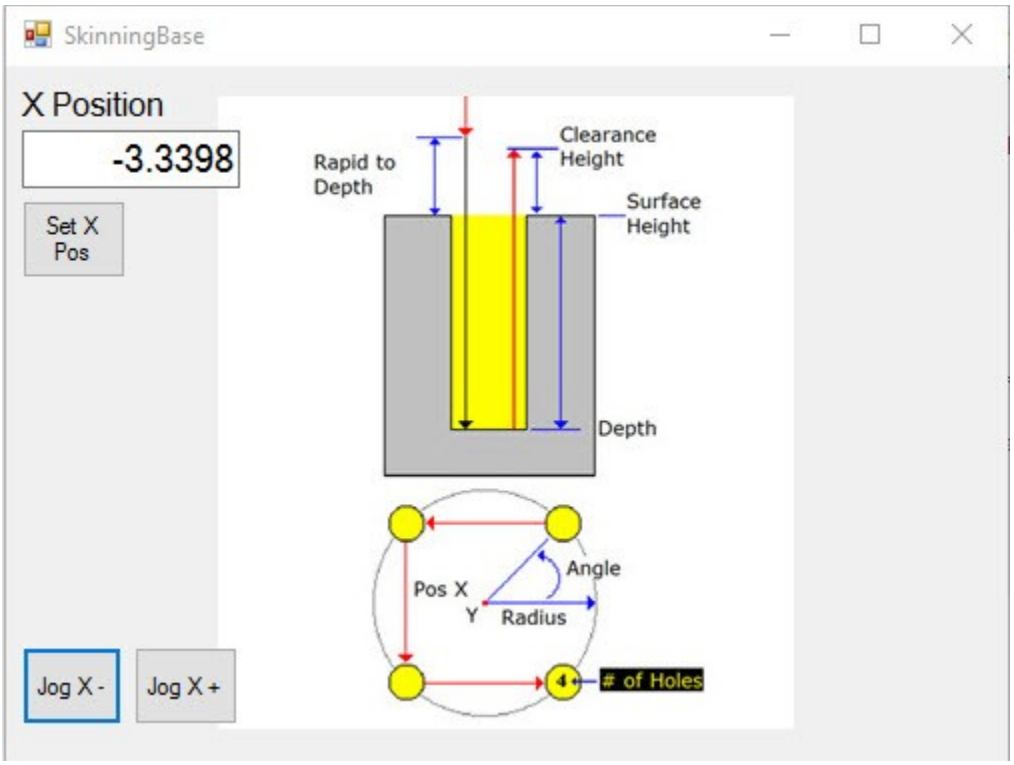
If your PLC program supports Skinning, your apps can also interact with the PLC through the packet based PLC functions. An example PLC program, Oak-skinned-mill.src is included in zip for this example program. Install it in the cncm directory and compile it to run jog the machine from the SkinningBase example. In the example project, find the event handlers for button1 and button2 MouseDown and MouseUp events and the code as shown below:

```
private void button1_MouseDown(object sender, EventArgs e)
{
  Cnc11Packets.SetMemBit(2);
}

private void button1_MouseUp(object sender, EventArgs e)
{
  Cnc11Packets.RstMemBit(2);
}

private void button2_MouseDown(object sender, EventArgs e)
{
  Cnc11Packets.SetMemBit(3);
}

private void button2_MouseUp(object sender, EventArgs e)
{
  Cnc11Packets.RstMemBit(3);
}
```



File Based Communication with CNC11

Using File Based Communication, CNC11 monitors for the existence of certain files and then responds to the existence and/content of these files. This is probably the simplest way to interact with CNC11 and can be accomplished with as little as a text editor. CNC11 v3.16 monitors for existence of these files by default and there is no action needed by the application programmer or the user to initiate this behavior.

The guidelines and procedure for using File Based Communications are as follows:

1. When CNC11 is not accepting commands from outside applications, it creates a file called clientlocked.txt in the CNC11 root directory (c:\cncm for Mill or c:\cnct for Lathe). This file on boot and at anytime that:

a) The machine has not been homed.
b) A job is in progress or MDI is active.
c) The control is in any screen other than the main screen. (Setup, Config, etc..)
d) A fault condition is present.

2. When none of these conditions are active, CNC11 deletes clientlocked.txt to indicate is ready to accept commands from outside applications.

3. Prior to creating a file based command, the application should confirm that clientwriting.txt does not exist and create clientwriting.txt in the CNC11 root directory. It is VERY important that the application create and confirm this file before creating any file commands as it Prevents CNC11 from performing file maintenance or doing other background tasks that may interfere or conflict with the actions being requested.

4. After confirming the presence of clientwriting.txt in the CNC11 root directory, the application should create any file commands that are required. If clientwiting.txt does not exist, CNC11 will attempt to process any file commands present whether the application is finished writing them or not.

5. After writing the file command(s), the application should delete clientwriting.txt to trigger processing of the file commands by CNC11.

6. CNC11 will delete all file commands after processing them and create servererror.txt if any errors occurred while processing a file command.

File Based Commands

clientwriting.txt

When an application needs to command an action from CNC11, it should first create clientwriting.txt to notify CNC11 that a file command is being prepared. This is very important as it Prevents CNC11 from performing file maintenance or doing other background tasks that may interfere or conflict with the incoming file command. This content of clientwriting.txt does not matter. The application should only create clientwriting.txt after first confirming that clientlocked.txt does not exist. After an application has created clientwriting.txt, it can now issue one or more of the following file based commands: Note: The file based command(s) are performed by CNC11 when the clientwriting.txt file is deleted.

loadjob.txt

An application can notify CNC11 to load a new G code program by creating a file called loadjob.txt which contains the path and job name that the appplication is requesting CNC11 to load. The path may be either relative to the CNC11 root directory or absolute. Both examples below instruct CNC11 to load a job named

myjob.cnc which is located in c:\cncm\ncfiles. Note: loadjob.txt only loads a job, it does not run it.

Content of loadjob.txt created in **c:\cncm** (CNC11 Mill root directory):

Using relative path: .\ncfiles\myjob.cnc

Using absolute path: c:\cncm\ncfiles\myjob.cnc

loadcommand.txt

An application can request that G or M code command, equivalent to an MDI command be performed immediately. The application should create of file named loadcommand.txt in the CNC11 root directory. The command is not case sensitive. Action(s) are performed by CNC11 when the clientwriting.txt file is deleted. The contents can be any single line of valid G or M code:

G92 X1 Y2 Z3 ;Sets current X position to 1, Y to 2 and Z to 3

NOTE: Command runs immediately and does not require cycle start to initiate action or motion!

loadconfig.xml

An application creates this file to make changes to the configuration of cnc11. The contents should be a complete, valid configuration file that includes any changes requested by the application. These changes/contents are then stored in cncmcfg.xml for mill and cnctcfg.xml for lathe by CNC11 when the clientwriting.txt file is deleted.

Loadparm.xml

An application creates this file to make changes to the parameters of cnc11. The contents should be a complete, valid parameter file that includes any changes requested by the application. These changes/contents are then stored in cncm.parm.xml for mill and cnct.parm.xml for lathe by CNC11 when the clientwriting.txt file is deleted.

servererror.txt

CNC11 writes any errors that occurred while processing file commands to this file. NOTE: File commands received while clientlocked.txt exists are ignored and no errors are recorded.

Packet Based Communication with CNC11

Packet Based Communication with CNC11 is a very powerful method of interfacing with CNC11. It allows application programmers to create modern, graphical applications applications in Visual Studio in any number of languages including C#, Visual Basic and C++. An easy to use API is provided allowing any application developed in these languages to asynchronously send and receive data with CNC11. The API allows interfacing with the PLC, getting positional information from up to 8 axes, loading jobs, changing CNC11 configurations etc.. This is definitely the most powerful and flexible way to interface your application with CNC11.

Packet Based Functions:

PLC functions:

bool GetInputStatus(int)

GetInputStatus() passes an integer representing the input # the application is requesting the status (state) of.. GetInputStatus() returns ***true*** if the input is closed as viewed in CNC11, ***false*** if the input is open as viewed in CNC11. Note: By "As viewed in CNC11", it is meant that it will return the boolean value of the input ***after*** CNC11 has applied any software inversions or forcing. An input that is electrically open, but has been inverted by CNC11 will return ***true.*** Example usage:

```
bool myInputStatus = GetInputStatus(11);  //Gets status of input 11 as viewed in CNC11
```

bool GetMemBitStatus(int)

GetMemBitStatus() passes an integer representing the memory bit # the application is requesting the status (state) of.. GetMemBitStatus() returns ***true*** if the membit is SET, (1) as viewed in CNC11, ***false*** if the membit is RST (0) as viewed in CNC11. Note: By "As viewed in CNC11", it is meant that it will return the boolean value of the membit ***after*** CNC11 has applied any software forcing. An membit that is has been forced by CNC11 will return ***true*** regardless of what the logic the PLC program tried to set it to. Example usage:

```
bool myMemBitStatus = GetMemBitStatus(20);  //Gets status of membit as viewed in CNC11
```

SetMemBit(int)*

SetMemBit() passes an integer representing the memory bit # the application wishes to SET the state of.. *This function requires a PLC program to support it. (example included) Example usage:

```
SetMemBit(20);  //SETS (1) membit 20*
```

Packet Based Functions:(cont)

PLC functions:(cont)

RstMemBit(int)*

RstMemBit() passes an integer representing the memory bit # the application wishes to RST the state of.. *This function requires a PLC program to support it. (example included) Example usage:

```
RstMemBit(20);  //RST's (0) membit 20*
```

bool GetOutputStatus(int)

GetOutputStatus() passes an integer representing the memory bit # the application is requesting the status (state) of.. GetOutputStatus() returns ***true*** if the Output is SET, (1) as viewed in CNC11, ***false*** if the Output is RST (0) as viewed in CNC11. Note: By "As viewed in CNC11", it is meant that it will return the boolean value of the Output ***after*** CNC11 has applied any software forcing. An Output that is has been forced by CNC11 will return ***true*** regardless of what the logic the PLC program tried to set it to. Example usage:

bool myOutputStatus = GetOutputStatus(20); //Gets status of Output as viewed in CNC11

SetOutput(int)*

SetOutput() passes an integer representing the Output # the application wishes to SET the state of. *This function requires a PLC program to support it. (example included) Example usage:

SetOutput(20); //SETS (1) Output 20*

RstOutput(int)*

RstOutput() passes an integer representing the Output # the application wishes to RST the state of. *This function requires a PLC program to support it. (example included) Example usage:

RstOutput(20); //RST's (0) Output 20*

int GetWordValue(*int wordNumber*)

GetWordValue() returns the value of 32bit integer PLC word #(W1-W128) specified by *int wordNumber* as passed. (See ***SetSkinningDataWord()*** to set the value of a 32 bit PLC integer word)

int GetDWordValue(*int wordNumber*)

GetDWordValue() returns the value of 64bit integer PLC word #(DW1-DW128) specified by *int wordNumber* as passed. (No function is available to set all 64 bits of a the value of a 64 bit integer PLC word. See ***SetSkinningDataWord()*** to set 32 bits of the value of a 64 bit PLC word)

float GetFWordValue(*int wordNumber*)

GetFWordValue() returns the value of 32bit float PLC word # (FW1-FW128) specified by *int wordNumber* as passed. (See ***SetSkinningDataFloatWord()*** to set the value of a 32 bit PLC float word)

float GetDFWordValue(*int wordNumber*)

GetDFWordValue() returns the value of 64bit float PLC word # (DFW1-DW128) specified by *int wordNumber* as passed. (See ***SetSkinningDataDoubleFloatWord()*** to set the value of a 64 bit PLC float word)

SetSkinningDataWord(*int index, int32 value, bool sendImmediately = true*)

Usage: Where ***index*** = # in SV_SKINNING_DATA_W#, ***value*** = value to set SV_SKINNING_DATA_W# to and ***sendImmediately*** = send data to PLC on function call.

CNC11 v3.16+ provides 12 system variables, SV_SKINNING_DATA_W1-SV_SKINNING_DATA_W12, to allow skinning to pass a 32 bit word values into the PLC. They are not directly mapped to any of the 32 bit word values (W1-W128) in the PLC and must be explicitly mapped to PLC word values in the PLC program if so desired.

PLC program mapping of a 32 bit integer PLC word value to a SV_SKINNING_DATA_W# value to allow skinning to set the value of the PLC word.
Example, assuming the PLC program has declared MyWordValue_W IS W10:

PLC program statement:

IF TRUE THEN MyWordValue_W = SV_SKINNING_DATA_W1

Set PLC W10 (MyWordValue) value = 65536 when Skinning calls SetSkinningDateWord(1, 65536, true);.

Additionally, the PLC examples referenced by this document, utilize SV_SKINNING_DATA_W12 for the calls to SetMemBit(), Rst MemBit(), SetOutput and RstOutput().

SetSkinningDoubleDataWord(*int index, int32 value, bool sendImmediately = true*)

Usage: Where ***index*** = # in SV_SKINNING_DOUBLE_DATA_W#, ***value*** = value to set SV_SKINNING_DOUBLE_DATA_W# to and ***sendImmediately*** = send data to PLC on function call.

CNC11 v3.16+ provides 11 system variables, SV_SKINNING_DOUBLE_DATA_W1-SV_SKINNING_DOUBLE_DATA_W12, to allow skinning to pass a 64 bit word values into the PLC. They are not directly mapped to any of the 64 bit word values (W1-W128) in the PLC and must be explicitly mapped to PLC word values in the PLC program if so desired.